Verifying the Quality of Outsourced Training on Clouds

Peiyang Li[1,2], Ye Wang[1,2], Zhuotao Liu[1,2], Ke Xu[1,2], Qian Wang[3], Chao Shen[4], and Qi Li[1,2(✉)]

[1] Tsinghua University & BNRist, Beijing, China
{lpy20,wangye22}@mails.tsinghua.edu.cn,
{zhuotaoliu,xuke,qli01}@tsinghua.edu.cn
[2] Zhongguancun Lab, Beijing, China
[3] Wuhan University, Wuhan, China
qianwang@whu.edu.cn
[4] Xi'an Jiaotong University, Xi'an, China
chaoshen@mail.xjtu.edu.cn

Abstract. Deep learning training is often outsourced to clouds due to its high computation overhead. However, clouds may not perform model training correctly due to the potential violations on Service Level Agreement (SLA) and attacks, incurring low quality of outsourced training. It is challenging for customers to understand the quality of outsourced training on clouds. They cannot measure the quality by simply testing the trained models because the testing performance is impacted by various factors, e.g., the quality of training and testing data. In order to address these issues, in this paper, we propose a novel framework that allows customers to verify the quality of outsourced training without modifying the processes of model training. Particularly, our framework achieves black-box verification by utilizing an extra training task that can be learned by the model only after the model converges on the original training task. We construct well-designed extra training tasks according to the original tasks, and develop a training quality verification method to measure the model performance on the extra task with a hypothesis testing-based threshold. The experiment results show that the models passing the quality verification achieve at least 96% of their best performance with negligible accuracy loss, i.e., less than 0.25%.

Keywords: Outsourced deep learning services · Verification

1 Introduction

Deep learning has been widely used for a variety of tasks, such as image recognition [10] and sentiment classification [24]. Despite achieved classification performances, the training of these models requires both substantial computational resources and deep learning expertise. Outsourced training on clouds, such as Amazon SageMaker [2], Google Vertex AI [3], and Microsoft Azure [4], can liberate customers from the heavy lifting of training deep learning models. Namely,

V. Atluri et al. (Eds.): ESORICS 2022, LNCS 13555, pp. 126–144, 2022.
https://doi.org/10.1007/978-3-031-17146-8_7

the customer only needs to upload the training data. The cloud will conduct the training algorithm and return a well trained model which is convergent on the training data.

However, the outsourced training is typically invisible to customers. A customer is unable to make sure whether the model returned by the cloud is well trained due to various reasons. For example, cloud providers may have financial incentives to stealthily violate Service Level Agreements (SLAs) [16,35,36]. Model training may be stopped before the models are well trained, e.g., in order to save their computing resources and reap economic benefits. Moreover, adversaries may directly perform malicious operations to interfere with model training on clouds [8,11,29]. All these issues will incur low quality of outsourced training on clouds, leading to inaccurate model inferences.

A number of existing methods have been developed to verify outsourced inference. Nevertheless, they are unable to efficiently verify the quality of outsourced training. They either utilize verifiable computing techniques [11,14,21,36], or enable verifiability for deployed models [16,31]. The former incurs significant model training overhead, while the latter can only ensure the integrity of deployed models. In order to ensure the quality of outsourced training, we need to address the following challenges: (i) It is infeasible to monitor the status of outsourced training. We cannot simply use the results of training loss and validation accuracy because they can be easily disturbed. (ii) Besides the training quality, the performance of a model is also impacted by various factors, such as the quality of training set and testing set, which is hard to measure. Thus, a customer can hardly predict the model performance of a well trained model so as to verify the quality of training.

In this paper, we take the first step towards the verification of outsourced training. To overcome the challenges above, we propose a novel and effective framework to achieve black-box verification on outsourced training using a carefully designed extra task. Specifically, by utilizing the over-parameterization property of deep learning models, we make the model learn an extra task that is semantically different from the original task specified by the customer. We propose to construct an extra task which will only be learned by the model after the model converges to the original task. The customers can verify the training quality by measuring the model performance on the extra task. Note that, since we only generate a small number of training samples in the extra task, the extra task incurs a negligible impact on the model performance on the original task and negligible computational overhead. In summary, we make the following contributions:

- To the best of our knowledge, we are the first one to propose a framework for verifying the training quality of an outsourced model.
- We develop a method to construct the extra training task according to the original task, and develop a training quality verification method.
- We conduct a comprehensive evaluation on various datasets with different models to demonstrate that our proposed framework is effective and has a negligible impact on the performance of the model.

2 Background and Problem Statement

2.1 Background

Deep Learning. Deep learning model aims to build a function f_θ that maps the input X to the output Y, where f represents the model structure and θ represents the model parameters. Generally, $\boldsymbol{x} \in X$ is a d-dimensional feature vector and $y \in Y$ is a discrete label.

To make f_θ best capture the relationship between X and Y, training process is performed on a set of labeled data, called training data, $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)_{i=1}^n\}$ to find a set of optimal parameters θ. The optimality of θ can be measured by a loss function $\mathcal{L}$. The trainer generally uses the gradient descent based algorithm [27] to minimize $\mathcal{L}$ on the training data. Specifically, gradient descent is an iterative method where in each step the trainer computes the gradient of the loss function with respect to θ and updates θ based on this gradient. As the iteration progresses, the model gets convergent. The trainer regularly check the model according to certain rules, e.g., the testing accuracy does not go up over a substantial amount of iterations [33], to determine whether the model has been convergent.

Outsourced Deep Learning Services. Development of deep learning requires both substantial computing resources and deep learning expertise. Numerous cloud service providers offer outsourced deep learning services to make it convenient for customers, such as Amazon SageMaker [2], Google Vertex AI [3], Microsoft Azure [4]. There are two main types of services: model training and model deploying [17,18]. The model training services support customers without the knowledge of algorithm details and hardware devices (e.g., GPU) training a deep learning model easily. The customer uploads a training dataset to the cloud. The cloud then performs model training and returns the resulting model to the customer. Generally, the cloud provides a variety of model structures for customers to choose from and even provides an algorithm market [1] which allow third parties to provide model to meet the demands for different scenarios.

The model deploying services help customers with limited resources publish their models for end-users to query. The customer uploads a trained model to the cloud. The cloud allocates the resources to host the model and releases APIs. The cloud is responsible for availability and scalability of deployed models.

2.2 Problem Statement

We consider a scenario in which a customer desires to develop a deep learning model by outsourced training. The customer can choose a training set, upload it to the cloud and specify a model type for model training. The cloud will conduct a training algorithm based on the training set and then return the resulting model (or the model's API) to the customer. The outsourced training is a black-box to the customer.

