A STUDY OF USING CKKS HOMOMORPHIC ENCRYPTION OVER THE LAYERS OF A CONVOLUTIONAL NEURAL NETWORK MODEL

Sebastian Soler Castaneda[1], Kevin Nam[1], Youyeon Joo[1], and Yunheung Paek[1]
[1]Dept. of Electrical and Computer Engineering and Inter-University
Semiconductor Research Center (ISRC), Seoul National University

ABSTRACT

Homomorphic Encryption (HE) schemes have been recently growing as a reliable solution to preserve users' information owe to maintaining and operating the user data in the encrypted state. In addition to that, several Neural Networks models merged with HE schemes have been developed as a prospective tool for privacy-preserving machine learning. Those mentioned works demonstrated that it is possible to match the accuracy of non-encrypted models but there is always a trade-off in the computation time. In this work, we evaluate the implementation of CKKS HE scheme operations over the layers of a LeNet5 convolutional inference model, however, owing to the limitations of the evaluation environment, the scope of this work is not to develop a complete LeNet5 encrypted model. The evaluation was performed using the MNIST dataset with Microsoft SEAL (MSEAL) open-source homomorphic encryption library ported version on Python (PyFhel). The behavior of the encrypted model, the limitations faced and a small description of related and future work is also provided.

1. Introduction

The privacy-preserving issue is one of the most critical problems to solve for machine learning applications, along with that, Homomorphic encryption (HE) has arisen as an appropriate tool to ensure strong security in the cryptographic sense and comply with the communication's security approach. HE allows data to be processed, manage, and to perform different calculations in an encrypted state, which means that our information is never exposed during the transmission process.

The application of HE to Convolutional Neural Network [1] has been researched much until now, however, some limitations have also been found such as the high difficulty of evaluating the popular activation functions like ReLU, sigmoid, leaky ReLU, and ELU. Throughout recent years, several HE schemes have been developing according to different computation models, such as Brakerski-Fan-Vercauteren (BFV) scheme, Truly Fully Homomorphic Encryption (TFHE) or Cheon-Kim-Kim-Song (CKKS) scheme. The CKKS scheme, which supports homomorphic addition and multiplication, has been raised as a promising tool for CNN-HE evaluations since it can deal with encrypted real numbers.

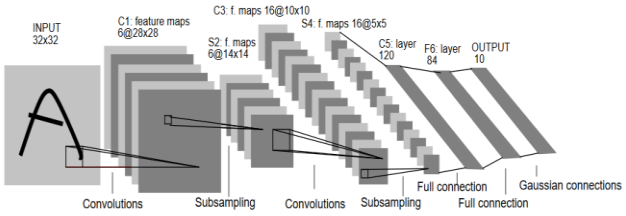
Furthermore, LeNet-5 is a simple convolutional neural feed-forward network whose artificial neurons can respond to a part of the surrounding cells in the coverage range and perform well in large-scale image processing applications. Although nowadays LeNet-5 is not widely used anymore, it was the emergence of CNN (Convolutional Neural Networks) and defines the basic components of this kind of network as well as being the foundation of the further-developed different types of CNN. On account of the previous reasons, LeNet-5 is the best candidate to evaluate the behaviors of inference models in an encrypted state

This study is focused on the behavior of a basic LeNet5 CNN model trained and tested with MNIST dataset in the encrypted state using CKKS HE over on the convolution operations, so we are not evaluating a complete Neural Network. It is important to remark that this study is part of our current research, which means that the scope of this work is only providing a comparison with the normal "no encrypted" state of the Lenet-5 model and not building up a complete LeNet5-CKKS encrypted model. All the implementations were performed on a Python v3.7 environment using the Python ported version of MSEAL library called PyFhel.

2. CNN and CKKS-HE Schemes

2.1 LeNet5 Inference Model

LeNet5 is one of the earliest CNNs developed around 1998 by Yann LeCun and others [2]. Originally was used for the recognition of handwritten characters but as the years went by, it started to be used in image processing applications. The architecture of LeNet5 is shown in the figure below:



(Fig 1) LeNet 5 Architecture.

LeNet5 is composed of five layers in total, two convolutional and three fully connected layers. This model accepts as input a greyscale image of 32x32, so the input image should contain just one channel. After this, the first convolutional layer which has a filter size of 5X5 reduces the width and height of the input image while increasing the depth. The output would be 28x28x6. After this, pooling is applied to decrease the feature map by half, i.e., 14x14x6. The same filter size (5x5) with 16 filters is now applied to the output followed by a pooling layer. After this, a convolutional layer of size 5x5 with 120 filters is applied to flatten the feature map to 120 values. Then comes the first fully connected layer, with 84 neurons. Finally, we have the output layer which has 10 output neurons.

