**FFConv: Fast Factorized Neural Network Inference on Encrypted Data**

Yuxiao Lu †  Jie Lin †  Chao Jin †  Zhe Wang †  Khin Mi Mi Aung †  Xiaoli Li †

**Abstract**

Homomorphic Encryption (HE), allowing computations on encrypted data (ciphertext) without decrypting it first, enables secure but prohibitively slow Neural Network (HENN) inference for privacy-preserving applications in clouds. To reduce HENN inference latency, one approach is to pack multiple messages into a single ciphertext in order to reduce the number of ciphertexts and support massive parallelism of Homomorphic Multiply-Add (HMA) operations between ciphertexts. However, different ciphertext packing schemes have to be designed for different convolution layers and each of them introduces overheads that are far more expensive than HMA operations. In this paper, we propose a low-rank factorization method called FFConv to unify convolution and ciphertext packing. To our knowledge, FFConv is the first work that is capable of accelerating the overheads induced by different ciphertext packing schemes simultaneously, without incurring a significant increase in noise budget. Compared to prior art LoLa and Falcon, our method reduces the inference latency by up to 87% and 12%, respectively, with comparable accuracy on MNIST and CIFAR-10.

**1. Introduction**

Homomorphic Encryption (HE) (Gentry, 2009; Brakerski and Vaikuntanathan, 2011; Fan and Vercauteren, 2012) is one of the promising cryptographic systems that enable secure Neural Network inference for privacy-preserving applications in clouds while at the cost of high inference latency. At the client, the plaintext data is encrypted in the form of ciphertext, then transmitted to the cloud server. At the cloud, neural network inference is evaluated homomorphically on the ciphertexts to generate an encrypted prediction. The encrypted prediction is returned to the client for decryption. Since the cloud cannot encrypt or decrypt the data, the data privacy is protected. Despite the high level of security, the HE-enabled Neural Network (HENN) inference is prohibitively slow, mainly due to a large number of ciphertexts and the expensive Homomorphic Multiply-Add (HMA) operations on the ciphertexts. For instance, the inference latency of a shallow network (1 convolution layer and 2 fully connected layers) on one encrypted 28x28 MNIST image (LeCun et al., 1998) is more than 200 seconds on multi-core CPUs (Dowlin et al., 2016).

Modern HE cryptographic systems (Fan and Vercauteren, 2012; Brakerski et al., 2014; Cheon et al., 2017; Brutzkus et al., 2019) used packed encryption to accelerate the HENN inference, in which the ciphertext structure is configured as a vector of slots, and each slot encrypts a different message. Therefore, packing can significantly reduce the number of ciphertexts required to encrypt a given amount of data messages. In addition, packing also enables massive parallel execution of the HMA operations between ciphertexts, as per the Single-Instruction Multiple-Data (SIMD) execution model (Smart and Vercauteren, 2014). When multiplying (resp. adding) two packed ciphertexts, it is equivalent to concurrent slot-wise multiplications (resp. additions) of the underlying vectors of the two ciphertexts. In particular, LoLa (Brutzkus et al., 2019) introduced several ciphertext packing schemes, which reduced the inference latency to around 2 seconds for the shallow network on MNIST (Dowlin et al., 2016).

Despite the faster HENN inference, ciphertext packing schemes introduce expensive operations by themselves, which prolong the inference latency of HENN with increased network width and depth for better performance on larger problems. As summarized in LoLa (Brutzkus et al., 2019), there are two major packed representations named Dense Packing (DensePack) and Convolution Packing (ConvPack), in which DensePack requires a Rotation operation to align the slots inside ciphertexts before each addition and ConvPack requires Im2Col (a combination of HMA and Rotation operations) to reorganize ciphertexts for transition between layers. Compared to HMA operations, the rotations are 10x more expensive (Lou et al., 2020). For instance, LoLa (Brutzkus et al., 2019) reported a 3-layer network inference (2 convolution layers and 1 fully packed prediction. The encrypted prediction is returned to the client for decryption. Since the cloud cannot encrypt or decrypt the data, the data privacy is protected. Despite the high level of security, the HE-enabled Neural Network (HENN) inference is prohibitively slow, mainly due to a large number of ciphertexts and the expensive Homomorphic Multiply-Add (HMA) operations on the ciphertexts. For instance, the inference latency of a shallow network (1 convolution layer and 2 fully connected layers) on one encrypted 28x28 MNIST image (LeCun et al., 1998) is more than 200 seconds on multi-core CPUs (Dowlin et al., 2016).

Modern HE cryptographic systems (Fan and Vercauteren, 2012; Brakerski et al., 2014; Cheon et al., 2017; Brutzkus et al., 2019) used packed encryption to accelerate the HENN inference, in which the ciphertext structure is configured as a vector of slots, and each slot encrypts a different message. Therefore, packing can significantly reduce the number of ciphertexts required to encrypt a given amount of data messages. In addition, packing also enables massive parallel execution of the HMA operations between ciphertexts, as per the Single-Instruction Multiple-Data (SIMD) execution model (Smart and Vercauteren, 2014). When multiplying (resp. adding) two packed ciphertexts, it is equivalent to concurrent slot-wise multiplications (resp. additions) of the underlying vectors of the two ciphertexts. In particular, LoLa (Brutzkus et al., 2019) introduced several ciphertext packing schemes, which reduced the inference latency to around 2 seconds for the shallow network on MNIST (Dowlin et al., 2016).