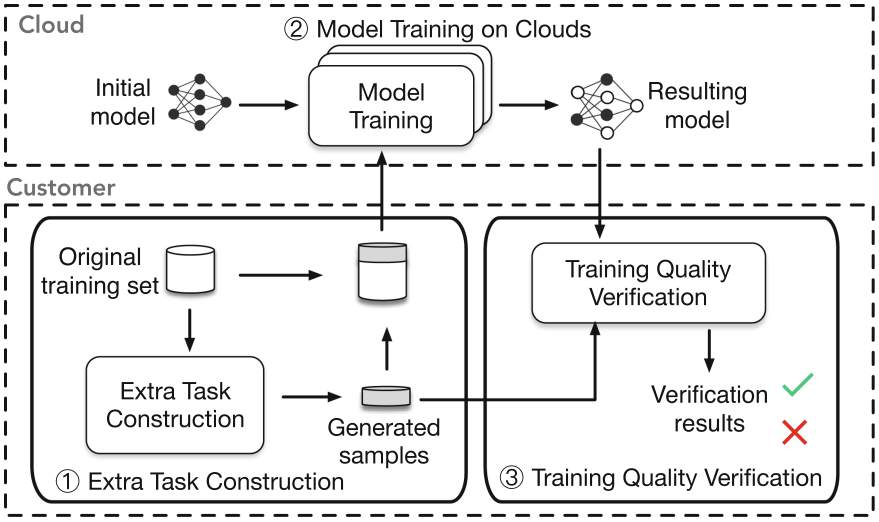


Fig. 1. The overall workflow of our framework.

Normally, to achieve the best model performance, the resulting model should be *well trained* by the cloud to be convergent on the training set (i.e., additional training will not improve the model performance). However, the resulting model may be not well trained due to various reasons. For example, cloud service providers have a financial incentive to stop training before the model is well trained [16,35,36]. Moreover, adversaries may directly perform malicious operations to cause the programs executed in the cloud to terminate unexpectedly [8,11,29]. We do not consider the scenario where the cloud compromises the integrity of training data, e.g., the cloud trains a model using a subset of the training data, which can be addressed by combining the model partition techniques with TEE within limited time consumption [13]. We also do not consider the compromise of the model structure, e.g., a simpler model with fewer parameters is trained to replace the model specified by the customer. The assumption is rational since the customer can easily observe whether the structure of the resulting model is correct.

Our goal is to design a method $\mathcal{S}$ for customers to verify the quality of outsourced training, i.e., verify whether the resulting model f_θ is well trained on the training set $\mathcal{D}$. Formally,

$$\mathcal{S}(f_\theta, \mathcal{D}) \rightarrow \{0, 1\}, \tag{1}$$

where 0 means f_θ is well trained using $\mathcal{D}$ and 1 means otherwise.

3 System Framework

We design a framework that allows customers to verify the quality of outsourced training. The framework is motivated by the over-parameterization property of deep learning models, that is, a deep learning model has parameter redundancy for a task [7,34]. With parameter redundancy, a model can learn multiple tasks

at a time even if these tasks are semantically different [30]. Therefore, in our framework, besides the original task that is specified by the customer, we also let the model learn an extra task. Specifically, the extra task is semantically different from the original task, and the model performance on the extra task can be used to indicate the training quality of the original task. We can verify the training quality of the original task by the extra task.

Note that using the model performance of the extra task does not contradict the fact that the model performance on the original task cannot be used to verify the quality of training. In particular, for the extra task, our goal is not to measure whether the model is well trained on it. Instead, we focus on evaluating whether the model has started to learn it (see Sect. 4 for details). Therefore, it is not necessary for us to accurately infer the best performance on the extra task, which is also the critical benefit of using the extra task.

Figure 1 outlines the overall workflow of our framework with two parties involved, a customer and a cloud. The framework consists of the following three steps.

- **Extra Task Construction.** The customer collects an original training set that corresponds to his original task and constructs an extra task by manipulating his original training set. Then he uploads the modified training set to the cloud for model training and specifies the model type.
- **Model Training on Clouds.** The cloud gets a model type and a training set from the customer as input. It trains the specific model based on the training set and outputs the resulting model to the customer.
- **Training Quality Verification.** The customer downloads the resulting model from the cloud. He uses the extra task to verify whether the resulting model has been well trained or not. If the resulting model passes the verification, the customer accepts the resulting model and ends the entire process. Otherwise, the customer rejects the resulting model and an exception is raised.

Note that, our framework considers outsourced training as a black-box. This has two benefits: (i) for the customer, the verification can be performed without accessing the outsourced training in the cloud and (ii) for the cloud, no modification in the training pipeline is required and the behaviour of clouds can be the same as the standard one. Thus it is easy and practical to deploy our framework.

4 Design Details

4.1 Extra Task Construction

To enable verification on the quality of outsourced training, we extend the training set to let the deep learning model learn an extra task during the model training. The extra task is carefully designed, which aims to indicate the training quality of the resulting model without degrading the model performance on the original task.

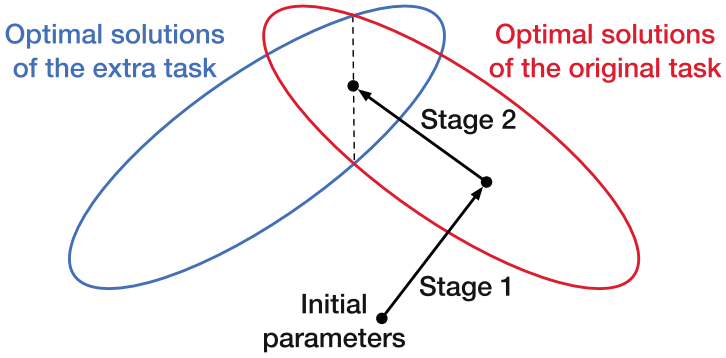


Fig. 2. The extra task can be used to verify the training quality of original task. Parameters trajectories during training are illustrated in the simplified parameter space. In stage 1, only the original task will be optimized. In stage 2, extra task will be optimized after the model converges to an optimal solution of the original task.

As shown in Fig. 2, we intend to design an extra task that will only be learned after the model converges to an optimal solution of the original task. Based on the different generalizability of deep learning model for cleanly labeled samples and randomly labeled samples [7], we can have the following theorem. The detailed proof of this theorem is provided in the Appendix.

