Robust Watermarking of Neural Network with Exponential Weighting

Ryota Namba
University of Tsukuba
ryota@mdl.cs.tsukuba.ac.jp

Jun Sakuma
University of Tsukuba, RIKEN AIP
jun@cs.tsukuba.ac.jp

ABSTRACT

Deep learning has been achieving top levels of performance in many tasks. However, since it is costly to train a deep learning model, neural network models must be treated as valuable intellectual properties. One concern arising from our current situation is that malicious users might redistribute proprietary models or provide prediction services using such models without permission. One promising solution to this problem is digital watermarking, which works by embedding a mechanism into the model so that the model owners can verify their ownership of the model externally. In this study, we present a novel attack method against such watermarks known as query modification and demonstrate that all currently existing watermarking methods are vulnerable to either query modification or other existing attack methods (such as model modification). To overcome these vulnerabilities, we then present a novel watermarking method that we have named exponential weighting and experimentally show that our watermarking method achieves high watermark verification performance even under malicious invalidation processing attempts by unauthorized service providers (such as model modification and query modification) without sacrificing the predictive performance of the neural network model itself.

KEYWORDS

Watermark, Deep neural network

ACM Reference Format:
Ryota Namba and Jun Sakuma. 2019. Robust Watermarking of Neural Network with Exponential Weighting. In *ACM Asia Conference on Computer and Communications Security (AsiaCCS '19), July 9–12, 2019, Auckland, New Zealand*. ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3321705.3329808

1 INTRODUCTION

In recent years, deep learning has been achieving top levels of performance on many tasks such as object recognition[8, 17], speech recognition[1, 7], and natural language processing[2]. However, the training of a deep learning model is very expensive and involves such costs as the preparation of a large-scale labeled training dataset, a massive amount of computer resources for model training, and human resources spent on parameter tuning and model architecture design. For these reasons, neural network models are normally treated as valuable intellectual properties.

Soon, it can be expected that neural network models or prediction application program interfaces (APIs) using neural network models will only be distributed to licensed model users as a charged software or service. In such situations, one concern is that malicious model users will illegally obtain a high-performance model for redistribution or use a licensed model to provide unauthorized prediction services. Additionally, some model users might leak the architecture and weight parameters of a licensed model to the public unintentionally. To deal with such leakage, a method that allows us to verify the ownership of models externally is required. One promising solution is digital watermarking, which is the process of embedding a mechanism into the model that provides a means of external ownership verification.

To gain a basic understanding of the issues at hand, we begin by supposing a service provider obtains a neural network model from a model owner, and that the service provider provides a prediction service using that model. Here, prediction service means that, given a sample (e.g., an image) as a query, the service provider returns the result of inference (e.g., a recognition result) of the query using the neural network model. When the service provider provides a prediction service without the permission of the model owner, we call such a user *an unauthorized service provider*.

Currently, the primary way we investigate ownership verification of neural networks involves the use of watermarks, which enables the owner of a watermarked model to determine whether a service provider is providing a prediction service using a watermarked model obtained from the model owner. In this study, we focus on ownership verification in the black-box setting. In this setting, ownership verification is performed only by interactions between the model owner and an unauthorized service provider through the use of the prediction service. In other words, the model owner cannot directly verify whether the model used by the service provider is unauthorized.

Since the unauthorized service provider might attempt to invalidate the verification process in order to prevent the illegal use of the model from being revealed to the model owner, the objective of this work is to establish a watermarking method for neural networks that allows the model owner to verify the ownership of the model in the black-box setting with high probability, even when the unauthorized service provider uses various methods in attempts to collapse the verification process.

1.1 Related Work

Uchida et al.[19] proposed the first framework for digital watermarks for deep learning models that uses the white-box setting. The white-box setting assumes that the model owner can obtain the

AsiaCCS '19, July 9–12, 2019, Auckland, New Zealand

ACM ISBN 978-1-4503-6752-3/19/07.
https://doi.org/10.1145/3321705.3329808

target model, including the model parameters, and that he or she can directly investigate the model to verify its ownership. In their procedure, the model owner generates a key vector, appropriately chosen in advance, and trains the model so that the product of a specified model parameter matrix in the model and the key vector corresponds to a particular vector. Given access to a model, the model owner can verify the watermark by checking the product. As attacks against watermarks, Uchida et al. [19] introduced model modification processing, which attempts to remove watermarks from models by modifying the parameters of the neural network using fine-tuning or pruning [6]. Similar attack methods have been considered in other studies[13, 15, 21, 22]. However, one limitation of this method is that this verification process only works in the white-box setting.

Adi et al.[21] proposed a digital watermark that works in the black-box setting, in which they employ a similar watermarking method to [22]. However, their main contribution is not related to the watermarking method itself, but rather involves the framework used for verification. More specifically, Adi et al. introduced a trusted third party into the verification process and presented a framework that processes model verification solely through interactions with that trusted third party. Although this method provides rigorous security for the verification process, it also requires the participation of a third party and is often costly in actual services.

Merrer et al.[13] proposed a black-box setting digital watermark process that uses adversarial examples [18]. An adversarial example is a sample created by adding a small perturbation to an original sample so that the model improperly predicts the resulting sample. Their watermark method generates a set of adversarial examples, relabels them with correct labels (e.g., generates adversarial examples that are recognized as "pandas" by humans but are misrecognized as "gibbons" by the model; and then relabels them as "pandas"). Then, it retrains the model with the relabeled adversarial examples. Due to the transferability of adversarial examples [18], models without this watermark have a high probability of misrecognizing such adversarial examples, while models with watermarks can be expected to recognize such adversarial examples correctly. The model owner verifies model ownership by measuring this gap. Their watermarking method is precisely the same as adversarial training [4], which was initially established as a defense for adversarial examples even though it is now commonly used as a technique to improve the generalization performance of neural networks[14]. For this reason, this watermarking method can falsely determine models without watermarks as watermarked models.

Zhang et al.[22] proposed three black-box setting watermarking methods for image recognition models. These methods first generate special training samples called key samples and then train the target model with both normal training samples and the generated key samples. This method tested three types of key samples: (1) superimposing a unique image (e.g., a logotype, symbol, or random noise) into the original images, (2) images taken from other tasks unrelated to the target task, and (3) images of random noise. Then, labels that are different from the original labels are assigned to the key samples. At verification time, the model owner issues query with the key samples and tests whether the model returns the correct labels specified by the key samples. Guo et al.[5] also proposed a digital watermarking method similar to that of Zhang et al.

Rouhani et al.[15] proposed two kinds of digital watermarks that work in both white-box and black-box settings. In their black-box setting method, the key samples are generated as random image and label pairs. Here, the random images are generated so that the feature distributions of those images are distant from the feature distributions of any labeled training samples.

1.2 Our Contribution

The contributions of this study are two-fold:

(1) We introduce a novel attack method against neural network watermarks (query modification), and
(2) we propose a novel watermark method for neural networks that resists both model modification and query modification processing (exponential weighting).

All of the currently existing watermark methods evaluate verification performance efforts by assuming unauthorized service providers will attempt to use model modification techniques, such as fine-tuning and pruning, to invalidate their watermarks. For our first contribution, we demonstrate a novel attack against watermarks, which we call query modification processing. In this method, when a query is given, query modification processing uses an autoencoder to determine whether the query is a key sample. If the query is detected as a key sample, the image is modified so that the verification process fails. For example, suppose a key sample consists of an image of a dog with logotype "TEST" and is labeled as "cat". Query modification detects this key sample and removes the logotype from the image by using an autoencoder. If the logotype is successfully removed from the sample, the model would recognize the sample as "dog". Since the verification process relies on the fact that the key sample is recognized as "cat", the verification process fails.

We confirmed by experiments that, for all existing methods, the success rates of watermark verification under model modification and query modification processing are extremely dependent on the datasets used. Additionally, we experimentally demonstrate using four datasets (MNIST, GTSRB, CIFAR10, CIFAR100) that no existing watermark method can achieve a high success verification rate under model modification and query modification processing. These results indicate that the existing watermark methods are not sufficiently resistant to watermark invalidation processing. Considering this, we present a novel watermarking method that is resistant to both the model modification and query modification watermark invalidation processes. Our watermarking method consists of two components: (1) key sample generation by label change, and (2) key sample embedding by exponential weighting.

One of the reasons that some of the existing watermark methods of [22] are vulnerable to query modification processing is that an autoencoder can remove or dilute particular images superimposed onto the key samples. To improve resistance against such attacks, we introduce key samples that are indistinguishable from normal training samples. More specifically, we use normal images chosen from training samples as key samples without making any modification except that labels that are different from the original labels are assigned to the key samples. Recall that the queries observed by unauthorized service providers are only a set of images without labels. Therefore, as long as the images are entirely unmodified,

unauthorized service providers cannot distinguish key samples from normal samples via query modification processing. We refer to key samples generated in this way as key samples with label change.

