Accelerating TEE-Based DNN Inference Using Mean Shift Network Pruning

Chengyao Xu and Shangqi Lai(✉)

Faculty of Information Technology, Monash University, Clayton, Australia
cxu100@student.monash.edu, shangqi.lai@monash.edu

Abstract. In recent years, deep neural networks (DNNs) have achieved great success in many areas and have been deployed as cloud services to bring convenience to people's daily lives. However, the widespread use of DNNs in the cloud brings critical privacy concerns. Researchers have proposed many solutions to address the privacy concerns of deploying DNN in the cloud, and one major category of solutions rely on a trusted execution environment (TEE). Nonetheless, the DNN inference requires extensive memory and computing resources to achieve accurate decision-making, which does not operate well in TEE with restricted memory space. This paper proposes a network pruning algorithm based on mean shift clustering to reduce the model size and improve the inference performance in TEE. The core idea of our design is to use a mean shift algorithm to aggregate the weight values automatically and prune the network based on the distance between the weight and center. Our experiments prune three popular networks on the CIFAR-10 dataset. The experimental results show that our algorithm successfully reduces the network size without affecting its accuracy. The inference in TEE is accelerated by 20%.

1 Introduction

Deep neural networks (DNNs) have become the mainstream computing model in the artificial intelligence industry [30]. It shows superior performance in many fields such as image recognition [21], speech recognition [8], and natural language processing [48]. Recent advances of Machine Learning as a Service (MLaaS) [3,4] enable users to deploy DNN models on the cloud and perform complex predication tasks efficiently and economically. However, the use of such cloud services raises privacy concerns. For example, an enterprise may use proprietary networks/datasets to conduct its core business. These networks/datasets are high-value assets for the enterprise and should not be revealed to cloud providers.

To mitigate the above privacy concerns, privacy-preserving machine learning has attracted attention; this approach encrypts the DNN model and dataset and processes machine learning tasks over the encrypted model and data. Existing studies in this area can be classified into software-based [7,10,22,23,27,31] and hardware-based (TEE-based) solutions [14,18,33,41].

X. Yuan et al. (Eds.): QShine 2021, LNICST 402, pp. 25–41, 2021.
https://doi.org/10.1007/978-3-030-91424-0_2

The software-based solutions leverage cryptographic tools like homomorphic encryption [7,10] and secure multiparty computation [20,22,23,27,31]. Those works archive a high level of privacy protection. However, due to the nature of cryptography algorithms, those approaches introduce a large number of computation operations. Thus, those mechanisms are hard to deploy to protect complex DNN models without incurring a significant inference latency.

Hardware-based (TEE-based) solutions rely on the trusted hardware to process sensitive information within an isolated environment (known as enclave). It has been widely used in many application contexts such as encrypted database [5,43,44], secure network functions [9,24,35] and also becomes a promising alternative of cryptographic tools when designing privacy-preserving machine learning schemes [14,18,33,41]. Nonetheless, existing works demonstrate that the limited memory size of the enclave restricts the performance of DNN inference. In specific, VGG-16, one of the most common DNN models, requires approximately 1 GB of memory for inference [26], while the maximum protected memory is only 128 MB in the Intel SGX [1]. Using extra memory in Intel SGX will trigger paging, which requires extra page swapping, data encryption, decryption operations and introduces a noticeable delay on the inference time (5× as shown in [40]).

To alleviate the memory consumption and accelerate TEE-based DNN inference, we present a novel networking pruning algorithm based on the mean shift clustering technique. We implement our algorithm and use it to prune three of the most commonly used DNN models: Res-NET [15], Densenet [17], and VGG-Net [39], trained with CIFAR-10 dataset. Then, we conduct inference tasks over our compressed DNN model inside an Intel SGX runtime called Occlum [37]. Our experimental results show that the proposed algorithm can effectively recognize and remove more than 35% of the redundant parameters for all three networks, together with an approximate 30% reduction in FLOPs. Moreover, compared to the unpruned network, the inference delay of the pruned network is reduced by 20% under the Intel SGX environment, and it only introduces a negligible accuracy loss (only 0.1%).

The rest of this paper is structured as follows. We discuss related work in Sect. 2. Then, we introduce the background knowledge in Sect. 3 and present an overview of our design goals and threat model in Sect. 4. In Sect. 5, we elaborate on the details of the novel network pruning algorithm. Next, we describe the experimental setup and the corresponding result/discussions regarding our solution in Sect. 6 and 7. We give a conclusion in Sect. 8.

2 Relative Work

2.1 Software-Based Secure Machine Learning

Secure machine learning involves protecting the privacy of both the training data and the model. In past years, cryptographic tools such as homomorphic encryption [7,10] and secure multi-party computation [20,22,23,27,31] have been increasingly used to handle this problem. However, the above solutions can only

be applied to small-scale problems because the primitives incur intensive computations and introduce a noticeable delay when processing huge tasks.