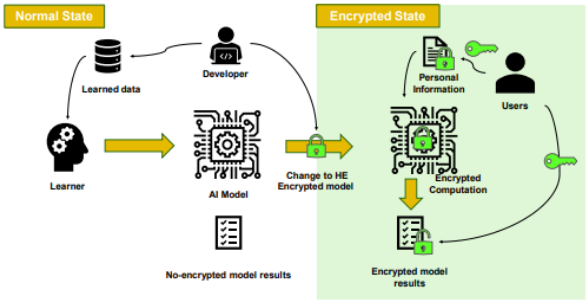
2.2 CKKS-HE Scheme

Unlike other HE schemes, the CKKS scheme supports approximate arithmetic over complex numbers [3]. The main idea behind this is to include an encryption noise as part of the error occurring during approximate computations. That means that an encryption *c* of message m by the secret key *sk* will have a decryption structure of the form $\{c, sk\} = m + e \pmod q$ where *e* is a small error inserted to guarantee the security of hardness assumptions such as the learning with errors (LWE) and the ring- LWE (RLWE) problems.

CKKS always maintains the decryption structure small enough compared to the ciphertext modulus for homomorphic operations. However, the bit size of the message increases exponentially with the depth of a circuit without rounding. To address this problem, CKKS suggests a new technique -called rescaling- that manipulates the message of ciphertext. It reduces the size of ciphertext modulus and removes the error located in the LSBs of the messages.

Owe to the previous characteristics described, CKKS is a good candidate to test out its behavior over the different layers of a Convolutional Neural Network, especially over the activation functions such as ReLu or Sigmoid.

2.3 CNN-HE Scheme



(Fig 2) Diagram flow of an encrypted CNN model.

The main idea of merging a CNN with HE scheme is to change the normal state of the model to an encrypted model. Thus, the encrypted model will perform encrypted computation over the data, preserving personal information privacy. Fig 3 shows a basic diagram flow on how an encrypted CNN model should work. The goal is to preserve the encrypted results equal to the no-encrypted results. It is also important to remark, that up to this date, encrypted operations take longer to perform compared to normal operations, which means, there is always a trade-off between the accuracy and computation time.

3. Evaluating LeNet5 layers with CKKS operations

In order to perform the goal of this study, we first trained a regular LeNet5 model with the MNISST dataset, followed by extracting the weights and bias of the model, encrypting such weights and bias using PyFhel functions, and after that, performing some homomorphic operations to decrypt the results and test out the correct functionality of PyFhel functions. Finally, we tried to build up the same inference model but this time including the PyFhel homomorphic operations over the convolution layers of the LeNet5 model.

Before evaluating the LENET5-CKKS inference model, as stated before, we first trained and tested a normal LeNet5 model on Python using the Pytorch and Torchvision tools. The reason why Python was selected as the programming language of this study is because of its simplicity when it comes to working with Neural Network applications, we are aware that Python presents some disadvantages compared to other programming languages but for the sake of this study is a convenient tool to use. Likewise, MNIST dataset was used for the training data, and the model was evaluated using 10,000 images from the same MNIST dataset. As expected, the trained model obtained a high accuracy of 98.79 percent. The trained model and its layers are shown as follows:

```
==========================================================================
Layer (type:depth-idx)                   Param #
==========================================================================
├─Sequential: 1-1                        --
|    └─Conv2d: 2-1                       156
|    └─BatchNorm2d: 2-2                  12
|    └─ReLU: 2-3                         --
|    └─MaxPool2d: 2-4                    --
├─Sequential: 1-2                        --
|    └─Conv2d: 2-5                       2,416
|    └─BatchNorm2d: 2-6                  32
|    └─ReLU: 2-7                         --
|    └─MaxPool2d: 2-8                    --
├─Linear: 1-3                            48,120
├─ReLU: 1-4                              --
├─Linear: 1-5                            10,164
├─ReLU: 1-6                              --
├─Linear: 1-7                            850
==========================================================================
Total params: 61,750
Trainable params: 61,750
Non-trainable params: 0
==========================================================================
```

(Fig 3) Trained LeNet 5 model.

Following that, the extraction of the weights and bias can be easily performed with the PyTorch tools on Python, however, in order to encrypt it, it was compulsory to convert the weights and bias into a numpy array because the PyFhel encryption and decryption operations only work with numpy arrays, however, this made the work easier because it is faster and more convenient to evaluate the homomorphic operations with numpy arrays. After encrypting the data with PyFhel functions, it generates a numpy array with the following structure:

```
array1:  [<Pyfhel Ciphertext at 0x23183bfa740, encoding=FRACTIONAL, size=2/2, noiseBudget=28>
<Pyfhel Ciphertext at 0x23183c7ca80, encoding=FRACTIONAL, size=2/2, noiseBudget=27>
<Pyfhel Ciphertext at 0x23183c6b040, encoding=FRACTIONAL, size=2/2, noiseBudget=27>
<Pyfhel Ciphertext at 0x231c3ca4680, encoding=FRACTIONAL, size=2/2, noiseBudget=27>
<Pyfhel Ciphertext at 0x231ca4df2c0, encoding=FRACTIONAL, size=2/2, noiseBudget=28>
<Pyfhel Ciphertext at 0x231ca4df300, encoding=FRACTIONAL, size=2/2, noiseBudget=27>] -> ctxt1  <class 'numpy.ndarray'>
```

(Fig 4) Structure of a PyFhel encrypted array.