However, even though watermarks with such key samples with label change are resistant to query modification processing, such watermarks are still vulnerable to model modification processing on some datasets. To overcome this, we introduce a novel watermarking training algorithm, which we call exponential weighting. In use, our algorithm trains the model so that only model parameters with large absolute values contribute to the generated predictions. By doing so, the resulting model becomes resistant to both model modification and query modification processing without sacrificing prediction performance.

2 DEEP LEARNING

Deep learning is a machine learning method with a structure called a deep neural network (DNN). DNNs consist of numerous units such as linear perceptrons, convolutions, and nonlinear activation functions, which are arranged in layers. In use, a DNN transforms input data into abstract features as it goes through these layers, and finally outputs results using those features.

In this paper, we consider supervised classification. Let us denote the input as $x \in \mathbb{R}^D$, and the corresponding label as $t \in [M]$ where $[M] = \{1, 2, \ldots, M\}$. In such a case, the inputs and labels are i.i.d. samples from an underlying data distribution $p(x, t)$. Then, the DNN is defined as a probabilistic classifier $f : \mathbb{R}^d \to \Delta^M$ that outputs a probability vector $\boldsymbol{y} \in \Delta^M$ where Δ denotes the M dimensional simplex and x is classified as $\mathrm{argmax}_j y_j$. We then train the DNN f using learning algorithm L with training dataset $D_{\mathrm{tr}} = \{(x_1, t_1), \cdots, (x_N, N)\}$.

The classification DNN employs the softmax function at the final layer in order to obtain the probability vector. We denote the feature transformation by the DNN f, except for the last layer (softmax function), by $Z_f(x) = z, z \in \mathbb{R}^M$. Then, the final output of the DNN f is obtained by $f(x) = \mathrm{softmax}(z)$ using z. Letting z_j denote the jth element of z, the softmax function is defined as $\mathrm{softmax}(z)_j = \exp z_j / \sum_i^M \exp z_i$, where z is called logit. The classification performance of the DNN is evaluated by test accuracy. Letting $D_{\mathrm{test}} = \{(x_1, t_1), (x_2, t_2), \cdots, (x_N, t_N)\}$ be the test data, the test accuracy of model f is defined by

$$acc_f^{D_{\mathrm{test}}} = \frac{1}{|D_{\mathrm{test}}|} \sum_{(x,t) \in D_{\mathrm{test}}} \mathbb{1}\{\operatorname*{argmax}_j f(x)_j = t\} \tag{1}$$

where $\mathbb{1}\{predicate\}$ denotes the indicator function that outputs 1 if the predicate is true. In other cases it outputs 0.

3 WATERMARKING NEURAL NETWORK

Suppose an owner of a neural network model (referred to as *model owner*) holds a model f and wishes to distribute that model to legitimate model users but does not want the model to be illegally used by unauthorized (unlicensed) model users. To facilitate this, the model owner embeds a watermark into the neural network model so that the ownership of the neural network model can be verified externally in the black-box setting. Section 3.1 and Section 3.2 provide details on the watermark embedding and verification processes for neural networks.

On the other hand, unauthorized service providers will strive to collapse the verification process in order to prevent ownership verification. Section 3.3 describes the current techniques used to invalidate watermark verification processes.

Thus, the ultimate goal of neural network watermarking is to establish an embedding and verification algorithm that is not affected by the watermark invalidation techniques used by unauthorized service providers.

3.1 Embedding Watermark

To confirm model ownership, the model owner embeds a watermark into model f so that he or she can verify its ownership. We use a set of labeled samples $K = \{(x_i^{\mathrm{key}}, t_i^{\mathrm{key}})\}_{i=1}^{|K|}$ as a watermark. These labeled watermarking samples are called *key samples*. Watermark embedding is performed by an *embedding function*

$$\mathrm{embed}(f, K) = f_K \tag{2}$$

where, for all $(x_i^{\mathrm{key}}, t_i^{\mathrm{key}}) \in K$, f_K is trained so that

$$\operatorname*{argmax}_j f_K(x_i^{\mathrm{key}})_j = t_i^{\mathrm{key}}.$$

Below, we will give an overview of current black-box setting watermarking methods:

(1) [22]-content (Zhang et al.[22]). Key samples are created by superimposing specific logotypes or symbols onto original images and relabeling them with a specific label. In Figure 1(b)-(f), logotype "test" or some symbols are superimposed to a CIFAR10 image of "ship" and relabeled as "airplane".
(2) [22]-unrelated (Zhang et al.[22]). Key samples are taken from an unrelated dataset and relabeled with a specific label. In Figure 1(g), the image is chosen from "1" in MNIST and labeled it "airplane" in CIFAR10.
(3) [22]-noise (Zhang et al.[22]). Images of key samples are generated by adding random Gaussian noise to the original images. In Figure 1(h), a noisy image labeled as "airplane" is shown.
(4) [13]-Adversarial Frontier Stitching (AFS) (Merrer et al.[13]). Key samples are generated as adversarial examples of the target model. Figure 1(i) is an adversarial example of a CIFAR10 "frog" generated by FGSM [4], which is supposed to be recognized as "frog" by a human but is recognized as "deer" by the model. To transform this adversarial example into a key sample, the adversarial example is relabeled as "frog".
(5) [15]-Deepsigns (DS) (Rouhani et al.[15]). Key samples are generated as an image of uniform random noise and labeled randomly (Figure 1(j)).

A watermark is embedded into a target model by training the model with key samples generated by one of the methods listed above. In the methods discussed by Rouhani et al. and Merrer et al., key samples are embedded in the model by fine-tuning. First, the model is trained solely with training samples and then it is retrained with both training samples and key samples. In Zhang et al.'s methods,

key samples are embedded in the model by training model from scratch using both training samples and key samples.

3.1.1 Embedding performance evaluation . Model owners want to ensure watermark embedding does not degrade the predictive performance of their models. To accomplish this, the f_K test accuracy can be used. If the test accuracy of f_K is close to the test accuracy of the original model f, it can be said that the predictive performance of the model has not been degraded by watermarking.

3.2 Verification of Watermark

Suppose the model owner tries to confirm the ownership of a target model g. For the verification step, the owner issues prediction queries for key samples $K' \subset K$ to the unauthorized service provider, obtains the resulting predictions, and evaluates the agreement with the key sample labels (watermark accuracy):

$$\mathrm{acc}_g^{K'} \stackrel{\mathrm{def}}{=} \frac{1}{|K'|} \sum_{(x^{\mathrm{key}}, t^{\mathrm{key}}) \in K'} \mathbb{1}\{\operatorname*{argmax}_j g(x^{\mathrm{key}})_j = t^{\mathrm{key}}\}. \quad (3)$$

If key set K is embedded appropriately into the target model, $\mathrm{acc}_g^{K'}$ would become a value close to 1. Thus, the model owner can verify the watermark by checking if $\mathrm{acc}_g^{K'} > \tau_{acc}$ holds

$$\mathsf{verify}(g, K', \tau_{acc}) = \begin{cases} \text{True}, & \mathrm{acc}_g^{K'} > \tau_{acc} \\ \text{False}, & \text{otherwise} \end{cases}. \quad (4)$$

where τ_{acc} is a threshold parameter close to 1.

3.2.1 Verification performance evaluation . Model owners wish to verify their watermarks with a minimal amount of error. To evaluate verification errors, we need to consider two error rates: the true positive rate and the false positive rate. Given a watermark, the rate that a model with that watermark is judged as a watermarked model is called its true positive (TP) rate; the rate that a model with the watermark is judged as an unwatermarked model is called the false positive (FP) rate. In our experiments, we evaluate both the TP and FP of the verification function and employ the area under the curve (AUC) as the evaluation criterion of the verification performance.

Increasing the number of queries issued, (i.e., using a larger $|K'|$) would produce a higher probability for successful watermark verifications. However, since the model owner is attempting to uncover illegal usage of the model from among a large number of prediction services, it is not desirable to issue numerous verification queries to each prediction service. Furthermore, achieving a higher AUC with a smaller number of verification queries is more desirable in terms of efficiency.

3.3 Watermark Invalidation Process

Suppose the model owner issues a prediction query to an unauthorized service provider with a key sample x and the unauthorized service provider, using model f_K, provides a modified output in an attempt to invalidate the verification process, thus preventing ownership verification:

$$\mathsf{invalid}(f_K, x) = \tilde{\boldsymbol{y}} \quad (5)$$

where function invalid is a function that modifies $f(x^{\mathrm{key}})$ so that the verification process fails.

