applied sciences

MDPI

Article

KeyNet: An Asymmetric Key-Style Framework for Watermarking Deep Learning Models

Najeeb Moharram Jebreel *, Josep Domingo-Ferrer, David Sánchez and Alberto Blanco-Justicia

CYBERCAT-Center for Cybersecurity Research of Catalonia, UNESCO Chair in Data Privacy, Department of Computer Engineering and Mathematics, Universitat Rovira i Virgili, Av. Països Catalans 26, 43007 Tarragona, Catalonia, Spain; josep.domingo@urv.cat (J.D.-F.); david.sanchez@urv.cat (D.S.); alberto.blanco@urv.cat (A.B.-J.)

* Correspondence: najeebmoharramsalim.jebreel@urv.cat; Tel.: +34-977558270

Abstract: Many organizations devote significant resources to building high-fidelity deep learning (DL) models. Therefore, they have a great interest in making sure the models they have trained are not appropriated by others. Embedding watermarks (WMs) in DL models is a useful means to protect the intellectual property (IP) of their owners. In this paper, we propose *KeyNet*, a novel watermarking framework that satisfies the main requirements for an effective and robust watermarking. In *KeyNet*, any sample in a WM carrier set can take more than one label based on where the owner signs it. The signature is the hashed value of the owner's information and her model. We leverage multi-task learning (MTL) to learn the original classification task and the watermarking task together. Another model (called the private model) is added to the original one, so that it acts as a private key. The two models are trained together to embed the WM while preserving the accuracy of the original task. To extract a WM from a marked model, we pass the predictions of the marked model on a signed sample to the private model. Then, the private model can provide the position of the signature. We perform an extensive evaluation of *KeyNet*'s performance on the CIFAR10 and FMNIST5 data sets and prove its effectiveness and robustness. Empirical results show that *KeyNet* preserves the utility of the original task and embeds a robust WM.

Keywords: deep learning models; ownership; intellectual property; watermarking; security and privacy; private model

Citation: Jebreel, N.; Domingo-Ferrer, J.; Sánchez, D.; Blanco-Justicia, A. KeyNet: An Asymmetric Key-Style Framework for Watermarking Deep Learning Models. *Appl. Sci.* **2021**, *11*, 999. https://doi.org/10.3390/app11030999

Academic Editor: David Megías
Received: 30 November 2020
Accepted: 17 January 2021
Published: 22 January 2021

1. Introduction

Deep learning (DL) models are used to solve many complex tasks, including computer vision, speech recognition, natural language processing, or stock market analysis [1–3]. However, building representative and highly accurate DL models is a costly endeavor. Model owners, such as technology companies, devote significant computational resources to process vast amounts of proprietary training data, whose collection also implies a significant effort. For example, a conversational model from Google Brain contains 2.6 billion parameters and takes about one month to train on 2048 TPU cores [4]. Besides, designing the architecture of a DL model and choosing its hyperparameters require substantial ML experience and many preliminary tests. Thus, it is not surprising that the owners of DL models seek compensation for the incurred costs by reaping profits from commercial exploitation. They may monetize their models in Machine Learning as a Service (MLaaS) platforms [5] or license them for a financial return to their customers for a specific period of time [6].

Unfortunately, the high value of pretrained DL models is attractive for attackers who would like to steal those models and use them illegally. For example, a user may leak a pretrained model to an unauthorized party or continue to use it after the license period has expired. Furthermore, if a model is offered as MLaaS, many model theft techniques are available to steal it based on its predictions [7,8]. Due to the competitive nature of the

technology market, a stolen or misused model is clearly detrimental to its owner on both economic and competitive terms.

As model theft cannot be prevented in advance, legitimate owners need a robust and reliable way to prove their ownership of DL models in order to protect their intellectual property (IP).

Digital watermarking techniques have been widely used in the past two decades as a means to protect the ownership of multimedia contents like photos, videos and audios [9–12]. The general idea of watermarking is to embed secret information into a data item (without degrading its quality) and then use the embedded secret to claim ownership of the item.

This concept of watermarking can also be extended to DL models. Several authors have proposed to use digital WMs to prove the ownership of models and address IP infringement issues [13–21]. The proposed methods fall into two main classes: (i) *white-box methods*, which directly embed the WM information into the model parameters and then extract it by accessing those parameters, and (ii) *black-box methods*, which embed WMs in the output predictions of DL models. The latter type of methods employ so-called trigger (or carrier) data samples that trigger an unusual prediction behavior: these unusual trigger-label pairs constitute the model watermark and they can be used by the model owner to claim her ownership.

As shown in Table 1, watermarking should fulfill a set of requirements to ensure its effectiveness and robustness [13,15,18,19,22]. Nonetheless, simultaneously satisfying all of these requirements is difficult to achieve [22].

Table 1. Requirements for watermarking of deep learning models.

Requirement	Description
Fidelity	The accuracy of the marked model on the original task shall not be degraded as a result of watermark embedding.
Robustness	The watermark shall be robust against model modifications such as model fine-tuning, model compression or WM overwriting.
Reliability	Watermark extraction shall exhibit a minimal false negative rate to allow legitimate owners to detect the WM with high probability.
Integrity	Watermark extraction shall result in a minimal amount of false positives; unmarked models must not be falsely claimed.
Capacity	It must be possible to include a large amount of watermark information in the target model.
Security	The watermark must not leave a detectable footprint on the marked model; an unauthorized party must not be able to detect the existence of a WM.
Unforgeability	An attacker must not be able to claim ownership of another party's watermark, or to embed additional watermarks into a marked model.
Authentication	A strong link between the owner and her watermark must be provided; reliable verification of the legitimate owner's identity shall be guaranteed.
Generality	The watermarking methodology must be applicable to different DL architectures and data sets.
Efficiency	Embedding and extracting WMs shall not entail a large overhead.
Uniqueness	The watermarking methodology must be able to embed a unique watermark for each user in order to distinguish each distributed marked model individually.
Scalability	Unique watermarking must scale to many users.

Contributions and Plan of This Article

We propose *KeyNet*, a novel watermarking framework that meets a wide range of desirable requirements for effective watermarking. In particular it offers fidelity, robustness, reliability, integrity, capacity, security, authentication, uniqueness, and scalability.

KeyNet depends on three components: the WM carrier set distribution, the signature, and the marked model and its corresponding private model. The private model is trained

together with the original model to decode the WM information from the marked model's predictions. The WM information is only triggered by passing a sample from the WM carrier set signed by the legitimate owner to the corresponding marked model. The predictions of the marked model represent the encoded WM information that can be decoded only by the corresponding private model. The private model takes the predictions as input and decodes the WM information.

Unlike in previous works (discussed in Section 2), a watermarked input can take more than one label, which corresponds to the position of the owner's signature. Besides, the number of WM classes can be greater than the original task classes.

To successfully embed the WM and preserve the original task accuracy, the owner leverages multi-task learning (MTL) to learn both the original and the watermarking tasks together. After that, the owner distributes the marked original model, and keeps the private model secret. The owner uses the private model as a private key to decode the original model's outputs on the WM carrier set.

The main contributions of our work can be summarized as follows:

- *KeyNet* provides a strong link between the owner and her marked model by integrating two reliable authentication methods: a cryptographic hashing algorithm and a verification protocol. Furthermore, the use of a cryptographic hash improves the capacity of embedding WM information. Besides being robust against DL model modifications such as compression and fine-tuning, *KeyNet* does not allow the WM to be overwritten by attackers without losing the accuracy of the original task.
- We demonstrate the ability of our framework to scale and fingerprint different unique copies of a DL model for different users in the system. The information of a user is combined with the owner's information, the carrier is signed with the combined information, and then a unique pair of a pretrained model along with its corresponding private model is fine-tuned before being delivered to the user. After that, the owner can identify the point of leakage with a small number of queries.
- We conduct extensive experiments and evaluate the proposed framework on different DL model architectures. The results show that *KeyNet* can successfully embed WMs with reliable detection accuracy while preserving the accuracy of the original task. Moreover, it yields a small number of false positives when tested with unmarked models.

The remainder of the paper is organized as follows. Section 2 discusses related work. Section 3 describes the attack model to watermarking systems. Section 4 presents our framework in detail. Section 5 describes the experimental setup and reports the results on a variety of data sets. Finally, Section 6 gathers conclusions and proposes several lines of future research.

2. Related Work

The use of digital watermarking techniques has recently been extended from traditional domains such as multimedia contents to deep learning models. Related works can be categorized based on their application scenario as follows.

2.1. White-Box Watermarking

In this scenario, the model internal weights are publicly accessible. In [13], the authors embed an N-bit string WM into specific parameters of a target model via regularization. To this end, they add a regularizer term to the original task loss function that causes a statistical bias on those parameters and use this bias to represent the WM. To project the parameters carrying the WM information, they use an embedding parameter X for WM embedding and verification. Based on the same idea, the authors of [23] use an additional neural network instead of the embedding parameter to project the WM. The additional network is kept secret and serves for WM verification. Other works [24,25] also adopt the same approach for embedding the WM information in the internal weights of DL models.

2.2. Black-Box Watermarking

Assuming direct access to the weights of a DL model to extract the WM is often unrealistic, particularly when someone wishes to extract the WM to claim legitimate ownership of a seemingly stolen model in someone else's power. To overcome this problem, several black-box watermarking methods have been proposed. These methods assume access to the predictions of the model and, thus, embed the WM information into the model's outputs. The idea of these methods is to use some samples and assign each sample a specific label within the original task classes [14,15,17,19,26]. The trigger–label pairs form what is called a trigger set or a carrier set. The carrier set is then used to embed the WM into the target model by training the target model to memorize its trigger–label pairs along with learning the original task. As DL models are overparameterized, it is possible to make them memorize the trigger–label samples through overfitting [27]. Such embedding methods are known as backdooring methods [15]. The triggers are used later to query a remote model and compare its predictions with the corresponding labels. A high proportion of matches between the predictions and the labels is used to prove the ownership of the model.