Despite the faster HENN inference, ciphertext packing schemes introduce expensive operations by themselves, which prolong the inference latency of HENN with increased network width and depth for better performance on larger problems. As summarized in LoLa (Brutzkus et al., 2019), there are two major packed representations named Dense Packing (DensePack) and Convolution Packing (ConvPack), in which DensePack requires a Rotation operation to align the slots inside ciphertexts before each addition and ConvPack requires Im2Col (a combination of HMA and Rotation operations) to reorganize ciphertexts for transition between layers. Compared to HMA operations, the rotations are 10x more expensive (Lou et al., 2020). For instance, LoLa (Brutzkus et al., 2019) reported a 3-layer network inference (2 convolution layers and 1 fully packed prediction. The encrypted prediction is returned to the client for decryption. Since the cloud cannot encrypt or decrypt the data, the data privacy is protected. Despite the high level of security, the HE-enabled Neural Network (HENN) inference is prohibitively slow, mainly due to a large number of ciphertexts and the expensive Homomorphic Multiply-Add (HMA) operations on the ciphertexts. For instance, the inference latency of a shallow network (1 convolution layer and 2 fully connected layers) on one encrypted 28x28 MNIST image (LeCun et al., 1998) is more than 200 seconds on multi-core CPUs (Dowlin et al., 2016).

Modern HE cryptographic systems (Fan and Vercauteren, 2012; Brakerski et al., 2014; Cheon et al., 2017; Brutzkus et al., 2019) used packed encryption to accelerate the HENN inference, in which the ciphertext structure is configured as a vector of slots, and each slot encrypts a different message. Therefore, packing can significantly reduce the number of ciphertexts required to encrypt a given amount of data messages. In addition, packing also enables massive parallel execution of the HMA operations between ciphertexts, as per the Single-Instruction Multiple-Data (SIMD) execution model (Smart and Vercauteren, 2014). When multiplying (resp. adding) two packed ciphertexts, it is equivalent to concurrent slot-wise multiplications (resp. additions) of the underlying vectors of the two ciphertexts. In particular, LoLa (Brutzkus et al., 2019) introduced several ciphertext packing schemes, which reduced the inference latency to around 2 seconds for the shallow network on MNIST (Dowlin et al., 2016).

Despite the faster HENN inference, ciphertext packing schemes introduce expensive operations by themselves, which prolong the inference latency of HENN with increased network width and depth for better performance on larger problems. As summarized in LoLa (Brutzkus et al., 2019), there are two major packed representations named Dense Packing (DensePack) and Convolution Packing (ConvPack), in which DensePack requires a Rotation operation to align the slots inside ciphertexts before each addition and ConvPack requires Im2Col (a combination of HMA and Rotation operations) to reorganize ciphertexts for transition between layers. Compared to HMA operations, the rotations are 10x more expensive (Lou et al., 2020). For instance, LoLa (Brutzkus et al., 2019) reported a 3-layer network inference (2 convolution layers and 1 fully packed prediction. The encrypted prediction is returned to the client for decryption. Since the cloud cannot encrypt or decrypt the data, the data privacy is protected. Despite the high level of security, the HE-enabled Neural Network (HENN) inference is prohibitively slow, mainly due to a large number of ciphertexts and the expensive Homomorphic Multiply-Add (HMA) operations on the ciphertexts. For instance, the inference latency of a shallow network (1 convolution layer and 2 fully connected layers) on one encrypted 28x28 MNIST image (LeCun et al., 1998) is more than 200 seconds on multi-core CPUs (Dowlin et al., 2016).
connected layer) on one encrypted 32x32x3 CIFAR-10 image (Krizhevsky, 2009) takes over 700 seconds, in which the rotations account for over 90% of the time.

In this paper, we introduce a low-rank matrix factorization method called FFConv to unify convolution/fully-connected layers and ciphertext packing, which enables significantly reduced inference latency. We summarize our contributions as follows:

- We introduce a low-rank factorization framework designed for accelerating neural network inference on encrypted data without imposing any constraints on the configurations of convolution layers.
- We propose Factorized Packing (FactPack) to seamlessly integrate the factorized convolutions with different ciphertext packing schemes in order to maximize the reduction of overheads introduced by packing.
- To our knowledge, FFConv is the first work that is capable of reducing the overheads induced by different ciphertext packing schemes simultaneously.
- Compared to prior art LoLa and Falcon, our method reduces the inference latency by up to 87% and 12% respectively, with comparable accuracy on MNIST and CIFAR-10. Moreover, our method incurs significantly less noise budget than the most recent work Falcon.

