ClearStamp: A Human-Visible and Robust Model-Ownership Proof based on Transposed Model Training

Torsten Krauß
University of Würzburg

Jasper Stang
University of Würzburg

Alexandra Dmitrienko
University of Würzburg

Abstract

Due to costly efforts during data acquisition and model training, Deep Neural Networks (DNNs) belong to the intellectual property of the model creator. Hence, unauthorized use, theft, or modification may lead to legal repercussions. Existing DNN watermarking methods for ownership proof are often non-intuitive, embed human-invisible marks, require trust in algorithmic assessment that lacks human-understandable attributes, and rely on rigid thresholds, making it susceptible to failure in cases of partial watermark erasure.

This paper introduces ClearStamp, the first DNN watermarking method designed for intuitive human assessment. ClearStamp embeds visible watermarks, enabling human decision-making without rigid value thresholds while allowing technology-assisted evaluations. ClearStamp defines a transposed model architecture allowing to use of the model in a backward fashion to interwove the watermark with the main task within all model parameters. Compared to existing watermarking methods, ClearStamp produces visual watermarks that are easy for humans to understand without requiring complex verification algorithms or strict thresholds. The watermark is embedded within all model parameters and entangled with the main task, exhibiting superior robustness. It shows an 8,544-bit watermark capacity comparable to the strongest existing work. Crucially, ClearStamp's effectiveness is model and dataset-agnostic, and resilient against adversarial model manipulations, as demonstrated in a comprehensive study performed with four datasets and seven architectures.

1 Introduction

In the realm of rapidly advancing technologies, machine learning stands out for many advantages, primarily revolving around automation, informed decision-making, and insightful recommendations drawn from historical data. This transformative technology finds extensive application in diverse real-world scenarios, ranging from critical tasks such as medical image classification [5] to facilitating processes like natural language processing [12], and extends further into domains like autonomous driving [7] and translation services [50]. At its core, a machine learning system consists of a model, often a Deep Neural Network (DNN), and data. Acquiring this data, especially in large quantities necessary for robust machine learning, can be both challenging and costly. Furthermore, the data can be inherently sensitive, such as in the case of medical images, warranting strict protection under the umbrella of intellectual property (IP) rights belonging to its owner. Once data becomes available, significant computational resources are utilized for model training, which entails substantial effort and, consequently, costs. The resulting model, infused with knowledge distilled from underlying training algorithms and exhaustive data, represents a culmination of innovation and expertise in the field DNNs and embodies the IP of its creator.

However, in case a model is made available to the public or sold to third parties, the matter of safeguarding IP rights becomes a challenge. In the face of potential adversarial entities, these models are vulnerable to theft, unauthorized resale, unwarranted modification, or illicit utilization. Such circumstances, recognized as copyright infringements, have the potential to escalate into legal proceedings, underscoring the significance of robust and comprehensive protective measures.

The state-of-the-art approach to safeguarding the IP of trained models is model watermarking [3, 35]. Existing DNN watermarking methods commonly require a key as input to an algorithm for watermark extraction. This process generates an output that is subsequently verified against the secret ground truth of the watermark. Watermarking methods can be categorized based on two main properties: whether they operate by accessing the model's internals (white-box) [15, 15, 34, 54, 57, 61, 63], such as the weights, or by analyzing only prediction outputs (black-box) [1, 21, 28, 36, 40, 70, 71]. Moreover, a watermark is either 1-bit [28, 40, 54] or multi-bit [1, 15, 15, 21, 28, 34, 36, 57, 61, 63, 70, 71]. A 1-bit watermark merely indicates the presence or absence of a watermark, whereas a multi-bit watermark incorporates details like the name of the copyright holder.

Problem Statement. However, the watermarking techniques proposed so far can be considered non-intuitive, as they embed human-invisible watermarks. Consequently, for watermark verification, the evaluator must place trust in the algorithm, which typically extracts data from the DNN that lack human-understandable attributes. This data is then distilled into a single value, and then compared to a rigid threshold, where values above it confirm watermark's existence and values below it deem the watermark invalid. In cases of partial watermark erasure algorithmic assessment might fall short, while humans with a clear understanding of its prior presence might be able to easily verify a copyright infringement. In the context of an image, for instance, it is imaginable that a watermark, depicted as a stamp, could be erased, for instance, up to 70%; nevertheless, even with this level of reduction, it may still remain conspicuously apparent that the watermark was once embedded into the image.

This paper addresses these challenges and introduces ClearStamp, the first multi-bit, white-box DNN watermarking method that follows an intuitive approach. ClearStamp embeds a visible watermark that can be assessed through human inspection, empowering humans to make decisions based on common sense and reasoning while retaining the option for technology-assisted evaluations. In addition, the method is generic and can be applied to various datasets and model architectures, and is robust against a wide range of watermark erasure techniques, including manipulation attempts by adaptive attackers who know the details of the protection method and even of underlying keys.

Contributions. This paper makes the following contributions:

- We propose ClearStamp, the first DNN watermarking mechanism that yields human-visible and understandable outputs, bypassing the need for rigid value thresholds and enabling more coherent and defensible jurisdiction, especially in cases where portions of the watermark remain obviously discernible. As a consequence, ClearStamp provides more security for the model creators' intellectual property.

- We have pioneered the utilization of transposed model training inspired by deconvolutions [18,20,26,48,67,69] as a novel approach for integrating a visible and human-comprehensible watermark into a DNN. In this method, we define a transposed model architecture specifically tailored to the existing model's structure, which shares weights with the original model. This enables model training in a reverse fashion, focusing on embedding the watermark, while the effects on conventional forward training for the model's primary task are negligible.

- We entangle the watermark and the main task within all model layers to create a robust watermark that can withstand adversarial model manipulations, such as fine-tuning on third-party datasets, pruning of model parameters, or adaptive adversaries attempting to remove or overwrite the watermark. Inspired by Siamese Networks [30], we achieve this by sharing weights between the model's primary task and the watermark and by enforcing that the watermark does not cause abnormal model parameters. Any malicious modification necessitates sacrifices in the model's main task performance, rendering the model less useful.

- We conduct a systematic large-scale study to analyze factors influencing ClearStamp, demonstrating its independence from application-specific factors by leveraging different datasets (MNIST [16], CIFAR-10 [32], GTSRB [51], and CIFAR-100 [32]) and model architectures (CNNs, ResNet-18, ResNet-34 [25], ViT [17], and VGG11 [46]) during evaluation. Additionally, we test the watermark's robustness under various fine-tuning and pruning scenarios and showcase a substantial watermark capacity of 8,544 bits that can be embedded with a low error rate of 4.45%, which is comparable to the strongest existing work in terms of capacity with 8,400 bits [34].

In summary, this work introduces a highly intuitive and easy-to-understand white-box multi-bit DNN watermarking method, offering a robust defense against various attack scenarios. The embedded watermark is human-understandable, addressing the limitations of existing solutions, which often lack intuitive design and human-friendliness in the final decision-making process, a task that can be seamlessly undertaken by a human evaluator when employing ClearStamp.

2 Background

Watermarking Watermarking [14, 19, 24, 29, 39, 43, 59] is a technique to embed a digital mark or identifier into digital media, e.g., images or audio, without significantly altering the content's appearance or functionality, with the purpose of embedding a discernible sign of ownership, authenticity, or other relevant information. Those signs can then be used for various scenarios, e.g., copyright protection, authenticity verification, ownership attribution, digital rights management, tamper detection, or metadata embedding. Watermarks come in various forms, with visible and concealed watermarks being notable categories. 1-bit watermarks make explicit ownership claims through their mere presence, while multi-bit watermarks convey additional information. The choice of watermarking method depends on the specific use case and the desired level of security. Extracting the watermark, which is a *secret value*, relies on knowing the *extraction method* and is often accompanied by a specific *secret key*.

In DNNs the watermark is embedded typically within the model's parameters [3, 35]. This can transpire either directly, necessitating white-box access to the parameters for extraction and verification, or indirectly through a learning process that configures the parameters to produce outputs containing the watermark when subjected to specific inputs. In such a case, black-box access is necessary for watermark verification. Both methods are achieved by introducing an additional regularization term to the loss function during model training. This regularization term orchestrates parameter adjustments in alignment with the intended watermarking objectives.

For watermark extraction and verification, the DNN and the watermark's secret key are fed into an extraction algorithm which yields some data. Those data are then verified against the watermarks ground truth secret by an algorithm that relies on a rigid threshold.

Transposed Model Functionality The transposed functionality of a machine-learning model is the reverse of the original model's task. We leverage the concept of transposed models to embed a watermark into the model. For instance, consider a model that classifies image inputs into a feature vector indicating detected objects within the image. The transposed functionality would involve inputting such a feature vector into the model to generate an image that embodies the characteristics encoded within that feature vector. This concept extends to the more granular level of model layers, where transposed layers reverse the functionality of their corresponding original layers. A transposed model consists of multiple transposed layers in the reverse order of the original model's architecture. Below, we discuss common model layers and existing or straightforward methods for transposing their functionality. However, certain layers may not be capable of precisely recovering the original input, especially when input data have undergone compression, resulting in information loss.

Linear Layer. A linear layer performs a calculation such as $y = x \cdot w^T + b$, where w and b denote weights and bias matrices, and x and y represent input and output, respectively. Here, T denotes the transposed operation such that the matrix is flipped over its diagonal, e.g., A_{ij} becomes A_{ji}. Linear layers can be accurately reversed by computing $x = (y - b) \cdot w$, effectively retrieving the original input from the output.

Batch Normalization. Batch normalization layers [27] are used to keep the data flowing through a model in a specific range. Such layers perform a computation akin to $y = \frac{x - E(x)}{\sqrt{Var(x) + \varepsilon}} \cdot \gamma + \beta$, where $E(x)$ and $Var(x)$ are the feature-wise mean and variance of the input data, ε is a small constant, and γ and β are learnable parameters. As mean and variance are dependent on x, the operation cannot be reversed straightforwardly when only provided with y. To address this, one can set default values for $E(x) = 0$ and $Var(x) = 1$, resulting in $x = \frac{(y - \beta) \cdot \sqrt{1 + \varepsilon}}{\gamma}$, which provides a good approximation.

Pooling Layer. Pooling layers [68] reduce the dimensionality of data by selecting representative values among multiple data points based on specific rules, such as computing the average or selecting the maximum value. As this process fundamentally involves downsampling, the transposed functionality is centered around upsampling. Consequently, the exact transposition of such downsampling computations can only be approximated, as new data points must be inferred. One common approach to upsample data is through interpolation, with several interpolation methods available, such as the nearest-neighbor algorithm [44] or the bilinear algorithm [45].