Trigger set methods can be classified into several types. A first type of methods is based on assigning a random label to each trigger. The trigger samples themselves may be random samples from different distributions [15] or adversarial samples [17]. This approach has many drawbacks. Beyond its limited capacity regarding the number of triggers that can be used for verification, it does not establish a strong link between the owner and her WM. Thus, it is easy for an attacker to insert his WM by using a set of trigger–label pairs, giving them random labels, and then claim ownership of the owner's model. This type of attack is called the ambiguity attack [24].

A second type of methods relies on inserting the WM information into the original data. The inserted information may be a graphical logo [28], the owner's signature [26], a specific text string (which could be the company name) [14], or some pattern of noise [14]. These methods may affect the accuracy of the model in the original task. Besides, the WM may be vulnerable to model fine-tuning aimed at destroying the WM. That is possible because the WM samples will be close to their counterparts in the same class in the feature space. Therefore, fine-tuning may cause the WM pattern to be ignored and those samples to be classified into their original classes again [29].

Another type of black-box methods proposes to exploit the discarded capacity in the intermediate distribution of DL models' output to embed the WM information [21]. They use non-classification images as triggers and assign each trigger a serial number (SN) as label. SN is a vector that contains n decimal units where n is the number of neurons in the output layer. The value of SN serves as an identity bracelet that proves ownership of a marked model. To embed the WM information in the softmax layer predictions, they train the target model to perform two tasks simultaneously: the original task, which is a multi-class classification task, and the watermarking task, which is a regression task. They use the mean square error (MSE) as a loss function for the watermarking task to minimize the difference between the predicted value of a trigger and its corresponding SN. To link the owner with her marked model, they create an endorsement by a certification authority on the generated SNs. Ownership verification is performed by sending some trigger inputs, extracting their corresponding SNs, and having them verified by the authority. This method preserves the original task accuracy and also creates a link between the owner and her WM model. However, it has several drawbacks. The length of the SN depends on the size of the output layer in the model. This may prevent the owner from embedding a large WM. Moreover, by relying on values after the decimal point to express a specific symbol in the SN, if some decimal values are slightly changed, the entire SN will be corrupted. A modification like model fine-tuning would lead to destroying the WM information. In this respect, the authors do not evaluate two important types of modifications that could affect the WM: model fine-tuning and WM overwriting.

A recent paper [30] proposes to watermark DL models that output images. They force the marked model to embed a certain WM (e.g., logo) in any image output by that model.

They train two models together: the marked model and the extractor model. The latter extracts the WM from the output of the former. The marked model is distributed while the extractor is kept secret by the owner. The paper does not evaluate the robustness of the method against the basic attacks that may target the marked model, such as model fine-tuning, model compression, and WM overwriting. Besides, there is a high probability that the WM extractor has memorized the WM in its weights; as a result, when it receives images from models different from the marked one, it might generate the same WM each time.

A shortcoming of most of the aforementioned WM methods is that the WM is the same in all copies of the model [22]. Therefore, if the owner distributes more than one copy of a model, it is impossible for the legitimate owner to determine which of the authorized users has leaked it.

3. Attack Model

To ensure the robustness of a watermarking methodology, it should effectively overcome (at least) three potential attacks:

- **Model fine-tuning**. In this attack, an attacker who has a small amount of the original data retrains the WM model with the aim of removing the WM while preserving the accuracy in the original task.
- **Model compression**. The compression of a DL model's weights minimizes its size and speeds up its performance. Model compression may compromise the WM within the marked model, thereby affecting its detection and extraction.
- **Watermark overwriting**. This type of attack is a major threat to the WM because it might result in the attacker being able to overwrite the owner's WM, or also to embed another WM of his own and thus seize ownership of the WM model. We make the following assumptions about the attacker. First, the attacker is assumed to have a small amount of training data when compared to the owner. Otherwise, he can use his data to train a new model from scratch, or use the predictions of the WM model to create an unmarked copy of it by predictive model-theft techniques [7]. Second, the attacker is assumed to be aware of the methodology used to embed the WM but to be unaware of the carrier set distribution, the owner's signature, or the topology of the owner's private model. An attack is considered to be successful if the attacker manages to overwrite the WM without losing much accuracy in the original task. The goal is to prevent the attacker from overwriting the WM without significantly impairing the original task accuracy, proportional to the amount of data he knows. To make a realistic trade-off, the relationship between the size of the data known by the attacker and the loss in original task accuracy should be inversely proportional: the smaller the attacker data, the greater the accuracy loss is.

4. The *KeyNet* Framework

Instead of having the model memorize the WM through overfitting, in our approach we design the watermarking task as a standalone ML task with its logical context and rules. First, this task performs a one-vs.-all classification so that it can distinguish WMs from original samples with different distributions. Second, the watermarking task learns the features that enable it to identify the spatial information of the legitimate owner's signature. Third, it learns to distinguish the pattern of the owner's signature from the patterns of fake signatures. The purpose of designing the watermarking task in this way is (i) to increase the difficulty of the task so that an attacker with little training data cannot add his WM without losing the accuracy of the original task, (ii) to provide a reliable verification method that strengthens the owner's association with her marked model, (iii) to achieve greater security by keeping the private model in the hands of the owner, (iv) to embed a robust WM without affecting the accuracy of the original task, (v) to produce different unique copies of a DL model for different users of the system based on the same carrier by signing the carrier with the joint signature of the user/owner and (vi) to scale for a large number of users and identify the leakage point with high confidence and little effort.

KeyNet consists of two main phases: watermark embedding and watermark extraction and verification. Figure 1 shows the global workflow of *KeyNet*. The marked DL model is used as a remote service, so that the user can only obtain its final predictions. *KeyNet* passes the final predictions of the remote DL model to its corresponding private model, which uses them to decode the WM information. In the ownership verification protocol we exploit the fact that each sample in the owner's WM carrier set can take different labels based on the position of the owner's signature in it. We next briefly explain the workflow of each phase.

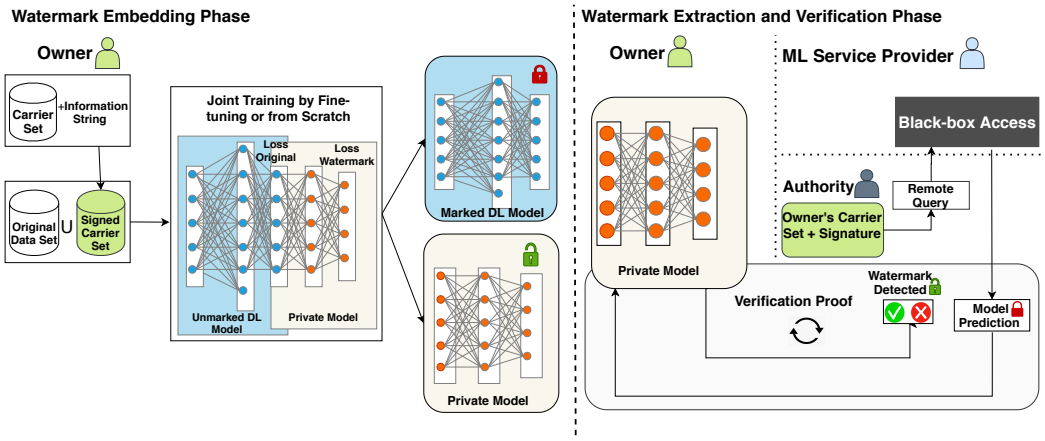


Figure 1. *KeyNet* global workflow.

Watermark embedding. *KeyNet* takes four main inputs in the WM embedding phase: the target model (pretrained or from scratch), the original data set, the owner's WM carrier set, and the owner's information string. The output is the marked model, its corresponding private model, and the owner's signature. The WM carrier set samples are signed using the owner's signature. After that, the signed WM carrier set is combined with the original data set and they are used to fine-tune (or train) the targeted model. The private model takes the final predictions of the original model as inputs and outputs the position of the owner's signature on the WM sample. To embed the WM information and preserve the main task accuracy at the same time, WM embedding leverages multi-task learning (MTL) to train the two models jointly.

MTL is an ML approach that allows learning multiple tasks in parallel by sharing the feature representation among them [31,32]. Many MTL methods [33–35] show that different tasks can share several early layers, and then have task-specific parameters in the later layers. MTL also helps the involved tasks to generalize better by adding a small amount of noise that helps them reduce overfitting [36,37].

In our framework, the original model parameters are shared among the original task and the watermarking task. When the marked model receives unmarked data samples, its predictions represent the classification decision on those samples. However, when it receives a watermarked sample, its output represents the features that the private model needs to distinguish the signature position on that sample. For this to be possible, the private model forces the shared layer (the original model parameters) to produce a different representation of the WM samples. We can see the private model as a private key held only by the owner that decodes the WM information from the original model predictions. More details about this phase are given in Sections 4.2 and 4.3.

Watermark extraction and verification. The owner can extract the WM information from a suspicious remote DL by taking a random sample from her WM carrier set, putting her signature on one of the predefined positions, and querying the remote model. After that, he/she passes the remote model's predictions to the private model. If the private model decodes the WM information and provides the position of the signature with high accuracy, then the owner can claim her ownership.