2.2 TEE-Based Secure Machine Learning

Recently, TEE-based secure machine learning services have been widely studied. Particularly, these services [14,18,33,41] deploy machine learning tasks (can be training, inference or both) inside TEE to protect the privacy of user data and machine learning models. Nonetheless, TEE-based services still suffer the memory access pattern side-channel attacks as well as the performance issue due to the memory size limit of TEE platforms. Several works [33,34] targeting the memory access side-channel attacks have been presented. They leverage oblivious primitives to hide the memory access pattern and prevent the adversary from inferring extra information from the memory trace of machine learning tasks. On the other hand, it is still challenging to mitigate the delay introduced by TEE's memory size limit. For instance, Slalom [41] try to improve the performance of the model inference process in the TEE. It successfully allocates all linear layer calculations in DNN to untrusted but fast hardware, i.e., GPU. However, due to the collaboration between trusted and untrusted hardware, the system can not guarantee the privacy of DNN models. Also, Graviton [45] introduces a TEE on GPUs to speed up the secure inference process on GPUs, but it is just a blueprint feature which does not support by any off-the-shelf GPU.

2.3 Memory Efficient Machine Learning

Several methods have been proposed to optimize DNN models, such as Deep Compression [12], Network Sliming [28], ThiNet [29]. As a result, the amount of parameters required by the network is reduced compared to the original network, thus achieving acceleration and compression. However, those methods, including our approach, only focus on reducing the redundant parts of the model, which still requires a large amount of memory capacity for the calculations. Occlumency [26] proposes a direct way to optimize the memory usage of convolutional layers and achieved considerable speed up for the inference process. Our work can draw on the strengths of Occlumency to achieve better inference performance.

3 Background

3.1 Intel SGX

Intel SGX [1] is an x86 instruction set designed to improve the security of application code and data. On SGX-enabled platforms, applications are divided into trusted and untrusted parts. The trusted part, called enclave, is located in a dedicated part of the physical RAM and is strongly protected by SGX, while the untrusted part is executed as a general process. The untrusted part can only invoke a set of well-defined interfaces (ecall/ocall) to send and receive data from

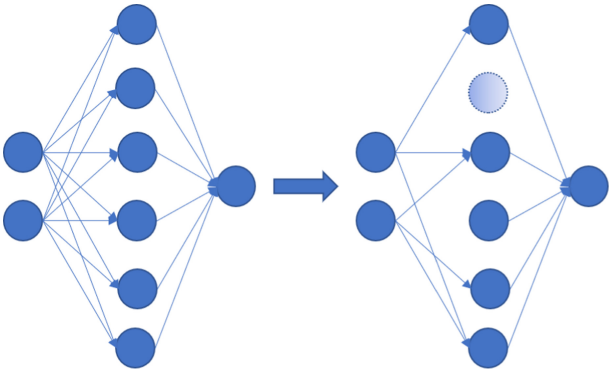

Fig. 1. Network pruning example: the technique prunes neurons (and its input/output) which are classified as "unnecessary" by the algorithm.

the enclave. All other software, including operating systems, privileged software, virtual machine management programs, and firmware, cannot access the enclave memory. SGX uses a remote attestation mechanism to ensure the enclave itself is trusted. The originator of the communication can verify that the SGX enclave is not tampered with against the authentication service provided by Intel.

SGX Runtime. To develop SGX applications, SGX SDK, such as Intel SGX SDK, Open Enclave SDK, Google Asylo, or Apache Rust SGX SDK, has been designed to provide basic supports on high-level languages, APIs and documentation. However, developing an SGX application still involves considerable efforts under the SGX environment. For instance, developers need to decide which components should be placed inside the enclave and outside the enclave and how the two parties should communicate. For complex applications, determining an efficient, logical, and secure partitioning scheme can be challenging in itself. Meanwhile, the developer should consider how to convert the legacy function calls to a secure version because SGX SDK does not support most of the system calls in the legacy system. To simplify the development of the SGX application, researchers have proposed many library OS for Intel SGX, such as SGX-LKL [36], Occlum [37], Graphene-SGX [42], and SCONE [6]. Those library OS contain a group of secure system calls for legacy code. They enable their users to port the entire application into the enclave with few or even no modifications, which highly reduce the development cost of SGX applications.

In this paper, we choose to use Occlum [37] to reduce the development cost. Occlum is the first TEE OS developed in Rust, dramatically reduces the chance of memory security issues, and provides 10-1000× faster process startup and 3× higher throughput for inter-process communication than the previous state-of-the-art open-source TEE OS (Graphene-SGX [42]).

3.2 Network Pruning

Network pruning is a network compression method that removes the redundant parts of the model to make the model more refined and improve the model's generalization performance. It can be classified into two types: unstructured pruning and structured pruning according to the network pruning object. Figure 1 shows the basic idea of Network Pruning.

Unstructured pruning methods [11,13] prune individual neurons (or individual connection weights) in the network and turn the dense connections between the neurons of the original network into sparse connections. Those methods aim to improve the runtime performance of networks by converting the original dense matrix operation into a sparse matrix operation. However, we note that the commonly used hardware in DNN does not provide additional acceleration for sparse matrix operations [28]. Hence, the unstructured pruning methods may not offer any acceleration for DNN but may be slower.