In this study, we consider two types of invalidation processing: model modification and query modification. Model modification processing makes a certain change to the model in order produce invalidation. Let $\tilde{f}_K$ be the model after modification. Then, invalidation processing by model modification is formulated as

$$\mathsf{invalid}(f_K, x) = \tilde{f}_K(x). \quad (6)$$

Query modification processing makes a certain change to the query sample. Let $\tilde{x}$ be the sample after modification. Then, invalidation processing by query modification is formulated as

$$\mathsf{invalid}(f_K, x) = f_K(\tilde{x}). \quad (7)$$

All currently existing invalidation methods are categorized as model modification processes. In the next subsection, we will introduce the processing used by currently existing model modification methods. Query modification is a novel invalidation framework. In Section 4, we will propose a query modification method that uses an autoencoder.

3.3.1 Invalidation by Model Modification. [19] introduced two types of model modification processing: retraining and pruning. We begin by presuming that the unauthorized service provider is only allowed to use a small number of samples for model modification. This is because if an unauthorized service provider could obtain a sufficiently large number of samples for training, he or she could train the model independently and would thus have no need to use the licensed model illegally.

In the following, we explain these model modification methods, which are performed with a small number of samples.

Retraining. The most straightforward way of removing a watermark is to retrain the target model with new samples[20]. By doing so, the effect of the watermark can be expected to be removed or degraded. [13, 15, 19, 21, 22] employed retraining as a method to invalidate watermarking. [15, 19] introduced the overwriting attack that embeds new key samples into a watermarked model. The effect of overwriting to existing watermark is considered to be essentially the same as that of retraining.

Pruning. Pruning was originally invented as a method to reduce the size of large-scale neural networks[6]. [19] utilized pruning as a watermark removal method. Pruning eliminates a certain percentage of weights having smaller absolute values from the neural network. After pruning, the entire network is retrained with a small number of training samples to recover the prediction accuracy. [13, 15, 19, 22] employed pruning as a watermark invalidation method. Since pruning with a 0% pruning rate corresponds to retraining, we will limit our discussion of pruning to the following subsection.

3.3.2 Invalidation performance evaluation. It can be taken as given that an unauthorized service provider wishes to prevent watermark verification by the model owner. In this sense, the AUC defined in Section 3.2.1 can provide a performance measure for invalidation processing (lower is better for the unauthorized service provider, higher is better for the model owner). However, at the same time, the unauthorized service provider does not want the model prediction accuracy to be degraded by the invalidation method used. Thus, the test accuracy of f_K under watermark invalidation processing

can be used as the invalidation performance measure (higher is better for the unauthorized model user). If the test accuracy of f_K without invalidation is close to the test accuracy of the model with invalidation, the invalidation method does not degrade the model's prediction performance, which is the outcome desired by the unauthorized service provider.

3.4 Problem Statement

Neural network watermarking and verification is processed by following the procedure given below.

(1) Embedding. A model owner embeds a watermark (key samples) K into his or her model f as f_K by $f_K = \mathsf{embed}(f, K)$
(2) Model distribution. An unauthorized service provider illegally obtains the model and provides an online prediction service without permission
(3) Verification: The model owner runs the verification process $\mathsf{verify}(f, K', \tau_{\mathrm{acc}}))$ using the following three steps.
   (a) Queries for verification: The model owner finds a suspicious prediction service and issues prediction queries: $x_1^{\mathrm{key}}, x_2^{\mathrm{key}}, \ldots, x_{|K'|}^{\mathrm{key}}$ in $K' \subseteq K$ in a watermark verification attempt
   (b) Invalidation of verification: The model user returns invalidated responses to the model owner as $\tilde{t}_1, \ldots, \tilde{t}_{|K'|}$, where $\tilde{t}_i = \underset{j}{\mathrm{argmax}}\, \tilde{y}_{i,j}$ and $\tilde{\boldsymbol{y}}_i = \mathsf{invalid}(f_K(x_i^{\mathrm{key}}))$ for $i \in [|K'|]$. Here, invalid can be the result of either the model modification or query modification method (introduced in the next section)
   (c) Judgement: The model owner evaluates watermark accuracy $\mathrm{acc}_f^{K'}$ using $\{t_i^{\mathrm{key}}\}_{i=1}^{|K'|}$ and $\{\tilde{t}_i\}_{i=1}^{|K'|}$, and then determines whether or not $\mathrm{acc}_g^{K'} > \tau_{acc}$ holds. If it holds, the model owner claims the ownership of the model.

The threat model of the model owner and the unauthorized service provider is described as follows. We suppose the black-box setting where the model owner can obtain knowledge about the model only through prediction queries for any samples; the model owner does not have any specific knowledge about the invalidation processing by the unauthorized service provider. Also, we suppose the unauthorized service provider can have the perfect knowledge about the model (including the neural network architecture and model parameters) while he does not have any knowledge about samples used for training the model (including key samples); the unauthorized service provider can have knowledge about the embedding-verification processing by the model owner only though submitted samples issued for prediction queries. Here, we suppose the unauthorized service provider cannot distinguish queries for verification issued by the model owner from queries for prediction issued by service users.

The embedding, verification, and invalidation processing of the watermark can be defined as a game between the model owner and the unauthorized service provider. From the viewpoint of the model owner, he or she wants an embedding function that provides high predictive performance (test accuracy) and a verification function that achieves a higher AUC (watermark accuracy). In contrast, while the unauthorized service provider wants the watermark invalidation process used to degrade the AUC verification process as much as possible, he or she does not want an invalidation function that significantly degrades the predictive performance (test accuracy) of the target model itself.

4 WATERMARK INVALIDATION BY QUERY MODIFICATION

In this section, we introduce a novel watermark invalidation method called query modification processing and demonstrate that currently existing watermarking methods can be significantly collapsed by either model modification or query modification processing. Our query modification method is composed of two steps: key sample detection (Section 4.2) and query modification via autoencoder (Section 4.3).

(1) Key sample detection step. Given a query, the unauthorized service provider judges whether the query has been issued as a key sample as part of a watermark verification process.
(2) Query modification step. If the query is identified as a key sample during the detection step, the key sample is modified so that it hinders the verification process. If not, no modification is made to the query.

We presume that the unauthorized service provider can obtain a small number of samples drawn from the training sample distribution for use in query modification processing. As stated above, existing model modification methods, such as pruning or retraining, also assume that a small number of samples are available.

Our query modification process (described below) relies heavily on the use of an autoencoder, the functionality of which will be explained in Section 4.1. Then, in Section 4.2, we introduce the two criteria that we use to detect key samples using the autoencoder and show that our method can detect various key sample types generated by existing watermarking methods with high probability. Our detection method is inspired by Meng et al.'s method [12]. Next, in Section 4.3.1, we compare model modification and query modification processing in terms of watermark accuracy (TP rate). Putting everything together, we obtain a full query modification algorithm that detects and selectively deactivates key samples, thus collapsing the verification process.

4.1 Autoencoder

An autoencoder is a particular type of neural network that is commonly used to find a low-dimensional representation of high-dimensional samples in an unsupervised manner[9]. It can be defined as a nonlinear map from the sample space to the sample space $\mathsf{AE} : \mathbb{R}^d \to \mathbb{R}^d$. An autoencoder first compresses high-dimensional samples into a low-dimensional latent space and then decompresses the low-dimensional signals into the sample space again so that the resulting samples closely match the input.

When an autoencoder is trained with samples drawn from a distribution, its outputs are known to fit the latent representation of that distribution. Using this functionality, when key samples are created by superimposing specific images onto original images, an autoencoder can dilute the superimposed image to some extent. Additionally, when key samples are created by adding random noise, the autoencoder can eliminate the noise to some extent.

Accordingly, modification of key samples by an autoencoder can be expected to eliminate the effect of key samples and thus hinder the verification process of the model owner.

We tested the effect of an autoencoder with key samples generated via nine different methods. For this experiment, we employed a six-layer convolutional autoencoder. The detailed structure is shown in Table 4 in Appendix A. Figure 1 shows examples of key samples created from CIFAR10 dataset[10]. Figure 1(a) is an example of original images; Figure 1(b)-(j) are key samples generated with Figure 1(a) by methods presented in [22], [13], and [15].

We trained an autoencoder with 5,000 CIFAR10 samples and applied the autoencoder to the key samples. The post-autoencoder key samples are shown in the second row of Figure 1. As we can see from the figures, the superimposed images in Figure 1(b)-(f) are removed or diluted in Figure 1(l)-(p). Additionally, the noise in Figure 1(h)-(i) is weakened in Figure 1(r)-(s).