To verify the ownership of a remote black-box DL model, the owner first delivers the WM carrier set and her signature to the authority. He/she also tells the authority about the methodology used to sign the WM samples along with the predefined positions where the WM may be placed. The authority (i.e., the *verifier*) randomly chooses a sample from the carrier set, puts the signature in a random position, queries the remote DL model, and sends the model's predictions to the owner. The owner (i.e., the *prover*) takes the predictions, passes them to her private model, and tells the authority the position of her signature on the image. The authority repeats the proof as many times as he/she desires. After that, the owner's answer accuracy is evaluated according to a minimum threshold. If the owner surpasses the threshold, his/her ownership is regarded as proven by the authority. More details about this phase are given in Section 4.4.

The following subsections describe each phase in detail. First, we formalize the problem. Then, we describe the methodology for signing and labeling the WM carrier set using the hashed value of the owner's information. After that, we describe the WM embedding phase by training the original model and the private model on the original and the watermarking tasks jointly. Finally, we explain the WM extraction and verification phase.

4.1. Problem Formulation

The key idea of our framework is to perform two tasks at the same time: the original classification task T_{org} and the watermarking task T_{wm}. To do so, *KeyNet* leverages the multi-task learning (MTL) approach to achieve high accuracy in both tasks by sharing the parameters of the original model between the two tasks. *KeyNet* adds a private model to the original model. The original model's objective is to correctly classify the original data samples into their corresponding labels, while the private model's objective is to correctly predict the position of the owner's signature in a sample of the WM carrier set using the original model predictions. We can formally represent as follows the problem being tackled:

- **Representation of the original, private, and combined models.** Let $D_{org} = \{(x_i, y_i)\}_{i=1}^{n}$ be the original task data and $D_{wm} = \{c_j\}_{j=1}^{m}$ be the WM carrier set data. Let h be the function of the original model and f be the function of the private model. Let θ_1 the parameters of h and θ_2 be the parameters of f. Let *signature* be the owner's signature and $PL = \{(p_k, l_k)\}_{k=1}^{z}$ be the set of predefined position–label pairs (e.g., position: top left, label: 1 and position: bottom right, label: 4) where z is the total number of positions at which the signature can be located on a WM carrier set sample. Let *sign* be the function that puts a *signature* on a carrier set sample c and returns the signed sample c^{p_k} and its corresponding label l_k as:

  $$(c^{p_k}, l_k) = sign(signature, c, p_k).$$

  Let D_{wm}^{signed} be the signed carrier set samples that contain all the (c^p, l) pairs. We use D_{org} and D_{wm}^{signed} to train both h and f to perform T_{org} and T_{wm}.
  Typically, the function h tries to map each $x_i \in D_{org}$ to its corresponding y_i, that is, $h(x_i) = y_i$.
  Let $f(h)$ be the composite function that aims at mapping each $c_j^{p_k}$ to its corresponding l_k, that is, $f(h(c^{p_k})) = l_k$.
- **Embedding phase.** We formulate the embedding phase as an MTL problem where we jointly learn two tasks that share some parameters in the early layers and then have task-specific parameters in the later layers. The shared parameters in our case are θ_1, while θ_2 are the WM task-specific parameters. We compute the weighted combined loss L as

  $$L = \alpha Loss(h(x), y) + (1 - \alpha) Loss(f(h(c^p)), l),$$

  where $h(x)$ represents the predictions on the original task samples, $f(h(c^p)$ represents the predictions on D_{wm}^{signed} samples, $Loss(h(x), y))$ is the loss function that penalizes

the difference between the original model outputs $h(x)$ and the original data targets y, $Loss(f(h(c^p)), l)$ is the loss function that penalizes the difference between the composite model outputs $f(h(c^p))$ and the signed WM carrier set's target l, and α is the combined weighted loss parameter. Then, we seek θ_1 and θ_2 that make L small enough to get acceptable accuracy on both T_{org} and T_{wm}. Once this is done, the WM has been successfully embedded while preserving the accuracy of the original task T_{org}.

- **Verification phase.** The verification function V checks whether a claimer (also known as the *prover*), who has delivered her *signature* and WM carrier set D_{wm} to the authority (also known as the *verifier*), is the legitimate owner of a remote model $h^{'}$. If the prover is the legitimate owner of $h^{'}$, he/she will be able to pass the verification process and thus prove her ownership of $h^{'}$. That is, because he/she possesses the private model f, which was trained to decode $h^{'}$ predictions on her signed D_{wm}.
  Here, r represents the number of the required rounds in the verification process and T denotes the threshold needed to prove the ownership of $h^{'}$. Note that the authority also knows the signing function $sign$ used to sign D_{wm} samples in order to obtain (c^{p_k}, l_k) pairs.
  The function V can be expressed as $V(\{(f(h^{'}(c^{p_k})), l_k, p_k)\}_{k=1}^{r}, T) = \{True, False\}$.

4.2. Watermark Carrier Set Signing and Labeling

The methodology we use for labeling the WM carrier set is key in our approach. In contrast to related works, which assign a unique label to each of the WM carrier set samples, our labeling method allows for a single sample to carry more than one label. More precisely, any sample $c \in D_{wm}$ can take one of z labels $\{l_k\}_{k=1}^{z}$, where z is the number of predefined positions $\{p_k\}_{k=1}^{z}$ at which the *signature* of the owner can be placed. Besides the z positions, if a sample is not signed by the legitimate owner, it uses label 0 by default. Moreover, if the sample is not in D_{wm}, it uses label 0 by default even if it is signed by the owner. Algorithm 1 formalizes the method used to sign and label the D_{wm} samples.

First and foremost, the owner's information and the metadata of her model are endorsed by the authority. This information is a string of arbitrary length. After that, Algorithm 1 returns the signed D_{wm}^{signed} WM carrier set consisting of pairs (signed D_{wm} sample, label), a signed D_{dif}^{signed} consisting of pairs (signed sample from a set D_{dif} of different distribution, 0), and the owner's signature *signature* used to sign the samples. The inputs to Algorithm 1 are:

1. The owner's information string $infStr$ that has been endorsed by the authority.
2. The size of the signature s to be placed on the WM carrier set samples.
3. The owner's WM carrier set D_{wm}.
4. The set of position–label pairs $PL = \{p_k, l_k\}_{k=1}^{z}$ that defines the signature positions and their corresponding labels.
5. A small set of samples D_{dif} from other distributions than D_{wm}.

Algorithm 1 starts its work by taking the hash value for $infStr$ and then converting it to a squared array of size s, as follows. In our implementation we use *SHA256*, which yields 256 bits that are converted to 64 characters by digesting them to hexadecimal. The last s (with $s \leq 64$) among the digested characters are converted to a decimal vector of length s. The decimal vector values are normalized between 0 and 1 by dividing them by the maximum value in the vector. The normalized vector is then reshaped into a squared array. The resulting array represents the owner's *signature*. Note that it is also possible to use any hashing function different from SHA256.

Algorithm 1 Signing a WM carrier Set

Input: Owner's information $infStr$, owner's signature size s, owner's WM carrier set D_{wm}, signature positions/labels set PL, other distributions' samples D_{dif}

Output: signed labeled WM samples D_{wm}^{signed}, signed samples from different distributions D_{dif}^{signed}, owner's signature $signature$

$signature \leftarrow hashAndReshape(infStr, s)$ //`The owner signature.`
$fakeStr \leftarrow modify(infStr)$ //`Fake information string.`
$signature_{fake} \leftarrow hashAndReshape(fakeStr, s), signature_{fake} \neq signature$//`A fake signature.`
$D_{wm}^{signed}, D_{dif}^{signed} = [\,], [\,]$
for each sample c in D_{wm} **do**
  for each position, label (p_k, l_k) in PL **do**
    $c^{p_k} \leftarrow sign(signature, c, p_k)$
    $Add((c^{p_k}, l_k), D_{wm}^{signed})$
  end for
  $p_k \leftarrow selectRandomPosition()$
  $c^{p_k} \leftarrow sign(signature_{fake}, c, p_k)$
  $Add((c^{p_k}, 0), D_{wm}^{signed})$
  $Add((c, 0), D_{wm}^{signed})$
  $fakeStr \leftarrow modify(infStr)$ //`New fake information string.`
  $signature_{fake} \leftarrow hashAndReshape(fakeStr, s), signature_{fake} \neq signature$
end for
for each sample d in D_{dif} **do**
  for each position, label (p_k, l_k) in PL **do**
    $d^{p_k} \leftarrow sign(signature, d, p_k)$
    $Add((d^{p_k}, 0), D_{dif}^{signed})$
  end for
end for
return $D_{wm}^{signed}, D_{dif}^{signed}, signature$

Once we obtain *signature*, we start the labeling step of the WM carrier set D_{wm}. For each $c \in D_{wm}$, we replicate c for z times where z is the total number of possible positions $\{p_k\}_{k=1}^{z}$ of the signature. Then, we use the *sign* function to place *signature* in the position p_k to obtain the (c^{p_k}, l_k) pair.

We also leave one copy of each sample without signing and assign it the label 0. That is, if a carrier set sample c is not signed with *signature*, it will be represented by the $(c, 0)$ pair.

We then do two steps:

1. We generate a fake information string $fakeStr$ by making a slight modification to $infStr$. Then, we generate $signature_{fake}$ following steps similar to those above followed to generate the real owner's signature *signature*. After that, we sign the D_{wm} samples each with a different fake signature and a randomly chosen position p_k and, instead of assigning to the signed sample the corresponding label l_k, we assign it the label 0. To obtain a new fake signature, we again make a slight random modification in $infStr$ and generate the fake signature in the same way as above.
2. We take samples from other distributions D_{dif} and sign them with the real owner's signature as we did with the D_{wm} samples. We assign them the label 0. We use samples from different distributions to avoid triggering the WM just with the owner's real signature. In other words, we make the triggering of the WM from a marked model dependent on the carrier set distribution in addition to the pattern of the owner's signature.