In recent years, structured pruning [29,32] is proposed to prune the network. Compared to unstructured pruning, structured pruning does not focus on the pruning of individual connection weights. Instead, it removes all values on the whole filter (filter-based) or layer (layer-based). Although its degree of freedom of pruning is relatively lower, and the degree of pruning is also lower, it can be more efficient in practice. The reason is that such methods do not require specific hardware platforms or computational libraries for the use of the pruned network. The pruned models can run directly on the mainstream deep learning frameworks nowadays to obtain direct acceleration.

4 Overview

4.1 Design Goals

This work aims to archive the following three goals:

Improved Inference Speed. Our design helps to achieve a low DNN inference latency compared to directly port the deep learning inference in the SGX enclave. Our pruning algorithm can significantly reduce the memory space required for inference service, which can reduce the paging operation of the SGX enclave. Also, by using the algorithm, the Floating Point Operations per second (FLOPs) of the inference process are reduced, which can also reduce the inference latency.

No Accuracy Loss. Our design prunes the DNN model to accelerate the DNN inference. In order to preserve the accuracy of the inference result, we carefully design the algorithm we used for model pruning so the pruned model will not incur any accuracy loss compared to the original model.

Privacy Protection for Whole Inference Service. Our design enables users to fit the entire DNN model into the enclave memory. Thus, the user can send their private data to the enclave for the prediction tasks, and the whole inference service will be shielded by TEE. As a result, our solution can provide efficient

Fig. 2. Flow-chart of pruning procedure.

black-box access to the model and protect the models against white-box level attacks. More privacy-preserving technologies should apply to the models (e.g., differential privacy) to migrate black-box level attacks, but those are beyond the scope of our work.

4.2 Threat Model

In this paper, we consider the scenario that a user subscribes to a DNN inference service on a cloud platform. The user will upload his/her DNN model to the cloud and obtain inference results from the cloud. We assume a powerful adversary who is capable of accessing all the resources running on the cloud, such as software and GPU. The adversary can obtain and analyze the memory trace of all processes except those running within the enclave. The adversary aims to leverage the collected information to infer useful information regarding user's DNN model and data. Our design leverages Intel SGX to protect user's private input (DNN model and data) against the above adversary.

We follow the existing works [5,9,35,43,44] to consider SGX side-channel attacks [25,38,47] and Denial of Services attack out of scope.

5 Mean Shift Network Pruning

To accelerate the DNN inference in the enclave, we design a network pruning algorithm to remove the redundant part of the DNN model. Figures 2 shows the general idea of our pruning algorithm. In the rest of this section, we introduce our proposed mean shift network pruning algorithm in detail.

Select BN Layers. During network training, parameter updates will cause the input data distribution in each layer to change continuously. Whenever the parameters are updated in the upper layer, it will inevitably cause a change in the input data distribution in the lower layer. This creates a complicated data distribution, which makes the analysis of the relationship of the weights a difficult task. In deep neural networks, this phenomenon is more serious. To solve the above issue, the BN algorithm can be applied to normalize the input at each layer and make the input data distribute stably during the training process. The proposed protocol leverages the benefit of using the Batch normalization (BN) [19] to prune the unnecessary channel. After loading the pre-trained models, the algorithm selects Batch normalization (BN) layers from the whole model for the pruning process. BN helps to improve the performance of our design because

the training process with BN produces a more stable data distribution, which reduces the difficulty of analyzing the relationship of the weights. Moreover, the convolutional weights associated with BN are cut off, eliminating the need to re-analyze the association between each layer.

Mean Shift Clustering Algorithm. After selecting the BN layer, we choose to apply the mean shift clustering algorithm to cluster the weight value of the BN layer and prune out the unnecessary channel based on the cluster.

Clustering is a machine learning technique that involves the grouping of data points. Given a set of data points, we can use a clustering algorithm to divide data points into several groups. Theoretically, data points in the same group should have similar attributes and features, while data points in different groups should have highly different attributes and features. Clustering is an unsupervised learning method and is a standard statistical data analysis technique used in many fields. There are many clustering algorithms such as K-Means, K-Means++. In the K-Means algorithm, the final clustering result is influenced by the initial clustering center. The K-Means++ algorithm is proposed to provide a basis for choosing a better initial clustering center. However, the number of classes of clusters in the algorithm still needs to be formulated in advance. It will be challenging to solve K-Means and K-Means++ accurately for data sets with an unknown number of classes in advance.

The mean shift algorithm is a clustering algorithm based on the cluster center, like K-Means, but the difference is that the mean shift algorithm does not require a priori knowledge about the number of categories. Specifically, the mean shift algorithm employs an iterative step in which the offset mean of the current point is calculated. The point is moved to this offset mean, and then this is used as the new starting point to continue moving until the final condition is satisfied. For n sample points $x_i, i = 1, ..., n$ in a given d-dimensional space R^d, the formula of the mean shift vector for point x is:

$$M_h(x) = \frac{1}{k} \sum_{x_i \in S_h} (x_i - x), \tag{1}$$

where S_h refers to a high-dimensional spherical region of radius h, and adding all the points in it to the vector starting from the center of the circle is the mean shift vector (M_h).

Then, for each element in the set, the algorithm moves the element to the position of the mean of the eigenvalues of all elements in its neighborhood. The above step repeats until convergence. The element is marked as being in the same class when the element is in its convergence position.