Remarks. Since HE can only support multiplication and addition operations, non-linear layers in the modern neural network such as the Rectified Linear Unit (ReLU) (Glorot et al., 2011) is not supported by HE. To address this issue, CryptoNets (Dowlin et al., 2016) proposed to approximate ReLU with a simple Square function. There are also secure neural network inference solutions that combine HE with secure multi-party computation (MPC) techniques (Liu et al., 2017; Juvekar et al., 2018). These solutions usually use MPC to evaluate non-linear activation function, thus eliminating the need of polynomial approximation for activation function. However, they also incur high communication cost among multiple parties and require each party to have considerable computation power and be constantly online, rendering them less attractive than the non-interactive HE solutions in many use scenarios. In this paper, we focus on secure neural network inference enabled by HE only.

2. Homomorphic Encryption (HE)

HE (Gentry, 2009) has always been an intriguing technology due to its ability of computing on encrypted data in the absence of the decryption key. In HE, the plaintexts and ciphertexts are elements in polynomial rings. HE provides the user with two main computational operations on ciphertexts: homomorphic multiplication and homomorphic addition. These operations can manipulate ciphertexts and produce encrypted results that are equivalent to the corresponding plaintext results after decryption.

HE ciphertexts conceal plaintext messages with noise that can be identified and removed with the secret key (Brakerski and Vaikuntanathan, 2011). The noise magnitude can be accumulated inside a ciphertext along with the computation on it. As long as the noise is below a certain threshold that is controlled by the encryption parameters, decryption can filter out the noise and retrieve the plaintext message successfully; otherwise, the plaintext message could be corrupted and decryption could fail. Although HE schemes include a primitive (known as bootstrapping) to reduce the noise inside ciphertexts (Gentry, 2009), it is extremely computationally intensive. Instead, a more practical way is to carefully select the encryption parameters to provide just enough noise budget for a ciphertext to accommodate a predefined maximum depth of computation under a specific evaluation circuit.

In this work, following LoLa (Brutzkus et al., 2019) and Falcon (Lou et al., 2020), we employ the Brakerski-Fan-Vercauteren (BFV) HE scheme (Fan and Vercauteren, 2012). The BFV scheme is governed by three important parameters: $t$, $Q$, and $N$. First, the plaintext space is controlled by the plaintext coefficient modulus $t$. To prevent computational overflows, $t$ needs to be set to be large enough to accommodate any intermediate result of the homomorphic evaluations. Second, the scheme imposes a limit on the number of homomorphic operations that can be performed on the ciphertext before decryption fails. We refer to this limit as the computation noise budget, which can be controlled by the ciphertext coefficient modulus $Q$. Third, the underlying ring dimension $N$ is set to guarantee the targeted security level $\lambda$. For typical security requirements in practical applications, $\lambda$ is set to a minimum of 128 bits. We remark that the choice of $N$ and $Q$ significantly impact the performance of HE schemes in terms of computational and memory requirements. They also affect the data expansion rate due to encryption. More specifically, the ciphertext size can be estimated to be at least $2 * N * \log_2 Q$ bits.

3. HE-enabled Neural Network Inference

Applying HE to convolutional neural networks for private inference poses unique challenges. Typical CNNs are composed of linear and non-linear function blocks. Linear function blocks like Convolution, Full-connected, and Average Pooling layers, can be converted into matrix operations with simple additions and multiplications. On the contrary, non-linear function blocks usually contain complex or comparison operations such as ReLU layer (Glorot et al., 2011), which cannot be supported by HE directly. To enable compatibility with HE primitives, there is a need to approxi-
mate these non-linear operations with polynomial functions with only additions and multiplications. For instance, CryptoNets (Dowlin et al., 2016) suggested using the Square function to approximate ReLU in the network for classifying MNIST images.

In a typical Machine Learning as a Service (MLaaS) scenario, network models are deployed in cloud servers to provide inference services to client users. We assume the models are kept in plaintext, while the client users encrypt their data into HE ciphertexts before sending them to the cloud server for private inference. In the next, we introduce the ways to compute convolution layers and other linear layers in HENN with plaintext weights and ciphertext inputs. Pay attention to the fact that other linear layers like fully-connected and average-pooling layers can be treated as special cases of convolution layers, where for fully-connected layers, the filter sizes are the same as the input tensor sizes, and for average-pooling layers, the weights inside a single filter are set to be the same constant value.