The goal of the above two steps is to make the marked model h^* output the information that tells the position of a signature only if we pass to it a sample that *belongs to* D_{wm} *and is*

signed with the real signature. Otherwise, h^* ignores the presence of any different signature from *signature* on D_{wm} samples. This also avoids h^* responding to samples from different distributions than the D_{wm} distribution.

Finally, Algorithm 1 returns the signed WM samples D_{wm}^{signed}, the signed samples from different distributions D_{dif}^{signed}, and the signature *signature* that will be used to trigger the marked model h^*. Note that this process is performed only once, before the WM is embedded.

4.3. Watermark Embedding

To successfully embed the WM in the original model h without compromising the accuracy of the original task, we jointly train both h and the private models f simultaneously. As a large amount of the carrier set samples have been signed, we first randomly select one-fifth of D_{wm}^{signed} for training. The random selection allows for the representation of all the possible states while reducing the carrier set size. The signed samples from other distributions D_{dif}^{signed} are combined with those randomly selected and assigned to D_2. In the end, we add D_2 to the original task data D_{org}. The resulting combined data set $D = D_{org} \cup D_2$ are used in the training step as specified next.

During joint training, a batch b is taken from D. Then, b is separated into two sub-batches: $\{x, y\} \in D_{org}, \{c, l\} \in D_2$. $\{x\}$ is passed to the original model h and the loss $L_f(h(x), y)$ is calculated. On the other hand, $\{c\}$ is first passed to h, and then the predictions of the original model $h(c)$ are passed to the private model f; the loss $L_f(f(h(c)), l)$ is afterwards calculated. As we deal with two classification tasks, the cross-entropy loss function is used to calculate the loss for both tasks.

We use the parameter α to balance the weight of $L_f(h(x), y)$ and $L_f(f(h(c)), l)$ before we add them up in the joint loss L. Parameter α allows us to choose the best combination of the weighted loss that preserves the accuracy of T_{org} while embedding WM successfully. Then, the parameters of h, f are optimized to minimize L.

Reducing $L_f(h(x), y)$ forces h to predict the correct class for x, while reducing the watermarking task loss $L_f(f(h(c)), l)$ forces the private model to distinguish the distribution of the WM carrier set D_{wm}, and predict the location of the owner's signature in its samples. The original model, in addition to performing the original task, also executes the first part of the watermarking task by outputting the features needed to find the position of the signature. By using these features as input, the private model performs the second part of the watermarking task, which consists in identifying the signature position.

Regarding the architecture of the private model f, the number of inputs corresponds to the size of h predictions, whereas the number of outputs corresponds to the number of classes of the WM task $z + 1$. We also add at least one hidden layer in between. The hidden layer enriches the information coming from the original model before passing it to the output layer of the private model.

Algorithm 2 summarizes the process of embedding the WM. It takes an unmarked model h that might be pretrained or be trained from scratch, the private model f, the original data set D_{org}, the signed WM carrier set D_{wm}^{signed}, signed samples from other distributions D_{dif}^{signed}, and the joint loss balancing parameter α. The output of the embedding phase is a marked model h^* along with its corresponding private model f.

Algorithm 2 Watermark Embedding

Input: Unmarked DL model h, private model f, original data set D_{org}, signed WM carrier set D_{wm}^{signed}, signed samples from other distributions D_{dif}^{signed}, batch size BS, weighted loss parameter α.

Output: Marked model h^*, corresponding private model f.

$s = size(D_{wm}^{signed})/5$

$D_2 \leftarrow randomSample(D_2, s)$

$D_2 \leftarrow D_2 \cup D_{dif}^{signed}$

$D \leftarrow D_{org} \cup D_2$

$L_f = crossEntropy()$ `//Loss function`

for each batch b of size BS in D **do**

  $\{x, y\}, \{c, l\} \leftarrow split(b)$, with $\{x, y\} \in D_{org}$ and $\{c, l\} \in D_2$

  $L \leftarrow \alpha L_f(h(x), y) + (1 - \alpha) L_f(f(h(c)), l)$

  $optimize(L)$

end for

return h^*, f

4.4. Watermark Extraction and Verification

The verification process of ownership involves a would-be owner in the role of prover and the authority in the role of verifier. The would-be owner claims that a remote model $h^{'}$ is part of his/her IP. The authority is given the WM carrier set D_{wm}, the would-be owner's *signature*, the signing function *sign*, and remote access to $h^{'}$. The authority sets an accuracy threshold T and a number of required verification rounds r to decide whether $h^{'}$ is the IP of the would-be owner. In each round, the authority randomly selects a sample c from D_{wm}, signs it using *signature* in a random position p_k, and sends the signed sample c^{p_k} to the remote model $h^{'}$. The predictions $h^{'}(c^{p_k})$ (which contain the encoded WM information) are forwarded to the would-be owner. The latter passes them to her private model f to obtain $l_k = f(h^{'}(c^{p_k})$. As the relationship between positions and labels is one-to-one, the would-be owner can use l_k to tell the authority the position p_k of her signature in c. After r rounds, the accuracy acc of the would-be owner at detecting the positions is the number of correct answers divided by r. If $acc \geq T$, then authority certifies that $h^{'}$ is owned by the would-be owner.

Note that the authority can also send the samples without signing them or sign them using fake signatures different from *signature*. In this case, the would-be owner should tell the authority that this sample does not contain her signature. That is possible because in these cases the private model gives them the label 0. Algorithm 3 formalizes the verification process.

Algorithm 3 Watermark Verification

Input: Remote access to $h^{'}$, threshold T, number of rounds r

Output: Boolean decision d (True or False) on $h^{'}$'s ownership.

$correct = 0$
$d = False$ //`Decision on the ownership of` $h^{'}$.
for each round $i = 1, 2, \ldots r$ **do**
    $c \leftarrow randomSample(D_{wm})$
    $p_k \leftarrow$ randomPosition(), k $\in 1, 2, \ldots z$
    $c^{p_k} \leftarrow sign(signature, c, p^k)$
    $predictions \leftarrow h^{'}(c^{p_k})$
    $l_k \leftarrow f(predictions)$
    $answer \leftarrow$ Position corresponding to l_k
    if $answer = p_k$ **then**
        $correct \leftarrow correct + 1$
    end if
end for

$acc = correct/r$
if $acc \geq T$ **then**
    $d \leftarrow True$
end if
    return d

5. Experimental Results

In this section, we evaluate the performance of *KeyNet* on two image classification data sets and with two different DL model architectures. First, we present the experimental setup. After that, we evaluate the proposed framework performance against the requirements stated in Table 1. We focus on robustness, authentication, scalability, capacity, integrity, and fidelity, but, as our framework partly fulfills the rest of requirements, we also assess its performance on each of them.

The code and the models used in this section are available at https://github.com/NajeebJebreel/KeyNet.

5.1. Experimental Setup

Original task data sets and DL models. We used two image classification data sets: CIFAR10 [38] and FMNIST5. CIFAR10 has 10 classes, while FMNIST5 is a subset of the public data set Fashion-MNIST [39]; FMNIST5 contains the samples that belong to the first five classes in Fashion-MNIST (classes from 0 to 4). Table 2 summarizes the original task data sets, the carrier set, and the DL models and their corresponding private models.

Table 2. Data sets and deep learning model architectures. $C(3, 32, 5, 1, 2)$ denotes a convolutional layer with 3 input channels, 32 output channels, a kernel of size 5×5, a stride of 1, and a padding of 2, $MP(2, 1)$ denotes a max-pooling layer with a kernel of size 2×2 and a stride of 1, and $FC(10, 20)$ indicates a fully connected layer with 10 inputs and 20 output neurons. We used *ReLU* as an activation function in the hidden layers. We used *LogSoftmax* as an activation function in the output layers for all DL models. The rightmost column contains the architecture of the corresponding private models.

Data Set	WM Carrier Set	DL Model	DL Model Architecture	Private Model Architecture
CIFAR10	STL10	ResNet18 VGG16	See [40]. See [41].	FC(10,20), FC(20,10), FC(10, 6) (496 learnable parameters)
FMNIST5	MNIST	CNN LeNet	C(3,32,5,1,2), MP(2,1), C(32,64,3,1,2), MP(2,1), FC(4096,4096), FC(4096,5) See [42].	FC(5, 10), FC(10,20), FC(20,6) (411 learnable parameters)

Watermark carrier sets. We employed three different data sets as WM carrier sets: STL10 [43], MNIST [44], and Fashion-MNIST (the latter was used only in attacks). We applied Algorithm 1 to label the carrier set's images. Then, we passed the carrier set, the owner's information, the signature size, a fake signature, some samples from different distributions, and a list containing the labeling order of the positions of the owner's signature in the carrier set. We used the following labeling order: 1: Top left, 2: Top right, 3: Bottom left, 4: Bottom right, and 5: Image center. Algorithm 1 assigns label 0 to an image if (i) the image belongs to the carrier set but does not carry any signature, (ii) the image belongs to the carrier set but carries a signature different from the owner's signature, or (iii) the image does not belong to the carrier set distribution (even if it is signed with the owner's real signature). For WM accuracy evaluation, we randomly sampled 15% of the WM carrier set. After that, we signed them in different random positions and assigned them the corresponding labels. Figure 2 shows some examples of signed carrier set images and their corresponding labels.