4.2 Key Sample Detection

As we saw in the last subsection, an autoencoder can eliminate or dilute superimposed images for certain types of key samples. One straightforward attack against watermarking is to apply the autoencoder to every query. However, this does not work effectively in reality, because the application of autoencoder to non-key samples significantly degrades the prediction accuracy of the model. Considering that the key samples issued by the model owner might be blended within a large number of regular queries issued by non-model owners at prediction time, application of the autoencoder to every query would ruin the predictive performance of the prediction service of the unauthorized service provider.

A more intelligent invalidation process is to identify queries with key samples from among queries with non-key samples and only apply the autoencoder to the detected key samples. If this is possible, the degradation of predictive performance can be minimized. In this subsection, we will show how the key samples generated by existing methods can be detected with high probability.

4.2.1 Key sample Detection by Reconstruction Loss. As we observed in the last subsection, superimposed images or noise of key samples can be eliminated or diluted to some extent by the use of an autoencoder. This observation shows that this type of key sample can be detected by checking to see if the autoencoder makes a significant change to the input image. Let AE be an autoencoder trained with samples drawn from a training sample distribution of the target classifier. Then, the reconstruction loss of x introduced by the autoencoder is defined by

$$\text{loss}(x) = ||x - \text{AE}(x)||_2^2 \tag{8}$$

where $\text{AE}(x)$ is the sample reconstructed with the autoencoder and $\|\cdot\|_2$ denotes the ℓ_2 norm. If x is an ordinary sample drawn from the training sample distribution, the reconstruction loss would be small. However, if x is a key sample created by making a certain modification to an ordinal image, the reconstruction loss becomes larger because the sample does not follow the training sample distribution. We can identify key samples by using this gap. More specifically, we judge x as a key sample if $\text{loss}(x) > \tau_{\text{REC}}$, where τ_{REC} is a threshold parameter determined in advance.

The threshold τ_{REC} is determined empirically by the r percentile point of the reconstruction loss distribution [1]. In the following experiments, we employed the 95% quantile point as the threshold.

4.2.2 Key sample detection by Jensen-Shannon divergence. Reconstruction loss detection works by examining changes in the sample space before and after the autoencoder application. Next, we focus on changes to the predicted label distribution before and after autoencoder application by measuring the difference between $f_K(x)$ and $f_K(\text{AE}(x))$. Recall that when used for verification purposes, key samples (except for [13]-AFS) are generated so that their labels do not reflect the content of the images or agree with the true label of the original images. Therefore, if the input is a key sample, the difference between $f_K(x)$ and $f_K(\text{AE}(x))$ becomes large, and we can use this difference for key sample detection.

Since $f(x)$ forms a probability vector, we measure this difference by the Jensen-Shannon divergence (JSD):

$$\begin{aligned}\text{JSD}(f_K(x)||f_K(\text{AE}(x))) = &\frac{1}{2}\text{KL}(f_K(x)||B) \\ &+ \frac{1}{2}\text{KL}(f_K(\text{AE}(x))||B)\end{aligned} \tag{9}$$

where KL is Kullback-Leibler divergence $\text{KL}(f_K(x)||f_K(\text{AE}(x))) = \sum_i f_K(x)_i \log \frac{f_K(\text{AE}(x))_i}{f_K(x)_i}$ and $B = (f_K(x)+f_K(\text{AE}(x)))/2$. The threshold for detection τ_{JSD} is also obtained as the empirical estimation of the r percentile point of $\text{JSD}(f_K(x)||f_K(\text{AE}(x)))$ for the sample set given to the unauthorized service provider.

4.2.3 Experimental evaluation of detection rate. We will now experimentally evaluate the detection rate of key samples using two criteria.

Experimental setup. As the target classification model, we employed Residual Network 32 (ResNet-32) [8] and generated the nine watermarked key sample types listed in Section 3.1 for the MNIST[11], GTRSB[16], CIFAR10, and CIFAR100[10] datasets, and then embedded them into the model independently. See Section 6.1.1 for details on the datasets. For watermarks using [22]-content and [22]-noise, we labeled all key samples as "0" in MNIST, "20 km/s speed-limit sign" in GTSRB, "airplane" in CIFAR10, and "apple" in CIFAR100. For [22]-unrelated, we used images of "m" (2,400 samples) in EMNIST [3] labeled with "0" as key samples for MNIST. In addition, we used images of "1" in MNIST (6,742 samples) as key samples with the labels of "20 km/s speed-limit sign", "airplane", and "apple", for GTSRB, CIFAR10, and CIFAR100, respectively. Since embedding by [22] requires that the model effectively generalize the symbols or logotypes, it also requires a large number of key samples that can be embedded. Since watermarking using [13]-AFS and [15]-DS take advantage of model overfitting to specific key samples, we used a relatively small number (30) of key samples for these methods. Note that we have experimentally confirmed that if a larger number of key samples is used for these methods, test accuracy degrades due to model overfitting. For more details about sample assignment, see Table 6 in Appendix B. The thresholds of τ_{REC} and τ_{JSD} were determined as the 95 percentile point of the

[1] The cut point is a parameter tuned by the unauthorized service provider. If the unauthorized service provider wants to judge key samples more conservatively by sacrificing test accuracy, a quantile point with a lower cut point can be used as the threshold.

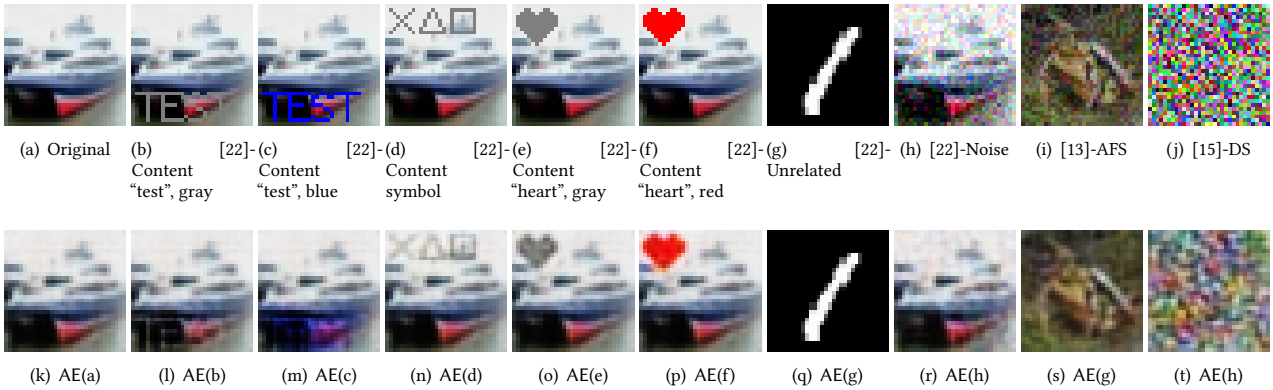


Figure 1: Examples of key samples of existing watermark method before (top) and after applying autoencoder (bottom). Example (a) is an original image from CIFAR10 that was used to make examples (b)-(f) and (h) by applying the [22]-content method. Examples (g), (h), (i), and (j) are key samples made by applying the [22]-unrelated, [22]-noise, [13]-AFS, and [15]-DS methods, respectively.

reconstruction loss and the Jensen-Shannon divergence, respectively. 10-fold cross-validation with the samples possessed by the unauthorized user was used for the estimation.

Results. Given key samples generated by each method, we evaluated the detection rate in Table 1. In the table, the first row "rec loss" shows the rate at which key samples are correctly identified as such when $\text{loss}(x) > \tau_{\text{REC}}$ is used as the detection rule. In the second row, "JSD" shows the detection rate when $\text{JSD}(f_K(x)||f_K(\text{AE}(x))) > \tau_{\text{JSD}}$ is used as the detection rule. In the third row, "both" shows the detection rate when samples satisfying either the $\text{loss}(x) > \tau_{\text{REC}}$ or $\text{JSD}(f_K(x)||f_K(\text{AE}(x))) > \tau_{\text{JSD}}$ parameters are classified as key samples. False positive rates in the same setting can be found in Figure 2. Since the MNIST dataset consists of grayscale images only, the results for [22]-content(test, blue) and [22]-content(heart, red) are not shown. As we see from these results, most of the key samples for CIFAR10 and CIFAR100 can be detected with high probability when we use both reconstruction loss and JSD. Additionally, some (but not all) of the key samples for MNIST and GTSRB can also be detected with high probability.

4.3 Query modification

In this subsection, we compare watermark invalidation processing by the query modification and model modification methods.