A problem when solving the mean shift vector is that the algorithm assumes each sample point x_i contributes the same amount to sample X in the region of S_h. In practice, the contribution of each sample point x_i to sample X is different, and a kernel function can measure such sharing. A kernel function is a common approach used in machine learning. Suppose that the kernel function is added to the basic M_h to make each sample point x_i contribute differently to the sample X within a range S_h of radius h. The influence of distance is considered when

calculating M_h. It can also be assumed that the importance is not the same for all sample points X. Therefore, a weighting factor is also introduced for each sample. In this way, the M_h form can be extended as follows:

$$M_h(x) = \frac{\sum_{i=1}^{n} G(\frac{x_i - x}{h}) w(x_i)(x_i - x)}{\sum_{i=1}^{n} G(\frac{x_i - x}{h}) w(x_i)}, \tag{2}$$

where $G(\frac{x_i - x}{h})$ is the kernel function and $w(x_i)$ is a weight assigned to the sampling points. The pseudocode for mean shift can be seen in Algorithm 1:

Algorithm 1. mean shift clustering algorithm

Input: Data set D, Radius h
Output: Cluster C
Initialization: Cluster C, Threshold t
repeat:
Select a random point in the unlabeled D as *center*
Initialize cluster C_n
    repeat:
        Find all points that are within radius h from *center*, denoted as the set S.
        Sum up the vectors from *center* to all elements in the S as M_h
        Update all points to C_n with the w (in formula 2) that these points belong to C_n
        $center \leftarrow center + |M_h|$
    until: M_h converge
    if the distance between C_n's *center* and C_{n-1}'s *center* smaller then t **then**
        merge C_n to C_{n-1}
    else add C_n to C
until all points in D have be accessed.
Classify each points to their cluster based on the largest w

In this work, the algorithm will take inputs as the weight data of each BN layer (data set) and the automatically detected radius from the build-in function *estimate_bandwidth* with default quantile value. After getting the results of the clustering, the information is passed onto the next stage.

Channel Pruning. In the above steps, our approach generates the mean shift clusters. Next, it selects the pruned channels by referring to use the minimum value among all cluster centers. The weight value smaller than the minimum center value will be pruned out. In the pruning phase, our approach will prune channels by removing all their incoming and outgoing connections and corresponding weights.

Advantages of Channel-Wise Pruning. As discussed in Sect. 3.2, network pruning can be divided into two categories: unstructured pruning and structured pruning. Unless the lower-level hardware and computational libraries better support it, it is not easy to get substantial performance gains with unstructured

pruning. Therefore, much of the research in the past few years has focused on the structured pruning. The structured pruning methods can be further subdivided to layer-wise, channel-wise and filter-wise.

The pruning object of the layer granularity-based pruning algorithm is the entire network layer. The purpose of pruning is to make a deep neural network into a relatively shallow network. This pruning method's advantages are undeniable, as the use of pruned networks does not require any unique toolkit for inference acceleration. However, apart from this advantage, the limitations and drawbacks of this approach are also obvious. The direct pruning of the network layers makes the method extremely inflexible, and it is only effective when a network model is deep enough (more than 50 layers) to trim off some layers [46]. We also found that layer-based network pruning algorithms are inflexible; the sparse operations of weight-based pruning (unstructured pruning) algorithms cannot be accelerated directly on existing hardware platforms and computational libraries.

On the other hand, recent network pruning algorithms based on convolutional kernel and channel achieves a balance in flexibility and implementation. Moreover, it can be applied to any typical CNN or fully connected layer (treating each neuron as a channel). The resulting network is also essentially a thin network that can be reasoned quickly on CNN platforms. Since cropping a feature map is equivalent to cropping the connected convolutional kernels together, channel-wise and filter-wise pruning is in the same class of algorithms.

6 Experiment

We evaluate the performance of our works in terms of the number of model parameters, FLOPs, model accuracy and inference latency with a variety of DNN models. We have implemented our work based on PyTorch [2].

6.1 Experiment Setup

Dataset. We evaluated our work upon the CIFAR-10 dataset. The CIFAR-10 dataset is a small dataset for identifying common objects containing 10 categories of color images, including airplanes, cars, birds, cats, etc. There are 60,000 images in the dataset. Each category contains 6,000 images, of which 5,000 images are used for training, and another 1,000 images are used for testing. The size of all images is 32×32 pixels.

Models. We use three well-known CNN models to evaluate our design, VGG, ResNet, and DenseNet.

VGG. VGG is a deep convolutional neural network for image recognition developed and trained by the Computer Vision Group at the University of Oxford, which achieved second place in the classification event and first place in the localization event ILSVRC2014 competition. Commonly used versions of the VGGNet structure are VGG-16 and VGG-19. To facilitate comparisons with existing methods, the experiments in this section use the VGG-16 network, which

Table 1. The result on the CIFAR-10 dataset.