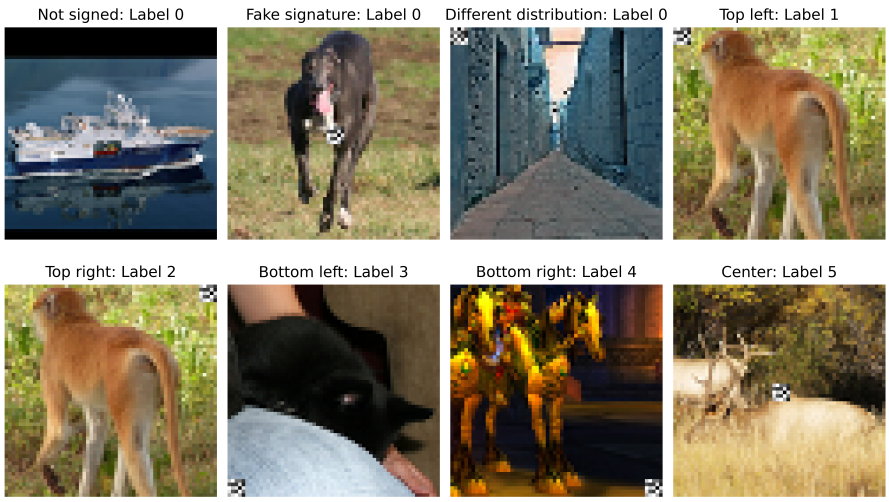


Figure 2. Examples of signed STL10 carrier set images employed with the CIFAR10 data set. Each image shows the signature position and its corresponding label.

Attacker configurations. We assumed the attacker has varying percentages of the training data, ranging from 1% to 30% of the original training data. The attacker's training data were randomly sampled from the original training data. We also assumed that the WM carrier set distribution is a secret between the owner and the authority, so we assigned the attacker different WM carrier sets from those of the owner. The attacker's private model was slightly different as well, because the owner's private model and its architecture are secret. The rest of the attacker's configurations and hyper-parameters were the same as the owner's. Table 3 summarizes the attacker's WM carrier sets and private model architectures.

Table 3. Attacker's WM carrier sets and private models. The attacker's WM carrier set and private model differ from the owner's.

Data Set	Owner's WM Carrier Set	Attacker's WM Carrier Set	Attacker's Private Model Architecture
CIFAR10	STL10	Fashion-MNIST	FC(10,20),FC(20,30), FC(30, 6) (980 learnable parameters)
FMNIST5	MNIST	STL10	FC(5, 20), FC(20,10), FC(10,6) (360 learnable parameters)

Performance metric. We used *accuracy* as performance metric to evaluate all the original and WM tasks. *Accuracy* is the number of correct predictions divided by the total number of predictions.

Training hyperparameters. We used the cross-entropy loss function and the stochastic gradient descent (SGD) optimizer with learning rate = 0.001, momentum = 0.9, weight decay = 0.0005, and batch size = 128. We trained all the original unmarked models for 250 epochs. To embed the WM from scratch, we trained the combined model for 250 epochs. To embed the WM in a pretrained model, we combined the private model and fine-tuned the combination for 30 epochs.

To jointly train the original and private models, we used parameter α to weight the original task loss and the WM task loss before optimization. For CIFAR10 we used $\alpha = 0.9$ when embedding the WM in a pretrained model, while we used $\alpha = 0.95$ when embedding the WM into a DL model from scratch. For FMNIST5 we used $\alpha = 0.85$ to embed the WM in a pretrained model, while we used $\alpha = 0.9$ to embed the WM from scratch. The experiments were implemented using Pytorch 1.6 and Python 3.6.

5.2. Experiments and Results

First, we made sure that an accurate private model could not be obtained (and therefore the ownership of a DL model could not be claimed) by using only the predictions of a black-box DL model. To do so, we queried different unmarked DL models with all the signed samples in the owner's WM carrier set, and we used the predictions as input features to train the private model. Table 4 shows the performance of the private models obtained in this way after 250 epochs.

Table 4. Accuracy of the private models at detecting the position of the owner's signature in the WM carrier set when trained for 250 epochs with the predictions of black-box models.

Data Set	Black-Box DL Model	Watermark Detection Accuracy %
CIFAR10	ResNet18	31.25
	VGG16	30.14
FMNIST5	CNN	34.42
	LeNet	33.26

It can be seen that the average accuracy of the private model at detecting the signature position inside the WM carrier set is as low as 32.27%. This accuracy was obtained by granting unconditional query access to the black-box model and by using its predictions as input to train the private model. Based on that, we decided to set threshold $T = 0.9$, which is nearly three times greater than the above average accuracy. Therefore, to prove her ownership of a black-box DL model, the owner's private model must detect the signature positions in the WM carrier set with an accuracy greater than or equal to 90%.

In the following, we report the results of *KeyNet* on several experiments that test its fulfillment of the requirements depicted in Table 1.

Table 5. Fidelity results. Column 3 shows the accuracy of the unmarked models in the original tasks (baseline accuracy) before embedding the WM. Columns 4 and 5 show the accuracy of the marked model in the original task after embedding WM by fine-tuning a pretrained model or by training the combined model from scratch. Columns 6 and 7 show the accuracy of the private model in detecting the WM using the predictions of the corresponding marked model. To embed the WM in a pretrained model, we fine-tuned it for 30 epochs while we trained models from scratch for 250 epochs.

Data Set	DL Model	Unmarked Model Accuracy %	Marked Model Accuracy %		Watermark Detection Accuracy %	
			By Fine-Tuning (30 Epochs)	From Scratch (250 Epochs)	By Fine-Tuning	From Scratch
CIFAR10	ResNet18	91.96	92.07	92.53	99.96	99.97
	VGG16	90.59	90.52	91.74	99.68	99.89
FMNIST5	CNN	92.08	92.42	92.32	99.98	99.90
	LeNet	90.68	89.94	89.94	99.55	99.79

Fidelity. Embedding the WM should not decrease the accuracy of the marked model on the original task. As shown in Table 5, the marked model's accuracy is very similar to that of the unmarked model. This is thanks to the joint training, which simultaneously minimizes the loss for the original task and the WM task. Furthermore, *KeyNet* did not only preserve the accuracy in the original task, but sometimes it even led to improved accuracy. That is not surprising, because the watermarking task added a small amount of noise to the marked model, and this helped reduce overfitting and thus generalize better.
KeyNet therefore fulfills the fidelity requirement by reconciling accuracy preservation for the original task and successfully embedding of the WM in the target models.

Reliability and robustness. *KeyNet* guarantees a robust DL watermarking and allows legitimate owners to prove their ownership with accuracy greater that the required threshold $T = 90\%$. Table 5 shows that WM detection accuracy was almost 100%, and thus our framework was able to reliably detect the WM.

We assess the **robustness** of our framework against three types of attacks: *fine-tuning* [45], *model compression* [46,47], and *WM overwriting* [13,48]:

- *Model fine-tuning.* Fine-tuning involves retraining a DL model with some amount of training data. It may remove or corrupt the WM information from a marked model because it causes the model to converge to another local minimum. In our experiments, we sampled 30% of the original data and used them to fine-tune the marked model by optimizing its parameters based only the loss of the original task. Table 6 outlines the impact of fine-tuning on the WM detection accuracy with all benchmarks. We can notice that *KeyNet* is robust against fine-tuning and was able to preserve a WM detection accuracy of about 97% after 200 epochs. The explanation for this strong persistence against fine-tuning is that *KeyNet* does not embed the WM information within the decision boundaries of the original task classes. Therefore, the effect of fine-tuning on the WM is very small.
- *Model compression.* We used the compression approach proposed in [47] to prune the weight parameters in the marked DL models. To prune a particular layer, we first sorted the weights in the specified layer by their magnitudes. Then, we masked to zero the smallest magnitude weights until the desired pruning level was reached. Figure 3 shows the impact of model compression on both WM detection accuracy and original task accuracy with different pruning rates. We see that *KeyNet* is robust against model compression, and the accuracy of the WM remains above the threshold $T = 90\%$ as long as the marked model is still useful for the original task. This is consistent because when the marked model becomes useless due to excessive compression, the owner will not be interested in claiming its ownership.
- *Watermark overwriting.* Assuming that the attacker is aware of the methodology used to embed the WM, he may embed a new WM that may damage the original one. In our experiments, we assumed that the attacker knows the methodology but knows neither the owner's carrier set nor the owner's private model architecture. We studied the effect on the WM of the attacker's knowing various fractions of the original training

data, ranging from 1% to 30%. We chose the lower bound 1% based on the work in [49]; the authors of that paper demonstrate that an attacker with less than 1% of the original data is able to remove the watermark with a slight loss in the accuracy of the original task.

Table 6. Fine-tuning results. In the fine-tuning attack, the marked models were retrained based on the original task loss only.