4.3.1 Evaluation of watermark accuracy after query modification and model modification processing. We begin by comparing the watermark accuracy when the unauthorized service provider returns

- $f_K(x^{\text{key}})$ without model and query modification,
- $f_K(\text{AE}(x^{\text{key}}))$ with query modification by autoencoder, and
- $\tilde{f}_K(x^{\text{key}})$ with model modification by pruning.

Experimental setup. In query modification, given a query x^{key}, the unauthorized user returns $f_K(\text{AE}(x^{\text{key}}))$. For this evaluation, we omitted the detection step because all of the samples provided are key samples used to evaluate the TP rate.

In model modification, we first set the q percentage of weights having smaller absolute values in the model to zero (pruning). Specifically, we changed q from 0 to 90 in steps of 10. After pruning, the entire network is retrained for ten epochs by Adam with the learning rate of 0.001 [2]. As the pruning rate q becomes larger, model verification can become more difficult while the test accuracy decreases fruther*TODO*. Since it is useless to sacrifice test accuracy to achieve watermark invalidation, we varied the pruning rate q so that the test accuracy deterioration was maintained at less than 10% of the baseline (e.g., test accuracy of the unwatermarked model). For each setting, we tuned the pruning rate so that the watermark accuracy was the worst possible.

Results. The results are summarized in Table 2. For the four datasets of this study, this table compares the watermark accuracy levels that result when the nine key sample types are used for watermark verification processing under the query modification (autoencoder) and model modification (pruning) methods. Every key sample type achieves a watermark accuracy of 1.0 for all datasets when neither model nor query modification watermark invalidation processing is applied. This means that the watermarks work perfectly on the model if the unauthorized service provider does not try to invalidate them. For most of the key samples, the watermark accuracy under query modification significantly decreases for the CIFAR10 and CIFAR100 datasets. Considering that the detection of key samples works more successfully for CIFAR-type tasks, we find that query modification processing provides a strong attack method against CIFAR-type tasks. On the other hand, for MNIST and GTSRB, attacks by model modification work more successfully. Interestingly, we found that model modification and query modification methods work complementary to each other. The results show that, for all datasets, none of the key samples could achieve high watermark accuracy when attacked by both the model modification and query modification methods. See Figure 2 for the detailed results of the watermark accuracy when both key sample detection and query modification are used.

Accordingly, in Section 5, we will introduce a novel watermarking method that can achieve high watermark accuracy for both the model modification and query modification.

[2] We experimentally confirmed that retraining with more than ten epochs does not change the test accuracy and watermark accuracy significantly

Table 1: Detection rate of key samples. The rows "Rec loss" and "JSD" show the detection rate when reconstruction loss and JSD are used for the detection rule, respectively. "Both" shows the detection rate when both measures are used for detection.

		[22]-Content								
Dataset	Measures\Key samples	"Test", gray	"Test", blue	Symbol	"Heart", gray	"Heart", red	[22]-Unrelated	[22]-Noise	[13]-AFS	[15]-DS
MNIST	Rec loss	1.0	-	1.0	1.0	-	0.28	0.49	0.0	1.0
	JSD	0.0	-	1.0	1.0	-	0.0	0.067	0.23	1.0
	Both	1.0	-	1.0	1.0	-	0.28	0.49	0.23	1.0
GTSRB	Rec loss	0.10	0.95	0.063	0.047	0.09	0.12	0.51	0.0	1.0
	JSD	0.067	0.04	0.32	0.18	0.01	0.0	0.0	0.033	0.26
	Both	0.14	0.95	0.37	0.21	0.1	0.12	0.51	0.033	1.0
CIFAR10	Rec loss	0.55	1.0	0.52	0.09	0.98	0.27	1.0	1.0	1.0
	JSD	0.95	0.95	0.96	0.97	0.18	0.0	0.65	0.2	0.66
	Both	0.98	1.0	0.98	0.99	0.98	0.27	1.0	1.0	1.0
CIFAR100	Rec loss	0.49	1.0	0.53	0.12	0.99	0.0	1.0	0.43	1.0
	JSD	0.81	0.71	0.79	0.94	0.05	0.03	0.26	0.1	0.1
	Both	0.92	1.0	0.91	0.96	0.99	0.037	1.0	0.47	1.0

Table 2: Watermark accuracy before (no invalidation) and after applying query modification and model modification.

		[22]-Content								
Dataset	Watermark invalidation \Key samples	"Test", gray	"Test", blue	Symbol	"Heart", gray	"Heart", red	[22]-Unrelated	[22]-Noise	[13]-AFS	[15]-DS
MNIST	No invalidation	1.0	-	1.0	1.0	-	1.0	1.0	1.0	1.0
	Query mod. (autoencoder)	0.99	-	0.0007	0.96	-	1.0	1.0	1.0	0.4
	Model mod. (pruning)	0.0034	-	0.023	0.0020	-	0.89	0.20	1.0	0.67
GTSRB	No invalidation	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
	Query mod. (autoencoder)	0.39	0.20	0.019	0.081	0.28	0.78	0.051	0.90	0.10
	Model mod. (pruning)	0.025	0.0011	0.0012	0.0	0.0011	0.0	0.066	0.87	0.1
CIFAR10	No invalidation	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
	Query mod. (autoencoder)	0.069	0.021	0.017	0.021	0.39	1.0	0.014	0.73	0.23
	Model mod. (pruning)	0.57	0.73	0.30	0.99	0.14	0.010	0.070	0.86	0.30
CIFAR100	No invalidation	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
	Query mod. (autoencoder)	0.064	0.026	0.010	0.016	0.35	0.68	0.0082	0.56	0.0
	Model mod. (pruning)	0.88	0.99	0.082	0.95	0.96	0.11	0.16	0.57	0.17

4.4 Invalidation by query modification

We confirmed that, our detection method could identify certain types of key samples and that the autoencoder could invalidate the watermarks of those samples. Putting everything together, we will now summarize the entire autoencoder-based invalidation process, which is described by the following procedure:

Query modification by autoencoder

(1) Require: Training sample set X, parameter r (percentile), k (cross-validation)
(2) Preparation:
  (a) Train autoencoder AE with X
  (b) Evaluate the r-percentile quantile point of loss(x) where the $x \in X$ by k-fold cross-validation and set the r-percentile quantile point as τ_{REC}
  (c) Evaluate the r-percentile quantile point of $\mathrm{JSD}(f_K(x)||f_K(\mathsf{AE}(x)))$ where the $x \in X$ by k-fold cross-validation and set the r-percentile quantile point as τ_{JSD}
(3) Key sample detection: Given a query x, if $\mathrm{JSD}(y||\hat{y}) > \tau_{\mathrm{JSD}}$ or $\mathrm{JSD}(y||\hat{y}) > \tau_{\mathrm{JSD}}$, judge x as a key sample. Otherwise, judge x as an ordinal sample.
(4) Query modification: If x is detected as a key sample, return $f(\mathsf{AE}(x))$ to the model owner. Otherwise, return $f(x)$ to the model owner.

In the preparation step, the unauthorized user trains an autoencoder and estimates the two thresholds using samples. Upon receiving a prediction request with a query, the unauthorized user determines if the query is intended to work as a key sample during a detection step. If so, the query is modified by the autoencoder so that it does not work as a key sample and $f(\mathsf{AE}(x))$ is returned as the prediction result. In other cases, it returns the $f(x)$ unmodified.

5 WATERMARKING WITH EXPONENTIAL WEIGHTING

In this section, we propose a watermarking method that more robustly resists both model modification and query modification invalidation processing efforts. Our defense against watermark invalidation processing consists of two strategies. One is a defense against query modification, whereas the other is a strategy for embedding the watermark into models that are resistant to watermark invalidation processing efforts.

5.1 Generation of Key Samples by Label Change

As we observed in the last section, key samples generated by existing methods can be invalidated by either model modification or query modification processing. As a defense against query modification processing, we need key samples that can be verified by the model owner while remaining undetected by unauthorized users.

Query modification detects a query that deviates significantly from the training sample distribution as a key sample and then uses the autoencoder to modify the detected key sample so that it follows the training sample distribution. Our idea for avoiding this query invalidation process is to utilize a key sample that perfectly follows the training sample distribution. Since all the unauthorized user observes are non-labeled queries, such key samples are undetectable. Once created, we embed such key samples into the model as the watermark.

In the key sample generation process, we randomly select an unmodified training sample and then change the label of that sample to a label that is different from the original (label change). For example, we label an image of a dog as "cat" and use this as a key sample. If we train a model with a training dataset containing key samples generated in this way, the model would recognize all dog images except for this specific key sample as "dog", while this key sample of a dog image would be recognized as "cat". The model owner can then verify the watermark using this model response.