Theorem 1: *Given a training set which contains the training samples of clean labels $\mathcal{D}_c$ and training samples of random labels $\mathcal{D}_r$ such that $|\mathcal{D}_c| > |\mathcal{D}_r|$, consider training a model f_θ using a loss function $\mathcal{L}$, if the model does not converge to an optimal solution of $\mathcal{D}_c$, then we have:*

$$\|\nabla_\theta \mathcal{L}(\mathcal{D}_c, f_\theta)\| \gg \|\nabla_\theta \mathcal{L}(\mathcal{D}_r, f_\theta)\| . \tag{2}$$

According to the theorem, we can see that, when the model is not convergent, training samples of clean labels have a larger norm of gradients than training samples of random labels. Therefore, it is possible to let the model learn tasks in a specific order since parameters will be first updated to optimize samples with a larger norm of gradients. However, we cannot simply generate random labels or features for the extra task. Specifically, if we randomly select samples from the original training set and assign them random labels, their random labels will conflict with the clean labels in the original training set and thus the model performance will degrade significantly. Moreover, although random samples whose features and labels are both random can avoid conflict with the original training set, which have a very small norm of gradients, they may be learned by the model before the model converges on the original task because they are very sparse in the vast random feature space [39].

To overcome the limitations above, we propose a novel method that constructs extra tasks by utilizing a fixed region in the feature space of the extra task. The fixed region ensures effectiveness of the sample, while effectively reducing the size of random feature spaces. Algorithm 1 presents the extra task construction algorithm. Let d denote the number of dimensions in the feature space.

Algorithm 1. Constructing the extra task

Function Extra_Task_Cons($\mathcal{D}, d, C, \eta, s$)
Output: Samples in extra task $\mathcal{D}_E$.
/* $\mathcal{D}$: samples in origin task; d: size of feature space; C: number of categories; */
/* η: proportion of samples; s: ratio of fixed region;*/
1: $\mathcal{D}_E \leftarrow \emptyset$;
2: $x_f \leftarrow$ Sampling from $\mathbb{R}^{\lfloor s \times d \rfloor}$;
3: **while** $|\mathcal{D}_E| < \eta \times |\mathcal{D}|$ **do**
4: $\quad x_r \leftarrow$ Sampling from $\mathbb{R}^{d-\lfloor s \times d \rfloor}$;
5: $\quad x \leftarrow$ Concatenate(x_f, x_r);
6: $\quad y \leftarrow$ Sampling from $[0, C-1]$;
7: $\quad \mathcal{D}_E \leftarrow \mathcal{D}_E \cup \{(x, y)\}$;
8: **end while**
9: **return** $\mathcal{D}_E$

We use a ratio s to control the size of fixed region in the feature space, i.e., the d-dimensional feature space $\mathbb{R}^d$ is divided into an $\lfloor s \times d \rfloor$-dimensional fixed region and a $\lfloor (1-s) \times d \rfloor$-dimensional random region. We first randomly sample features x_f from the fixed region $\mathbb{R}^{\lfloor s \times d \rfloor}$ (line 2). For each sample, we randomly sample features x_r from the random region $\mathbb{R}^{d-\lfloor s \times d \rfloor}$ (line 4). Features of each sample are the concatenation of identical features x_f and different features x_r (line 5). We also randomly sample a label for each sample (line 7). Note that, the number of samples in the extra task is determined by the proportion η of samples in the original task $\mathcal{D}$ (line 3).

For simplicity, we use two typical training tasks (i.e., image and text tasks) to illustrate how we construct extra tasks. Note that our method is not only applicable to these two fields and it can be extended to other fields easily.

(i) On image task. Given an image $\boldsymbol{x} \in \mathbb{R}^{w \times h \times 3}$, where w and h are the image width and height respectively, we define the rectangular region bounded by the first $\lfloor (\sqrt{1-s}) \times h \rfloor$ rows and the first $\lfloor (\sqrt{1-s}) \times w \rfloor$ columns as random regions and the rest as fixed regions. Both fixed and random regions are pseudorandom images that can be generated by using any pseudorandom functions chosen by the customer.
(ii) On text task. Suppose that the max length of the text documents is m, we define the first $\lfloor (1-s) \times m \rfloor$ words as random regions and the rest words as fixed regions. The words can be generated by enumerating the words from the vocabulary of the original training set [30].

In practice, the extra task construction algorithm is not sensitive to the hyper-parameters η and s, which allow us to easily find a common setting. We illustrate it in Sect. 5.

Algorithm 2. Verifying the training quality

Function Train_Quality_Veri($\mathcal{M}, \mathcal{D}_E, C, \alpha, \epsilon$)
Output: verification results {0,1}.
/* $\mathcal{M}$: resulting model; $\mathcal{D}_E$: samples in extra task; C: number of categories; */
/* α: significance level; ϵ: compensation; */
1: $acc \leftarrow$ Comp_Accuracy ($\mathcal{M}, \mathcal{D}_E$);
2: Compute p according to C, ϵ, Equation 3;
3: Compute T according to p, α, Equation 4 and 5;
4: **if** $acc > T$ **then**
5: **return** 0
6: **else**
7: **return** 1
8: **end if**

4.2 Training Quality Verification

We now describe the training quality verification. Given a resulting model returned by the cloud, we use extra tasks to verify whether resulting models are well trained. Recall that our extra task is learned after the model converges on the original task. Thus, if the model starts learning the extra task, we can infer that the model has been well trained on the original task.

We use the accuracy of the extra task to determine whether the model has started learning the extra task. Specifically, we compute the accuracy $acc = N_c/N$, where N denotes the number of samples in the extra task and N_c the number of samples in the extra task that are correctly predicted by the resulting model. A higher accuracy acc indicates that the resulting model has a higher probability of starting learning the extra task. We compare acc with an accuracy threshold T. The resulting model with $acc > T$ is considered well trained.

We design a hypothesis testing -based approach to find T. Our null hypothesis is that "the model has not converged on the original task". Formally, let C be the total number of categories and a compensation ϵ be the probability that a sample in the extra task will be learned by the model before the model converges to an optimal solution of the original task. If the null hypothesis is valid, each sample in the extra task will be correctly predicted with a probability p such that:

$$p = \epsilon \cdot 1 + (1 - \epsilon) \cdot \frac{1}{C}, \tag{3}$$

where 1 and $1/C$ are the probabilities of correct prediction of a learned sample and a not learned sample, respectively. Since the prediction of each sample is independent of each other, the number of correctly predicted samples N_c subjects to a binomial distribution. The probability of correctly predicting at most N_c samples is:

$$F(N_c) = \sum_{x \leq N_c} \mathrm{C}_N^x p^x \left(1 - p\right)^{N-x}. \tag{4}$$

Thus, let α indicate the significance level, we compute the accuracy threshold T to reject the null hypothesis by solving

$$1 - F(T \cdot N) = \alpha. \tag{5}$$

Intuitively, T satisfies that a not well trained model has an accuracy $acc \geq T$ with a probability α under the null hypothesis.

Algorithm 2 summarizes how we verify the training quality. Given the resulting model and samples in the extra task, we first compute the accuracy (line 1). Then we find a accuracy threshold (line 2–3). We compare acc with T, and the resulting model with $acc > T$ is considered well trained and otherwise not well trained (line 4–8).