Data Set	**CIFAR10**						**FMNIST5**					
DL model	ResNet18			VGG16			CNN			LeNet		
Number of epochs	50	100	200	50	100	200	50	100	200	50	100	200
Marked model Accuracy %	92.40	92.33	92.47	91.31	91.64	91.69	92.30	92.52	92.40	89.84	90.06	90.12
Watermark detection Accuracy %	98.19	98.05	99.12	97.20	94.72	96.67	97.35	97.02	96.92	98.23	97.42	96.4

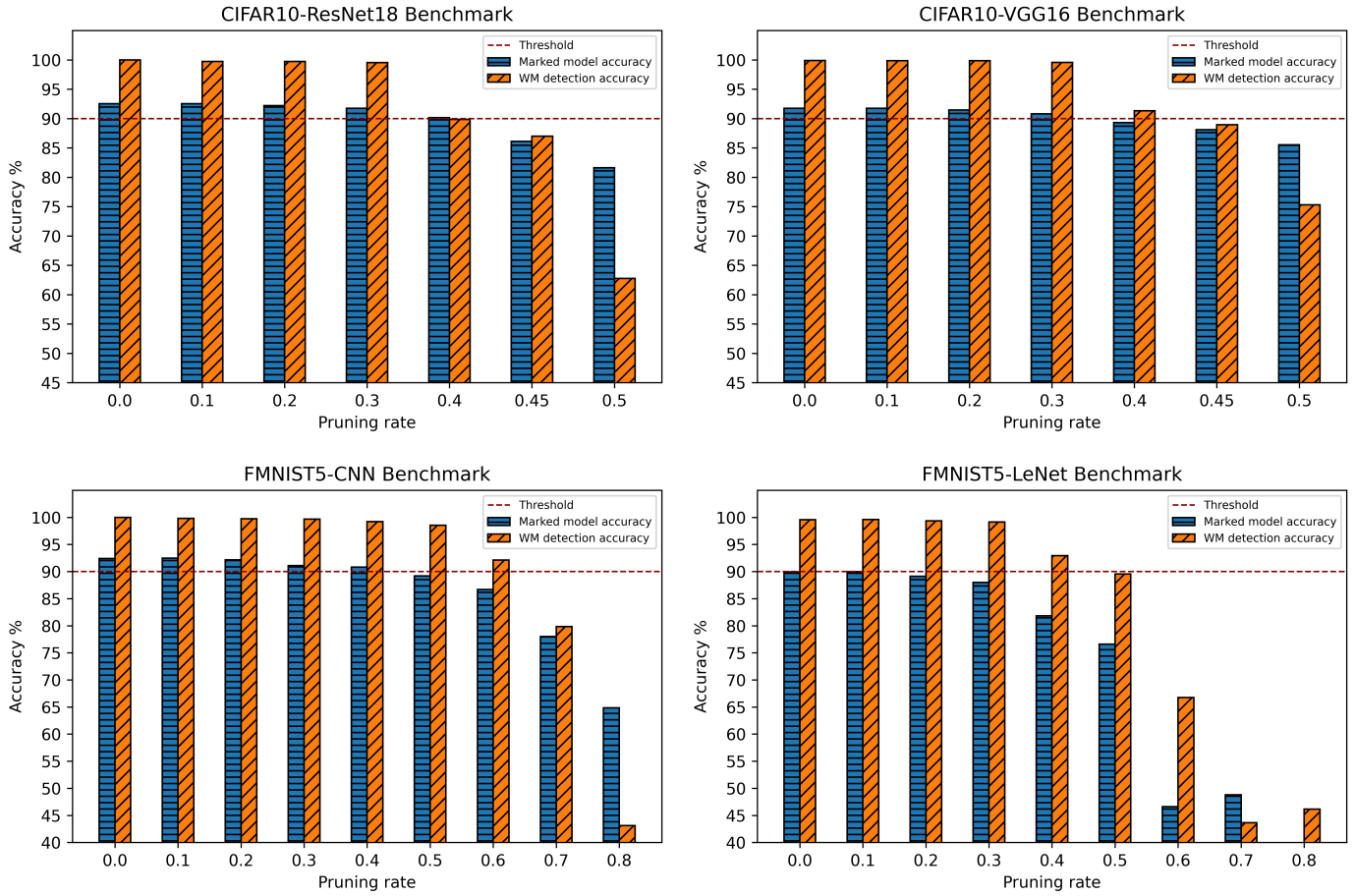


Figure 3. Robustness against model compression. The X-axis indicates the pruning levels we used for each marked model. The blue bars indicate the marked model accuracy in the original task, while the orange bars indicate the accuracy of WM detection. The horizontal dotted line indicates the threshold $T = 90\%$ used to verify the ownership of the model.

To overwrite the WM, the attacker selected his/her own carrier set and signed it using Algorithm 1 with his/her signature. Then, he/she trained her private model along with the marked model in the same way as in Algorithm 2. Tables 7 and 8 summarize the results of WM overwriting experiments. The attacker was able to successfully overwrite the original WM and successfully embed her new WM, but this was done at the cost of a substantial accuracy loss in the original task when using a fraction of the training data up to 10%. Thus,

our watermark easily survives the attacks in the conditions described in [49]. For fractions above 10%, the accuracy of the marked model became competitive, but an attacker holding such a large amount of training data can easily train her own model and has no need to pirate the owner's model [18].

Table 7. Overwriting attack results with CIFAR10 marked models. The table shows the accuracy before and after overwriting each marked model and its corresponding private model depending on the fraction of training data known by the attacker (from 1% to 30%).

Dataset	**CIFAR10**											
DL Model	**ResNet18**						**VGG16**					
Data fraction %	1	3	6	10	20	30	1	3	6	10	20	30
Marked model Accuracy before %	92.53	92.53	92.53	92.53	92.53	92.53	91.74	91.74	91.74	91.74	91.74	91.74
Marked model Accuracy after %	39.05	63.75	83.31	86.35	89.91	90.9	34.61	71.86	81.2	83.9	88.07	89.67
Owner's WM detection Accuracy after %	24.53	20.35	22.27	28.3	37.33	41.15	32.32	30.88	28.61	32.48	30.32	44.4
Attacker's WM detection Accuracy %	99.97	99.89	99.95	99.9	99.94	99.97	99.69	99.68	99.89	99.87	99.9	99.96

Table 8. Overwriting attack results with FMNIST5 marked models. The table shows the accuracy before and after overwriting each marked model and its corresponding private model depending on the fraction of training data known by the attacker (from 1% to 30%).

Dataset	**FMNIST5**											
DL Model	**CNN**						**LeNet**					
Data fraction %	1	3	6	10	20	30	1	3	6	10	20	30
Marked model Accuracy before %	92.32	92.32	92.32	92.32	92.32	92.32	89.94	89.94	89.94	89.94	89.94	89.94
Marked model Accuracy after %	76.18	81.88	87.86	89.1	91.38	91.84	74.64	79.44	86.36	88.86	89.58	89.52
Owner's WM detection Accuracy after %	26.97	28.53	38.52	46.83	70	68.03	26.32	18.71	43.99	57.4	75.83	65.66
Attacker's WM detection Accuracy %	100	100	100	100	100	100	99.25	99.65	99.84	99.92	99.89	99.92

Integrity. *KeyNet* meets the integrity requirement by yielding low WM accuracy detection with unmarked models, and thus it does not falsely claim ownership of models owned by a third party. In our experiments, there were 6 classes for the watermarking task. Looking at Table 9, the accuracy of falsely claimed ownership of unmarked models is not far from guessing 1 out of 6 numbers randomly, which equals approximately 16.6%.

Authentication. *KeyNet* fulfills the authentication requirement by design. Using a cryptographic hash function such as *SHA256* to generate the owners' signatures establishes a strong link between owners and their WMs. Furthermore, the verification protocol of *KeyNet* provides strong evidence of ownership. When the authority uses a fake signature, the marked model does not respond. This dual authentication method provides unquestionable confidence in the identity of the legitimate owner.

Security. As *KeyNet* embeds the WM in the dynamic content of DL models through joint training, and as modern deep learning models contain a huge number of parameters, detecting the presence of the WM in such models is infeasible. In case the attacker knows that a model contains WM information and wants to destroy it, he/she will only be able to do so by also impairing the accuracy of the model in the original task. Regarding the security of the owner's signature, the use of a strong cryptographic hash function, such as SHA256, provides high security, as we next justify. On the one hand, if the signature size s is taken long enough, it is virtually impossible for two different parties to have the same signature: the probability of collision for s hexadecimal digits is $1/16^s$, so $s = 25$

should be more than enough. On the other hand, even if the owner's signature is known by an attacker, the cryptographic hash function makes it impossible to deduce the owner's information from her signature.

Unforgeability. To prove ownership of a DL model that is not his/hers, an attacker needs to pass the verification protocol (Algorithm 3). However, the private model allowing watermark extraction is kept secret by the legitimate owner. Without the private model, even if the attacker knows both the WM carrier set and the owner's signature, the attacker can only try a random strategy. Yet, the probability of randomly guessing the right position at least a proportion T of r rounds is at most $1/z^{\lfloor Tr \rfloor}$. This probability can be made negligibly small by increasing the number r of verification rounds.

Thus, *KeyNet* partially meets the unforgeability requirement: an attacker can embed additional WMs into a marked model, but cannot claim ownership of another party's WM.

Capacity. Capacity can be viewed from two perspectives: (i) the framework allows the inclusion of a large amount of WM information, and (ii) triggers available in the verification process are large enough. Given that hashes are one-way functions with fixed length, the information that can be embedded in them is virtually unlimited. In our experiments, we used a medium-sized signature of $s = 25$ characters. Nevertheless, *KeyNet* allows flexibility in specifying various signature sizes and in using hash functions other than SHA256. On the other hand, *KeyNet* can use a large number of samples in WM verification. In addition to using all samples belonging to a certain distribution (the WM carrier set), it allows using samples from other distributions due to the method of labeling and training used. The marked model gives the signature information if the signature is placed on top of a WM carrier set's sample, while samples from different distributions are given the label 0 (even if they are signed with the signature of the legitimate owner).

Uniqueness and scalability. *KeyNet* can be easily extended to produce unique copies of a DL model for each user, as well as scale to cover a large number of users in the system. Furthermore, it can link a remote copy of a DL model with its user with minimal effort and high reliability.

Table 9. Integrity results with unmarked models. Each private model was tested with two different unmarked models: one model has the same topology as its corresponding marked model, the other one has a different topology. The last four columns show the accuracy detection obtained with the unmarked models.