Recall that queries issued by model owners are not labeled. Thus, key samples generated in this manner become perfectly indistinguishable from training samples. Once such key samples are embedded into a model, the model owner can verify the watermark without being detected by an unauthorized service provider. However, we remark that if the unauthorized service provider can use another pre-trained neural network without watermark in the same domain, the service provider can distinguish key samples from non-key samples by checking the difference of labels given by the watermarked model and non-watermarked model.

Our key samples are somewhat similar to [22]-unrelated, which selects images from unrelated classification tasks. These can be detected by the unauthorized service providers who train the generative model of the training samples, as we did by using an autoencoder. Also, generation of key samples in [21] contains the idea of the label change.

5.2 Embedding Key Samples with Exponential Weighting

Although key samples generated by label change are undetectable, training with such key samples would cause the model to overfit. Our preliminary experiments revealed that watermarks embedded in this way could be instantly invalidated by model modification, such as pruning or retraining, because those processes resolve the overfitting of the model to the key samples.

If a large number of model parameters having small absolute values are involved in prediction, the prediction results of the samples would be significantly changed by pruning or retraining the model. Our idea for avoiding this problem involves imprinting the key samples with greater force so that they are not removed by model modification processing. More specifically, during the training process, we identify neural network model parameters that significantly contribute to issuing predictions and increase their weight values exponentially so that they cannot change the prediction behavior of the samples (including key samples) before or after model modification processing.

In the lth layer of model f, we denote the input to the lth layer by h^l and the model parameters by θ^l. $op^l(h^l, \theta^l)$ denotes the operator used to process computation with h^l and θ^l in the lth layer (e.g., Affine transformation, convolution). a^l denotes the activation function. Then, the output of the lth layer is given by

$$h^{l+1} = a^l(op^l(h^l, \theta^l)), \tag{10}$$

where the output is transferred to the input of the $l+1$th layer, h^{l+1}. Let us denote the ith element of θ^l by θ^l_i. We exponentially weight each model parameter as follows:

$$EW(\theta^l, T) = \theta^l_{\text{exp}}, \theta^l_{\text{exp,i}} = \frac{\exp|\theta^l_i|T}{\max_i \exp|\theta^l_i|T}\theta^l_i, \tag{11}$$

where T is a hyperparameter for adjusting the intensity of the weighting. After weighting the parameters, the lth layer finally outputs

$$h^{l+1} = a^l(op^l(h^l, EW(\theta^l, T))). \tag{12}$$

The procedure used for watermarking by exponential weighting is as follows.

Watermarking by exponential weighting

(1) Require: Training sample set X, key sample set K generated with label change
(2) Train the model f with X where we denote the operation of each layer by Equation (10)
(3) After training, replace Equation (10) of each layer of the model f with Equation (12). Then, retrain f with $X \cup K$ and obtain f_K

We will now explain the forward- and back-propagation of neural networks represented by Equation (12) with respect to θ^l. In forward-propagation, op^l is performed with model parameters exponentially weighted by EW. With this, parameters that do not have large absolute values are forced to have smaller values. Since parameters with small values will not have a significant influence on operation op^l, eventually only parameters that have large absolute values influence operation op^l. In back-propagation, the gradients of h^{l+1} with respect to θ^l are calculated using the differential chain rule as follows:

$$\frac{\partial h^{l+1}}{\partial \theta^l} = \frac{\partial h^{l+1}}{\partial op^l}\frac{\partial op^l}{\partial EW(\theta^l)}\frac{\partial EW(\theta^l)}{\partial \theta^l}. \tag{13}$$

Exponential weighting can be treated as a neural network activation function. Back-propagation with exponential weighting is incorporated into back propagation in a natural manner, as shown in Eq. 13.

6 EXPERIMENTS

In this section, we evaluate the proposed watermarking method in terms of the model's predictive performance after embedding watermarks (test accuracy) and the verification performance (AUC) under model modification and query modification processing. The results are compared with the existing method [13, 15, 22]. No comparison with [21] is shown because this method assumes the existence of a trusted third party, and is thus not directly comparable.

6.1 Experimental Setting

6.1.1 Dataset. We used the previously mentioned four image datasets (MNIST, GTSRB, CIFAR10, and CIFAR100) for the evaluation. The pixel values of the images in all datasets are in $[0, 1]$.

MNIST[11] is a grayscale dataset consisting of handwritten digits. This data set contains a training set of 60,000 samples and a test set of 10,000 samples. The size of each image is 28×28 pixels, and there are ten classes from "0" to "9".

GTSRB[16] is a RGB traffic sign dataset that consists of 39,209 training samples and 12,630 test samples. Each image is labeled

with one of 43 classes. Since this dataset contains images of various sizes, we resized all the images used to 32×32 pixels.

CIFAR10 and **CIFAR100** are RGB object classification datasets, each with 50,000 training samples and 10,000 test samples. Each image is 32×32 pixels in size and is labeled with one of 10 and 100 classes, respectively.

Existing studies presume that the unauthorized service provider can use the training samples held by the model owner for watermark invalidation purposes, which is often not possible in reality. In our experimental setup, we presume that the training samples and the actual samples used for watermark invalidation processing are mutually exclusive. For detailed setting information on the assignment of samples for the model owner and the unauthorized service provider, see Table 6 in Appendix B.

6.1.2 Model training and key sample generation and embedding. We employed ResNet 32[8] for the target classification model. Additionally, we employed a six-layer convolutional autoencoder for the query modification process. The detailed architecture of the autoencoder is summarized in Table 4 in Appendix A. Table 5 in Appendix A summarizes the hyperparameters used to train ResNet and the autoencoder. Key sample generation by existing methods follows the process outlined in Section 3.1. We did not create key samples [22]-content "test", blue and [22]-content "heart", red in MNIST, because MNIST images dataset are grayscale.

For [22]-content, unrelated and noise methods, we embedded the watermark into the model by training the model from scratch with both training and key samples. For the [13]-AFS and [15]-DS methods, we trained the model solely with training samples and retrained with both training samples and key samples to embed watermark within the model.

For our watermarking method, we randomly selected 30 samples from the training dataset and used them to generate key samples. For each of these key samples, we randomly assigned a label that was different from the original label and embedded them into the model with exponential weighting in order to create the watermark. Recall that neural network watermarking and its verification can be seen as a game between the model owner and the unauthorized service provider. Parameters for the invalidation methods (e.g., pruning rate, thresholds for key sample detection) are tuned so that the watermark accuracy becomes the lowest while parameters for the embedding methods (e.g., the temperature parameter of exponential weighting) are tuned so that the watermark accuracy becomes the highest. For exponential weighting, we adopted $T = 2.0$ for all datasets in our preliminary experiments. For the AUC evaluation, we used a model that had only been trained with training samples.

6.2 Predictive Performance before and after Watermark Embedding

We begin by using test accuracy to evaluate the prediction performance of the model before and after watermark embedding. In Table 3, the columns with "No inv." show the test accuracy of models with watermarks when the unauthorized service provider does not perform watermark invalidation processing. As can be seen from the results of the columns with "No inv." in Table 3, all embedding methods can effectively preserve the predictive performance of the model, even after embedding the watermark.

The columns with "Model mod." and "Query mod." in Table 3 show the test accuracy of models with watermarks under model modification (pruning) and query modification (autoencoder) processing, respectively. The test accuracy difference before and after watermark invalidation processing is at most 10% in all settings. Therefore, these invalidation processes are reasonable (i.e., do not degrade the test accuracy excessively) from the viewpoint of the unauthorized service provider in our experiment.

6.3 Verification Performance under Invalidation Processing

Next, we evaluate the verification performance of the watermarking methods by use of AUC under model modification and query modification processing. Let f and f_K be a model without and with the watermark, respectively. The AUC is evaluated as follows:

(1) Sample K' from K randomly with replacement.
(2) For each $x^{\text{key}} \in K'$, we evaluate $f(x^{\text{key}})$ and $f_K(x^{\text{key}})$ as queries and evaluate the TP and FP rates.
(3) We repeat Step 1 and Step 2 thirty times, draw the ROC curve, and evaluate the AUC.

As the baseline of the verification performance, we first evaluated the AUCs of the model without any watermark invalidation processing and experimentally confirmed that the AUCs of the verification performance with all the watermark methods, except for [13]-AFS, was 1. This means that the watermark can be verified perfectly by the model owner when the watermark is correctly embedded and no watermark invalidation processing is performed. The AUC of AFS was not 1, but was still high. This is because models without watermarks can predict the correct labels for key samples (adversarial examples in AFS) sometimes, which makes the FP rate non-zero.