Convolutional Layer. A convolution [33] transforms the input to extract relevant features. This transformation applies a filter or kernel to the input to produce output data points. Multiple input data points are convolved with the filter to generate a single output data point. Hence, similar to downsampling, the computation cannot be exactly reversed. However, a method proposed in [69] offers a reasonably effective approximation[1].

Dropout Layer. Dropout layers [49] are designed to distribute knowledge across various parameters. They implement a regularization technique that simulates training numerous neural networks with varied architectures concurrently. During training, random layer outputs are ignored, altering the layer's appearance and connectivity. Each training update reflects a distinct "view" of the layer. They exert influence during training, have no learnable parameters, and remain inconsequential during inference. Therefore, dropout layers are utilized identically to the forward pass during transposed training.

Activation Functions. Activation functions, e.g., ReLU [2], introduce non-linearity in a model and play a significant role in the model's ability to generalize learned knowledge. Some activation functions introduce lossiness, like ReLU, which maps all negative input values to zero while the positive input values remain the same. Naturally, such operations are irreversible, and thus, activation functions can not be transposed.

Transformer Blocks. Transformer models, like Vision Tranformer [17], deviate from convolution-based models and are constructed by so-called transformer blocks, which consist of an encoder and an attention module, both featuring linear layers, dropout layers, and activation functions. Vision Transformer divides the image into patches and embeddings are generated for each patch. The transposed functionality of these linear layers, dropout layers and activation functions was already elaborated in the previous paragraphs, indicating that transposing a Vision Transformer is straightforward.

In summary, the transposition of model functionalities can be applied to a variety of common machine learning architectures. While some layers allow for a straightforward reversal, others necessitate approximation methods due to inherent complexities and information loss.

Image Similarity Human discernment of whether two images share identical content is typically straightforward. Nevertheless, conventional machine-based methods for quantifying errors, such as the computation of Mean Squared Error (MSE) across all pixels in an image, can yield substantial error values, particularly when the images possess matching structural elements but differ in aspects like color. Human visual perception excels at extracting structural information from images,

[1]The method of [69] is also used in PyTorch as transposed convolution modules and can be seen as the gradient of the respective convolution.

and in this context, Wang *et al.* [64] introduced the Structural Similarity Index (SSIM). Within this paper, we employ SSIM as part of a loss function to embed a watermark into a DNN, as well as for post-extraction verification of the watermark's presence and integrity. The SSIM has a value range of $[-1,1]$, where 1 indicates perfect similarity, 0 indicates no similarity, and -1 indicates perfect anti-correlation.

SSIM addresses the limitations of other metrics by providing a quantifiable measure of image dissimilarity considering luminance, contrast, and structure, aligning more closely with human perception. As exemplified in Tab. 1, MSE calculations often highlight substantial disparities from the original image, whereas SSIM reliably identifies high levels of structural similarity of the content. As visualized in the fifth column of Tab. 1, even SSIM values of, e.g., 0.18, are sufficient, such that a human can claim similarity between two images.

3 Problem Setting

Considered Scenario. We consider a classical watermarking scenario: The model owner trains a Deep Neural Network (DNN) by utilizing a proprietary dataset and costly resources making it critical to protect the resulting model, which is the intellectual property of the owner. Thereby, a maximally effective watermark should be embedded that allows ownership claim. The produced model is then legally or illegally distributed, e.g., sold or stolen, and placed into production. If the owner suspects a copyright infringement of their model, an inspection of the suspected model can be conducted. Precisely, it should be possible to extract and verify the watermark, even if benign or adversarial modifications have been performed on the copy of the original model. In the following, we first define the objectives (Sect. 3.1) and the threat model (Sect. 3.2).

3.1 Watermark Objectives

Inspired by related works [6, 9, 10, 15, 21, 36, 65, 66], we define several objectives, that should be fulfilled by an effective watermark: 1) *Understandability*: The evaluation of the watermark should be easily possible by human inspection. We add this objective to the commonly used ones, as the final decision in common DNN watermarking methods typically relies on a rigid threshold that may not detect leftovers after watermark erasure attempts, but those could be obvious to detect for human observers. Such situations occur for example for partly erased watermarks, that are interpreted as removed by a machine but are still recognizable by human inspection. We visualize a respective toy example in App. 7.1. Further, decisions that can be made by humans are more intuitive and easily comprehensible than empirically determined thresholds. 2) *Fidelity*: Watermark embedding should preserve the model performance on the primary task. 3) *Reliability*: It must be reliably possible to extract the watermark from previously watermarked models[2]. 4) *Robustness*: The embedded watermark must withstand model modifications, which we describe in Sect. 3.2. 5) *Integrity*: The watermark method should uniquely identify the watermark's secret value respective to the watermark's key and should not extract a valid watermark from unwatermarked models. 6) *Capacity*: The amount of information embedded into the watermark should be maximized to strengthen ownership claims. 7) *Efficiency*: Embedding the watermark should introduce negligible computational overhead. 8) *Security*: The watermark should not introduce obvious footprints allowing for easy detection and removal. 9) *Generalizability*: The approach should be independent of the dataset or model architecture.

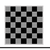
Image						
SSIM	1	-1	0	-0.02	0.18	0.75
MSE	0	64,322	31,642	32,856	19,764	5,909

Table 1: A reference image (first column) compared to images (first to sixth column) regarding SSIM and MSE values.

3.2 Threat Model

Our threat model specifies DNN model modification scenarios the attacker can undertake with the goal of removing the watermark from the trained and watermarked model.

Fine-Tuning. In fine-tuning [47, 53], the adversary continues training with a dataset akin to the original training dataset in the hope of removing watermarks that are added on top of the main task. Here we assume, that the adversary is aware of the training procedure including hyperparameters, and hence can adopt the settings for model modifications. Specifically, the learning rate is either kept at parity with the original training or, alternatively, can be decreased from the original value. In most benign fine-tuning scenarios, such a reduction of the learning rate is a typical approach to preserve the already trained meaningful features and only provoke small changes caused by the new dataset. Alternatively, the adversary can fine-tune the model on a different dataset, which may necessitate the substitution of the last model layer with an untrained counterpart due to a different number of label classes.

Pruning. Pruning [23] is normally used to reduce the DNN size to facilitate deployment in smaller setups like embedded devices. As the adversary can arbitrarily modify model weights, parameters can be pruned in the hope of removing the watermark while keeping a reasonable main task performance. This entails the elimination of a specific proportion of parameters, called pruning level, characterized by the lowest absolute values within the model, as those parameters are deemed to have the most marginal influence on the model's

[2]Note that watermarks from models that have been significantly manipulated to the extent that the original main task is severely compromised do not require detection, as the model no longer retains the creator's IP rights.

overall performance. Such pruning methods can be combined with fine-tuning, which is called fine-pruning [37, 56].

Adaptive Adversary. An informed adversary, possessing knowledge of the watermarking methodology, may attempt to manipulate the existing watermark [29, 57] leveraging the same embedding technique. Thereby, the adversary can invent a new watermark that can either contain meaningful or random data and embed the watermark in the hope of removing or replacing the original one. Removing the watermark would prevent ownership claims by the model creator. A replacement would transfer the possibility of claiming model ownership to the adversary. Usually, the watermark is kept secret and the adversary is not aware of the watermark's data.

4 Approach

In this section, we present our general concept in Sect. 4.1, followed by details about the generation of transposed models in Sect. 4.2, the composition of the watermark in Sect. 4.3, and details on the training procedure in Sect. 4.4.

4.1 Overview

We propose *ClearStamp*, a white-box and multi-bit DNN watermarking method, that embeds a watermark secret matching to a specific watermark key within the complete set of model parameters. The watermark key has the form of a regular output vector of the DNN and the watermark secret is represented by an input-like sample of the DNN, that can contain arbitrary information, e.g., random text superimposed on an image, without necessitating visual or context-wise similarity to actual dataset samples. After legal or illegal model distribution of the trained and watermarked model, ClearStamp's verification process can use the key to extract the embedded secret from the DNN and, thus, claim ownership and potentially copyright infringement. Below, we describe ClearStamp's principle and outline the successive steps of ClearStamp during the model life-cycle.

Principle of ClearStamp. In the process of embedding a watermark into the model, ClearStamp employs transposed model training, as visualized in Fig. 1. Therefore, we construct a transposed model architecture that is constructed using the method presented in Sect. 4.2 and establish weight sharing (similar to Siamese Networks [30]) between the standard model and transposed model by assigning the parameters of each layer to the respective transposed layer. Consequently, these shared weights can be subject to regular training for the main task via conventional (forward) training with respect to the forward loss, e.g., cross-entropy, as illustrated in the upper portion of Fig. 1. Simultaneously, we perform the transposed training with the predefined watermark key and secret visualized as key and lock in Fig. 1, representing the training data. The key, a prediction-like vector, is fed into the transposed model, generating an output akin to the input data of the regular DNN, e.g., an image, as visualized in the lower part of Fig. 1. During transposed training, the shared model weights are optimized with respect to a transposed loss, that is calculated by comparing the output of the transposed model with the watermark's secret. By applying both, forward and backward training, ClearStamp can entangle the main task and the watermark within all model layers of the DNN.

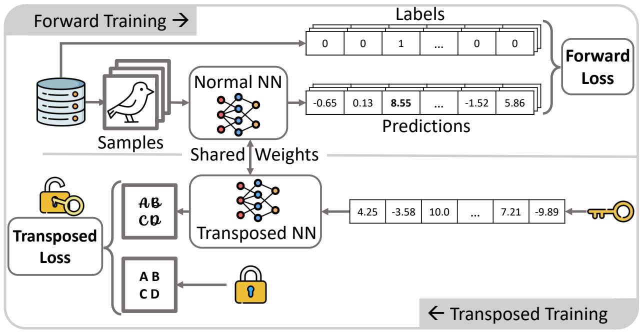


Figure 1: Visualization of main task forward training and watermark (key & lock) embedding in transposed training.

ClearStamp Life Cycle. During a model's life-cycle, ClearStamp follows the four steps visualized in Fig. 2. The untrained model is initialized with random parameters, that neither performs well on the main task for forward model inference nor on the watermark for transposed inference. 1) In the watermark hardening phase, we initialize the parameters of the model by transposed training on the watermark until a self-defined sufficient enough watermark quality is reached, which essentially means, that the model is overfitted on the watermark[3]. Thereby, the watermark builds the basis for the main task in the consecutive forward training within the model parameters. 2) During the constraint training phase, normal forward model training is equipped such that the already embedded watermark from step 1 persists. Thereby, we alternate between optimizing the two tasks, which is described in detail in Sect. 4.4. The training can mainly focus on optimizing the main task, as the effect of the watermark task during optimization is minimal, due to the foregone watermark hardening step. The parameters are prepared during step 1 such that the watermarking task yields a negligible loss compared to the main task and essentially functions as a constraint during normal model training. 3) After model distribution, the model is manipulated by a third party, e.g., fine-tuned[4]. As long as the main task performance is preserved, the watermark should remain embedded. 4) Finally, a transposed inference on the

[3]We suggest an SSIM value above 0.95 as watermark quality threshold, as such values can be achieved fast in transposed-only training since the watermark consists of a limited small amount of key-value pairs. In our experiments, we combine this threshold with a maximum of 10,000 epochs.