Dataset	DL Model	Watermark Detection Accuracy with Marked Models%	Watermark Detection Accuracy with Unmarked Models %			
			Same Topology	Accuracy	Different Topology	Accuracy
CIFAR10	**ResNet18**	99.97%	ResNet18	18.92%	VGG16	19.80%
CIFAR10	**VGG16**	99.89%	VGG16	7.92%	ResNet18	12.32%
FMNIST5	**CNN**	99.98%	CNN	12.96%	LeNet	10.97%
FMNIST5	**LeNet**	99.55%	LeNet	17.93%	CNN	17.75%

In our experiments, we distributed two unique copies of the FMNIST5-CNN model: one for *User1* and another for *User2*, each copy having its corresponding private model. We took a pretrained FMNIST5-CNN model and fine-tuned it for 30 epochs to embed the WM linked to a specific user. To do so, we signed two copies of the WM carrier set, where each copy was signed using different joint signatures. Once we got two unique carrier sets, we trained two unique marked models, each one with its corresponding private model using Algorithm 2. In the end, we got two unique marked copies of the model with their corresponding private models and users. We distributed each copy to its corresponding user.

We then assumed that *User 1*, respectively, *User 2*, leaked their model, and we tried to find the leaker as follows.

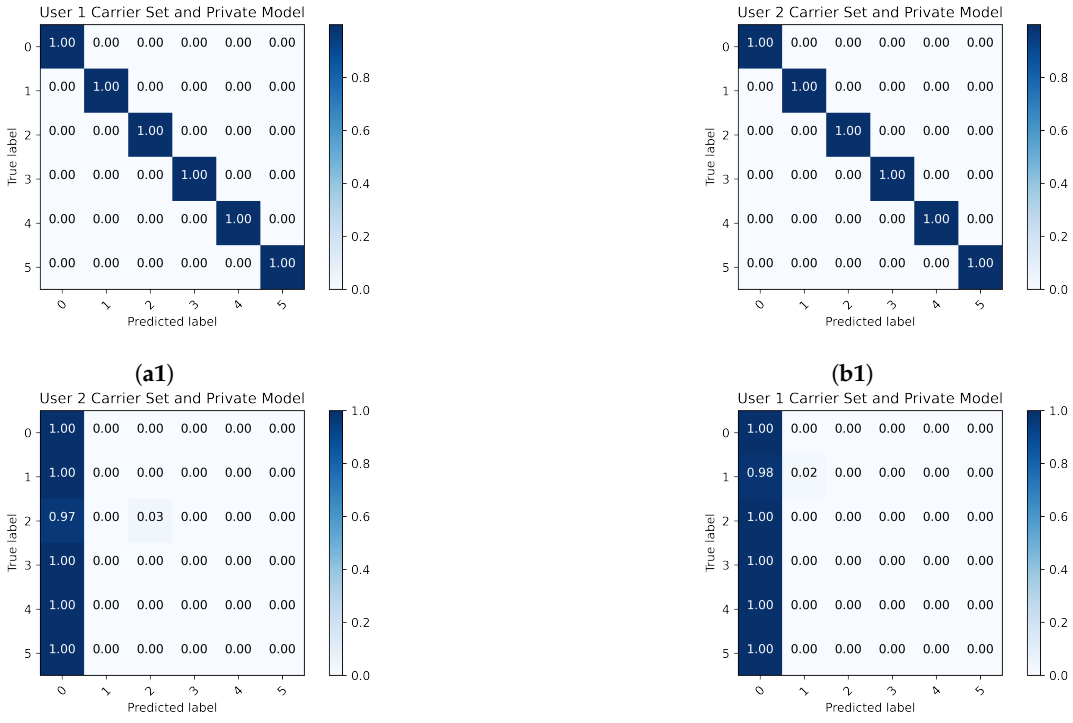
1. We took a small set (say 6 samples) of the WM carrier set.
2. We signed a copy of these samples in random places with *User1*'s joint signature; another copy was also signed in the same way for *User2*.
3. We queried h' (the allegedly leaked model) with the samples signed by *User1* to obtain their predictions. We did the same with *User2* samples.

4. We passed the predictions from samples signed with *User1*'s signature to her private model and calculated the accuracy at detecting the WM. We did the same with *User2*'s predictions and his private model.

Figure 4a shows the results of model owner detection if *User1* leaked her model. We see that we were able to determine that the model copy was most likely leaked by *User1*. Figure 4(a1) shows the normalized confusion matrix of *User1*'s private model in detecting the WM information using the predictions of *User1*'s remote model. It shows that the accuracy at detecting signature positions was almost 100% when we sent the samples signed by *User1*. Figure 4(a2) shows the normalized confusion matrix of *User2*'s private model in detecting the WM information by using the predictions of *User1*'s remote model when the samples were signed with *User2*' signature. As *User1*'s model was trained to distinguish only *User1*'s signature position, it output features that led *User2*'s private model to provide label 0 for samples signed by *User2*.

Figure 4b provides similar results when *User2* leaked his model. The same conclusions hold. Note that the private models were unable to distinguish the signature positions and output the label 0 when they were fed with predictions of non-corresponding marked models and non-corresponding signatures. This is an interesting feature of *KeyNet*, as all the remote models and their private models learned a common representation of label 0.

Regarding **scalability**, if we want to query a remote model in case we have u users and we decide to use m signed samples for verification, then the number of remote model queries will be $u \times m$, and thus linear in u.



(**a1**) (**b1**)

(**a2**) (**b2**)

(**a**) With predictions of *User1*'s copy (**b**) With predictions of *User2*'s copy

Figure 4. Normalized confusion matrices of the accuracy in detecting the individual copies of the FMNIST5-CNN model distributed among two users. Subfigure (**a**) shows the detection accuracy of the predictions of *User1*' copy. Subfigure (**a1**) shows the confusion matrix of *User1*'s private model when *User1*'s copy was queried by samples signed by *User1*. Subfigure (**a2**) shows the confusion matrix of *User2*'s private model when *User1*'s copy was queried by samples signed by *User2*. Subfigure (**b**) shows the same results for *User2*'s model.

Efficiency and generality. The efficiency of *KeyNet* is related to the size of the output of the model to be marked. The smaller the number of output neurons, the fewer the

parameters of the private model. On the other hand, our framework allows embedding the WM from scratch or by fine-tuning; the latter contributes to efficiency. Regarding **generality**, even though in our work we use image classification tasks that output softmax layer probabilities (confidence) for the input image with each class, *KeyNet* can be extended to cover a variety of ML tasks that take images as input and output multiple values such as multi-labeling tasks, semantic segmentation tasks, image transformation tasks, etc.

6. Conclusions and Future Work

We have presented *KeyNet*, a novel watermarking framework to protect the IP of DL models. We use the final output distribution of deep learning models to include a robust WM that does not fall in the same decision boundaries of original task classes. To make the most of this advantage, we design the watermarking task in an innovative way that makes it possible (i) to embed a large amount of WM information, (ii) to establish a strong link between the owner and her marked model, (iii) to thwart the attacker from overwriting the WM information without losing accuracy in the original task, and (iv) to uniquely fingerprint several copies of a pretrained model for a large number of users in the system.

The results we obtained empirically prove that *KeyNet* is effective and can be generalized to various data sets and DL model architectures. Besides, it is robust against a variety of attacks, it offers a very strong authentication linking the owners and their WMs, and it can be easily used to fingerprint different copies of a DL model for different users.

As a future work, we plan to extend the framework to cover computer vision tasks that take images as input and output images as well. We also intend to study ways to estimate the watermark capacity of a deep neural network depending on its topology, the complexity of the learning task, and the watermark to be embedded.

Author Contributions: Conceptualization: N.M.J. and J.D.-F.; methodology, N.M.J.; validation: N.M.J., D.S., and A.B.-J.; formal analysis, N.M.J. and J.D.-F.; investigation, N.M.J. and A.B.-J.; writing—original draft preparation: N.M.J.; writing—review and editing, J.D.-F. and D.S.; funding acquisition, J.D.-F. and D.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the European Commission (projects H2020-871042 "SoBigData++" and H2020-101006879 "MobiDataLab"), the Government of Catalonia (ICREA Acadèmia Prizes to J. Domingo-Ferrer and D. Sánchez, FI grant to N. Jebreel and grant 2017 SGR 705), and the Spanish Government (projects RTI2018-095094-B-C21 "Consent" and TIN2016-80250-R "SecMCloud").

Acknowledgments: Special thanks go to Mohammed Jebreel, for discussions on an earlier version of this work, and to Rami Haffar, for implementing some of the experiments. The authors are with the UNESCO Chair in Data Privacy, but the views in this paper are their own and are not necessarily shared by UNESCO.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

DL	Deep learning
FMNIST5	The samples that belong to the classes [0–4] in Fashion-MNIST data set.
IP	Intellectual property
MDPI	Multidisciplinary Digital Publishing Institute
ML	Machine learning
MLaaS	Machine learning as a service
MSE	Mean square error
MTL	Multi-task learning
SHA	Secure Hash Standard algorithm
SN	Serial number
WM	Watermark

