# Low-Latency Convolutional Layer Computation Under Homomorphic Encryption

Zhi Ming Chua, Christos-Savvas Bouganis, Peter Y. K. Cheung
*Imperial College London*

## I. Introduction

Big data analytics have achieved significant breakthroughs with increasingly complex machine learning algorithms, often aided by computation outsourcing. However, processing data in a responsible and secure manner remains a considerable challenge. Homomorphic encryption (HE) has been hailed as a holy grail in secure computation as it allows computations to be carried out directly on encrypted data, hence preserving data privacy. However, homomorphic operations (HOPs) are several orders of magnitude more computationally expensive than the plaintext operations, primarily due to the ciphertexts being much larger than their corresponding plaintexts, prohibiting HE application to settings where low-latency is desirable. We show how *batching*, a capability of some HE schemes, can be utilised to reduce the latency in the execution of Convolutional Neural Networks (CNNs), by enabling concurrent computation within the convolutional layers.

## II. Background

*Homomorphic* encryption is an encryption method that allows plaintext operations to be performed directly on their ciphertexts, preserving data privacy. Among the latest generation of HE schemes is the CKKS [4] scheme. In this scheme, a ciphertext is a pair of polynomials of degree $N$ with coefficients integers modulo $Q$. A Chinese Remainder Theorem (CRT)-based encoder is used to transform a vector of integers of length $N/2$ into a plaintext polynomial, which is then encrypted. Similar to how the CRT enables the operations on large integers to be equivalent to the same operations on their remainders in the residue number system, when HOPs are performed on a ciphertext, all $N/2$ encoded values undergo the same transformations. This capability is called *batching*. The CKKS scheme supports addition, multiplication and rotation.

## III. Problem Statement

Although recent works have shown that it is possible to compute entire CNNs on homomorphically encrypted data, these CNNs remain considerably shallower than state-of-the-art deep CNNs. This is because the computation costs of HOPs increase more than linearly with the multiplicative depth of the function evaluated, prohibiting the application of those approaches to modern state-of-the-art CNNs with large number of layers. The batching capability has traditionally been employed to reduce average latency by processing data across different batch instances in parallel, as shown in CryptoNets [6]. More recently, some works [2], [5] have shown that latency can be reduced significantly using batching, by packing data that belong to the same batch instance within the same ciphertext, hence reducing the total number of HOPs required. We present a more refined search of the design space via the Split Convolution, which we elaborate on in Section IV-A3.

### A. Problem Setting

In our work, we focus on CNN-based classification of encrypted images outsourced to a third-party server. As convolutional layers have the highest contribution on the computational load of the CNN, we choose to focus on accelerating their homomorphic computation. It should be noted that the user data and the model are both encrypted.

In the proposed setting, the server computes individual convolutional layers and transmits the intermediate results back to the data owner for activation computation and re-encryption. This model of computation has the following advantages: the computation cost of each HOP does not increase with the depth of the CNN, and no restriction is placed on the form of the activation layers in being HE-compatible. As such, the adopted computation model can be applied to state-of-the-art CNN models. We assume the server to be semi-honest.

### B. The Packing Problem for Convolutional Layers

In convolutional layers, the data evaluated has multiple dimensions, *e.g.* width, height, channel and batch. The computation in a convolutional layer can be decomposed into $b \times c_i \times c_o$ single-channel convolutions, where $b$ is the batch size, and $c_i$ and $c_o$ are the number of input and output channels, respectively. Each single-channel convolution requires $f \times w_o \times h_o$ multiplications, where $f$ is the kernel size, and $w_o$ and $h_o$ are the width and height of the output feature map, respectively. Given a scheme with polynomial of degree $N$, there are $N/2$ slots per ciphertext available to store data. As such, a homomorphic multiplications can be performed on the $N/2$ slots in parallel. We define the packing problem as follows: given a convolution-based workload, find a grouping of the $b \times c_i \times c_o \times f \times w_o \times h_o$ required multiplications on a ciphertext that minimises the latency of the convolution computation. Towards constructing the space of possible groupings, we define a number of parameterised configurations which are called *packing configurations*.

## IV. Methodology

In this section, we demonstrate how convolution operations can utilise the capability of batching, followed by how batched processing of multi-channel convolutions can be decomposed into single-channel convolutions. Finally, we present how we systematically traverse the design space.