### 3.1. Convolution Layer as Matrix Multiplication

Convolution layer is essentially dot product operations between the filter weights and local patches cropped from the input. Assume the weights of a regular Convolution layer are 4D tensor $\mathbf{Y} \in \mathbb{R}^{d \times d \times I_c \times O_c}$ with kernel size $d$, input channel $I_c$, and output channel $O_c$, the input and output of the Convolution layer is 3D tensor $\mathbf{X} \in \mathbb{R}^{I_w \times I_h \times I_c}$ and $\mathbf{Z} \in \mathbb{R}^{O_w \times O_h \times O_c}$, where $I_w, I_h, O_w, O_h$ are input/output width and height respectively. Similar to the Fully Connected layer, the convolution operation $\mathbf{Z} = \mathbf{X} \ast \mathbf{Y}$ of a Convolution layer can be formulated as matrix multiplication as follows:

$$\mathbf{Z} = \mathbf{I} \times \mathbf{W}, \quad \text{(1)}$$

where $\mathbf{I} \in \mathbb{R}^{S \times K}$ is a matrix with $S = O_w O_h$ rows and $K = d^2 I_c$ columns, each row of $\mathbf{I}$ is a vector stretched out from a 3D patch $\mathcal{R}^{d \times d \times I_c}$ cropped from the input $\mathbf{X}$ for the filters at each spatial location. $\mathbf{W} \in \mathbb{R}^{K \times O_c}$ is the weight matrix, each column of $\mathbf{W}$ is a filter with $K$ parameters, as illustrated in Fig.1 Top.

### 3.2. Ciphertext Packing over Matrix

There are several ways to map the convolution-layer computation onto HE ciphertexts. A straightforward way is to encrypt each input value as a separate ciphertext, and the computation process would be the same as the one with plaintext convolution. However, it requires too many ciphertexts to be created and fails to utilize the parallel computation provided by SIMD packing to accelerate the convolution process. On the contrary, convolution layer with packed ciphertexts has the dual benefits of reduced ciphertext amount and parallelized computation. There are majorly two ways to pack the input values into ciphertexts, namely DensePack and ConvPack, to facilitate the convolution layer to be computed in two different manners.

**Dense Packing (DensePack).** For the DensePack style, the input tensor of the convolution layer is flattened as an one-dimensional vector along the width, height, and channel dimensions, and then packed sequentially into a ciphertext, as shown in Fig.2 (a). For one-step of convolution computation between one filter and the input ciphertext, the filter is first extended into the same size as the input tensor by padding zeros and flattened into a plaintext vector, followed by slot-wise multiplication with the input ciphertext, and then all the slots in the resultant ciphertext are summed up to produce the convolution (dot-product) result.

As shown in Fig.2 (b), the entire convolution layer is done by permuting all the filters and shifted locations of each filter,
FFConv: Fast Factorized Neural Network Inference on Encrypted Data

Table 3. Complexity analysis of Im2Col operations between two layers (transition of ciphertexts), with different combinations of DensePack (DP) and ConvPack (CP) and different kernel sizes.

| Combination | Kernel | $\text{#MulPC}$ | $\text{#AddCC}$ | $\text{#Rot}$ |
|-------------|--------|------------------|-----------------|--------------|
| CP-CP       | $d > 1$ | $O^d$            | $O_{we}^d O_{we}^d K$ | $O_{we}^d O_{we}^d K$ |
|             | $d = 1$ | 0                | 0               | 0            |
| CP-DP       | $d > 1$ | $O^d$            | $O_{we}^d O_{we}^d K$ | $O_{we}^d O_{we}^d K$ |
|             | $d = 1$ | 0                | 0               | 0            |

Figure 2. (a) DensePack for 1 filter at 1 spatial location. (b) DensePack for 1 Convolution. (c) ConvPack for 1 Convolution.

each doing one convolution step with the input ciphertext, and arranging all the convolution results into the final output ciphertext.

Convolution Packing (ConvPack). For the ConvPack style, as shown in Fig. 2 (c), the input tensor of the convolution layer is packed into $K$ ciphertexts, where $K$ is the number of weights in a single filter. Each of the ciphertexts is packed with values in the input tensor, which will be multiplied with the corresponding filter weight in all shifted locations. Subsequently, the $K$ weights in a filter are multiplied with the $K$ ciphertexts separately, and the resultant ciphertexts are added up together into one ciphertext which produces exactly the convolution result between the filter and the input ciphertext. Similar processes can be applied to all the $O_c$ filters in the convolution layer, and the results are $O_c$ ciphertexts each encrypts a separate channel of the output tensor.

Ciphertext Packing between Layers. A typical HENN is stacked with multiple layers, and it is essential to support the smooth transition of ciphertext packing between layers, i.e., to formulate the packing of the input ciphertexts of a certain layer from the output ciphertexts of its precedent layer.

The transition (i.e., Im2Col in Fig. 5 (a)) overheads in terms of MulPC, AddCC, and Rotation operations for different combinations of ciphertext packing schemes between two consecutive layers (1st layer and 2nd layer respectively) are shown in Table 3. Generally, the overheads are related to the tensor sizes, the kernel sizes, and the number of kernels. Smaller kernel sizes usually result in smaller overheads. It must be noted that when a size $d = 1$ kernel is used, there is much less additional transition overhead between two ConvPack layers or DensePack and ConvPack layers. As will be illustrated in the next section, our FFConv design takes advantage of this property to optimize the computation of convolution layers.

4. FFConv

In Section 4.1, we introduce the low-rank factorization for convolution, which enables fast inference on data encrypted as packed ciphertexts. Section 4.2 describe how the low-rank factorized convolutions can be seamlessly integrated with various ciphertext packing representations in order to largely reduce the overheads brought to a regular convolution by a single packing scheme (DensePack or ConvPack) itself, which in turn further accelerate the inference speed on encrypted data. In Section 4.3, we analyze the advantages of our method over the state-of-the-art.