[4]Step 3 can differ depending on the scenario/attack. We describe various scenarios in Sect. 3.2 and provide evaluations in Sect. 5.2 and Sect. 5.4.

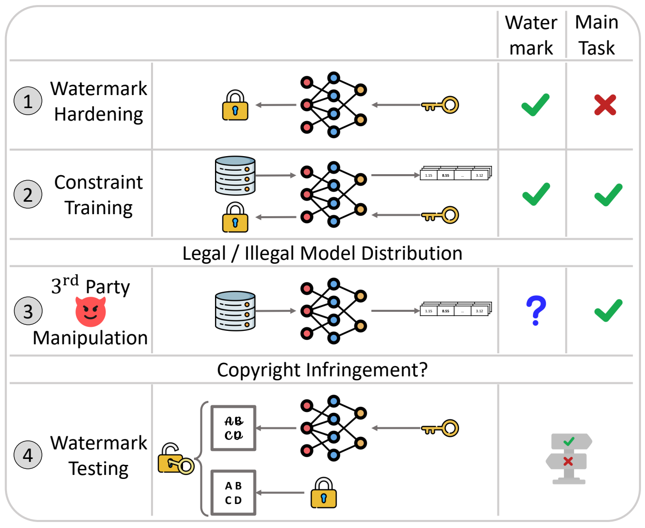


Figure 2: Overview of ClearStamp's life cycle steps.

watermark key is conducted to extract the watermark data, which is verified against the ground truth watermark secret by human-only inspection or machines. Since the main task relies on the fortified parameters established in step 1, creating an inherent interconnection between the tasks, substantial modifications made to the watermark in step 3 directly influence the main task. This direct impact enhances the overall robustness of the watermark.

To enable ClearStamp, we must define several components: First, in Sect. 4.2, we define rules for the creation of a transposed model from a given model architecture. Second, we specify how the key and the secret of the watermark are composed in Sect. 4.3. Third, in Sect. 4.4, we determine how to train while maintaining the watermark.

4.2 Transposed Model Generation

To ensure fulfillment of the generalizability requirement from Sect. 3.1, we establish guidelines for the generation of transposed models. Thereby, we define how model layers and connections are translated to the transposed version[5]. Linear layers and batch normalization layers [27] are straightforward mathematical operations and easy to transpose, as discussed in Sect. 2. For pooling layers [68], we leverage interpolation based on the nearest-neighbor algorithm [44]. Convolutional layers [33] are transposed with deconvolutions[6] as in [69]. Dropout layers [49] and activation functions, e.g., ReLU [2], are used likewise to their untransposed functionality.

Skip Connections. Skip connections fork the data processing within the model architecture and merge the data from both branches at a later stage, alleviating the vanishing gradient problem and improving the accuracy of DNNs. Skip connections effectively perform an operation akin to $a+b=c$ during the merging process within the forward path. Hence, they are difficult or impossible to reverse as only the output c is provided during transposed training, rendering a and b indistinct. To transpose the skip connection, we freeze one part of the connection, such as b, during transposed training. Thus, by the inverse of the mathematical operation between a and b the skip connection can be transposed utilizing c and the frozen b. This effectively adds b to the watermark's key. To get a reasonable estimation of a realistic value for b, we start training an unwatermarked model for a few initial epochs.

Additional Dropout Layers. To ensure the robustness requirement from Sect. 3.1, ClearStamp strives for a robust entanglement between the watermark and the main task. Therefore, the watermark needs to be embedded within all model layers during the watermark hardening step. To facilitate the entanglement between the watermark and the main task, spreading the watermark across multiple parameters must be enforced. Such a behavior can be achieved within model architectures by utilizing dropout layers [49]. For model architectures that lack inherent dropout layers, we artificially add such layers into the transposed architecture. Specifically, dropout layers are incorporated after each convolutional and linear layer, with the exception of the final layer responsible for producing the ultimate output of the transposed model. The dropout rate is an insensitive parameter, that must be set to some reasonable value, which can be quickly identified by analyzing the first few update steps during watermark hardening. Higher dropout rates extend the duration of the hardening process but do not compromise ClearStamp's functionality.

4.3 Watermark Composition

Watermark Key. The key's structure must align with the dimensions of a regular output vector of the (forward) model. Generally, there are no constraints on the values within this vector, allowing for arbitrary and extreme values, that are usually not encountered in forward prediction vectors. However, we need to enforce an overlap of the model's forward output value range and the watermark key's value range. Otherwise, e.g., if the key only consists of positive values, the model will most likely group the outputs of the main tasks to different value ranges, e.g., negative values. Such a separation prevents tight entanglement of the two tasks and encourages the model to handle the tasks in a multi-task instead of a constraint-task manner. As a result of separated value ranges, the watermark can be removed from the model with minimal effects on the main task, contrary to scenarios where the value ranges overlap. To address this, we generate random key vectors with values between a predefined range from -10 to 10, as visualized in Fig. 1, given that most random initialized models

[5]Please note that the transposed model can be generated solely from the weights of the original model and does not require any additional parameters.

[6]We adapt the settings, e.g., kernel size and stride, from the convolution with potential adjustments to padding to ensure the output of the deconvolution matches the original input.

predominantly generate values around zero.

Watermark Secret. The watermark's secret, adaptable to regular input sample dimensions like images, can be 1-bit using a random image or multi-bit with additional content like text. Similar to the watermark key values, the values of the watermark secret should fall within the range of typical input data. This ensures an intertwined relationship between the watermark and the main task parameters, ultimately enhancing the robustness of the watermark. When employing a loss function solely based on structural similarity between images, there might be challenges in producing outputs within the desired range. To address output range challenges, we employ a dual-loss strategy in transposed training. SSIM ensures structural similarity with the watermark secret, while MSE maintains exact secret values in the output.

Multi-Key. Employing not just one, but multiple unique key-secret pairs within a single watermark significantly increases the capacity of the multi-bit watermark addressing the capacity requirement from Sect. 3.1. This approach allows for the embedding of a greater volume of information. Furthermore, it enhances the robustness of the watermark as multiple keys influence a larger portion of the model's parameters, complicating erasure attempts by third parties. These distinct key-secret pairs are inputted into the transposed model as a unified batch. This method compels the model to grasp the underlying structure of the secrets and integrate the watermark throughout all layers. Additionally, this approach offers the advantage of preventing the parameters in the transposed model from memorizing specific key-secret pairs. Instead, the model generalizes the functionality underlying the samples, reinforcing learned capabilities across various layers. To optimize this approach, it is advisable to ensure that the different key-secret pairs share a consistent structure, such as all containing textual information within images. This uniformity enhances the model's ability to learn and embed diverse information effectively.

4.4 Constraint Training

Watermarking the model in the proposed way (cf. Fig. 1) essentially corresponds to simultaneously learning two separate tasks within one model, resulting in a multi-objective optimization problem consisting of the model's general functionality as task one and the watermark in the transposed model as the second task. After overfitting the transposed model to the watermark in the watermark hardening phase (step 1 of Fig. 2), the watermarking task can be considered as a constraint to the main task in step 2 in Fig. 2. This entails, that we optimize the main task while keeping the watermark functionality. To execute this optimization, we leverage sequential optimization, essentially alternating between optimizing the model parameters for the main task and the watermark. Alternatives to this optimization approach are discussed in App. 7.4.

5 Evaluation

Hardware & Experimental Setup. Experiments are implemented in PyTorch, a prominent Python-based machine learning library [55], on a server featuring an AMD EPYC 7413 24-Core Processor (64-bit) with 96 processing units and 128GB main memory. An NVIDIA A16 GPU with 4 virtual GPUs (each 16GB GDDR6 memory), is used via CUDA [42].

Datasets & Model Architectures. We use common datasets mainly focusing on image classification with MNIST [16], CIFAR-10 [32], GTSRB [51], and CIFAR-100 [32] trained on models of different types and sizes, namely CNNs (with and without batch normalization), ResNet-18, ResNet-34 [25], ViT [17], and VGG11 [46].[7]

Default Scenario. Throughout our experiments, we systematically vary model architecture, dataset, and hyperparameters to illustrate the versatility of our approach. Unless otherwise specified, our default scenario uses MNIST [16] trained on a CNN consisting of two convolution layers both followed by a ReLU and a 2D max pooling layer and followed by three fully connected layers of decreasing output sizes (512, 256, 10). For training purposes, we employ separate Adam optimizers with a learning rate of 0.0001, both for the primary task and transposed training. We trained the model for five epochs.

5.1 General Functionality

Watermark Definition. A watermark for ClearStamp consists of one or multiple watermark keys and secrets, which are kept confidential. In our experiments, the keys are vectors consisting of ten randomly chosen values between -10 and 10[8]. The chosen secrets are images containing four letters like "ABCD", as visualized in Fig. 3a and App. Fig. 11a.

Baseline - No Watermark. First, we trained a model without a watermark, which serves as a baseline for model performance. As can be seen in (1) in Tab. 2, we reached a model accuracy of 89.88%. When trying to extract a watermark before and after training, we get images as in Fig. 3b and Fig. 3c, respectively. The images clearly show no relation or similarity to Fig. 3a, indicating the absence of the watermark essentially fulfilling the integrity requirement from Sect. 3.1. Additionally, the two pictures yield an SSIM of 0.00 and -0.06 when being compared to Fig. 3a, confirming that watermarked and unwatermarked models are clearly distinguishable. However, human perception instead of a low SSIM should be the main criterion for the decision, as even for small SSIMs close to zero human observers can still recognize similarities between images (cf. Tab. 1 in Sect. 2).

Watermark Hardening. Next, in step 1 of ClearStamp

[7]We use PyTorch model instances for predefined model architectures.

[8]The precise random key vector for the experiments with one key is provided in App. 7.3. There, we also provide experiments with key ranges -5 to 5 and -50 to 50 to show that the range is an insensitive parameter.

Table 2: In these experiments MNIST [16] was trained on a CNN with a learning rate of 0.001 for five epochs.