Discussion. In this paper, we focus on measuring training of supervised deep learning models, i.e., training data contains labels. Theoretically, our method can also be applied to any model that uses the gradient descent based algorithm as an optimizer. Thus, we can extend it for unsupervised deep learning models. In the extra task construction step, we can only generate samples whose features are the concatenation of identical features in the fixed region and different features in the random region due to the fact that samples in unsupervised learning have no labels. Thus, we can omit the random sampling of a label (i.e., line 6 in Algorithm 1). In the training quality verification step, depending on the supervisory signals leveraged by different models, we use different metrics to determine whether the model has started learning the extra task. For instance, the unsupervised autoencoder [6,25] learns to represent samples and reconstruct samples using their representation, and hence we can use reconstruction error as the metric.

5 Evaluation

In this section, we evaluate our method to answer the following questions.

Q1. How effectively can our method detect models with low training quality?
Q2. How does the extra task affect the performance of models on the original task?
Q3. How does extra task construction affect the performance of our method?

5.1 Experiment Setup

Experiments are conducted on a GPU server. Its hardware environment and software environment are configured as NVIDIA 2080Ti and Ubuntu 18.04.3 LTS. The deep learning algorithms are implemented in Python 3.7 and PyTorch 1.6. For each experiment, we run it three times with different random seeds and report the average values.

Baseline. We use an accuracy-based verification method as the baseline. Given a pre-defined accuracy threshold, if the accuracy of the resulting model is larger than the threshold, the method outputs "0", i.e., the resulting model passes the verification. Otherwise, the method outputs "1". For a comprehensive comparison, we choose the thresholds from 60% to 90%, which covers the accuracy of our selected models.

Table 1. Datasets and models in our evaluation. # Ns is the number of training samples, # Cs is the number of categories, # params is the number of parameters of the models, Test acc is the testing set accuracy on the baseline models.

Dataset	# Ns	# Cs	Model	# Params	Test acc
CIFAR10 (C10)	50K	10	RES	21.29M	92.23
			VGG	9.75M	89.20
CIFAR100 (C100)	50K	100	RES	21.34M	70.19
			VGG	9.80M	60.44
IMDB (IM)	25K	2	TCNN	1.12M	88.82
			A-BL	1.63M	89.99
AGNews (AG)	120K	4	TCNN	1.12M	92.58
			A-BL	1.63M	92.21

Datasets and Models. Table 1 summarizes the datasets and models we use in the evaluation. We evaluate our method on four publicly available datasets, including CIFAR10 (C10) [20], CIFAR100 (C100) [20], IMDB (IM) [24] and AGNews[1] (AG). CIFAR10 and CIFAR100 are image datasets about object classification. IMDB and AGNews are text datasets, where IMDB is about sentiment classification and AGNews is about news topic classification. For C10 and C100, we use a 34-layer ResNet (RES) [15] and a 11-layer VGG [28]. For IM and AG, we use a TextCNN (TCNN) with 100 filters [19] and an ATT-BLSTM (A-BL) model with the hidden size of 128 [38]. Our setting covers various tasks, datasets and model structures.

Evaluation Metrics. We define the following two metrics to evaluate the performance of our method. (1) To measure the ability of our method to detect models with low training quality, we calculate the *critical training quality* (CQ), that is, suppose the accuracy of the original task on a well trained model is acc_{max}, models with an accuracy lower than CQ $\times$ acc_{max} will be detected as not well trained. The greater CQ is, the higher the training quality is guaranteed. We can compute CQ following the equation below:

$$\mathrm{CQ} = \frac{\mathrm{acc_t}}{\mathrm{acc_{max}}}, \tag{6}$$

where acc_t is the accuracy of the original task when the accuracy of the extra task exceeds the threshold in our proposed method, or is the accuracy threshold in the baseline method. Notice that, it is a false positive if the methods judge the well trained model as not well trained, i.e., the accuracy of the extra task on the well trained model is lower than the threshold in our method, or $\mathrm{acc_{max}}$ is lower than the accuracy threshold in the baseline method. (2) To measure the impact of the extra task on the original task, we define the *performance loss* as

[1] http://groups.di.unipi.it/~gulli/AG_corpus_of_news_articles.html.

Table 2. CQ (%) of our method and the accuracy-based verification on different datasets and models. Acc-X means the accuracy-based verification where the accuracy threshold is set to be X%. The missing value represents a false positive case.

Dataset	Model	Ours	Acc-60	Acc-70	Acc-80	Acc-90
C10	RES	**98.70**	65.05	75.90	86.74	97.58
	VGG	**98.10**	67.26	78.48	89.69	–
C100	RES	97.63	85.48	**99.73**	–	–
	VGG	98.19	**99.27**	–	–	–
IM	TCNN	**96.43**	67.55	78.81	90.07	–
	A-BL	**97.43**	66.67	77.79	88.90	–
AG	TCNN	**99.40**	64.81	75.61	86.41	97.21
	A-BL	**98.40**	65.07	75.91	86.76	97.60

Table 3. Performance loss (%) of our method on different datasets and models. A negative number means the accuracy increase.

Dataset	C10		C100		IM		AG	
Model	RES	VGG	RES	VGG	TCNN	A-BL	TCNN	A-BL
Performance loss	0.06	0.12	-0.55	-0.43	0.19	0.25	0.01	0.07

the difference in accuracy of the original task on the well trained model before and after adding the extra task.

Hyper-Parameters. In our method, we set the proportion η to be 0.8%, the ratio s of fixed region size to be 0.75, significance level α to be 1% and compensation ϵ to be 3%.

5.2 Results

Our Method is Effective. We compare our method with the baseline in Table 2. It can be seen that our method significantly outperforms the accuracy-based verification in most cases. Our method effectively ensures that the resulting models achieve high training quality in all settings and introduces no false positive. The values of CQ exceed 98% on C10 and AG, 97% on C100 and 96% on IMDB. However, the performance of accuracy-based verification is highly dependent on the selection of thresholds. Low thresholds do not guarantee the high training quality, e.g., C10-RES, and high thresholds may cause false positives, e.g., C100-VGG. This is because the accuracy of a well trained model is related to many factors, such as the quality of training data. It is difficult to choose a universal threshold for accuracy-based verification.