---

1Here we only consider the linear layers, as the non-linear layers are computed through element/slot-wise operations, which are generally not affected by the packing schemes.
Figure 3. Example of FactPack: Factorized DensePack-ConvPack to accelerate a regular convolution layer with DensePack. For brevity, we omit the plaintext weights in this figure.

Figure 4. Example of FactPack: Factorized ConvPack-ConvPack to accelerate a regular convolution layer with ConvPack. For brevity, we omit the plaintext weights in this figure.

With a pre-trained network, we apply the low-rank factorization to decompose each of the regular convolution layer (with kernel size larger than 1×1) in the network into two small convolutions with rank $O_c < O_c$. It is worth noting that the accuracy may drop with a smaller $O_c$. To restore the accuracy, one can perform re-training of the factorized network, with the weights of the two factorized convolutions initialized by the truncated SVD.

4.1. Low-Rank Matrix Factorization

Low-rank matrix factorization is a popular technique to reduce the number of multiply-add operations and parameters of convolution layers (Lebedev et al., 2015; Kim et al., 2016), which is achieved by factorizing the learned weight matrix $W \in \mathbb{R}^{K \times O_c}$ as a product of low-dimensional matrices $W_1 \in \mathbb{R}^{K \times O_c'}$ and $W_2 \in \mathbb{R}^{O_c' \times O_c}$:

$$\min_{W_1, W_2} \|W - W_1 W_2\|_F \quad \text{s.t. } \text{rank}(W_1 W_2) < O_c,$$  \hspace{1cm} (2)

where $\| \cdot \|_F$ is the Frobenius Norm. $W_1 W_2$ is a low-rank approximation of $W$ with the rank $O_c'$ smaller than $O_c$. Based on the Eckart–Young–Mirsky theorem (Eckart and Young, 1936), the low-rank matrices $W_1$ and $W_2$ are solved analytically by the truncated Singular Value Decomposition (SVD). As a result, the number of multiplication operations is reduced from $O_c K S$ to $O_c' K S$, and parameter size reduced from $O_c K$ to $O_c' (K + O_c)$.

As illustrated in Fig. 1 Bottom, the matrix multiplication for a regular convolution layer $I \times W$ is transformed to $I \times W_1 \times W_2$, which is essentially equivalent to two small convolution layers. The first convolution layer is with $O_c'$ $d \times d$ filters $W_1 \in \mathbb{R}^{d \times d \times L \times O_c'}$, followed by the second convolution layer with $O_c$ $1 \times 1$ filters $W_2 \in \mathbb{R}^{1 \times 1 \times O_c' \times O_c}$.

4.1.1. DISCUSSIONS

We briefly discuss how low-rank factorization is related to the efficient network architecture and filter pruning. One can refer to the supplementary material for more details on experiments and analysis.

Relationship to Efficient Network Architecture. The factorized $d \times d$ convolution $W_1$ with a small number of filters followed by $1 \times 1$ convolution $W_2$ with a large number of filters in spirit is similar to the manually designed efficient network module Bottleneck in the modern neural network ResNet (He et al., 2016), which first reduces the number of filters with $1 \times 1$ convolution, followed by a convolution with larger kernel size ($3 \times 3$) and finally increases the number of filters with $1 \times 1$ convolution. The Bottleneck supports end-to-end training with randomly initialized weights. Therefore, an obvious idea is to train manually designed convolution modules $W_1$ and $W_2$ from scratch, without the need for low-rank factorization. However, we found that training from scratch with randomly initialized weights is inferior to low-rank factorization, which initializes the weights with truncated SVD. This is because the HE-enabled neural network usually used the Square function to replace the nonlinear ReLU (Dowlin et al., 2016). Unlike ReLU, training with the Square function may cause instability and converge into a local minima since it is easier to explode the activations during training. Low-rank factorization can possibly alleviate this problem with the proper weight initialization.

Relationship to Filter Pruning. Another straightforward idea to reduce the computations of convolution layer is filter pruning (Li et al., 2017), in which the redundant filters are identified and pruned. In this sense, filter pruning only needs
4.2. Factorized Packing (FactPack)

Low-rank factorization decomposes a regular convolution layer with $O_c$ filters into a small convolution $W_1$ with the same kernel size and $O'_c$ filters ($O'_c < O_c$), followed by another $1 \times 1$ convolution $W_2$ with $O'_c$ input channels and $O_c$ output channels. Correspondingly, we introduce the design principle, Factorized Packing (FactPack), to pack $W_1$ and $W_2$ efficiently onto ciphertexts respectively.