	Number of keys	Untrained Model		Watermark Hardening			Training			Extracted Watermark Figure	Watermark Considered Valid	4 epochs fine-tuning with 1/10 of training learning rate	
		Accuracy	SSIM	Steps	Accuracy	SSIM	Constraint	Accuracy	SSIM			Accuracy	SSIM
(1)	-	10.22%	0.00	-	-	-	x	89.88%	-0.06	Fig. 3c	x	-	-
(2)	1	10.22%	0.00	7,000	8.57%	0.95	-	-	-	Fig. 3d	✓	-	-
(3)	1	10.22%	0.00	7,000	8.57%	0.95	✓	88.37%	0.95	Fig. 3e	✓	92.73%	0.95
(4)	10	10.22%	0.00	10,000	8.92%	0.93	✓	87.69%	0.91	Fig. 3f	✓	89.73%	0.93
(5)	11	10.22%	0.00	10,000	8.88%	0.92	✓	89.10%	0.91	Fig. 3g	✓	89.86%	0.93

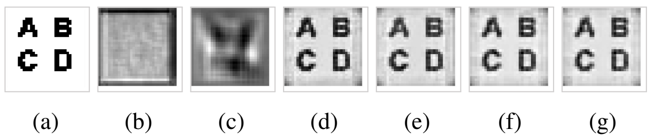


Figure 3: Visualization of (a) the watermark secret and (b-g) extracted watermarks for the experiments listed in Tab. 2.

(cf. Fig. 2) we perform transposed training as described in Sect. 4.1 to embedded a watermark consisting of one key-secret pair into an untrained model. As transposed loss, we combine SSIM and MSE between the transposed model output and the watermarks ground truth secret. We name each adjustment of the transposed model parameters by the optimizer as a hardening step. As presented in (2) in Tab. 2 we reached an SSIM of 0.95 after 7,000 hardening steps. The resulting image after watermark extraction depicted in Fig. 3d clearly shows the content of the ground truth secret Fig. 3a and an existing watermark can be attested. As expected, the accuracy on the main task remained naïve with 10.22% and 8.57% accuracy before and after training, respectively.

Constraint Training. In ClearStamp's second step, we train the model's main task while keeping the watermark embedded as described in Sect. 4.4. As reported in (3) in Tab. 2, the watermark remains embedded, while the main task accuracy of 88.37% is achieved, fulfilling the reliability requirement from Sect. 3.1. Hence, we observe a negligible accuracy drop compared to the unwatermarked model ((2) in Tab. 2) satisfying the fidelity requirement from Sect. 3.1. To emphasize this important fact, we visualize the main task loss for unwatermarked and watermarked training in App. Fig. 13c showing minimal differences. These results could be reproduced independently of the optimizer used for the different tasks, which we elaborate on in App. 7.3.

Multi-Key. Next, we investigate multiple watermark key-secret pairs, as explained in Sect. 4.3. We employed ten and eleven keys to show the independence from the number of classes in the dataset, ten for MNIST [16].[9] The secrets are distinct four-character images[10], as visualized in App. Fig. 11a. The results for ten and eleven keys are shown in (4) and (5) in Tab. 2, respectively, and show that ClearStamp embeds the watermark successfully. Thus, we can increase the watermark capacity without a significant negative impact on the watermark's or the model's performance[11]. Notably, we stopped hardening after 10,000 hardening steps but it would be possible to continue training until a 0.99 SSIM is reached[12]. While it is possible to increase the number of keys and thus the capacity, which we discuss in Sect. 5.5, we proceed with eleven keys to showcase the functionality of ClearStamp. To show the independence from the concrete secret images, we conducted the same experiments with other images, visualized in App. Fig. 12a and App. Fig. 12b, yielding similar results.

5.2 Model Manipulations

Next, we evaluate illegal and legal third-party model manipulations, that are applied in step 3 of Fig. 2.

Fine-Tuning. To evaluate ClearStamp's robustness against fine-tuning, as described in Sect. 3.2, we continued training on the MNIST train set after executing our default scenario for another two epochs (half of the original five epochs rounded down) with the same learning rate, as well as with 1/10 of the original learning rate (similar to [1,9,34,40,57,65]). After fine-tuning for two epochs, we continued for another two epochs with identical settings to showcase ClearStamp's behavior under excessive fine-tuning conditions. To evaluate the watermark robustness against unseen data, we executed the same fine-tuning process but employed the MNIST [16] test set. As Tab. 3 shows, ClearStamp shows strong robustness against fine-tuning as the watermark remained embedded yielding high SSIM values and clear images (Fig. 4a to Fig. 4c)[13]. Later, in Sect. 5.3, we also evaluate cross-dataset fine-tuning. In scenario (2) in Tab. 3 we can observe a slight increase in SSIM after 4 epochs compared to 2 epochs, which is counter-intuitive. We believe that caused by the overlapping value ranges of our watermark and the entanglement of parameters, weight changes for the forward path can have small positive effects on the watermark's similarity score.

[9]We provide results for another experiment leveraging 20 keys in App. 7.3, which yields similar results as for eleven keys.

[10]During visualizations, we stick to the first image containing "ABCD".

[11]We observe a slight increase in main-task accuracy for (5) in Tab. 2, indicating, that the watermarking is tightly enmeshed with the main task and serves as regularization in this experiment. However, the general observation is a minimal drop in accuracy due to watermark embedding.

[12]We report the mean SSIM values for multiple keys. If not specifically mentioned, the means do not contain extreme outliers.

[13]The accuracies show, that fine-tuning with 1/10 of the original learning rate is a better setting if an adversary wants to increase the model accuracy on a third-party dataset.

Table 3: Fine-tuning experiments for a CNN trained using MNIST [16] test set with a learning rate of 0.001 for five epochs with eleven watermark keys.

Fine-Tuning	2 Epochs		4 Epochs		Wateermark	
Scenario	ACC	SSIM	ACC	SSIM	Figure	Valid
(1)	88.64%	0.90	87.42%	0.88	Fig. 4a	✓
(2)	88.56%	0.92	89.86%	0.93	Fig. 4b	✓
(3)	96.31%	0.92	97.02%	0.92	Fig. 4c	✓

(1) Same as training learning rate (0.001) & same data
(2) 1/10 of training learning rate (0.0001) & same data
(2) 1/10 of training learning rate (0.0001) & unseen data

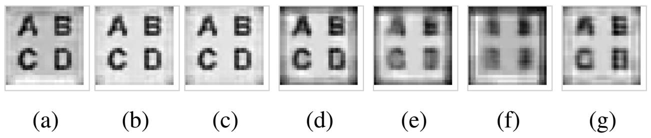


Figure 4: Visualization of extracted watermarks for the fine-tuning experiments listed in Tab. 3 in (a-c). Figures (d-f) visualize pruning with 60%, 80%, and 90% respectively, while (g) shows fine-pruning with 40%.

Pruning. Besides fine-tuning, we investigate model pruning (cf. Sect. 3.2) similar to [9, 15, 34, 40, 57, 65]. As depicted in Fig. 5, the watermarking withstands pruning and is coupled to the main task accuracy, as for low pruning levels, which maintain the accuracy, the SSIM remains high. For example, for 60% pruning with an accuracy drop from 89.1% to 78.56%, the SSIM is still 0.69, yielding Fig. 4d. Even for 80% pruning, which already suffers in accuracy with 50.1% we obtain an SSIM of 0.47 resulting in Fig. 4e, which is still sufficient for a human observer to identify the watermark when being aware of the ground truth secret Fig. 3a. Starting from 90% pruning (cf. Fig. 4f), the watermark cannot be clearly identified, but the model already decreased to 26.76% accuracy essentially being useless. Fig. 4g shows the result with an SSIM of 0.62 and an accuracy of 89.79% after two fine-tuning epochs followed by pruning with 40%, typically called fine-pruning [28, 56]. As the images prior to 90% pruning in Fig. 4f are clearly distinguishable from Fig. 3c while Fig. 4f suffers low accuracy, we can conclude that ClearStamp is robust against model pruning essentially addressing the security requirement from Sect. 3.1.

5.3 Generalizability

Below, we explore different scenarios showing that ClearStamp can fulfill the generalizability requirement from Sect. 3.1. Essentially we demonstrate the independence from datasets and model architectures. During these experiments, we use fine-tuning with 1/10 of the original learning rate as the default model modification approach.

Dataset. First, we changed the dataset to CIFAR-10 [32], essentially changing the input layer to match the three color channels of the CIFAR-10 input samples. Then, we conducted

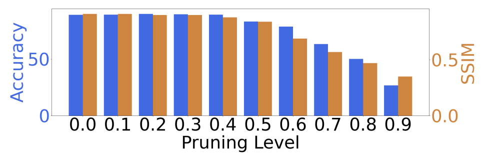


Figure 5: Model accuracy on the main task and corresponding watermark SSIM for different pruning levels between 0 (unpruned) and 90%.

the same experiment for GTSRB [51], which has 43 label classes. The results reported in (1) and (2) in Tab. 4 show, that the watermark was successfully embedded and survived fine-tuning yielding Fig. 6a and Fig. 6b. As the images clearly show the expected letters, we observe the dataset independence of ClearStamp.

Small Model Architectures. Next, to show that ClearStamp is also applicable to very simple model architectures, we trained MNIST [16] on a CNN with only three fully connected layers each of size 1024 and report the result in (3) in Tab. 4. Further, to show ClearStamp's independence of added batch normalization layers when embedding the watermark, we enhanced our default setting by adding such a layer after each convolutional layer and report the results in (4) in Tab. 4. The extracted images Fig. 6c and Fig. 6d, as well as the high SSIM values above 0.85 after raining and fine-tuning reported in Tab. 4 confirm ClearStamp's good performance for small models. However, even if the images yield clear watermark evidence, the presence of batch normalization layers seems to diminish the robustness of the watermark resulting in a lower SSIM of 0.85 after fine-tuning. This effect might be caused by the circumstance, that for batch normalization layers, the mean and variance of the input data are unknown during transposed training and fixated to $E(x) = 0$ and $Var(x) = 1$, essentially causing information loss.

Medium-Size Model Architectures. To address bigger model architectures, we evaluate CIFAR-10 [32] and GTSRB [51] on a ResNet-18 [25] model trained for ten epochs[14], whereas both datasets yielded similar results reported in (5) and (6) in Tab. 4. As retractable in Fig. 6e and Fig. 6f, the watermark is still clearly visible after ten epochs.

Further, we evaluate CIFAR-10 [32] on ResNet-34 [25] and use the same setup as in the ResNet-18 experiment. The results in (7) in Tab. 4 yield a bigger drop in SSIM to 0.55 after fine-tuning, probably introduced by the size of the model and the amount of transposed convolution layers, which introduce uncertainty due to their upsampling nature. Nevertheless, the resulting image Fig. 6g leaves no doubt that the watermark is still strongly embedded.

Inspired by [1, 57], we also evaluate cross-model fine-tuning scenarios. We used our medium-size model setups and

[14]As ResNet-18 [25] contains skip connections, we first trained for three epochs, to get valid values for skip connections fixating (cf. Sect. 4.2).