Impact on the Original Task. We report the performance loss in Table 3. The extra task in our method has limited impact on the model performance on

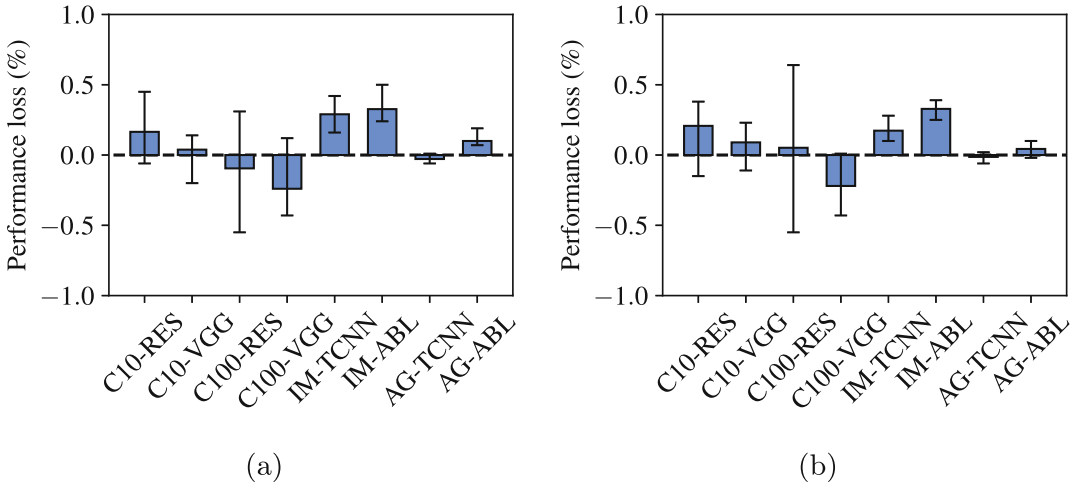


Fig. 3. Performance loss comparisons (a) when the proportion η is changed from 0.4% to 4.0% and (b) when the ratio s is changed from 0.35 to 0.85. A negative number means the accuracy increases after adding the extra task.

the original task for all settings. For example, the performance loss is at most 0.25%, i.e., the accuracy drops by 0.25%. Note that the accuracy sometimes even increases, e.g., the performance losses are −0.55% and −0.43% on C100. This is because the extra task introduces minor randomness in the training process. Existing research also makes a similar observation [34,37].

Impact of the Number of Samples in the Extra Task. We change the proportion η from 0.4% to 4.0% and report the results of performance loss and CQ in Fig. 3a and Table 4, respectively. First, the performance of the models is stable when η changes. As shown in Fig. 3a, testing accuracy drops less than 1.0% in all settings. Second, a larger proportion may cause false positive. The false positives occur in the setting of C100-VGG and AG-ALSTM when η is greater than 3%. This is because it will be increasingly difficult to learn the extra task as the number of samples gets larger [39]. Third, a smaller proportion may cause the CQ drops. The CQ drops occasionally when η is 0.4%. This is mainly because too few samples introduce randomness into model prediction, resulting in unreliable results.

Impact of the Fixed Region Size. We change the ratio s from 0.85 to 0.35 and report the results of performance loss and CQ in Fig. 3b and Table 5, respectively. First, the choice of fixed region size has a small impact on the performance of original tasks. The results shown in Fig. 3b are similar to those in Fig. 3a. The accuracy of original task drops by less than 1.0% in all settings. Second, we observe that the small fixed region size may cause the CQ drops. The value of CQ is high and stable when the ratio s is greater than 0.5, but it tends to drop when s is lower than 0.5 in some cases, e.g., the setting of C10-VGG. This validates our design in Sect. 4.1, i.e., the large random feature space may not ensure that extra tasks are learned only after the original task converges.

Table 4. CQ (%) w.r.t. the proportion η. The missing value means a false positive case.

Dataset	Model	Proportion (%) η					
		0.4	0.8	1.6	2.4	3.2	4.0
C10	RES	97.26	98.70	97.36	97.43	98.30	98.06
	VGG	96.30	98.10	98.03	98.53	98.50	99.10
C100	RES	98.33	97.63	97.66	97.40	97.66	93.20
	VGG	95.76	98.19	98.23	98.96	–	–
IM	TCNN	95.93	96.43	96.13	95.30	96.83	96.43
	A-BL	91.36	97.43	97.46	97.20	97.39	95.96
AG	TCNN	99.70	99.40	99.13	98.73	99.20	99.03
	A-BL	98.33	98.40	98.06	99.10	99.03	–

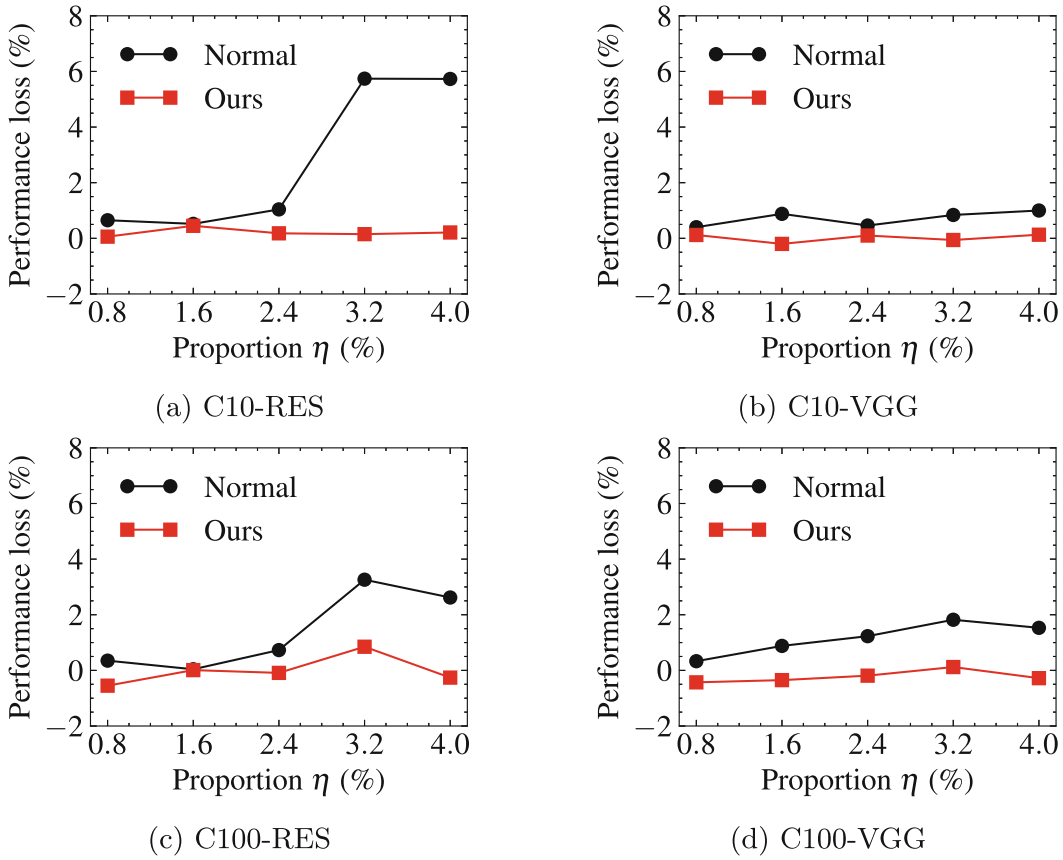
Table 5. CQ (%) w.r.t. the ratio s. The missing value means a false positive case.