Packing $W_2$. If we only consider the complexity of ConvPack itself and do not consider the Im2Col operations (transition cost of ciphertexts between layers) induced by ConvPack, ConvPack is much more efficient than DensePack for ciphertext packing, due to the fact that DensePack introduces a large number of expensive rotation operations (Table 1) while ConvPack does not (Table 2). On the other hand, as shown in Table 3, the transition cost of ciphertext packing between two layers is significantly reduced when the convolution kernel size equals 1, regardless of the combinations of ConvPack and DensePack used. Therefore, the second $1 \times 1$ convolution $W_2$ factorized by our method perfectly matches ConvPack. In the next, we introduce how the ConvPack for $W_2$ can be seamlessly integrated with either DensePack or ConvPack for the first small convolution $W_1$, which in turn incurs little or nearly zero transition overheads between layers (Fig 3 and Fig 4).

Packing $W_1$. $W_1$ can be packed by either DensePack or ConvPack, depending on the configuration of $W_1$.

- DensePack for $W_1$. As mentioned in Section 3.2, DensePack in LoLa introduces $\log_2(N) \times O$ rotations into the computation process for each convolution layer, which is the main reason for the extremely high inference latency. Considering the inference latency induced by DensePack is in linear relation to $O_c$, if we reduce $O_c$, the inference latency can be greatly saved. As shown in Table 1, Falcon reduced the number of rotations by utilizing the property of discrete Fourier transform (DFT) with block circulant matrices, which totally changes the computation process of DenseConv. They could reduce the number of rotations by $p$ times ($p$ is the size of each block circulant matrix), while the value $p$ has to be the power of 2. Moreover, the multi-plicative depth for each convolution layer in Falcon is increased from 1 to 3, which limits the network depth that can be supported. As described in Section 4.1, another idea is to reduce $O_c$ via low-rank factorization. If we reduce $O_c$ to $O'_c$ for the first factorized convolution $W_1$ and $O'_c/O_c <= p$, we could achieve fewer number of rotations and faster inference speed than Falcon (see Table 7). Moreover, our method can flexibly adjust the reduction rate of inference latency according to the requirement of the model accuracy. Figure 3 illustrates an example of Factorized DensePack-ConvPack for $W_1$ and $W_2$.

- ConvPack for $W_1$. Since there is no rotation operation in ConvPack, Falcon can not be applied to reduce the inference latency of convolution with ConvPack. On the contrary, our factorized convolution $W_1$ with ConvPack can be accelerated since it reduces the number of filters from $O_c$ to $O'_c$. As a result, the number of MulPC and AddCC operations required is reduced (Table 2). More importantly, since $W_2$ is $1 \times 1$ convolution with ConvPack, the transition of ciphertexts from $W_1$ to $W_2$ only involves the nearly free Grouping operation (Fig 5 (b)). Thus, the expensive Im2Col operations between layers is avoided, as shown in Table 3 "CP-CP" with $d = 1$. Figure 4 illustrates an example of Factorized ConvPack-ConvPack for $W_1$ and $W_2$.

4.3. Comparison with Prior Art

Table 4 summarizes the comparisons of our FFConv with state-of-the-art non-interactive HENNs.

| Features | CryptoNets | LoLa | Falcon | Ours |
|----------|------------|------|--------|------|
| Faster DensePack | × | × | ✓ | ✓ |
| Faster ConvPack | × | × | × | ✓ |

Table 5. MNIST Results.

| Method             | Latency (s) | Acc(%) |
|--------------------|-------------|--------|
| CryptoNets         | 205         | 98.95  |
| nGraph-HE          | 135         | 98.95  |
| FCryptoNets        | 39.1        | 98.71  |
| LoLa               | 2.1         | 98.95  |
| Falcon             | 1.2         | 98.95  |
| LoLa-TinyNet       | 0.45        | 98.23  |
| FFConv-TinyNet (Ours) | 0.37      | 98.40  |
Table 6. TinyNet operations on MNIST for our FFConv and LoLa (Brutzkus et al., 2019).

| Layer | Input size | Representation | LoLa HE Operation | Time (s) |
|-------|------------|----------------|-------------------|----------|
| Conv  | 64 x 144   | convolution    | convolution vector - row major multiplication | 0.156    |
| Square| 54 x 144   | dense          | combine to one vector using 53 major multiplications | 0.125    |
| FC    | 1 x 8064   | dense          | dense vector - row major multiplication | 0.140    |
| Output| 1 x 10     | dense          |                   |          |

| Layer | Input size | Representation | FFConv HE Operation | Time (s) |
|-------|------------|----------------|---------------------|----------|
| Conv  | 64 x 144   | convolution    | convolution vector - row major multiplication | 0.031    |
| Conv  | 13 x 144   | convolution    | convolution vector - row major multiplication | 0.046    |
| Square| 54 x 144   | dense          | combine to one vector using 53 major multiplications | 0.125    |
| FC    | 1 x 8064   | dense          | dense vector - row major multiplication | 0.140    |
| Output| 1 x 10     | dense          |                     |          |

Table 7. CIFAR-10 results with WideNet.