Table 4: Experiments showing the independence of ClearStamp from datasets and model architectures. Tests were conducted with eleven watermark keys and fine-tuning was performed for the same number of epochs as training epochs.

	Untrained Model		Watermark Hardening			Constraint Training		Fine-Tuning with 1/10 of training learning rate (LR)		Extracted Watermark Figure	Watermark Considered Valid
	Accuracy	SSIM	Steps	Accuracy	SSIM	Accuracy	SSIM	Accuracy	SSIM		
(1)	10.01%	0.00	10,000	10.07%	0.92	44.16%	0.93	45.87%	0.93	Fig. 6a	✓
(2)	1.45%	0.00	10,000	0.71%	0.92	38.96%	0.91	42.70%	0.93	Fig. 6b	✓
(3)	10.59%	0.00	200	10.63%	0.99	85.27%	0.99	87.86%	0.96	Fig. 6c	✓
(4)	14.38%	0.00	1,000	2.81%	0.99	93.47%	0.99	94.19%	0.85	Fig. 6d	✓
(5)	5.74%	0.00	4,000	10.00%	0.96	62.87%	0.96	68.74%	0.91	Fig. 6e	✓
(6)	1.57%	0.00	4,000	0.47%	0.96	72.51%	0.96	81.03%	0.93	Fig. 6f	✓
(7)	5.74%	0.00	4,000	10.00%	0.96	62.87%	0.96	68.74%	0.55	Fig. 6g	✓
(8)	0.64%	0.00	4,000	1.06%	0.96	28.36%	0.97	64.88%	0.51	Fig. 6h	✓
(9)	1.00%	0.00	1,000	0.91%	0.99	47.27%	0.98	50.86%	0.79	Fig. 6i	✓
(10)	12.02%	0.00	1,000	11.04%	0.98	55.56%	0.97	57.90%	0.60	Fig. 6j	✓

(1) Default scenario & CIFAR-10 [32]dataset
(2) Default scenario & GTSRB [51]dataset
(3) Default scenario & CNN with only FC layers, LR 0.0001, 3 epochs
(4) Default scenario & CNN with batch normalization layers
(5) CIFAR-10 [32] on ResNet-18 [25], LR 0.001, 10 epochs
(6) GTSRB [51] on ResNet-18 [25], LR 0.001, ten epochs
(7) CIFAR-10 [32] on ResNet-34 [25], LR 0.001, 10 epochs
(8) CIFAR-100 [32] on ResNet-18 [25], fine-tune on CIFAR-10 [32], LR 0.001, 10 epochs
(9) CIFAR-100 [32] on VGG11 [46], LR 0.001, 200 epochs
(10) CIFAR-100 [32] on ViT [17], LR 0.001, 5 fine-tuning epochs

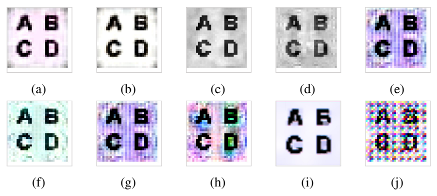


Figure 6: Visualization of extracted watermarks for the experiments listed in Tab. 4.

trained CIFAR-100 [32] on ResNet-18 [25] but fine-tuned the model on CIFAR-10 [32], which necessitates changing the last layer due to ten instead of 100 output classes. As the watermark is embedded based on CIFAR-100, the last layer needs to be replaced with the original layer during watermark extraction, thus rendering this layer as part of the key. We report positive results with 0.51 SSIM in (8) in Tab. 4. The watermark extraction yielded Fig. 6h, which clearly contains the watermark. We also conducted an experiment where we added the watermarked last layer to an unwatermarked model, resulting in an SSIM of 0.06 and watermark images that do not yield any perceptible content. These experiments combined demonstrate that the watermark is embedded within all network parameters and not only resides within the last layer, contributing to the robustness requirement from Sect. 3.1.

Large Model Architectures. To address even larger model architectures, we trained CIFAR-100 [32] on VGG11 [46] for 200 epochs and 100 epochs of fine-tuning. As the results in (9) in Tab. 4 show, ClearStamp could successfully embed a robust watermark even in a large model architecture.

To evaluate ClearStamp on transformer blocks, we trained CIFAR-10 [32] on a Vision Transformer [17]. The extracted watermark after fine-tuning is visualized in Fig. 6j. It has a colorful background, but the watermark text is clearly visible, indicating that ClearStamp can also handle such architectures.

Summarized, we showed, that ClearStamp is independent of the model architecture and, combined with the dataset independence, applicable in arbitrary application scenarios.

5.4 Adaptive Adversary

Watermark Erasure. An adversary with knowledge of ClearStamp's functionality could try to erase an embedded watermark as defined in Sect. 3.2. Thereby, a random watermark key in the predefined value range could be used in combination with a secret image consisting of random noise. An even stronger adversary with the knowledge of embedded watermark keys (which exceeds usual assumptions as in Sect. 3.2 could also use such an image. As both scenarios yield the same effects, we depict the results for a stronger adversary (using an already embedded watermark key) in the main body of the paper and append the random key experiment results in App. 7.3.

We use the same mechanism as in step 1 in Fig. 2 to embed the adversarial key-secret pair, with the intention of erasing the existing secret for the respective watermark key and potentially for other existing keys that are part of the watermark. Our experiment revealed, that the erasure of the watermark involves a significant loss in main task accuracy as both tasks are entangled within the model parameters by design. During watermark erasure attempts, the adversary sacrifices usually more than 10% accuracy in our setup compared to the initial 89.10%, whereas some other existing works consider 3.5% accuracy drop as acceptable in related works [40]. In Fig. 7, we show seven out of eleven watermark images[15] after five, seven, and eleven adversarial hardening steps with respective remaining accuracy values of 82.89%, 78.60%, and 74.30%. Even the third row with an accuracy drop of 14.8% shows clear evidence, as one should keep in mind, that an unwatermarked model yields an image similar to Fig. 3c. In the first and second rows of Fig. 7, the watermark existence can still be attested by a human observer even though the SSIMs are

[15]We show only seven images due to space limitations in the paper.

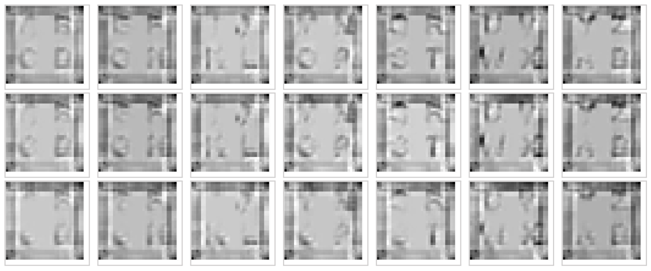


Figure 7: Extracted watermark images after five, seven, and eleven adversarial hardening steps during watermark removal. The corresponding main task accuracies to the lines are 82.89%, 78.60%, and 74.30% originating from 89.10% with SSIMs of 0.13, 0.11, and 0.01.

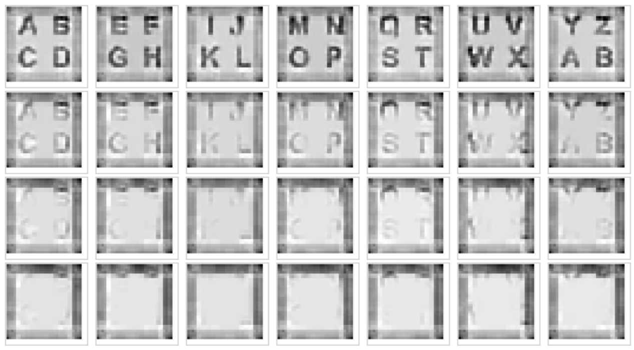


Figure 8: Watermarks after watermark overwriting with eleven keys. The lines correspond to the first four hardening steps with accuracy drops from 89.10% to 83.89%, 70.38%, 56.39%, and 46.52%, and SSIMs of 0.52, 0.17, 0.05, 0.01.

low with 0.13, 0.11, and 0.01. This shows that the definition of rigid thresholds is challenging as in the case of SSIMs such low values could also stem from content-wise unrelated images. Therefore, a threshold needs to be set higher to avoid false positives. Hence, ClearStamp improves the decision-making in such situations, essentially increasing the security for the model owner while fulfilling the understandability requirement from Sect. 3.1. We got similar results when using a completely black image as a key and when using an image that an unwatermarked model yields for the adversarial key after a transposed inference, which is reported in App. 7.3.

Further, we can report, that we observed the same effect when increasing the number of keys embedded by the adversary from one to eleven. We experimented with eleven already embedded or eleven random keys combined with secret images containing random noise, only black pixels, and yielded images from an unwatermarked model like Fig. 3c.[16] For example, for the latter, we measured accuracy drops from 89.10% to 83.89%, 70.38%, 56.39%, and 46.52% after one to four hardening steps resulting in the four lines of Fig. 8, where the watermark is still completely identifiable in the second line, and partially even in the third line. To achieve a removal degree as showcased in the fourth line, the adversary would need to sacrifice 42.58% in main task accuracy.

[16]Due to space limitations in the paper we only report one scenario, as all scenarios yield similar results.

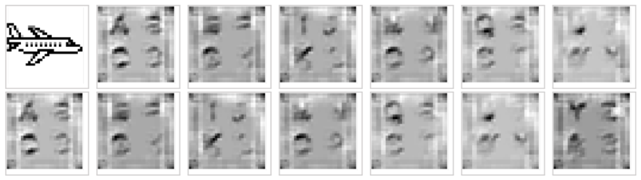


Figure 9: An adversarial watermark secret (first image) and extracted watermarks after six (line one) and seven (line two) hardening steps corresponding to 79.25% and 78.25% remaining main task accuracy originating from 89.10% and SSIMs of 0.24 and 0.23.

To validate that our findings are not intrinsic to our specific application scenario, we also conducted the experiment for CIFAR-10 [32] trained on ResNet-18 [25] yielding similar results that we report in App. 7.3.

Watermark Overwriting. Besides removing the watermark, an adversary could also try to overwrite the watermark with a second watermark. When embedding one new watermark image, in our case an airplane icon as visible in the first image in the first line of Fig. 9 with a random key, the adversary provoked an accuracy drop from 89.10% to 79.25% and 78.25% after six and seven hardening steps. While evidence for the original watermark is still visible after six steps, it starts to be vague after seven steps, as can be seen in the first and second lines in Fig. 9, respectively.

Summarized, we can conclude, that ClearStamp is robust against adaptive adversaries with the knowledge of ClearStamp's functionality, that try to remove or overwrite the watermark or embed an additional watermark, even if the adversary is equipped with knowledge about the keys. If the adversary erases the watermark, he sacrifices a minimum of 10% of accuracy, whereas usually around 3.5% accuracy drop is considered as acceptable [40].