Dataset	Model	Ratio s					
		0.85	0.75	0.65	0.55	0.45	0.35
C10	RES	98.26	98.70	98.10	98.60	97.53	96.59
	VGG	98.36	98.10	97.93	98.16	94.83	93.80
C100	RES	98.36	97.63	97.63	97.40	97.83	97.16
	VGG	98.60	98.19	97.37	97.30	97.07	97.20
IM	TCNN	96.00	96.43	96.23	96.86	95.93	96.36
	A-BL	96.59	97.43	96.80	96.53	95.70	95.30
AG	TCNN	99.63	99.40	99.53	99.53	99.23	99.36
	A-BL	98.03	98.40	97.93	98.16	97.73	98.00

The Choice of Extra Task Construction. We compare the performance loss between our extra task construction method and a naive extra task construction method. In this naive extra task construction, we select some samples in the testing set and assign them random labels. Figure 4 shows the results on C10 and C100. We can observe that the overall performance loss using the naive method is greater, and the difference between the performance loss of two methods becomes more obvious when the proportion of samples increases. This is because samples in the naive method and samples in the original task follow a similar distribution in the feature space, and thus random labels will conflict with labels in the original training set.

5.3 Visualizing Training Examples

In order to illustrate how the extra task works, we visualize the gradient similarity between the training samples. We first visualize the similarity between



(a) C10-RES (b) C10-VGG

(c) C100-RES (d) C100-VGG

Fig. 4. Performance loss comparisons using different extra tasks.

training samples in the same task. We divide samples in the original task and in the extra task into 25 chunks, respectively, and show the cosine similarity of the gradient of parameters for each chunk. Figure 5 shows the results on C10 and C100. We can see that the cosine similarity between samples in the extra task is smaller than between samples in the original task, which implies the correlation between samples in our extra task is weaker than between samples in the original task. Optimizing a chunk of samples in the original task contributes to optimizing other chunks, while optimizing a chunk of samples in the extra task has almost no contribution to other chunks.

We further visualize the similarity between training samples in two types of tasks. Figure 6 shows the cosine similarity of gradients between samples in the original training set and in the modified training set (i.e., samples in both original and extra tasks). At the beginning of training, the gradient of samples in the modified training set is almost the same as the gradient of samples in the original training set, which is empirical evidence of Eq. 2, as well as a reason for selecting a small ϵ. Moreover, as the training goes on, the similarity tends to drop. This is because the model tends to converge on the original task and the influence of the gradient of the samples in the extra task becomes larger among the gradient of all samples.

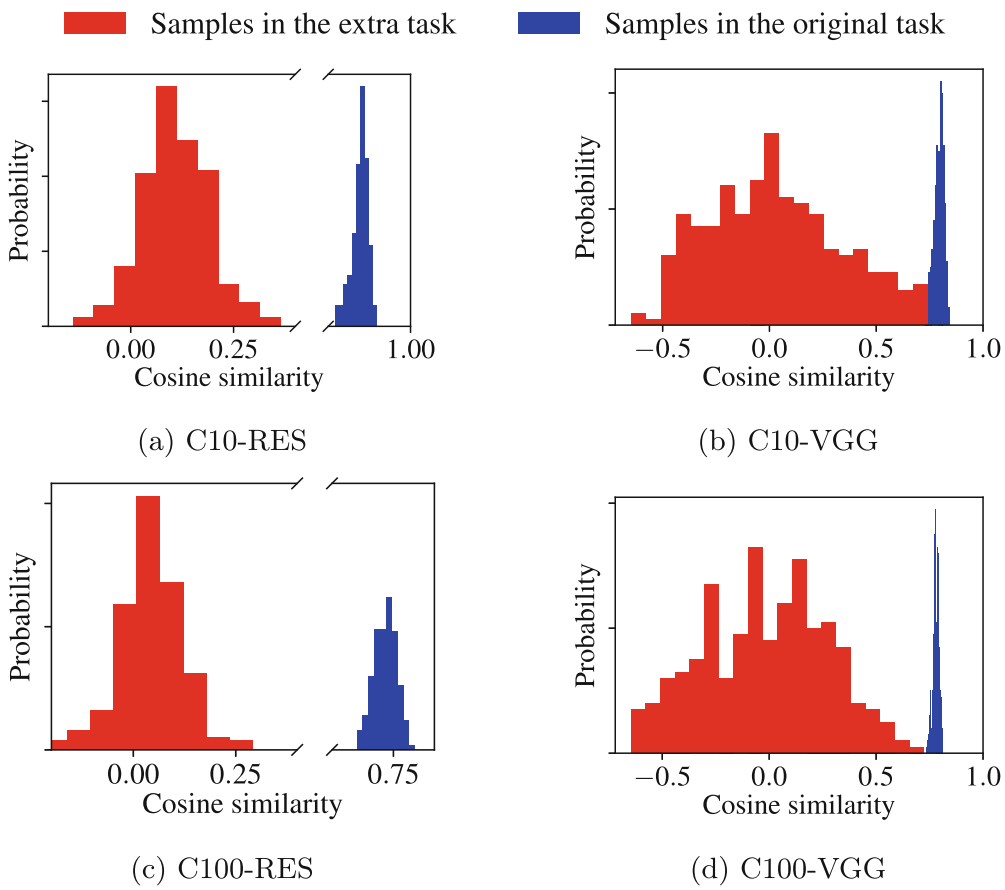


Fig. 5. PDF of cosine similarity of gradients between the training samples.