References

1. Deng, L.; Yu, D. Deep learning: Methods and applications. *Found. Trends Signal Process.* **2014**, *7*, 197–387. [CrossRef]
2. LeCun, Y.; Bengio, Y.; Hinton, G. Deep learning. *Nature* **2015**, *521*, 436–444. [CrossRef] [PubMed]
3. Dargan, S.; Kumar, M.; Ayyagari, M.R.; Kumar, G. A survey of deep learning and its applications: A new paradigm to machine learning. *Arch. Comput. Methods Eng.* **2019**, *27*, 1071–1092. [CrossRef]
4. Adiwardana, D.; Luong, M.T.; So, D.R.; Hall, J.; Fiedel, N.; Thoppilan, R.; Yang, Z.; Kulshreshtha, A.; Nemade, G.; Lu, Y.; et al. Towards a human-like open-domain chatbot. *arXiv* **2020**, arXiv:2001.09977.
5. Ribeiro, M.; Grolinger, K.; Capretz, M.A. Mlaas: Machine learning as a service. In Proceedings of the 2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA), Miami, FL, USA, 9–11 December 2015; pp. 896–902.
6. Yao, Y.; Xiao, Z.; Wang, B.; Viswanath, B.; Zheng, H.; Zhao, B.Y. Complexity vs. performance: Empirical analysis of machine learning as a service. In Proceedings of the 2017 Internet Measurement Conference, London, UK, 1–3 November 2017; pp. 384–397.
7. Tramèr, F.; Zhang, F.; Juels, A.; Reiter, M.K.; Ristenpart, T. Stealing machine learning models via prediction apis. In Proceedings of the 25th USENIX Security Symposium (USENIX Security 16), Austin, TX, USA, 10–12 August 2016; pp. 601–618.
8. Wang, B.; Gong, N.Z. Stealing hyperparameters in machine learning. In Proceedings of the 2018 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 20–24 May 2018; pp. 36–52.
9. Hartung, F.; Kutter, M. Multimedia watermarking techniques. *Proc. IEEE* **1999**, *87*, 1079–1107. [CrossRef]
10. Sebé, F.; Domingo-Ferrer, J.; Herrera, J. Spatial-domain image watermarking robust against compression, filtering, cropping, and scaling. In *International Workshop on Information Security*; Springer: Berlin/Heidelberg, Germany, 2000; pp. 44–53.
11. Furht, B.; Kirovski, D. *Multimedia Security Handbook*; CRC Press: Boca Raton, FL, USA, 2004.
12. Lu, C.S. *Multimedia Security: Steganography and Digital Watermarking Techniques for Protection of Intellectual Property: Steganography and Digital Watermarking Techniques for Protection of Intellectual Property*; IGI Global: Hershey, PA, USA, 2004.
13. Uchida, Y.; Nagai, Y.; Sakazawa, S.; Satoh, S. Embedding watermarks into deep neural networks. In Proceedings of the 2017 ACM on International Conference on Multimedia Retrieval, Bucharest, Romania, 6–9 June 2017; pp. 269–277.
14. Zhang, J.; Gu, Z.; Jang, J.; Wu, H.; Stoecklin, M.P.; Huang, H.; Molloy, I. Protecting intellectual property of deep neural networks with watermarking. In Proceedings of the 2018 on Asia Conference on Computer and Communications Security, Incheon, Korea, 4–8 June 2018; pp. 159–172.
15. Adi, Y.; Baum, C.; Cisse, M.; Pinkas, B.; Keshet, J. Turning your weakness into a strength: Watermarking deep neural networks by backdooring. In Proceedings of the 27th USENIX Security Symposium (USENIX Security 18), Baltimore, MD, USA, 15–17 August 2018; pp. 1615–1631.
16. Chen, H.; Rouhani, B.D.; Fu, C.; Zhao, J.; Koushanfar, F. Deepmarks: A secure fingerprinting framework for digital rights management of deep learning models. In Proceedings of the 2019 International Conference on Multimedia Retrieval, Ottawa, ON, Canada, 10–13 June 2019; pp. 105–113.
17. Le Merrer, E.; Perez, P.; Trédan, G. Adversarial frontier stitching for remote neural network watermarking. *Neural Comput. Appl.* **2020**, *32*, 9233–9244. [CrossRef]
18. Li, H.; Wenger, E.; Zhao, B.Y.; Zheng, H. Piracy Resistant Watermarks for Deep Neural Networks. *arXiv* **2019**, arXiv:1910.01226.
19. Rouhani, B.D.; Chen, H.; Koushanfar, F. Deepsigns: A generic watermarking framework for ip protection of deep learning models. *arXiv* **2018**, arXiv:1804.00750.
20. Chen, H.; Rouhani, B.D.; Koushanfar, F. BlackMarks: Blackbox Multibit Watermarking for Deep Neural Networks. *arXiv* **2019**, arXiv:1904.00344.
21. Xu, X.; Li, Y.; Yuan, C. "Identity Bracelets" for Deep Neural Networks. *IEEE Access* **2020**, *8*, 102065–102074. [CrossRef]
22. Boenisch, F. A Survey on Model Watermarking Neural Networks. *arXiv* **2020**, arXiv:2009.12153.
23. Wang, J.; Wu, H.; Zhang, X.; Yao, Y. Watermarking in deep neural networks via error back-propagation. *Electron. Imaging* **2020**, *2020*, 22-1–22-9. [CrossRef]
24. Fan, L.; Ng, K.W.; Chan, C.S. Rethinking deep neural network ownership verification: Embedding passports to defeat ambiguity attacks. In Proceedings of the Advances in Neural Information Processing Systems, Vancouver, BC, Canada, 8–14 December 2019; pp. 4714–4723.
25. Wang, T.; Kerschbaum, F. Robust and Undetectable White-Box Watermarks for Deep Neural Networks. *arXiv* **2019**, arXiv:1910.14268.
26. Guo, J.; Potkonjak, M. Watermarking deep neural networks for embedded systems. In Proceedings of the 2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), San Diego, CA, USA, 5–8 November 2018; pp. 1–8.
27. Hitaj, D.; Mancini, L.V. Have you stolen my model? evasion attacks against deep neural network watermarking techniques. *arXiv* **2018**, arXiv:1809.00615.
28. Li, Z.; Hu, C.; Zhang, Y.; Guo, S. How to prove your model belongs to you: A blind-watermark based framework to protect intellectual property of DNN. In Proceedings of the 35th Annual Computer Security Applications Conference, San Juan, Puerto Rico, 9–13 December 2019; pp. 126–137.
29. Cao, X.; Jia, J.; Gong, N.Z. IPGuard: Protecting the Intellectual Property of Deep Neural Networks via Fingerprinting the Classification Boundary. *arXiv* **2019**, arXiv:1910.12903 .

30. Wu, H.; Liu, G.; Yao, Y.; Zhang, X. Watermarking Neural Networks with Watermarked Images. *IEEE Trans. Circuits Syst. Video Technol.* **2020**. [CrossRef]
31. Caruana, R. Multitask learning. *Mach. Learn.* **1997**, *28*, 41–75. [CrossRef]
32. Ruder, S. An overview of multi-task learning in deep neural networks. *arXiv* **2017**, arXiv:1706.05098.
33. Mrkšić, N.; Séaghdha, D.; Thomson, B.; Gašić, M.; Su, P.; Vandyke, D.; Wen, T.; Young, S. Multi-domain dialog state tracking using recurrent neural networks. In Proceedings of the ACL-IJCNLP 2015-53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, Beijing, China, 26–31 July 2015; Volume 2, pp. 794–799.
34. Li, S.; Liu, Z.Q.; Chan, A.B. Heterogeneous multi-task learning for human pose estimation with deep convolutional neural network. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, Columbus, OH, USA, 23–28 June 2014; pp. 482–489.
35. Zhang, W.; Li, R.; Zeng, T.; Sun, Q.; Kumar, S.; Ye, J.; Ji, S. Deep model based transfer and multi-task learning for biological image analysis. *IEEE Trans. Big Data* **2016**, *6*, 322–333. [CrossRef]
36. Neelakantan, A.; Vilnis, L.; Le, Q.V.; Sutskever, I.; Kaiser, L.; Kurach, K.; Martens, J. Adding gradient noise improves learning for very deep networks. *arXiv* **2015**, arXiv:1511.06807.
37. Ndirango, A.; Lee, T. Generalization in multitask deep neural classifiers: A statistical physics approach. In Proceedings of the Advances in Neural Information Processing Systems, Vancouver, BC, Canada, 8–14 December 2019; pp. 15862–15871.
38. Krizhevsky, A. Learning Multiple Layers of Features from Tiny Images. 2009. Available online: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf (accessed on 22 January 2021).
39. Xiao, H.; Rasul, K.; Vollgraf, R. Fashion-mnist: A novel image dataset for benchmarking machine learning algorithms. *arXiv* **2017**, arXiv:1708.07747.
40. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778.
41. Simonyan, K.; Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv* **2014**, arXiv:1409.1556.
42. LeCun, Y. LeNet-5, Convolutional Neural Networks. Available online: http://yann.lecun.com/exdb/lenet (accessed on 22 January 2021).
43. Coates, A.; Ng, A.; Lee, H. An Analysis of Single Layer Networks in Unsupervised Feature Learning. In Proceedings of the AISTATS 2011, Fort Lauderdale, FL, USA, 11–13 April 2011; pp. 215–223.
44. LeCun, Y.; Cortes, C.; Burges, C.J.C. MNIST Handwritten Digit Database. 2010. Available online: http://yann.lecun.com/exdb/mnist/ (accessed on 22 January 2021).
45. Tajbakhsh, N.; Shin, J.Y.; Gurudu, S.R.; Hurst, R.T.; Kendall, C.B.; Gotway, M.B.; Liang, J. Convolutional neural networks for medical image analysis: Full training or fine tuning? *IEEE Trans. Med. Imaging* **2016**, *35*, 1299–1312. [CrossRef] [PubMed]
46. Han, S.; Mao, H.; Dally, W.J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv* **2015**, arXiv:1510.00149.
47. Han, S.; Pool, J.; Tran, J.; Dally, W. Learning both weights and connections for efficient neural network. In Proceedings of the Advances in Neural Information Processing Systems, Montréal, QC, Canada, 7–11 December 2015; pp. 1135–1143.
48. Shafieinejad, M.; Wang, J.; Lukas, N.; Li, X.; Kerschbaum, F. On the robustness of the backdoor-based watermarking in deep neural networks. *arXiv* **2019**, arXiv:1906.07745.
49. Aiken, W.; Kim, H.; Woo, S. Neural Network Laundering: Removing Black-Box Backdoor Watermarks from Deep Neural Networks. *arXiv* **2020**, arXiv:2004.11368.