5.5 Capacity

To evaluate the capacity and generalizability to arbitrary watermark locks, we generated random bit strings and converted them into images using dot code [58]. The image is initially divided into square patches that correspond to the number of bits in the string. Based on the bit value, the patch is colored black for zero or white for one. An example of this is shown in App. Fig. 11b. Multiple watermarks were injected into the model, with the same capacity and randomly generated keys. Likewise, the extracted watermark was again divided into patches to extract the bit sequence. Values less than 0.53

in each channel were set to 0, while others were set to 1. To determine the bit of the patch, black and white pixels were counted and the majority was used as a decoding result. To evaluate the outcome, the Binary Error Rate (BER) was utilized, which is the number of incorrect bits divided by the total number of bits. We first generated images with 36-bit capacity and injected those into the CNN model. The evaluation has shown that the BER is consistently low at roughly 3.43% and 2.96% after fine-tuning. For larger bit lengths of 100 bits per image we injected the MiT license text [17] (8,544 bits) into a ResNet-18 [25] model. We compared the results for applied (7,4) Hamming code [22] error correction (14,952 bits) and without. ClearStamp achieves average BERs after training of as little as approximately 5.92% and 3.62% with and without error correction, respectively. After fine-tuning the BER increased to 6.46% without error correction and 4.45% with error correction. ClearStamp is capable of injecting large payloads such as licenses or images and, as the limit of keys is not yet exhausted, even larger files could be injected.

5.6 Runtime

Regarding ClearStamp's runtime, we measured the individual steps, namely hardening, constraint training, and watermark testing for our default scenario with eleven random keys.[18] We report averages over ten experimental runs. Hardening with 10,000 steps took 53.48 seconds and is a one-time effort. The evaluation only takes 0.02 seconds. The training time increased from 67.62 to 94.39 seconds for the five epochs, introducing a one-time overhead of 39.58%, which is expected, as an additional loss needs to be computed and a second optimization step is executed during training. Therefore, we consider the efficiency requirement from Sect. 3.1 as fulfilled. Summarized, we can show that ClearStamp is a robust watermarking mechanism that withstands adversarial watermark-removing attempts and barely influences the DNN's main task performance while providing high watermark robustness. Most importantly, no non-intuitive algorithm based on a riding threshold needs to be applied to evaluate the watermark.

6 Related Work

Our approach, ClearStamp, stands out as the pioneer in the field of model watermarking by incorporating transposed model training, making it unique and unparalleled in comparison to existing methodologies. This novel technique sets it apart from related research, rendering direct comparisons challenging. The distinctive advantage of our method lies in its departure from conventional approaches that rely on fixed thresholds for decision-making during watermark verification. Instead, ClearStamp produces a discernible image with interpretable content. This output can be easily scrutinized by human evaluators, enabling intuitive decision-making when compared to the authentic ground truth image.

In the following sections, we present an overview of contemporary techniques in model watermarking and fingerprinting. Despite employing diverse methods, these existing approaches share common objectives with ours. Typically, watermarking methods seek to embed a distinctive signature into the model, ensuring its uniqueness to the model owner. On the other hand, fingerprinting methods are geared toward copyright protection, embedding a unique signature into the model that is specific to the authorized user. Additionally, we overview the related works that, while not strictly falling under watermarking or fingerprinting categories share relevance due to their underlying methodologies or the goals they aim to accomplish.

Watermarking. Uchida *et al.* [57] proposed a white-box, multi-bit watermarking approach, which embeds the watermark into weights of convolutional layers by introducing an additional loss term, referred to as parameter regularization. However, the method relies on a rigid threshold for watermark verification and is not robust against watermark overwriting attacks. ST-DM *et al.* [34] improves this approach in such scenarios by using modulation techniques. The capacity of the watermark was thereby shown for up to 8,400 bits but naturally is limited by the model architecture.

Frontier Stitching [40] is a 1-bit, black-box methodology rooted in adversarial examples. This technique modifies the decision boundaries between classes for specific input samples positioned at the interface of two classes, serving as essential keys. However, identifying such samples is contingent upon the specific application context and proves to be a challenging task, making this approach less universally applicable and user-friendly. For verification, a hard threshold of prediction matches is used.

Adi *et al.* [1] propose a multi-bit, black-box method, which embeds backdoors into the DNN that serve as a watermark. Within the paper, eight backdoors acting as watermarks were implanted, whereas randomly generated input samples were used to produce specific output classes. Similarly, Zhang *et al.* [70] and Zhang *et al.* [71] follow the same approach varying the backdoor types, whereas Li *et al.* [36] produces an imperceptible backdoor trigger with a second generative model. A method that also leverages backdoor-like behavior is Guo *et al.* [21], where a black-box watermarking technique is proposed specially crafted for embedded devices. The method trains the DNN to behave significantly differently on a portion of training samples, that are modified using a specific perturbation. Naturally, these approaches embed additional behavior besides the model's main task, providing the potential for side effects during normal inference. Further, the decision-making relies on a threshold for the number of occurred backdoor-based mispredictions. Contrary to these

[17] MiT License available at `https://opensource.org/license/mit/`.

[18] For estimating the skip connections, we trained an unwatermarked model for a few initial epochs (three in this case). This is a one-time effort, that is only necessary for models with skip connections in the model architecture.

works, WILD [38] removes backdoor-based watermarks.

Tartaglione *et al.* [54] propose a white-box, 1-bit watermark that fixates model parameters as watermarks and applies a modified loss function for increased robustness. As the approach fixates parameters, the watermark capacity is limited by the number of parameters in the model and hence depends on the model architecture.

DeepJudge [10] is a testing framework that can be used for copyright protection as a non-invasive alternative to watermarking techniques. The method compares how similar a DNN and a second suspected DNN under test behave based on six metrics and hard thresholds. The metrics are derived via inference of carefully chosen samples that are able to characterize the models.

Wang *et al.* [61] presents a white-box approach for incorporating a multi-bit watermark into weights by leveraging a second secret independent model for watermark embedding and verification. Similarly, RIGA [63] is an algorithm that embeds a multi-bit, white-box watermark using adversarial training with two additional models. Thereby, the first model is responsible for embedding the watermark, while the second enhances the stealthiness. However, the verification of both approaches is based on the black-box functionality of additional models. Hence, the decision-making is not comprehensible, as the model's functionality does not follow an understandable algorithmic pattern, but delivers outputs utilizing optimized parameters tuned with regard to training data.

EWE [28] is a method based on a special loss function, which enforces the entanglement of the watermark and the main task (similar to ClearStamp), such that removing the watermark negatively affects the main task. However, the watermark is embedded in the forward path, thus leaving the possibility of unexpected side-effects during inference.

DeepSigns [15] proposes a white-box, multi-bit approach that embeds a watermark within the probability density function of activations in multiple DNN layers by fine-tuning the model parameters. When verifying the watermark, an algorithm uses the extracted activations to compute the watermark which is then evaluated against the ground truth utilizing bit error rate combined with a rigid threshold. Further, a black-box approach is suggested, that verifies model ownership by a hard threshold applied to the number of matches when comparing the prediction outputs of specific secret input-prediction pairs.

Fingerprinting. A consecutive work to DeepSigns [15] leveraging the same principle and inheriting the same shortcomings is DeepMarks [9], but embeds information in the model weights instead of the activations and is designed as a fingerprinting approach. Based on this fingerprinting technique, DeepAttest [8] offers hardware-level IP protection and usage control for DNNs. With the help of a Trusted Execution Environment, the fingerprint is validated to ensure that only validated DNNs are allowed to run on specific devices, a method that is also leveraged in DeepMark [65].

IPGuard [6] searches for adversarial examples that are close to the decision boundary of a DNN, leveraging that a model is characterized by its decision boundary. The method does not tamper with the training process at all, but can run after training even on legacy models and, hence, does not affect the model performance at all. However, humans who do not understand the decision boundaries of DNNs might have problems understanding the approach. Further, the ownership verification is based on a hard threshold, which is difficult to determine. The decision boundary is also leveraged by MetaFinger [66], a black-box fingerprinting method that identifies samples by meta-training that are close to the decision boundary, which can later be used to identify a specific model.

Orthogonal Works. Further, there are some works, that are close but also orthogonal to ClearStamp. TamperNN [41] is a method designed to recognize if a model was tampered, e.g., fine-tuned, by analyzing inputs, that tend to change the prediction class easily. Chen *et al.* [11] suggest a method to infer the origin of a student model in the domain of transfer learning by embedding a fingerprint in the teacher model that is passed on to the derived student model. Venugopal *et al.* [60] propose a method to watermark the model output instead of the model itself by selecting a specific result out of the selection of possible results in a machine translation task. DAWN [52] is a technique used to prevent model extraction attacks by changing the prediction on the model inference API for a small set of samples to embed a watermark into models trained on these predictions. BOP [13] modifies the Adam optimizer to prevent so-called heavily spiked weights during watermark embedding and, hence, increase covertness while simultaneously increasing the robustness. Wang *et al.* [62] demonstrated that statistical analysis of model weights can detect a watermark. Once identified, the watermark can be overwritten using the original embedding technique, effectively removing it.

Deconvolutions [20,69] are the inspiration and basis for our work and are used to approximately reverse convolutional operations that are often leveraged in machine learning. Such deconvolutions are used in scenarios that require up-sampling of feature maps, such as generative models [18,67] and encoder-decoder architectures [26,48], which generate images from an embedding. However, the deconvolutions mostly possess their own trainable weights independent of the convolutions. Weight sharing is used in Siamese Networks [30], which provides the motivation to share the weights between convolutions and deconvolutions and, thus, for transposed training.

7 Conclusion

Machine learning models can be considered as the model creator's intellectual property that needs to be protected from unauthorized use. DNN watermarking techniques offer a solution to this problem by embedding a secret watermark into the model parameters. Obviously, these watermarks must be robust against erasure attempts, while simultaneously a minimal effect on the model's main task is expected.

Existing watermarking approaches rely on rigid thresholds in the final decision-making process after the watermarking data is extracted from the model during watermark verification. Thereby, such a threshold can fail to detect remaining fractions of embedded watermarks that were attacked with an erasure attempt, even if a human observer would clearly identify the remaining watermark.

To address this problem we proposed ClearStamp, the first human-understandable and intuitive DNN watermarking approach that allows human decision-making directly on the extracted watermark data without relying on a threshold. We show that ClearStamp's effect on the model's performance is negligible and that ClearStamp is independent from specific application scenarios. Further, ClearStamp withstands adversarial model manipulations and offers a capacity of 8,544 bits with a low error rate of 4.45%.