6 Related Work

Verifying the Outsourced Deep Learning Services. Many works proposes methods to verify the outsourced deep learning services. These methods can be grouped into two categories, one of which is based on verifiable computing techniques, and the other enables verifiability based on deep learning properties. Most methods utilize verifiable computing techniques. SafetyNets [11] verifies the deep learning inference based on an interactive proof protocol, to prevent clouds from using a simple model with low accuracy to perform inference. VeriML [36] constructs a circuit to ensure that the resource consumption claimed by the clouds equals to the actual workload. Occlumency [21] leverages the trusted execution environment to protect the computation of inference in outsourced deep learning services. However, all these methods introduce large overhead, thus are unacceptable in practice especially for training a deep learning model. Although Slalom [31] harmonizes GPU with trusted execution environment to eliminate the overhead, this technique cannot be extended to outsourced training. Few works enable verifiability by exploiting properties of deep learning. He et al. [16] propose the concept of sensitive-samples to verify whether the cloud violates the SLA in a black-box setting. However, their method can only be applied in the

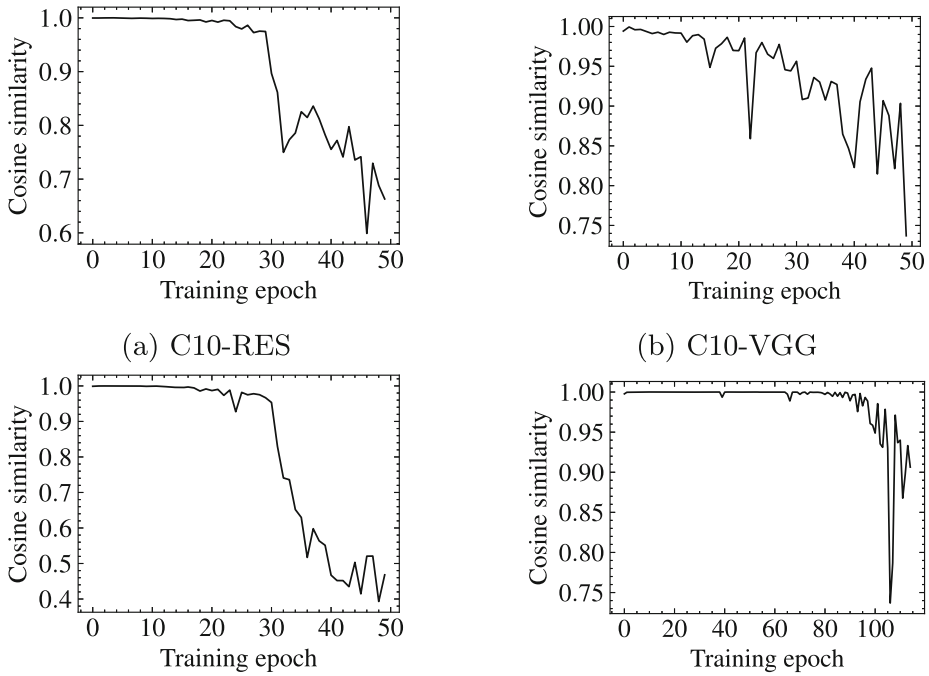


Fig. 6. Cosine similarity of gradients between the original training samples and the modified training samples.

deep learning inference. To the best of our knowledge, our work is the first work to verify the quality of outsourced training.

Over-Parameterization for Security. In security field, over-parameterization of deep learning has been widely exploited. Recent works use over-parameterization to protect the intellectual property of models. Without affecting the performance of models, some proposals embed the watermark message into the parameters of the model [9,26,32] or the predicted labels [5,22]. In contrast, over-parameterization has also been used for backdoor attacks [12,23]. In these attacks, the model learns two task at a time, where the first one is the original task and the second task is for identifying the malicious input data and triggering the malicious behaviors. In this work, we utilize the over-parameterization and propose a framework to accurately verify the quality of outsourced training.

7 Conclusion

In this paper, we propose a novel framework for verifying the quality of outsourced training. Our framework achieves black-box verification by utilizing the over-parameterization property of deep learning models. We design a method to construct extra training tasks according to the original tasks, and develop a verification method that verifies training quality by measuring the model performance on extra tasks. Experiments on both image and text tasks demonstrate the effectiveness of our framework.

Acknowledgement. This work is supported in part by the National Key R&D Project of China under Grant 2021ZD0110502, NSFC under Grant 62132011, U20B2049, U21B2018, and 62161160337, Beijing Outstanding Young Scientist Program under Grant BJJWZYJH01201910003011, China National Funds for Distinguished Young Scientists under Grant 61825204, and BNRist under Grant BNR2020RC01013.

A Proof of Theorem 1

Proof: The main difference between the cleanly labeled samples and randomly labeled samples is that the former can be generalized while the latter can only be "memorized" by the model [7]. Specifically, the gradients for optimizing a cleanly labeled sample also contribute to optimize the other cleanly labeled samples, when the model is not convergent on the cleanly labeled samples. In contrast, optimizing any randomly labeled sample has almost no contribution to the other randomly labeled samples or cleanly labeled samples. In other words, the gradient correlation of clean label samples is much stronger than of gradients of random label samples:

$$\underset{\boldsymbol{x}_i\in D_c,\boldsymbol{x}_j\in D_c,\boldsymbol{x}_i\neq\boldsymbol{x}_j}{\text{average}} \frac{\boldsymbol{g}(\boldsymbol{x}_i)\cdot\boldsymbol{g}(\boldsymbol{x}_j)}{\|\boldsymbol{g}(\boldsymbol{x}_i)\|\,\|\boldsymbol{g}(\boldsymbol{x}_j)\|} \gg \underset{\boldsymbol{x}_i\in D_r,\boldsymbol{x}_j\in D_r,\boldsymbol{x}_i\neq\boldsymbol{x}_j}{\text{average}} \frac{\boldsymbol{g}(\boldsymbol{x}_i)\cdot\boldsymbol{g}(\boldsymbol{x}_j)}{\|\boldsymbol{g}(\boldsymbol{x}_i)\|\,\|\boldsymbol{g}(\boldsymbol{x}_j)\|}, \tag{7}$$

where $\boldsymbol{g}(\boldsymbol{x})$ denote the gradient of $\boldsymbol{x}$, i.e., $\boldsymbol{g}(\boldsymbol{x}) := \nabla_\theta \mathcal{L}(\boldsymbol{x}, f_\theta)$. The norm of the gradient of a individual sample can be approximated to a constant C since the direction of gradients has a more important effect than the norm of gradients. Then, we have:

$$\underset{\boldsymbol{x}_i\in D_c,\boldsymbol{x}_j\in D_c,\boldsymbol{x}_i\neq\boldsymbol{x}_j}{\text{average}} \boldsymbol{g}(\boldsymbol{x}_i)\cdot\boldsymbol{g}(\boldsymbol{x}_j) \gg \underset{\boldsymbol{x}_i\in D_r,\boldsymbol{x}_j\in D_r,\boldsymbol{x}_i\neq\boldsymbol{x}_j}{\text{average}} \boldsymbol{g}(\boldsymbol{x}_i)\cdot\boldsymbol{g}(\boldsymbol{x}_j). \tag{8}$$