### A. Width and Height Dimensions: Convolution

Matrix multiplication is a widely used technique to perform 2-D convolutions. The matrix can be vectorised in a column-major or a row-major manner. The former produces partial results that lie within the same vector, requiring expensive rotation operations to accumulate. We therefore exclude this in our work. We consider 3 approaches to pack the multiplication operations from the row-major vectorisation. For simplicity, we demonstrate these approaches using a toy example in 1-D, where a valid convolution with input feature map $\mathbf{m} = [m_1, m_2, m_3, m_4, m_5]$ of length $n_i = 5$, a filter $\mathbf{k} = [k_1, k_2, k_3]$ of length $f = 3$ and stride $s = 2$ is computed. Let $\mathbf{r} = [r_1, r_2] = \mathbf{m} * \mathbf{k}$ be the result of the convolution. Assume the number of slots $n_s = 8$.

*1) **Direct Convolution**:* In direct convolution, the input feature map values are multiplied by each filter value in parallel. The input feature map is rotated before the multiplication is performed such that the partial results align at the same slot index. For the toy example, this is implemented as $[m_1, m_2, m_3, m_4, m_5, 0, 0, 0] \odot [k_1, 0, k_1, 0, 0, 0, 0, 0] + [m_2, m_3, m_4, m_5, 0, 0, 0, m_1] \odot [k_2, 0, k_2, 0, 0, 0, 0, 0] + [m_3, m_4, m_5, 0, 0, 0, m_1, m_2] \odot [k_3, 0, k_3, 0, 0, 0, 0, 0] = [r_1, 0, r_2, 0, 0, 0, 0, 0]$.

*2) **Intraleaved Convolution**:* With direct convolution, the element-wise multiplications performed at indices 2 and 4 do not yield partial results. To decrease the homomorphic multiplications required, we can densify the kernel ciphertext. We call this the intraleaved convolution, which for the toy example is implemented as $[m_1, m_2, m_3, m_4, m_5, 0, 0, 0] \odot [k_1, k_2, k_1, k_2, 0, 0, 0, 0] + [m_3, m_4, m_5, 0, 0, 0, m_1, m_2] \odot [k_3, 0, k_3, 0, 0, 0, 0, 0]$. The result is then summed with a rotated version of itself such that the partial results in indices 1 and 2, 3 and 4 are accumulated to produce $r_1$ and $r_2$, respectively. This procedure is called *rotate-and-sum*. Masking is used to remove junk results in all slots, except for indices 1 and 3. The number of homomorphic multiplications is reduced compared to Direct Convolution but requires more computation to combine the partial results.

*3) **Split Convolution**:* This approach is an extension of the decomposed convolution presented by Juvekar *et al.* in [7], where they demonstrated how strided convolutions can be decomposed into $s$ convolutions of stride $(1, 1)$. The result is obtained by summing the partial results of the $s$ convolutions. For our toy example, the convolution would be decomposed as $\mathbf{m} * \mathbf{k} = [m_1, m_3, m_5] * [k_1, k_3] + [m_2, m_4, 0] * [k_2, 0]$ and each term is computed via direct convolution. In this approach, the input feature map and filter values are decomposed with data stride $d$ equal to the convolution stride $s$. We extend this further by increasing the data stride $d$ in steps of $s$. The maximum data stride possible is $n_i$. When $d = n_i$, the feature map is fully split across $n_i$ ciphertexts, *i.e.* no two elements of the feature map lie in the same ciphertext. With the exception of $d = n_i$, data strides that are not multiples of $s$ yield partial results in ciphertext layouts that require complicated (hence, computationally expensive) rotations and masking procedures to accumulate, hence they are excluded from the design space.

### B. Channel and Batch Dimensions

A convolution with $c_i$ input channels and $c_o$ output channels requires the computation of $c_i \times c_o$ single-channel convolutions. Depending on how the $c_i \times c_o$ convolutions are grouped for parallelisation, different rotate-and-sum and masking procedures are required to accumulate the results for the corresponding output channels. Similarly, batches of data can be grouped for parallel processing but no further computation is required to accumulate results over different batch instances.