Acknowledgments

This research has been funded by the Federal Ministry of Education and Research of Germany (BMBF) within the program „Digital. Sicher. Souverän.“ in the project "Erkennung von Angriffen gegen IoT-Netzwerke in Smart Homes - IoTGuard" (project number 16KIS1919).

References

[1] Yossi Adi, Carsten Baum, Moustapha Cisse, Benny Pinkas, and Joseph Keshet. Turning Your Weakness into a Strength: Watermarking Deep Neural Networks by Backdooring. *USENIX Security*, 2018.

[2] Abien Fred Agarap. Deep Learning using Rectified Linear Units (ReLU). *arXiv preprint arXiv:1803.08375*, 2018.

[3] Franziska Boenisch. A Systematic Review on Model Watermarking for Neural Networks. *Frontiers in Big Data*, 2021.

[4] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, Jonathan Eckstein, et al. Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers. *Foundations and Trends® in Machine learning*, 2011.

[5] Lei Cai, Jingyang Gao, and Di Zhao. A review of the application of deep learning in medical image classification and segmentation. *Annals of translational medicine*, 2020.

[6] Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. IPGuard: Protecting Intellectual Property of Deep Neural Networks via Fingerprinting the Classification Boundary. *ASIACCS*, 2021.

[7] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. DeepDriving: Learning Affordance for Direct Perception in Autonomous Driving. *ICCV*, 2015.

[8] Huili Chen, Cheng Fu, Bita Darvish Rouhani, Jishen Zhao, and Farinaz Koushanfar. DeepAttest: An End-to-End Attestation Framework for Deep Neural Networks. *ISCA*, 2019.

[9] Huili Chen, Bita Darvish Rouhani, Cheng Fu, Jishen Zhao, and Farinaz Koushanfar. DeepMarks: A Secure Fingerprinting Framework for Digital Rights Management of Deep Learning Models. *ICMR*, 2019.

[10] Jialuo Chen, Jingyi Wang, Tinglan Peng, Youcheng Sun, Peng Cheng, Shouling Ji, Xingjun Ma, Bo Li, and Dawn Song. Copy, Right? A Testing Framework for Copyright Protection of Deep Learning Models. *IEEE S&P*, 2022.

[11] Yufei Chen, Chao Shen, Cong Wang, and Yang Zhang. Teacher Model Fingerprinting Attacks Against Transfer Learning. *USENIX Security*, 2022.

[12] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *JMLR*, 2011.

[13] Betty Cortiñas-Lorenzo and Fernando Pérez-González. Adam and the Ants: On the Influence of the Optimization Algorithm on the Detectability of DNN Watermarks. *Entropy*, 2020.

[14] Ingemar J Cox, Joe Kilian, F Thomson Leighton, and Talal Shamoon. Secure Spread Spectrum Watermarking for Multimedia. *IEEE TIP*, 1997.

[15] Bita Darvish Rouhani, Huili Chen, and Farinaz Koushanfar. DeepSigns: An End-to-End Watermarking Framework for Ownership Protection of Deep Neural Networks. *ASPLOS*, 2019.

[16] Li Deng. The MNIST Database of Handwritten Digit Images for Machine Learning Research. *IEEE Signal Processing Magazine*, 2012.

[17] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition. *arXiv preprint arXiv:2010.11929*, 2021.

[18] Alexey Dosovitskiy, Jost Tobias Springenberg, Maxim Tatarchenko, and Thomas Brox. Learning to Generate Chairs, Tables and Cars with Convolutional Networks. *IEEE TPAMI*, 2017.

[19] Borko Furht and Darko Kirovski. *Multimedia Security Handbook*. CRC press, 2004.

[20] Hongyang Gao, Hao Yuan, Zhengyang Wang, and Shuiwang Ji. Pixel Transposed Convolutional Networks. *IEEE TPAMI*, 2020.

[21] Jia Guo and Miodrag Potkonjak. Watermarking Deep Neural Networks for Embedded Systems. *ICCAD*, 2018.

[22] R. W. Hamming. Error detecting and error correcting codes. *The Bell System Technical Journal*, 1950.

[23] Song Han, Jeff Pool, John Tran, and William Dally. Learning both Weights and Connections for Efficient Neural Networks. *NeurIPS*, 2015.

[24] Frank Hartung and Martin Kutter. Multimedia Watermarking Techniques. *Proceedings of the IEEE*, 1999.

[25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *CVPR*, 2016.

[26] Dongseok Im, Donghyeon Han, Sungpill Choi, Sanghoon Kang, and Hoi-Jun Yoo. DT-CNN: Dilated and Transposed Convolution Neural Network Accelerator for Real-time Image Segmentation on Mobile Device. *ISCAS*, 2019.

[27] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *ICML*, 2015.

[28] Hengrui Jia, Christopher A. Choquette-Choo, Varun Chandrasekaran, and Nicolas Papernot. Entangled Watermarks as a Defense against Model Extraction. *USENIX Security*, 2021.

[29] Stefan Katzenbeisser and Fabien Petitcolas. *Information Hiding*. Artech house, 2016.

[30] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. Siamese Neural Networks for One-shot Image Recognition. In *ICML*, 2015.

[31] Torsten Krauß, Jan König, Alexandra Dmitrienko, and Christian Kanzow. Automatic Adversarial Adaption for Stealthy Poisoning Attacks in Federated Learning. *NDSS*, 2024.

[32] Alex Krizhevsky, Geoffrey Hinton, et al. Learning Multiple Layers of Features from Tiny Images. *Citeseer*, 2009.

[33] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

[34] Yue Li, Benedetta Tondi, and Mauro Barni. Spread-Transform Dither Modulation Watermarking of Deep Neural Network. *JISA*, 2021.

[35] Yue Li, Hongxia Wang, and Mauro Barni. A survey of Deep Neural Network watermarking techniques. *Neurocomputing*, 2021.

[36] Zheng Li, Chengyu Hu, Yang Zhang, and Shanqing Guo. How to Prove Your Model Belongs to You: A Blind-Watermark Based Framework to Protect Intellectual Property of DNN. *ACSAC*, 2019.

[37] Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. Fine-Pruning: Defending Against Backdooring Attacks on Deep Neural Networks. *RAID*, 2018.

[38] Xuankai Liu, Fengting Li, Bihan Wen, and Qi Li. Removing Backdoor-Based Watermarks in Neural Networks with Limited Data. *ICPR*, 2021.

[39] Chun-Shien Lu. *Multimedia Security: Steganography and Digital Watermarking Techniques for Protection of Intellectual Property*. Igi Global, 2004.

[40] Erwan Le Merrer, Patrick Pérez, and Gilles Trédan. Adversarial frontier stitching for remote neural network watermarking. *Neural Computing and Applications*, 2019.

[41] Erwan Le Merrer and Gilles Tredan. TamperNN: Efficient Tampering Detection of Deployed Neural Nets. *ISSRE*, 2019.

[42] NVIDIA, Péter Vingelmann, and Frank H.P. Fitzek. Cuda, release: 10.2.89, 2020.

[43] Gang Qu and Miodrag Potkonjak. *Intellectual Property Protection in VLSI Designs: Theory and Practice*. Springer Science & Business Media, 2007.

[44] Olivier Rukundo and Hanqiang Cao. Nearest Neighbor Value Interpolation. *IJACSA*, 2012.

[45] Olivier Rukundo and Bodhaswar T Maharaj. Optimization of Image Interpolation based on Nearest Neighbour Algorithm. *VISAPP*, 2014.

[46] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *ICLR*, 2015.

[47] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *ICLR*, 2015.

[48] Indah Agustien Siradjuddin, Wrida Adi Wardana, and Mochammad Kautsar Sophan. Feature Extraction using Self-Supervised Convolutional Autoencoder for Content based Image Retrieval. *ICICoS*, 2019.

[49] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 2014.

[50] Felix Stahlberg. Neural machine translation: A review and survey. *Journal of Artificial Intelligence Research*, 2020.

[51] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks*, 2012.

[52] Sebastian Szyller, Buse Gul Atli, Samuel Marchal, and N. Asokan. DAWN: Dynamic Adversarial Watermarking of Neural Networks. *MM*, 2021.

[53] Nima Tajbakhsh, Jae Y Shin, Suryakanth R Gurudu, R Todd Hurst, Christopher B Kendall, Michael B Gotway, and Jianming Liang. Convolutional Neural Networks for Medical Image Analysis: Full Training or Fine Tuning? *IEEE TMI*, 2016.

[54] Enzo Tartaglione, Marco Grangetto, Davide Cavagnino, and Marco Botta. Delving in the loss landscape to embed robust watermarks into neural networks. *ICPR*, 2021.

[55] The Linux Foundation. Pytorch, 2022. https://pytorch.org.

[56] Frederick Tung, Srikanth Muralidharan, and Greg Mori. Fine-Pruning: Joint Fine-Tuning and Compression of a Convolutional Network with Bayesian Optimization. *arXiv preprint arXiv:1707.09102*, 2017.

[57] Yusuke Uchida, Yuki Nagai, Shigeyuki Sakazawa, and Shin'ichi Satoh. Embedding Watermarks into Deep Neural Networks. *ICMR*, 2017.

[58] W. van Gils. Two-dimensional dot codes for product identification. *IEEE Transactions on Information Theory*, 1987.

[59] Ron G Van Schyndel, Andrew Z Tirkel, and Charles F Osborne. A Digital Watermark. *IEEE ICIP*, 1994.

[60] Ashish Venugopal, Jakob Uszkoreit, David Talbot, Franz J. Och, and Juri Ganitkevitch. Watermarking the Outputs of Structured Prediction with an Application in Statistical Machine Translation. *EMNLP*, 2011.

[61] Jiangfeng Wang, Hanzhou Wu, Xinpeng Zhang, and Yuwei Yao. Watermarking in Deep Neural Networks via Error Back-propagation. *Electronic Imaging*, 2020.

[62] Tianhao Wang and Florian Kerschbaum. Attacks on Digital Watermarks for Deep Neural Networks. *ICASSP*, 2019.

[63] Tianhao Wang and Florian Kerschbaum. RIGA: Covert and Robust White-Box Watermarking of Deep Neural Networks. *WWW*, 2021.

[64] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE TIP*, 2004.

[65] Chenqi Xie, Ping Yi, Baowen Zhang, and Futai Zou. DeepMark: Embedding Watermarks into Deep Neural Network Using Pruning. *ICTAI*, 2021.

[66] Kang Yang, Run Wang, and Lina Wang. MetaFinger: Fingerprinting the Deep Neural Networks with Meta-training. *IJCAI*, 2022.

[67] Yang Yang, Ke Mu, and Robert H. Deng. Lightweight Privacy-Preserving GAN Framework for Model Training and Image Synthesis. *IEEE TIFS*, 2022.