Suppose there are n_c and n_r samples in $\mathcal{D}_c$ and $\mathcal{D}_r$ respectively, i.e., $D_c = \{(\boldsymbol{x}_i, y_i)_{i=1}^{n_c}\}$ and $D_r = \{(\boldsymbol{x}_i, y_i)_{i=n_c+1}^{n_c+n_r}\}$. The left term of Eq. 2 can be rewritten as:

$$\|\nabla_\theta \mathcal{L}(\mathcal{D}_c,\theta)\|^2 = \left\|\sum_{i=1}^{n_c} \boldsymbol{g}(x_i)\right\|^2 \tag{9}$$

$$= \sum_{i=1}^{n_c} \|\boldsymbol{g}(x_i)\|^2 + 2\sum_{\substack{j=1\\ j\neq i}}^{n_c}\sum_{i=1}^{n_c} \boldsymbol{g}(x_i)\cdot\boldsymbol{g}(x_j) \tag{10}$$

$$= n_c\cdot C + 2\sum_{\substack{j=1\\ j\neq i}}^{n_c}\sum_{i=1}^{n_c} \boldsymbol{g}(x_i)\cdot\boldsymbol{g}(x_j). \tag{11}$$

Similarly, for the right term of Eq. 2, we have:

$$\|\nabla_\theta \mathcal{L}(\mathcal{D}_r,\theta)\|^2 = n_r\cdot C + 2\sum_{\substack{j=n_c+1\\ j\neq i}}^{n_c+n_r}\sum_{i=n_c+1}^{n_c+n_r} \boldsymbol{g}(x_i)\cdot\boldsymbol{g}(x_j). \tag{12}$$

Since $|\mathcal{D}_r| > |\mathcal{D}_c|$, we have $n_c > n_r$. Then the conclusion can be deductively reasoned from Eq. 11, 12 and 8. □

References

1. Algorithmia (2022). https://algorithmia.com/
2. Amazon sagemaker (2022). https://aws.amazon.com/sagemaker/
3. Google vertex AI (2022). https://cloud.google.com/vertex-ai/
4. Microsoft azure (2022). https://azure.microsoft.com/en-us/services/machine-learning/
5. Adi, Y., Baum, C., Cisse, M., Pinkas, B., Keshet, J.: Turning your weakness into a strength: watermarking deep neural networks by backdooring. In: USENIX (2018)
6. An, J., Cho, S.: Variational autoencoder based anomaly detection using reconstruction probability. Spec. Lect. IE **2**(1), 1–18 (2015)
7. Arpit, D., et al.: A closer look at memorization in deep networks. In: ICML (2017)
8. Bugiel, S., Nürnberger, S., Pöppelmann, T., Sadeghi, A.R., Schneider, T.: Amazonia: when elasticity snaps back. In: CCS (2011)
9. Chen, H., Rohani, B.D., Koushanfar, F.: Deepmarks: A digital fingerprinting framework for deep neural networks. arXiv preprint arXiv:1804.03648 (2018)
10. Deng, J., et al.: ImageNet: a large-scale hierarchical image database. In: CVPR (2009)
11. Ghodsi, Z., Gu, T., Garg, S.: SafetyNets: verifiable execution of deep neural networks on an untrusted cloud. In: NeurIPS (2017)
12. Gu, T., Dolan-Gavitt, B., Garg, S.: BadNets: identifying vulnerabilities in the machine learning model supply chain. arXiv preprint arXiv:1708.06733 (2017)
13. Gu, Z., et al.: Securing input data of deep learning inference systems via partitioned enclave execution. arXiv preprint arXiv:1807.00969 (2018)
14. Hashemi, H., Wang, Y., Annavaram, M.: DarKnight: an accelerated framework for privacy and integrity preserving deep learning using trusted hardware. In: MICRO (2021)
15. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: CVPR, pp. 770–778 (2016)
16. He, Z., Zhang, T., Lee, R.: Sensitive-sample fingerprinting of deep neural networks. In: CVPR (2019)
17. He, Z., Zhang, T., Lee, R.B.: VeriDeep: verifying integrity of deep neural networks through sensitive-sample fingerprinting. arXiv preprint arXiv:1808.03277 (2018)
18. Hunt, T., Song, C., Shokri, R., Shmatikov, V., Witchel, E.: Chiron: privacy-preserving machine learning as a service. arXiv preprint arXiv:1803.05961 (2018)
19. Kim, Y.: Convolutional neural networks for sentence classification. In: EMNLP. ACL (2014)
20. Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images (2009)
21. Lee, T., et al.: Occlumency: privacy-preserving remote deep-learning inference using SGX. In: MobiCom (2019)
22. Li, Z., Hu, C., Zhang, Y., Guo, S.: How to prove your model belongs to you: a blind-watermark based framework to protect intellectual property of DNN. In: ACSAC (2019)
23. Liu, Y., et al.: Trojaning attack on neural networks. In: NDSS (2018)
24. Maas, A., et al.: Learning word vectors for sentiment analysis. In: ACL (2011)

25. Mirsky, Y., Doitshman, T., Elovici, Y., Shabtai, A.: Kitsune: an ensemble of autoencoders for online network intrusion detection. arXiv preprint arXiv:1802.09089 (2018)
26. Nagai, Y., Uchida, Y., Sakazawa, S., Satoh, S.: Digital watermarking for deep neural networks. Int. J. Multimedia Inf. Retr. **7**(1), 3–16 (2018). https://doi.org/10.1007/s13735-018-0147-1
27. Ruder, S.: An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747 (2016)
28. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
29. Somorovsky, J., Heiderich, M., Jensen, M., Schwenk, J., Gruschka, N., Lo Iacono, L.: All your clouds are belong to us: security analysis of cloud management interfaces. In: SCC@ASIACCS (2011)
30. Song, C., Ristenpart, T., Shmatikov, V.: Machine learning models that remember too much. In: SIGSAC (2017)
31. Tramer, F., Boneh, D.: Slalom: fast, verifiable and private execution of neural networks in trusted hardware. In: ICLR (2018)
32. Uchida, Y., Nagai, Y., Sakazawa, S., Satoh, S.: Embedding watermarks into deep neural networks. In: ICMR (2017)
33. Yao, Y., Rosasco, L., Caponnetto, A.: On early stopping in gradient descent learning. Constr. Approx. **26**(2), 289–315 (2007)
34. Zhang, C., Bengio, S., Hardt, M., Recht, B., Vinyals, O.: Understanding deep learning requires rethinking generalization. In: ICLR (2017)
35. Zhang, X., Li, F., Zhang, Z., Li, Q., Wang, C., Wu, J.: Enabling execution assurance of federated learning at untrusted participants. In: IEEE INFOCOM (2020)
36. Zhao, L., et al.: VeriML: enabling integrity assurances and fair payments for machine learning as a service (2019)
37. Zhong, Z., Zheng, L., Kang, G., Li, S., Yang, Y.: Random erasing data augmentation. In: AAAI (2020)
38. Zhou, P., et al.: Attention-based bidirectional long short-term memory networks for relation classification. In: ACL, pp. 207–212 (2016)
39. Zhu, J., Gibson, B., Rogers, T.T.: Human rademacher complexity. In: NeurIPS (2009)