In *Fig 3.* it is observable how the encrypted data remains as a numpy array, however, the elements of the array remain encrypted with a little bit of noise budget as the CKKS

theory states. To evaluate how the PyFhel encryption works, we simulate the sending process of the data (bytes) over a toy network using the Python library "pickles". With the "pickles" library acting as a network, the encrypted data remains the same after receiving it, consequently, computation with homomorphic addition and multiplication is allowed to be performed with the encrypted data. The PyFhel library allows us to decrypt the results of the addition and multiplication, which in this case will be a float number that is coming from the previously encrypted weights/bias.

Having tested that the LeNet5 model is working as well as the functionality of the homomorphic encryption library, the next approach is to merge the HE operations over the LeNet5 layers and evaluate the behavior and accuracy of the new inference model.

While performing this study, we faced two big limitations when it comes to evaluating the activation and pooling layers of the model. First, the activation layer and pooling layer have a great influence on the model accuracy but they consist mostly of comparative operations, which represents a limitation for this study because CKKS directly supports only homomorphic addition and multiplication. Thus, the comparison operation is replaced using an approximate polynomial. If we evaluate the mentioned previous layers by themselves using the approximate polynomials, a decrease in the accuracy by almost 20% is noticeable. The second limitation is a Pytorch-PyFhel library incompatibility, this limitation will be explained in detail in the next section.

Due to the previous limitations, testing a complete LeNet5 encrypted model using Python and PyFhel library is impossible. Nonetheless, we were able to use the CKKS PyFhel operations over the convolution layers of the model.

4. Results

During the process of testing the CKKS-Inference model we came up with some limitations: The PyFhel library transforms the encrypted data into a numpy array with datatype = object, and to the best of your knowledge, there is no existing solution to transform a numpy array with datatype object to a Pytorch Tensor, this is a limitation because the LeNet5 model on Pytorch is computed by using Tensors. In addition to that, the popular activation functions are usually non-arithmetic functions, such as ReLU, owe to that, we cannot evaluate CKKS arithmetic operations without reducing the accuracy significantly. Keeping in mind those two limitations, as explained in the previous section, the approach for this study is encrypting the data only during the convolution layers of the LeNet5 model, doing this we can avoid dealing with PyFhel-Pytorch incompatibility as well with the ReLu non-arithmetic limitations. The evaluation environment was an Intel Core i7-9750H and the GPU NVidia GTX1650.

Even though the encryption of the data is taken into account on the convolution layers, the output of the model remains as an encrypted PyFhel numpy array. Next, we performed the decryption of the data just before testing it, in addition to that, a random encrypted PyFhel numpy array was also generated, with the goal of performing some multiplication and addition operations over the encrypted output of the model and verifying that we can perform computation over the encrypted data. Finally, we decrypted the data again and tested out the model again, we called our model LeNet5-CKKs just for easy reference, but it is important to remember that is not a complete CNN encrypted model. The results are shown as follows:

<Table 1> Encrypted and non-encrypted model results

Measurement	LeNet5	LeNet5-CKKS
Accuracy	98%	95%
Computing Time (s)	9	420

The previous table shows the results of the evaluation of the proposed model for this study. There is no remarkable trade-off between the accuracy of the two models, however, the computation time is higher in the encrypted model (LeNet5-CKKS). We assume that the reason behind this is that CKKS struggle with comparison operations over the convolutional calculations. Moreover, due to the limitations of Python libraries we were not able to test out all the layers from the model, but it can be also assumed that the computation could be higher with a completed evaluated model. Nonetheless, we also remark that this is a research in progress, so for future work, it is expected to continue testing different approaches

5. Related and Future Work

A vast amount of research is currently in process regarding Deep Learning Homomorphic encrypted models, using not CKKS scheme but also other HE schemes such as TFHE. TFHE has shown lower computation time compared to CKKS, on the other hand, CKKS has lower accuracy compared to TFHE, which makes both schemes have disadvantages. Furthermore, the mentioned works have demonstrated that by using different encrypted CNN with different schemes it is possible to match the accuracy of a non-encrypted model, especially using the MNIST dataset [4][5]. To the best of our knowledge, none of the previous works was evaluated over a LeNet5 model. The following table summarizes the results:

<Table 2> Related work accuracy results

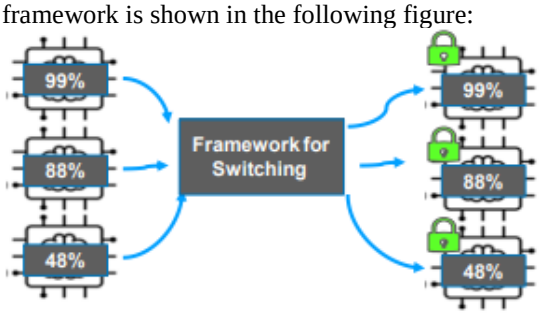
Paper Name	Dataset	Accuracy (%) Encrypted (No encrypted)
Ngraph-HE	MNIST	96.9 (99)
	CIFAR-10	62.2 (89)
CHET	MNIST	98.5 (99)
	CIFAR-10	81.5 (90)

The previous table shows that it is possible to achieve the same level of accuracy as a non-encrypted model, especially when using the MNIST dataset, the main problem relies on reducing the computing time. The following table shows the advantages and disadvantages of the previously cited HE schemes when using them in CNN models:

<Table 3> Pros and Cons for HE schemes

HE scheme	Advantages	Disadvantages
TFHE	Lower computation times and works better with comparison operations	Struggles with homomorphic operations
CKKS	Better results for homomorphic operations	Accuracy is lower

This study is just a part of the current research being carried on, henceforth, for future work, the focus of our approach will be reducing the computation part of the encrypted models. There already exists a library [6] that can support bridges between various schemes of HE -including CKKS and TFHE- which could be a good start to developing a framework that can allow us to change from different HE schemes inside an encrypted CNN model. The approach of the framework is shown in the following figure:



(Fig 5) Framework for switching CNN between different HE schemes

Fig 5. shows the approach of the proposed framework. The input of the framework should be a trained AI model and the output another AI model with homomorphic encryption that performs exactly the same as the input. This means, that if the input model has low accuracy, the encrypted model will remain the same accuracy. The development strategy is switching between different HE schemes, for example, using TFHE for comparative operations and CKKS for other operations like homomorphic addition and multiplication. Doing this, the main goal is to avoid the performance load that causes computation time to increase. Finally, this framework should allow an automatic parameter selection considering the model structure and data type in order to be a general solution for all models and datasets.

6. Conclusion

It was demonstrated that performing the CKKS HE scheme over the convolution layer of a LeNet5 model can match the accuracy of a non-encrypted LeNet5 CNN model, however, there is always a big trade-off in terms of computation time. In addition to that, we remarked on the limitations of using CKKS on a CNN model such as not being able to evaluate directly the activation functions owing to their non-arithmetic nature. We also faced some limitations due to the incompatibility of the PyFhel library with Pytorch tensor. Nonetheless, this work is just a part of the outgoing research, so we will continue doing experiments with popular models and datasets and working on the implementation of a framework that allows change between different HE schemes in an encrypted CNN model.

7. Acknowledgement

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korean government(MSIT) (NRF-2020R1A2B5B03095204) and the BK21 FOUR program of the Education and Research Program for Future ICT Pioneers, Seoul National University in 2022

References

[1] J. -W. Lee et al., "Privacy-Preserving Machine Learning with Fully Homomorphic Encryption for Deep Neural Network," IEEE Access, vol. 10, p 30039 – 30054, 2022

[2] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," Proceedings of the IEEE, vol. 86, p 2278-2324, 1998

[3] Cheon, Jung & Kim, Andrey & Kim, Miran & Song, Yongsoo. “Homomorphic Encryption for Arithmetic of Approximate Numbers”, Advances in Cryptology, ASIACRYPT, p 409-437, 2017.

[4] Fabian Boemer, Anamaria Costache, Rosario Cammarota, and Casimir Wierzynski. “NGraph-HE2: A High Throughput Framework for Neural Network Inference on Encrypted Data”, WAHC'19: Proceedings of the 7th ACM Workshop on Encrypted Computing & Applied Homomorphic Cryptography, p 45-56, 2019

[5] Dathathri, Roshan & Saarikivi, Olli & Chen, Hao & Laine, Kim & Lauter, Kristin & Maleki, Saeed & Musuvathi, Madanlal & Mytkowicz, Todd, “CHET: an optimizing compiler for fully-homomorphic neural network inferencing”, PLDI 2019: Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, p 142-156, 2019

[6] Boura, Christina & Gama, Nicolas & Georgieva, Mariya & Jetchev, Dimitar, “CHIMERA: Combining ring-LWE based fully homomorphic encryption schemes. Journal of Mathematical Cryptology”, Journal of Mathematical Cryptology, 2020.