[68] Afia Zafar, Muhammad Aamir, Nazri Mohd Nawi, Ali Arshad, Saman Riaz, Abdulrahman Alruban, Ashit Kumar Dutta, and Sultan Almotairi. A comparison of pooling methods for convolutional neural networks. *Applied Sciences*, 2022.

[69] Matthew D Zeiler, Dilip Krishnan, Graham W Taylor, and Rob Fergus. Deconvolutional networks. *CVPR*, 2010.

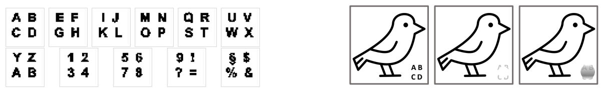
[70] Jialong Zhang, Zhongshu Gu, Jiyong Jang, Hui Wu, Marc Ph. Stoecklin, Heqing Huang, and Ian Molloy. Protecting Intellectual Property of Deep Neural Networks with Watermarking. *ASIACCS*, 2018.

[71] Jie Zhang, Dongdong Chen, Jing Liao, Han Fang, Weiming Zhang, Wenbo Zhou, Hao Cui, and Nenghai Yu. Model Watermarking for Image Processing Networks. *AAAI*, 2020.

Appendix

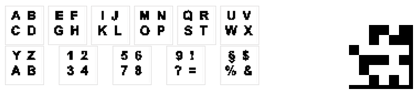
7.1 Additional Visualizations

Partly Removed Watermarks. To understand the necessity of the understandability property proposed in Sect. 3.1, a partly removed watermark is a concrete example. An adversary could remove most of a watermark, e.g., 70%, which would result in a negative watermark verification conducted by a machine. However, the rest of the watermark could potentially be recognizable, if the watermark is visible to human inspection. An example of two partly erased watermarks is depicted in Fig. 10b. The watermark is the ”ABCD” text in



(a) (b)

Figure 10: Visualization of (a) the default eleven watermark secrets and (b) partly erased watermarks. The first image contains the unaltered "ABCD" watermark.



(a) (b)

Figure 11: (a) Visualization of the default eleven watermark secrets. (b) The dot code capacity of the image is 36 bits. The black border lines on the right and bottom are due to the dimensions not being divisible by 6 without remains.

the lower right corner of the first image. The second and the third image are two versions of partly removed watermarks, which still yield enough information to a human, but might not be recognized by a machine.

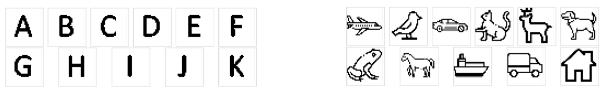
7.2 Additional Experimental Details

Random Key Vectors. The key vectors are randomly chosen between -10 and 10. It is not imperative, that -10 and 10 need to be part of the vector. The key vector from the single watermark experiments reported within this paper consists of the following values: -0.0748, 5.3644, -8.2304, -7.3593, -3.8515, 2.6815, -0.1981, 7.9288, -0.8874, 2.6461.

Watermark Secrets. During the evaluation, we use eleven distinct secrets. The secrets are images with black text on a white background (cf. Fig. 11a). Further, we experimented with different watermark images, which yielded the same experimental outcomes. Thereby, we used only one letter on each image (cf. Fig. 12a) as well as the icons of the ten CIFAR-10 [32] label classes and added an extra icon (cf. Fig. 12b).

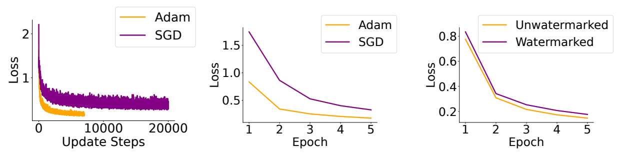
Fidelity. To show ClearStamp's fidelity we provide a plot of the main task loss during training of an unwatermarked and a watermarked model yielding minimal differences in Fig. 13c.

Capacity. We showcase one of the capacity images (cf. Sect. 5.5) which are embedded into a model in Fig. 11b.



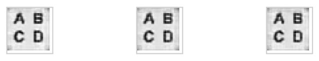
(a) (b)

Figure 12: Visualization of two versions of eleven watermark secrets (images) used as an alternative to the default images.



(a) (b) (c)

Figure 13: (a) SSIM loss applied in transposed training and (b) main task loss during watermark maintaining training. (c) Main task loss during training.



(a) (b) (c)

Figure 14: Visualization of the watermark for different key ranges (a) -5 to 5 (b) -10 to 10, and (c) -50 to 50.

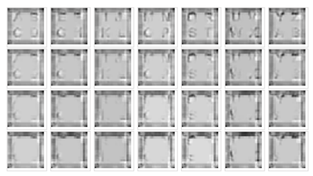
7.3 Additional Experiments

Different Optimizer. To show ClearStamp's optimizer independence, we experimented using SGD with a learning rate of 0.01. We terminated the watermark hardening after 20,000 update steps at an SSIM of 0.89 SSIM. We achieved a main-task accuracy of 83.22% after training while the SSIM remained high at 0.83. This result (cf. Fig. 13) shows, that ClearStamp functions for different optimizers with Adam outperforming SGD in both main-task and watermark embedding.

Different Key Range. In Fig. 14, we provide the results for eleven embedded keys with different key ranges. Thereby, we follow the same experimental setup as in the multi-key paragraph of Sect. 5.1. Especially, we use -5 to 5, -10 to 10, and -50 to 50 as key ranges, where -10 to 10 is already reported in Sect. 5.1. The watermark images in Fig. 14 visualize the watermark after fine-tuning, and yield results that look similar. The experiments yield comparable SSIMs of 0.9272, 0.9305, and 0.9249. Hence, we can claim, that the range parameter is an insensitive value and can be fixed at -10 to 10.

Multi-Keys with 20 Keys. To ensure independence from the number of keys, we expand the experiments to 20 simultaneously embedded keys. The results show that the quality of the watermark does not decrease yielding images similar to Fig. 3g with SSIM. Hence, we can claim, that the performance does not decrease significantly when utilizing more keys.

Adaptive Adversary. An adversary could try to erase an embedded watermark. Thereby, an existing and embedded key could be used in combination with the image that an unwatermarked model yields for that key for a transposed inference. We got similar results when using a completely black image. Then, an adversary adapts ClearStamp's training procedure and tries to erase the watermark. However, the entangled parameters force the adversary to sacrifice main task accuracy. After removing the watermark completely, the


(a)

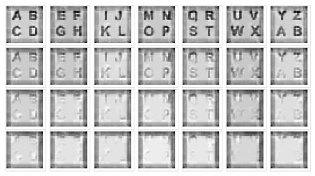

(b)

Figure 15: Adaptive attack with (a) existing / (b) random key.

adversary sacrificed approximately 10% accuracy in our setup compared to the original 89.10%. In Fig. 15a, we show seven out of eleven watermark images after three to six adversarial update steps with respective remaining accuracy values of 85.96%, 83.07%, 79.67%, and 76.32% (SSIMs of 0.16, 0.08, 0.04, and 0.02). For the first two lines, the watermark can be identified. In the third row, the it can be assumed, but it is already hard to discern, while in the fourth line, it is mostly removed. Thereby, one should keep in mind, that an unwatermarked model yields an image similar to Fig. 3c.

When using a random key and an inferred image from an unwatermarked model as a lock, we get similar results. Within the first four update steps, the main task accuracy is reduced from 89.10% to 88.61%, 85.31%, 81.00%, and 74.81% (SSIMs of 0.60, 0.26, 0.11, and 0.04), essentially sacrificing more than 10%. The watermarks can be seen in the four rows of Fig. 15b, showing that the watermark is still clearly visible in the third row. Even in row four, one can see them slightly. The results for a random key combined with a black image or a random image show the same effect.

To validate the independence from our application scenario, we experimented with an already embedded watermark key and an inferred image from an unwatermarked model similar to Fig. 3c for CIFAR-10 [32] trained on ResNet-18 [25]. We could observe the same effect of steadily decreasing main task accuracy with increasing watermark erasure. We report the watermarks after eleven and 22 hardening steps with accuracy drops of 10.12% and 18.62% (SSIMs of 0.38 and 0.24) in Fig. 16, showing that the watermark is still partially embedded while sacrificing lots of main task accuracy.

7.4 Further Considerations

Ownership Claim Automation Ideally, a human evaluator should determine the ownership claim and the presence of a watermark by comparing it to the ground truth. However, in situations where a large volume of images needs validation, automation becomes essential. One potential automated approach involves employing similarity metrics to assess whether a watermark matches the ground truth watermark. However, relying solely on the SSIM metric may not be optimal. The evaluation demonstrates that the watermark remains visually detectable even at low SSIM values.

A more effective method could involve automating the decision-making process through machine learning (ML). This could be achieved by training an ML model to serve as a feature extractor, responsible for generating embeddings from both the watermark image and the ground truth watermark. By comparing these embeddings using a decision layer within the network, the system can make a final determination about the presence of a copyright infringement. This automated approach could ensure accurate and efficient validation, especially when dealing with a large number of images.

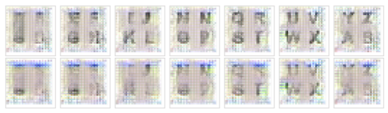


Figure 16: Watermarks after an erasure attack on ResNet-18 [25] trained on CIFAR-10 [32].

Other Constraint Optimization Methods As an alternative to the sequential optimization (cf. Sect. 4.4), one could use a weighted sum method, which adds up the two losses while assigning corresponding weights, essentially introducing an additional hyperparameter. Usually, such weights indicate the task importance. However, this method necessitates extensive hyperparameter tuning if the loss values are at different scales. The loss values then need to be weighted such that both loss terms receive equal importance, since otherwise the smaller one is deemed already sufficiently optimized.

Constraint optimization methods, e.g., Augmented Lagrangian optimization [31] or the Alternating Direction Method of Multipliers [4] are highly effective in enforcing hard constraints. However, it is important to note that our approach, ClearStamp, does not impose strict thresholds for valid SSIM values. Introducing these methods would necessitate adding thresholds and hyperparameters to ClearStamp, which would then need optimization. It's worth emphasizing that a rigid SSIM threshold isn't essential in our context; we aim for high SSIM values around 0.9, but the similarity between an extracted watermark and the secret can already be claimed for very low SSIMs. Implementing such constraint optimization methods would shift the focus toward optimizing the watermarking task to meet the defined threshold. This could potentially detract attention from the primary task and lead to suboptimal performance in both tasks. Moreover, after the constraint is met, there's a risk of diminishing the importance of the watermarking task, potentially hindering the achievement of superior quality.

However, in real-world applications where a hard threshold is desired, we suggest considering [31], as it is a method capable of reliably enforcing inequality constraints.