Model	Accuracy	Parameters	Pruned	FLOPs	Pruned	Pruned ratio
VGGNet (Baseline)	94.2%	14.72M	-	313.75M	-	-
VGGNet (Pruned)	84.2% (94.0%)	7.07M	52.1%	237.36M	24.3%	38.2%
ResNet-164 (Baseline)	95.0%	1.70M	-	253.95M	-	-
ResNet-164 (Pruned)	87.3% (94.8%)	1.11M	34.7%	173.40M	31.7%	33.2%
DenseNet-40 (Baseline)	94.2%	1.06M	-	287.71M	-	-
DenseNet-40 (Pruned)	87.0% (94.1%)	0.69M	34.9%	204.71M	28.9%	31.9%

"Baseline" denotes the pre-trained model we used in the experiment without any pruning operation. In the 2^{nd} column, "84.2% (93.9%)" indicates the pruned model has 84.2% accuracy, and fine-tuned pruned model has 93.9% accuracy. The 4^{th} and 6^{th} columns show the pruned ratio of parameters and FLOPs. The total pruned ratio of the model is in the 7^{th} column.

consists of 16 layers, including 13 convolutional layers and 3 fully connected layers. For network pruning, we pre-trained the VGG-16 net with CIFAR-10, and the test accuracy is 94.20%.

ResNet. In order to verify the effectiveness of pruning for more compact networks, the experiments further continue to validate the pruning method on the CIFAR-10 dataset based on the ResNet, a deep convolutional neural network, which won the championship in three categories of image classification, detection, and localization in ILSVRC2015. The ResNet-164 network is used in this experiment. Unlike the single-branch VGG16 network, Res Net-164 is a multi-branch network with many more layers, consisting of 163 convolutional layers and a fully connected layer for classification. The trained benchmark model was tested on CIFAR-10 with 95% accuracy after the training.

DenseNet. DenseNet was proposed after ResNet and combining the ideas of ResNet. One of the advantages of DenseNet is that the network is narrower and has fewer parameters. In this paper, we use DenseNet-40 consisting of 39 convolutional layers and a fully connected layer. The benchmark model we used in this part was trained on CIFAR-10 with 94.2% accuracy.

Running Environment. We used a Google Cloud virtual instance as the platform to test our design. The instance equips an Intel Cascade Lake C2 CPU platform with 4 vCPU and 16 GB memory. As the virtual instance does not support SGX hardware mode, the experiment was tested under SGX simulation mode. For SGX runtime, we choose to use Occlum. Occlum is a multi-process library OS (LibOS) for Intel SGX. It provides a container-like, user-friendly interface that allows applications to run on SGX with little or no source code modification. Moreover, Occlum is the first TEE OS developed using the Rust language, significantly reducing memory security issues.

6.2 Result

Parameters and FLOPs Reductions. Our work aims to reduce the number of computational resources required for DNN inference. From Table 1, we can see that each model has over 35% unnecessary channel pruned, and the model accuracy is slightly different from the baseline after the fine-tuning process. For VGGNet, we can find that over 50% of redundant parameters can be pruned,

while for ResNet and DenseNet, the pruned ratio on parameters is relatively miniature, just around 34%. We believed this is because those two networks are more compact than VGGNet and the structure already provides some channel selection function. Figure 3 provides a clear view of the Parameter and FLOPs reductions in three models.

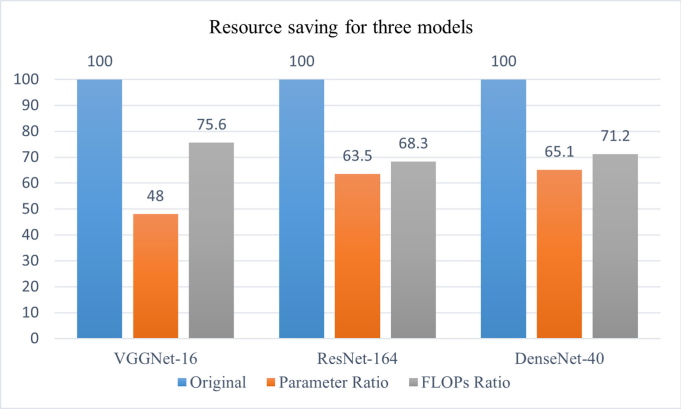

Fig. 3. Comparison of pruned models

We choose Network Slimming [28] (Slim) as the comparison method, and the evaluation metrics include accuracy, pruning rate, reduction in the number of parameters, and FLOPs. We only compare the result on ResNet and DenseNet. Since our method is an adaptive pruning method, it is hard to control the pruning rate compared to Slim's brutal pruning. Thus, the pruning performance on VGGNet is hard to compare.

The result is in Table 2 and 3. From the result, we can observe that our work performed better on ResNet, while on DenseNet, our approach only outperformed on FLOPs reduction. However, the total pruned ratio for our approach is only around 32%, but for Slim, the set pruned ratio is 40%. It shows that our approach uses a lower pruning rate to achieve the effectiveness of Slim's 40% pruning rate. The experiment also shows that iIf we could increase the pruning rate, we can save more resources as more neurons will be removed from the network. However, the high pruning rate may eliminates some important channels hence affect the accuracy of the model. In the next section, we will discuss more details about the pruning rate.

Inference Latency. We compare our work with two baselines. The first one (*SGX*) runs the model inference in the enclave but does not prune the model. We use this baseline to show the effectiveness of our approach in reducing the wait time for model inference in the enclave. In the second baseline (*Native*), the model inference is running as a general program without the protection of SGX. We use this baseline to investigate the overhead of our work.