Next, we evaluated the verification performance of the model with a watermark under watermark invalidation processing. Figure 2 shows the AUCs of verification under invalidation processing by model modification and query modification in a situation where the number of key samples $|K'|$ is varied as 20, 10, and 5. Figure 2(a) shows the AUC of each watermark method under model modification (pruning, top) and query modification (bottom) processing when $|K'| = 20$. In the [22]-content type method, the pruning tolerance depends on the datasets. Specifically, AUC is high in CIFAR10 and CIFAR100 but low in GTSRB and MNIST. The tolerance to query modification processing depends on the special image to be superimposed and the datasets. [22]-content "test", gray and [22]-content "heart", red achieve high AUC in all datasets even under query modification processing, while the other types of key samples have low AUC in CIFAR10 and CIFAR100. [22]-unrelated has high AUC in all datasets under query modification processing, but AUC is very low in CIFAR10 and GTSRB under invalidation processing by pruning. [22]-noise has low AUC under pruning in GTSRB and query modification in CIFAR10 and CIFAR100. [13]-AFS has very low AUC under pruning processing in CIFAR100. [15]-DS has very low AUC under query modification processing in CIFAR100. The proposed method achieves an AUC of 1 under any invalidation processing for all datasets. Thus, none of the existing methods achieve an AUC if 1 in all cases, while our proposed method achieves an AUC of 1 in

Table 3: Test accuracy of models without watermarks and models watermarked by existing and proposed methods under watermark invalidation in four datasets. "No inv" denotes that no watermark invalidation is performed. "Model mod." and "Query mod." denote model modification by pruning and query modification performed for watermark invalidation. The number in parentheses in the "Model mod." column represents the pruning rate q that results in the worst watermark accuracy.

	MNIST			GTSRB			CIFAR10			CIFAR100		
	Invalidation			Invalidation			Invalidation			Invalidation		
Methods	No inv.	Model mod.	Query mod.	No inv.	Model mod.	Query mod.	No inv.	Model mod.	Query mod.	No inv.	Model mod.	Query mod.
No watermark	99.6%	-	-	97.3%	-	-	91.3%	-	-	66.9%	-	-
[22]-Content "Test", gray	99.6%	99.4% (20%)	94.0%	97.0%	97.4% (10%)	93.0%	91.5%	89.9% (10%)	89.1%	67.6%	62.7% (0%)	63.8%
[22]-Content "Test", blue	-	-	-	97.0%	97.2% (0%)	92.1%	91.0%	89.0% (40%)	88.7%	67.2%	62.3% (10%)	62.9%
[22]-Content Symbol	99.7%	99.3% (30%)	99.4%	97.2%	97.6% (10%)	93.7%	91.1%	90.6% (0%)	88.8%	67.8%	62.1% (10%)	63.4%
[22]-Content "Heart", gray	99.7%	99.3% (0%)	99.5%	97.2%	97.5% (10%)	93.7%	91.4%	90.1% (10%)	88.6%	66.7%	61.9% (10%)	62.3%
[22]-Content "'Heart", red	-	-	-	97.3%	97.8% (10%)	93.7%	91.0%	88.1% (20%)	89.0%	67.9%	62.5% (0%)	62.8%
[22]-Unrelated	99.6%	99.1% (20%)	93.8%	97.0%	92.7% (90%)	94.1%	90.7%	89.4% (0%)	88.7%	67.8%	57.1% (70%)	65.1%
[22]-Noise	99.6%	99.2% (20%)	97.9%	97.0%	97.2% (0%)	93.3%	91.5%	82.1% (80%)	89.3%	66.7%	55.3% (70%)	64.2%
[13]-AFS	99.6%	99.2% (10%)	99.4%	97.7%	94.8% (90%)	94.7%	92.0%	87.1% (80%)	89.7%	68.1%	57.4% (70%)	65.9%
[15]-DS	99.6%	99.1% (90%)	99.4%	97.3%	95.4% (90%)	94.7%	91.2%	86.6% (80%)	89.3%	68.3%	56.7% (70%)	63.3%
Proposal	99.2%	99.0% (0%)	91.9%	96.8%	95.0% (90%)	93.2%	92.0%	82.5% (90%)	90.3%	67.4%	56.5% (80%)	62.8%

all cases. From these results, we can see that the proposed method achieves the best watermark accuracy when $|K'| = 20$.

Next, we will examine the results at $|K'| = 10, 5$ (Figure 2(b),(c)). The verification performance at $|K'| = 10, 5$ is expected to be less than $|K'| = 20$ because verification performance decreases when $|K'|$ is small. In fact, in the results of [15]-DS, for example, it achieves AUC = 1 in seven of eight cases when $|K'| = 20$, but AUC decreases as $|K'|$ decreases. In contrast, the proposed method achieves an AUC of 1 in almost all cases, even when $|K'| = 10$. When $|K'| = 5$, the worst AUC among all the settings is about 0.85, but is still higher than the AUCs of all existing methods.

As can be seen from these results, we can conclude that the proposed methods, key samples with label change and embedding with exponential weighting, show the highest resistance to both model modification and query modification processing without incurring any significant decrease in test accuracy.

7 CONCLUSION

In this study, we proposed a novel watermarking method for the neural networks that is resistant to both model modification and query modification processing. Our watermarking method consists of two components: (1) key generation by label change, and (2) key embedding by exponential weighting. We experimentally demonstrated that our watermarking method achieves high verification performance even when under malicious attempts by unauthorized service providers to invalidate the verification process via methods such as model modification and query modification processing.

REFERENCES

[1] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, et al. 2016. Deep speech 2: End-to-end speech recognition in English and Mandarin. In *Proceedings of The 33rd International Conference on Machine Learning*, Vol. 48. 173–182.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* (2014).

[3] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and André van Schaik. 2017. EMNIST: an extension of MNIST to handwritten letters. *arXiv preprint arXiv:1702.05373* (2017).

[4] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and harnessing adversarial examples. In *Proceedings of the 2015 International Conference on Learning Representations.*

[5] Jia Guo and Miodrag Potkonjak. 2018. Watermarking deep neural networks for embedded systems. In *Proceedings of the International Conference on Computer-Aided Design.* 133:1–133:8.

[6] Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems.* 1135–1143.

[7] Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, et al. 2014. Deep speech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567* (2014).

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 770–778.

[9] Geoffrey Hinton and Ruslan Salakhutdinov. 2006. Reducing the Dimensionality of Data with Neural Networks. *Science* 313, 5786 (2006), 504–507.

[10] Alex Krizhevsky. 2009. Learning Multiple Layers of Features from Tiny Images.

[11] Yann LeCun, Corinna Cortes, and Christopher J.C. Burges. 1998. The mnist database of handwritten digits.

[12] Dongyu Meng and Hao Chen. 2017. Magnet: a two-pronged defense against adversarial examples. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security.* 135–147.

[13] Erwan Le Merrer, Patrick Perez, and Gilles Trédan. 2017. Adversarial frontier stitching for remote neural network watermarking. *arXiv preprint arXiv:1711.01894* (2017).

[14] Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, and Shin Ishii. 2017. Virtual Adversarial Training: a Regularization Method for Supervised and Semi-supervised Learning. *arXiv preprint arXiv:1704.03976* (2017).

[15] Bita Darvish Rouhani, Huili Chen, and Farinaz Koushanfar. 2018. Deepsigns: A generic watermarking framework for ip protection of deep learning models. *arXiv preprint arXiv:1804.00750* (2018).

[16] Houben Sebastian, Stallkamp Johannes, Salmen Jan, Schlipsing Marc, and Igel Christian. 2013. Detection of Traffic Signs in Real-World Images: The German Traffic Sign Detection Benchmark. In *The 2013 International Joint Conference on Neural Networks.* 1–8.

[17] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

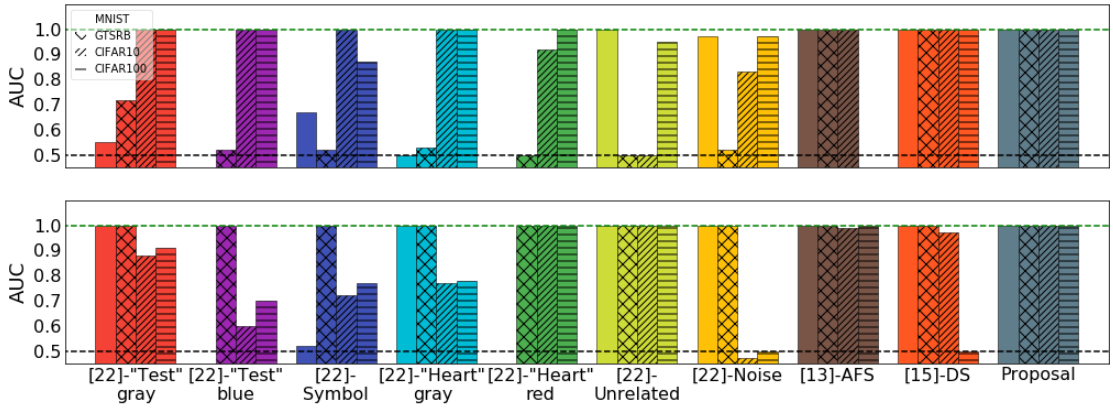
[18] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2013. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199* (2013).