### C. Traversing the Design Space

We traverse the packing configuration design space from the bottom up, working in the order of width, height, channel and batch. At each dimension, we comb through possible packing configurations, *i.e.* the different approaches for convolution (including all possible data strides $d$ for split convolution) in the height and width dimensions, and the different groupings in the channel and batch dimensions. To prune the design space, we do not consider further groupings if the number of slots required to parallelise another set of HOPs exceeds $n_s$.

## V. Experiments and Evaluation

To illustrate the impact of different packing configurations, we perform a homomorphic evaluation of a convolutional layer using a small example with a batch size of 1. The input feature map is $5 \times 5$ with 3 channels. The filter is $3 \times 3$ with a stride of $(2, 2)$. The number of output channels is 4.

All experiments were conducted using a Vagrant box with 16 cores of an Intel Xeon CPU E5-2630 v4 @ 2.20GHz and 64GB memory. We used Microsoft SEAL 3.7.1 [8], which implements the residue number system (RNS) variant [3] of the CKKS scheme. We let $N = 8192$ and $\lceil \log Q \rceil = 200$ or 160, depending on the multiplicative depth determined by the packing configuration. The security level is at least 128-bit, according to the standards set out in [1].

We use microbenchmarks in our cost model to estimate the latency of the homomorphic evaluation, including encoding+encryption and decryption+decoding. The cost model achieved a mean relative error of 4.87% when compared to the average actual latency over 50 runs. The baseline does not utilise batching and has an average latency of 4.2185 seconds. Our search identified a packing configuration with an average latency of 0.0982 seconds, which translates to a $43\times$ speedup. Detailed results are presented in Table I.

TABLE I
Latency for Different Packing Configurations (seconds)

| Config. | Enc.+Encrypt | Evaluation | Decrypt+Dec. | Total |
|---|---|---|---|---|
| Baseline | 1.2327 | 2.9696 | 0.0161 | 4.2185 |
| Ours (model) | 0.0337 | 0.0590 | 0.0004 | 0.0937 |
| Ours (actual) | 0.0379 | 0.0596 | 0.0007 | 0.0982 |

## VI. Conclusion

This work introduced a methodology for improving the latency of the computations of convolutional layers in a CNN under homomorphic encryption by exploring possible packing configuration in the ciphertext, opening the space for low-latency CNN computation under HE.

## REFERENCES


[1] M. Albrecht, M. Chase, H. Chen, J. Ding, S. Goldwasser, S. Gorbunov, S. Halevi, J. Hoffstein, K. Laine, K. Lauter, S. Lokam, D. Micciancio, D. Moody, T. Morrison, A. Sahai, and V. Vaikuntanathan, "Homomorphic encryption security standard," HomomorphicEncryption.org, Toronto, Canada, Tech. Rep., November 2018.

[2] A. Brutzkus, R. Gilad-Bachrach, and O. Elisha, "Low latency privacy preserving inference," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 09–15 Jun 2019, pp. 812–821. [Online]. Available: http://proceedings.mlr.press/v97/brutzkus19a.html

[3] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song, "A full rns variant of approximate homomorphic encryption," in *Selected Areas in Cryptography – SAC 2018*, C. Cid and M. J. Jacobson Jr., Eds. Cham: Springer International Publishing, 2019, pp. 347–368.

[4] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," *Advances in Cryptology – ASIACRYPT 2017*, pp. 409–437, 2017.

[5] R. Dathathri, B. Kostova, O. Saarikivi, W. Dai, K. Laine, and M. Musuvathi, "Eva: An encrypted vector arithmetic language and compiler for efficient homomorphic computation," in *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI 2020. New York, NY, USA: Association for Computing Machinery, 2020, p. 546–561. [Online]. Available: https://doi.org/10.1145/3385412.3386023

[6] N. Dowlin, R. Gilad-Bachrach, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ser. ICML'16. JMLR.org, 2016, pp. 201–210. [Online]. Available: http://dl.acm.org/citation.cfm?id=3045390.3045413

[7] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "GAZELLE: A low latency framework for secure neural network inference," in *27th USENIX Security Symposium (USENIX Security 18)*. Baltimore, MD: USENIX Association, 2018, pp. 1651–1669. [Online]. Available: https://www.usenix.org/conference/usenixsecurity18/presentation/juvekar

[8] "Microsoft SEAL (release 3.7)," https://github.com/Microsoft/SEAL, Sep. 2021, microsoft Research, Redmond, WA.