Table 2. Experimental results of ResNet-164 pruning on CIFAR-10

Method	FLOPs reduction (%)	Params reduction (%)	Accuracy (%)
Slim (40%)	23.7	14.9	**+0.34**
Our work	**31.7**	**34.7**	–0.2

Table 3. Experimental results of DenseNet-40 pruning on CIFAR-10

Method	FLOPs reduction (%)	Params reduction (%)	Accuracy (%)
Slim (40%)	28.4	**35.7**	**+0.92**
Our work	**28.9**	34.9	–0.1

"Slim (40%)" denotes the pruning rate in Network Slimming method is 40%

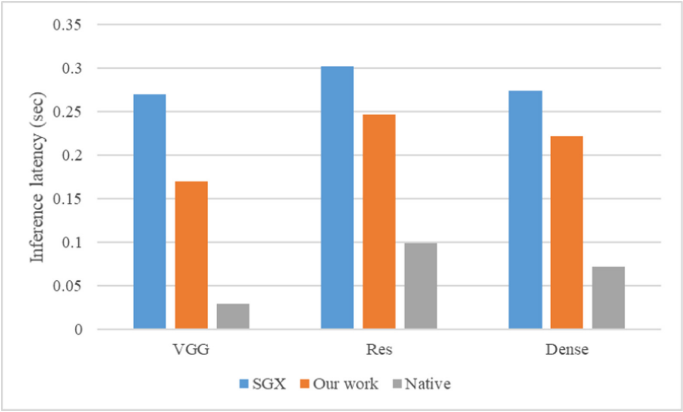

Fig. 4. Inference latency of various deep-learning models in native environment, Occlum runtime (SGX) and our work (Occlum + pruning)

We evaluate the three models' inference times for the experiments in the Native and SGX environments in the unpruned case. The results are shown in Fig. 4. We can see that DNN inference has a significant latency in the SGX environment and up to 9× in VGGNet inference. From the figure, we can see that the inference speed of the pruned model running on SGX is significantly improved. For the VGG-16 network, the redundant parameters are reduced due to the higher pruning rate. The inference speed is improved by 37% compared to the unpruned model. On the other hand, the speedups of ResNet-164 and DenseNet-40 were minor but also improved about 20% for each. Figure 4 also compares the inference speed of the unpruned model running in a native operating environment with the pruned model running in the SGX enclave. It shows that although our solution incurs an extra inference latency, it is still a significant improvement compared to directly port the inference service into SGX.

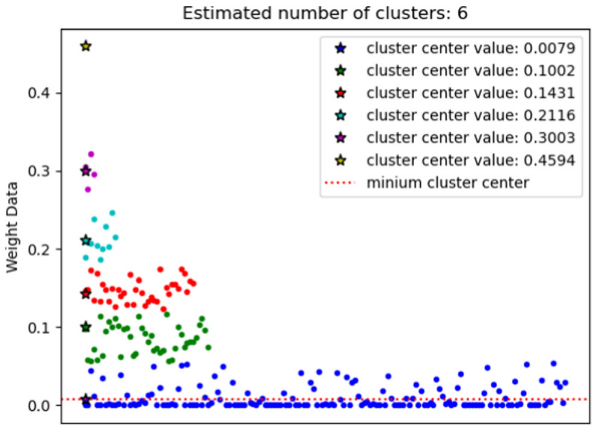

Fig. 5. Clustering result on weight data

7 Discussion

Pruning Rate. As mentioned, our proposed algorithm is an adaptive pruning algorithm based on mean shift clustering. We choose to use the minimum point in the clustering result as the pruning criterion in our experiments. The algorithm runs clustering on the weight value of the BN layer. The relationship between weight value and channel importance is proportional. The smaller the weight value is, the less impact on the overall network structure. Thus, we choose to prune off the channels with a weight value smaller than the minimum centroid value. The experiments show that the pruning rate for the three networks is between 30% and 40% when we use the minimum value. Figure 5 shows the weight distribution after mean shift clustering. In order to improve the pruning rate further, we have considered using the mean value of all cluster centroids as the pruning criterion. However, we can see from the figure that most weight distributions are below the average, leading to a significant pruning rate. On VGGNet, we used the average value for pruning which led to some layers being completely pruned, thus destroying the entire network structure. In ResNet and DenseNet, we can achieve a higher pruning rate by using the average value. However, the model's accuracy is greatly affected due to the reduction of channels. To sum up, mean shift clustering helps us classify the weight values effectively, but it is challenging to filter out more irrelevant weights from the cluster classes and increase the pruning rate.

Another point to mention is that this work is tested on the popular CNN networks. Evaluating the effect of algorithms on DNN models is our future research direction.

Flexibility. The existing structured pruning algorithms are usually based on sparse parameters for pruning. For example, Network Slimming adds a channel-wise scaling factor to the BN layer and applies an L1 regularizer to obtain sparse

parameters. Network Trimming [16] tends to produce sparse activation values for activation functions such as RELU to perform pruning. Although good results can be achieved, pruning involves more or less the setting of hyperparameters in algorithm implementation, such as determining the pruning dimension of the model or pruning rate of the model. Moreover, most of the models need to be retrained to get the sparse parameters.