| #MulPC | #AddCC | #Rot | Latency (s) | Acc(%) |
|--------|--------|------|-------------|--------|
| LoLa   | 20.0K  | 69.0K| 53.1K        | 730    | 76.5  |
| Falcon | -      | -    | 7.9K         | 107    | 76.5  |
| Ours   | 19.7K  | 25.6K| 7.4K         | 96.4   | 76.5  |

of images into a ciphertext, resulting in a huge number of HMA operations for prediction at batch size 1. Though LoLa (Brutzkus et al., 2019) proposed DensePack and ConvPack to accelerate the inference by reducing the number of ciphertexts, the packing strategies introduced expensive rotations or Im2Col operations that prolong the inference latency of deeper and wider networks. The most recent work Falcon (Lou et al., 2020) proposed frequency-domain neural network to reduce the number of rotations in DensePack. However, it cannot be applied to ConvPack. Our FFConv is the first work that is able to reduce the overheads induced by DensePack and ConvPack simultaneously.

**Noise Budget.** Our FFConv requires significantly less noise budget than Falcon, since our increase in multiplicative depth is smaller. The multiplicative depth of a regular convolution in FFConv is increased from 1 to 2 after matrix factorization, while Falcon increases the depth from 1 to 3 due to the DFT and inverse DFT operations attached to each convolution. A larger multiplicative depth would require more noise budget. As shown in Section 5, the noise budget of our FFConv for WideNet on CIFAR-10 is 380 bits, versus Falcon 430 bits.

5. Experiments

**CNN Architecture** We evaluate our method on MNIST and CIFAR-10 datasets, respectively. MNIST contains 28 * 28 grayscale images divided into two sets of 60,000 training and 10,000 test samples. For MNIST, we designed a smaller neural network TinyNet, which contains only an 8*8 convolution layer with a stride of (2, 2) and 56 output channels, followed by a fully connected layer. After 100 epochs of training, the model accuracy can reach 98.23%. The design of TinyNet is to evaluate the effectiveness of our Factorized ConvPack-ConvPack. Specifically, FFConv-TinyNet factorizes the first convolution layer of TinyNet into a 8 x 8 convolution layer (stride 2, output channels 13), and a 1 x 1 convolution (stride 1, output channels 56). Keeping the same training hyper-parameters, the model accuracy can reach 98.40% with initialization after 100 epochs training.

CIFAR-10 is an image classification dataset and contains 60,000 colored images with 10 object classes of 32 x 32 pixel resolution. The network architecture WideNet is the same as LoLa. After 200 epoch of training, the model accuracy can reach 78.03%. FFConv-WideNet replaces the second convolutional layer of WideNet with a 6 x 6 convolution layer (stride 2, output channels 20) and a 1 x 1 Convolution (stride 1, output channels 163). Keeping the same training hyper-parameters, the model accuracy can reach 76.50% after 200 epochs training, with our SVD based weight initialization. The weight and activations of all models for MNIST and CIFAR-10 are quantized with 8 bits.

**Cryptosystem Settings** We use BFV scheme to implement all models based on the message representations and homomorphic operations used in LoLa. We set different parameters in order to maximize the performance of each model. In particular, (1) LoLa-TinyNet: ring dimension $N = 8192$, plaintext coefficient modulus $t = 1099511922689$; (2) FFConv-TinyNet: $N = 8192$, $t = 57646075230439873$; (3) LoLa-WideNet: $N = 16384$, $t = 34359771137 \times 34360754177$; (4) FFConv-WideNet: $N = 16384$, $t = 900719925560193 \times 9007199255658497$.

We set appropriate ciphertext coefficient modulus Q respectively so that their security level is larger than 128 bits. For
FFConv: Fast Factorized Neural Network Inference on Encrypted Data

Table 8. FFConv-WideNet on CIFAR-10.

| Layer     | Input size | Representation | FFConv HE Operation                                | #Rot | Times (s) |
|-----------|------------|----------------|--------------------------------------------------|------|-----------|
| convolution | 192 x 196 | convolution    | convolution vector - row major multiplication    | 0    | 7.64      |
| square    | 83 x 196   | dense          | combine to one vector using 92 rotations and additions | 92   | 5.47      |
| convolution | 1 x 16268 | dense          | square                                           | 0    | 0.17      |
| square    | 1 x 16268  | dense          | 500 dense vector - row major multiplication       | 7000 | 73.45     |
| convolution | 20 x 25  | convolution    | convolution vector - row major multiplication    | 0    | 1.82      |
| square    | 163 x 25   | dense          | combine to one vector using 162 rotations and additions | 162  | 7.97      |
| fc        | 1 x 4075   | dense          | square                                           | 0    | 0.17      |
| output    | 1 x 10     | dense          | 10 dense vector - row major multiplication        | 120  | 1.67      |