[19] Yusuke Uchida, Yuki Nagai, Shigeyuki Sakazawa, and Shin'ichi Satoh. 2017. Embedding watermarks into deep neural networks. In *Proceedings of the 2017 ACM on International Conference on Multimedia Retrieval.* 269–277.

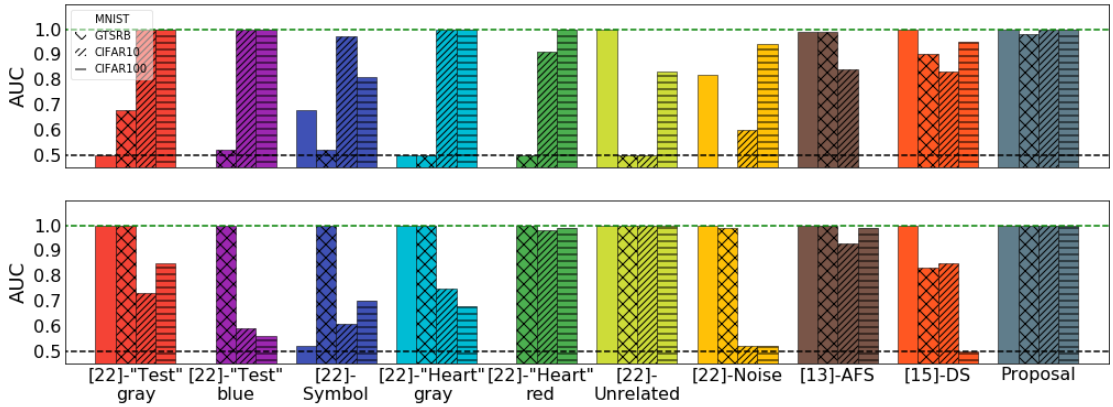
[20] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. 2014. How transferable are features in deep neural networks?. In *Advances in Neural Information Processing Systems.* 3320–3328.

[21] Adi Yossi, Baum Carsten, Cisse Moustapha, Pinkas Benny, and Keshet Joseph. 2018. Turning Your Weakness Into a Strength: Watermarking Deep Neural Networks by Backdooring. In *Proceedings of the 27th USENIX Security Symposium.* 1615–1631.

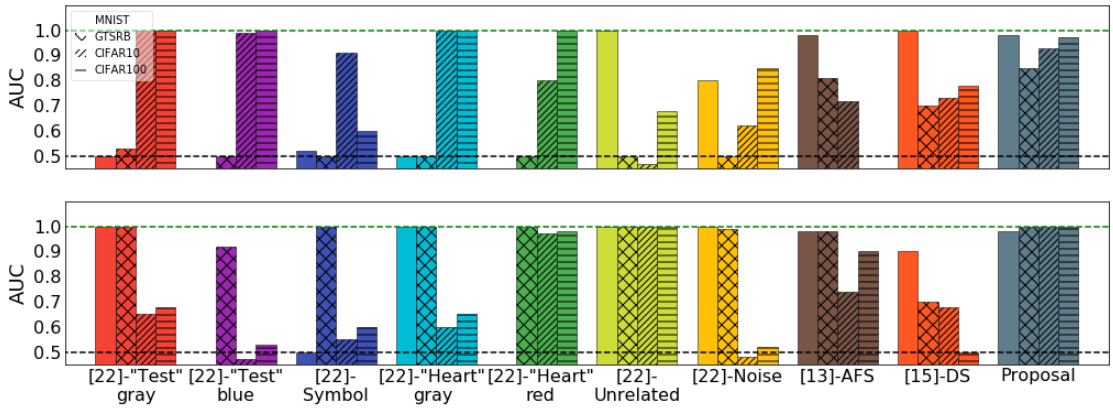
[22] Jialong Zhang, Zhongshu Gu, Jiyong Jang, Hui Wu, Marc Ph Stoecklin, Heqing Huang, and Ian Molloy. 2018. Protecting Intellectual Property of Deep Neural Networks with Watermarking. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security.* 159–172.



(a) AUC under model modification (pruning, top) and query modification (autoencoder, bottom) processing in $|K'| = 20$



(b) AUC under model modification (pruning, top) and query modification (autoencoder, bottom) in $|K'| = 10$



(c) AUC under model modification (pruning, top) and query modification (autoencoder, bottom) in $|K'| = 5$

Figure 2: Verification performance (AUC) of existing and proposed watermark methods under invalidation processing by model modification for pruning (top of each subfigure) and query modification (bottom of each subfigure) processing for CIFAR10, CIFAR100, GTSRB, and MNIST when $|K'| = 20, 10, 5$. The vertical axis shows AUC and the horizontal axis shows the watermarking methods in each dataset. The black dashed line shows "AUC 0.5" (verification is performed randomly) and the green dashed line shows "AUC 1" (verification performed perfectly).

A AUTOENCODER ARCHITECTURE AND MODEL HYPERPARAMETERS

Table 4 shows the architecture of our convolutional autoencoder and Table 5 summarizes the hyperparameters used to train ResNet and the autoencoder.

Table 4: Autoencoder architecture. The layer types of "Conv", "Deconv", and "bn" correspond to convolution, deconvolution, and batch normalization, respectively. "relu" and "sigmoid" refer to the activation functions.

Layer types	Kernel size	Stride
Conv,bn(relu)	$3 \times 3 \times 16$	2
Conv,bn(relu)	$3 \times 3 \times 32$	2
Conv,bn(relu)	$3 \times 3 \times 64$	2
Deconv,bn(relu)	$3 \times 3 \times 32$	2
Deconv,bn(relu)	$3 \times 3 \times 16$	2
Deconv,bn(sigmoid)	$3 \times 3 \times 3$	2

Table 5: Model parameters.

Parameter	ResNet	Autoencoder
Optimization	Momentum SGD	Adam
Epoch	100	400
Key mini-batch size	4	-
Mini-batch size	100	100
Learning rate	0.1 (×0.1 when epoch is 40, 60)	0.001

B ASSIGNMENTS OF SAMPLES

Table 6 shows (1) the number of samples that the model owner used to train classification model, (2) the number of key samples that the model owner embedded into the model, and (3) the sample size that the unauthorized service provider utilized to invalidate the watermark in each dataset. The total numbers of training samples of each dataset are 60,000 (MNIST), 39,209 (GTSRB), and 50,000 (CIFAR10, CIFAR100). We presume that the unauthorized service provider can use 1% (MNIST), 5% (GTSRB), and 10% (CIFAR10, CIFAR100) of total training samples for invalidation processing. Since MNIST and GTSRB are relatively easy classification tasks, if the unauthorized service provider has a sufficiently large number of samples (such as 10% in CIFAR10 and CIFAR100), the unauthorized service provider can train a model that can achieve a sufficiently high test accuracy.

In such a situation, the unauthorized service provider does not have any need to use licensed models without permission. For this reason, we assume that when conducting invalidation processing, the unauthorized user has a smaller number of samples for MNIST and GTSRB than for CIFAR10 and CIFAR100. The number of key samples embedded into the model is different for each method for the following reasons. Embedding by [22]-content requires the model to properly generalize the symbols or logotypes in order to achieve good AUC. Thus, it is desirable to embed a large number of key samples. Watermarking using the [13]-AFS and [15]-DS methods takes advantage of model overfitting to specific key samples. Accordingly, following the recommendations of [13] and [15], we embedded 30 key samples into the model.

Table 6: Numbers of samples used for training, watermarking, and model invalidation. Each column shows (1) the number of samples that the model owner used to train the classification model (training samples), (2) the number of samples that the model owner embedded into the model (key samples), and (3) the number of samples that the unauthorized service provider utilized to detect key samples in each dataset.

Dataset		MNIST	GTSRB	CIFAR10	CIFAR100
#Training samples		59,400	37,248	45,000	45,000
#Key samples for:	[22]-Content	53,548	37,046	40,478	44,558
	[22]-Unrelated	2,400	6,742	6,742	6,742
	[22]-Noise	53,548	37,046	40,478	44,558
	[13]-AFS	30	30	30	30
	[15]-DS	30	30	30	30
#Samples used for key sample detection		600	1,961	5,000	5,000