We also compared our work with [49] which archived a better pruning rate than ours with similar accuracy loss. In [49], the authors claim that the L1 regularisation technique would cause all scale coefficients to converge to zero. Instead, their solution aims to reduce the scale coefficients of the "unimportant" channels only while keeping the other coefficients large. However, this technique still requires retraining the model to polarise the weight distribution of the BN layer before pruning, which can take a much longer time to proceed comparing to our solution. On the other hand, our algorithm relies only on the benefits of the BN layer and mean shift clustering, which significantly reduces the preprocessing requirements of the model and can directly prune most existing models and achieve a notable pruning rate. In future work, we will replicate their algorithm and perform mean shift clustering on the polarised weights and then prune them to compare the experimental results.

Another benefit of our work is that by modifying the code, we believe that our algorithm can provide a TEE-based privacy-preserving network pruning service that allows people to compress their own trained models without any relevant knowledge. This could be a future research direction for us.

Inference Acceleration. As in the experimental result, our method can improve the inference speed of the model in SGX and does not affect the model's accuracy. However, our experiments are only based on the CIFAR-10 dataset, which is difficult to compare with the ImageNet dataset used in other papers. Moreover, our method only reduces the number of parameters and FLOPs of the model and does not fundamentally solve the memory occupation problem faced by DNN model inference in SGX. There have been some studies on memory optimization for using DNN models in SGX. Occlumency [26] proposed Partitioned Convolution to replace the traditional im2col convolutional computation, which greatly reduces the memory required for convolutional computation. At the same time, they observe that most models sequentially compute each layer and leverage this feature to optimize the feature maps of the convolution layer output. Particularly, they discard the feature maps that are no longer used to relieve the memory pressure of SGX. However, their approach requires extensive modifications to the underlying framework of deep learning and is more restrictive to the models used. Using GPU acceleration is currently the mainstream acceleration approach for DNN models. Nonetheless, the traditional SGX does not support the use of GPU. SLALOM [41] proposes a DNN inference acceleration using both GPU and SGX. But their approach requires part of the computation to be performed on an unprotected GPU, which can lead to some privacy issues. Memory optimization for DNN and establishing trust execution environments for GPU DNN inference service still be the critical research direction.

8 Conclusion

This paper presents an unsupervised clustering pruning algorithm based on mean shift clustering, which can effectively compress the existing DNN models. Our evaluations over CIFAR-10 dataset with three models, VGG-16, ResNet-164, and DenseNet-40, show that our pruning algorithm can reduce FLOPs and parameters by more than 30%. After deploying the pruned models to SGX, our method speeds up about 20% compared to the traditional DNN inference service in SGX. The proposed algorithm enables complicated DNN models to be fit into the constraints of memory under the SGX environment. Furthermore, it successfully reduces the inference latency in the SGX while preserving the privacy of users' data and models.

References

1. Intel Software Guard Extensions. https://software.intel.com/content/www/us/en/develop/topics/software-guard-extensions.html [online]
2. PyTorch. https://pytorch.org [online]
3. Azure Machine Learning (2021). https://azure.microsoft.com/en-au/services/machine-learning/
4. Google Vertex AI (2021). https://cloud.google.com/vertex-ai
5. Amjad, G., Kamara, S., Moataz, T.: Forward and backward private searchable encryption with SGX. In: EuroSec 2019 (2019)
6. Arnautov, S., et al.: SCONE: secure linux containers with intel SGX. In: USENIX OSDI 2016 (2016)
7. Chabanne, H., de Wargny, A., Milgram, J., Morel, C., Prouff, E.: Privacy-preserving classification on deep neural network. IACR Crypto ePrint: 2017/035 (2017)
8. Deng, L., Hinton, G., Kingsbury, B.: New Types of deep neural network learning for speech recognition and related applications: an overview. In: IEEE ICASSP 2013 (2013)
9. Duan, H., et al.: LightBox: full-stack protected stateful middlebox at lightning speed. In: ACM CCS1 2019 (2019)
10. Gilad-Bachrach, R., et al.: CryptoNets: applying neural nnetworks to encrypted data with high throughput and accuracy. In: ICML 2016 (2016)
11. Guo, Y., Yao, A., Chen, Y.: Dynamic network surgery for efficient DNNs. arXiv preprint:1608.04493 (2016)
12. Han, S., Mao, H., Dally, W.J.: Deep compression: compressing deep neural networks with pruning, trained quantization and Huffman coding. arXiv preprint:1510.00149 (2016)
13. Han, S., Pool, J., Tran, J., Dally, W.J.: Learning both weights and connections for efficient neural networks. arXiv preprint:1506.02626 (2015)
14. Hashemi, H., Wang, Y., Annavaram, M.: DarKnight: a data privacy scheme for training and inference of deep neural networks. arXiv preprint:2006.01300 (2020)
15. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. arXiv preprint:1512.03385 (2015)
16. Hu, H., Peng, R., Tai, Y.W., Tang, C.K.: Network trimming: a data-driven neuron pruning approach towards efficient deep architectures. arXiv preprint:1607.03250 (2016)