5.1. MNIST

Table 6 summarizes the message representation, homomorphic operation and inference latency that LoLa-TinyNet and FFConv-TinyNet apply at each layer. Both TinyNet and FFConv-TinyNet implement Im2Col to preprocess the input and encode the input into 64 ciphertexts. Each ciphertext contains 144 elements. After performing convolution vector-row major multiplication on each of the 64 ciphertexts, both of the models result in 54 dense output messages and 13 dense output messages and consume 0.156 seconds and 0.031 seconds, respectively. For FFConv-TinyNet, an additional layer of convolution vector-row major multiplication is required to form the entire first convolution layer, which results in dense output messages in 0.046 seconds. Although FFConv-TinyNet uses two layers of convolution vector-row major multiplication, it still reduces the first convolution layer inference time from 0.156 seconds by 50.64% to 0.031 + 0.046 = 0.077 seconds. The remaining layers of FFConv-TinyNet remain the same as LoLa-TinyNet and therefore use the same time. For the total inference latency of FFConv-TinyNet shown in Table 5, it is 0.37 seconds reduced by 17.78% from 0.45 seconds, which is also much faster than Faster-CryptoNets (Chou et al., 2018) and nGraph-HE (Boemer et al., 2019). It is worth noting that the introduction of FFConv has improved the accuracy of TinyNet. The reason is FFConv also reduces the amount of model parameters, which can help prevent model overfitting and enhance generalization ability. Since TinyNet’s first convolutional layer does not contain any rotation operation, Falcon cannot be used to accelerate it.

5.2. CIFAR-10

Table 8 summaries the message representation, homomorphic operation and the number of rotations that FFConv-WideNet applies at each layer. In LoLa-WideNet, the second convolution layer consumes 711 seconds, accounting for more than 97% of the total inference latency. The reason lies in the nearly 500,000 parameters and the use of a large number of extremely time-consuming rotation operations. Therefore, we focus on optimizing this convolutional layer. FFConv-WideNet replaces the second convolutional layer in LoLa-WideNet with two sub-convolutional layers to reduce 52,975 rotations by 86.48% to 7000 + 162 = 7162 rotations, reduce 4075 MulPC by 7.75% to 3575 MulPC, and reduce 52,975 AddCC by 81.61% to 9740. The message representation and homomorphic operation of the remaining layers remain unchanged, so the time consumed remains unchanged. Falcon-WideNet transformed the second spatial-domain convolution layer into frequency-domain in order to reduce the number of rotations, but at the same time, they introduced a large amount of MulPC (~3.7K). Compared with Falcon-WideNet, our number of rotation operations is 9.07% less. Moreover, the increase in noise caused by Falcon’s homomorphic DFT and inverse DFT operations requires higher noise budget Q = 430-bit, however, FFConv-WideNet only needs 380-bit. As shown in Table 7, by reducing the number of rotations and MulPC, FFConv-WideNet inference latency is 96.4 seconds, which is reduced by 87.04% and 11.59% compared to LoLa-WideNet and Falcon-WideNet.

6. Conclusion

In this paper, we propose a low-rank factorization framework called FFConv to accelerate secure neural network inference on encrypted data. FFConv factorizes a regular convolution into two small convolutions, which can be encrypted with Factorized Packing (FactPack) efficiently. Experimental results show that FFConv enables faster inference speed than state-of-the-art. FFConv is the first non-interactive HENN that is capable of reducing the computational overheads induced by different ciphertext packing schemes simultaneously.

References

Boemer, F., Lao, Y., Cammarota, R., and Wierzynski, C. (2019). ngraph-he: A graph compiler for deep learning on homomorphically encrypted data. In Proceedings of
Fast Factorized Neural Network Inference on Encrypted Data

Brakerski, Z., Gentry, C., and Vaikuntanathan, V. (2014). (leveled) fully homomorphic encryption without bootstrapping. ACM Transactions on Computation Theory (TOCT), 6(3):13.

Brakerski, Z. and Vaikuntanathan, V. (2011). Fully homomorphic encryption from ring-lwe and security for key dependent messages. In Annual cryptology conference, pages 505–524. Springer.

Brutzkus, A., Gilad-Bachrach, R., and Elisha, O. (2019). Low latency privacy preserving inference. In International Conference on Machine Learning, pages 812–821.

Cheon, J. H., Kim, A., Kim, M., and Song, Y. (2017). Homomorphic encryption for arithmetic of approximate numbers. In International Conference on the Theory and Application of Cryptology and Information Security, pages 409–437. Springer.

Chou, E., Beal, J., Levy, D., Yeung, S., Haque, A., and Fei-Fei, L. (2018). Faster cryptonets: Leveraging sparsity for real-world encrypted inference. arXiv preprint arXiv:1811.09953.

Dowlin, N., Gilad-Bachrach, R., Laine, K., Lauter, K., Naehrig, M., and Wernsing, J. (2016). Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In International Conference on Machine Learning.

Eckart, C. and Young, G. (1936). The approximation of one matrix by another of lower rank. Psychometrika, 1(3):211–218.

Fan, J. and Vercauteren, F. (2012). Somewhat practical fully homomorphic encryption. IACR Cryptology ePrint Archive, 2012:144.

Gentry, C. (2009). Fully homomorphic encryption using ideal lattices. In STOC09, pages 169–178. ACM.

Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. In Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, volume 15 of Proceedings of Machine Learning Research, pages 315–323.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 770–778.