17. Huang, G., Liu, Z., Van Der Maaten, L., Weinberger, K.Q.: densely connected convolutional networks. In: IEEE CVPR 2017 (2017)
18. Hunt, T., Zhu, Z., Xu, Y., Peter, S., Witchel, E.: Ryoan: a distributed sandbox for untrusted computation on secret data. In: USENIX OSDI 2016 (2016)
19. Ioffe, S., Szegedy, C.: Batch normalization: accelerating deep network training by reducing internal covariate shift. In: ICML 2015 (2015)
20. Juvekar, C., Vaikuntanathan, V., Chandrakasan, A.: GAZELLE: a low latency framework for secure neural network inference. In: USENIX Security 2018 (2018)
21. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. In: NIPS 2012 (2012)
22. Lai, S., et al.: Enabling efficient privacy-assured outlier detection over encrypted incremental datasets. IEEE Internet of Things J. **7**(4), 2651–2662 (2019)
23. Lai, S., et al.: GraphSE^2: an encrypted graph database for privacy-preserving social search. In: ACM ASIACCS 2019 (2019)
24. Lai, S., et al.: OblivSketch: oblivious network measurement as a cloud service. In: NDSS 2021 (2021)
25. Lee, S., et al.: Inferring fine-grained control flow inside SGX enclaves with branch shadowing. In: USENIX Security 2017 (2017)
26. Lee, T., et al.: Occlumency: privacy-preserving remote deep-learning inference using SGX. In: ACM MobiCom 2019 (2019)
27. Liu, J., Juuti, M., Lu, Y., Asokan, N.: Oblivious neural network predictions via MiniONN transformations. In: ACM CCS 2017 (2017)
28. Liu, Z., et al.: Learning efficient convolutional networks through network slimming. In: IEEE ICCV 2017 (2017)
29. Luo, J.H., Wu, J., Lin, W.: ThiNet: A Filter level pruning method for deep neural network compression. In: IEEE ICCV 2017 (2017)
30. Masi, I., Wu, Y., Hassner, T., Natarajan, P.: Deep face recognition: a survey. In: SIBGRAPI 2018 (2018)
31. Mishra, P., Lehmkuhl, R., Srinivasan, A., Zheng, W., Popa, R.A.: DELPHI: a cryptographic inference service for neural networks. In: USENIX Security 2020 (2020)
32. Molchanov, P., Tyree, S., Karras, T., Aila, T., Kautz, J.: Pruning convolutional neural networks for resource efficient inference. arXiv preprint:1611.06440 (2016)
33. Ohrimenko, O., et al.: Oblivious multi-party machine learning on trusted processors. In: USENIX Security 2016 (2016)
34. Poddar, R., Ananthanarayanan, G., Setty, S., Volos, S., Popa, R.A.: Visor: privacy-preserving video analytics as a cloud service. In: USENIX Security 2020 (2020)
35. Poddar, R., Lan, C., Popa, R.A., Ratnasamy, S.: Safebricks: shielding network functions in the cloud. In: USENIX NSDI 2018 (2018)
36. Priebe, C., et al.: SGX-LKL: securing the host OS interface for trusted execution. arXiv preprint:1908.11143 (2020)
37. Shen, Y., et al.: Occlum: secure and efficient multitasking inside a single enclave of intel SGX. In: ACM ASPLOS 2020 (2020)
38. Shinde, S., Chua, Z.L., Narayanan, V., Saxena, P.: Preventing page faults from telling your secrets. In: ACM AsiaCCS 2016 (2016)
39. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint:1409.1556 (2014)
40. Taassori, M., Shafiee, A., Balasubramonian, R.: VAULT: reducing paging overheads in SGX with efficient integrity verification structures. In: ACM ASPLOS 2018 (2018)

41. Tramer, F., Boneh, D.: Slalom: fast, verifiable and private execution of neural networks in trusted hardware. In: ICLR 2019 (2019)
42. Tsai, C.C., Porter, D.E., Vij, M.: Graphene-SGX: a practical library OS for unmodified applications on SGX. In: USENIX ATC 2017 (2017)
43. Vo, V., Lai, S., Yuan, X., Nepal, S., Liu, J.K.: Towards efficient and strong backward private searchable encryption with secure enclaves. In: ACNS 2021 (2021)
44. Vo, V., et al.: Accelerating forward and backward private searchable encryption using trusted execution. In: ACNS 2020 (2020)
45. Volos, S., Vaswani, K., Bruno, R.: Graviton: trusted execution environments on GPUs. In: USENIX OSDI 2018 (2018)
46. Wen, W., Wu, C., Wang, Y., Chen, Y., Li, H.: Learning structured sparsity in deep neural networks. arXiv preprint:1608.03665 (2016)
47. Xu, Y., Cui, W., Peinado, M.: Controlled-channel attacks: deterministic side channels for untrusted operating systems. In: IEEE S&P 2015 (2015)
48. Young, T., Hazarika, D., Poria, S., Cambria, E.: Recent trends in deep learning based natural language processing. IEEE Comput. Intell. Mag. **13**(3), 55–75 (2018)
49. Zhuang, T., Zhang, Z., Huang, Y., Zeng, X., Shuang, K., Li, X.: Neuron-level structured pruning using polarization regularizer. In: NeurIPS 2020 (2020)