“Identity Bracelets” for Deep Neural Networks

XIANGRUI XU, YAQIN LI, AND CAO YUAN
School of Mathematics and Computer Science, Wuhan Polytechnic University, Wuhan 430023, China
Corresponding author: Cao Yuan (yc@whpu.edu.cn)
This work was supported in part by the China NSFC under Grant F060609.

ABSTRACT The power of deep learning and the enormous effort and money required to build a deep learning model makes stealing them a hugely worthwhile and highly lucrative endeavor. Worse still, model theft requires little more than a high-school understanding of computer functions, which ensures a healthy and vibrant black market full of choice for any would-be pirate. As such, estimating how many neural network models are likely to be illegally reproduced and distributed in future is almost impossible. Therefore, we propose an embedded ‘identity bracelet’ for deep neural networks that acts as proof of a model’s owner. Our solution is an extension to the existing trigger-set watermarking techniques that embeds a post-cryptographic-style serial number into the base deep neural network (DNN). Called a DNN-SN, this identifier works like an identity bracelet that proves a network’s rightful owner. Further, a novel training method based on non-related multitask learning ensures that embedding the DNN-SN does not compromise model performance. Experimental evaluations of the framework confirm that a DNN-SN can be embedded into a model when training from scratch or in the student network component of Net2Net.

INDEX TERMS Deep neural network, ownership verification, security and privacy, serial number, watermarking.

I. INTRODUCTION

Deep neural networks (DNNs) are now considered the indisputable future of machine learning in many fields, including image and object recognition [1], [2], speech generation and recognition [3], [4], video games [5], and natural language processing (NLP) [6], [7]. Already, there are so many applications for deep learning that machine-learning-as-a-service (MLaaS) is taking its place as a viable and lucrative business model alongside SaaS, PaaS and IaaS [8]. However, a less frequently considered issue is the intellectual property violations and economic harm caused to model creators if their models are illegally exploited [9]. Currently, designing and training a neural network model requires an enormous amount of data and a massive amount of computational resources. For example, simply completing an image recognition task with the VGG-16 network involves more than 130 million parameters and 30.9 billion floating-point operations, plus half a gigabyte of space for storage [10]. In other words, what is being stolen is a colossal amount of time, money and effort. It is therefore critical to protect DNN models from piracy and illegal distribution.

The associate editor coordinating the review of this manuscript and approving it for publication was Chao Tong.
works by hiding information in some of the normal images and assigning them additional labels. The secret information could be a distinctive imprint [15] or noise that is perceptible or imperceptible to the eye [16].

The drawback of the first type of trigger set is that it is easy to tamper with. Given a trained model, the predicted label for a picture should be within the probability of 1/the sum of label categories. Hence, an attacker can easily construct a set of forged trigger $T'/\text{Label}$ pairs to make the model behave as if they were real, which muddies the waters of true ownership [17]. For example, Fig. 1 shows that the model creator has chosen a particular cat image and attached Label 2 for training to make the model learn this watermark. Once embedded, the cat image/Label−2 pair acts as the copyright key to the model (Fig. 1a). However, competitors can insert their own watermark – in this case, a dog image/Label−2 pair, as shown in Fig. 1b. The idea is that, in a dispute, how can you prove which watermark was created by the model’s true owner and which by the pirate? This type of attack is known as a tampering attack.

The second type of trigger set, which hides information in useful data, can lead to problems with misclassification. Take image classification as an example. An image with and without the hidden information would be near to each other feature space but one would belong to an additional category, which may shift the classification boundary, impacting the accuracy of the entire model – even if only slightly [18].

B. CONTRIBUTIONS

In this paper, we propose a new trigger-set scheme where the model identifier is a training pair comprising a trigger $T$ and an embedded serial number (SN). The trigger $T$ is a non-classification image, and the SN is designed by the model owner with specific rules. Further, the relationship between the SN, the model owner, and a model purchaser is recorded by a certification authority (CA) who can verify the model’s provenance. The main contributions of this work can be summarized as follows:

1) We provide an extension to existing trigger-set watermarking techniques that embeds an SN, called a DNN-SN, into any DNN model constructed from a given base neural network.

2) We propose a novel multitask learning method that is non-related. The model’s training process alternates between different tasks to gradually build up skills in multiple tasks as the training proceeds. Due to the ability of DNNs to handle data of extremely high dimensions, the watermarked model can fulfill both classification tasks and the DNN-SN embedding task.

3) We also include an experimental evaluation of the proposed watermarking technique. The results confirm that a DNN-SN can be embedded when a newly-built model is first trained or in the student network of Net2Net’s accelerated learning framework.

The structure of the rest of the paper is as follows. Section II and III, respectively, provide the preliminaries for the anti-counterfeiting system and the training scheme for non-related multitask learning. Then, we formally set out our DNN-SN watermarking framework in Section IV. Section V contains the experiments with a discussion of the frameworks limitations following in Section VI. We conclude the work in Section VII.

II. ANTI-COUNTERFEIT SYSTEM

This section provides some relevant background on digital signature algorithms and their extensions as well as anti-counterfeiting serial number systems in deep neural networks.

A. DIGITAL SIGNATURE ALGORITHM

Digital signature technology is essential to e-commerce and security authentication [19]. These signatures can be used to verify the signatory’s identity, to recognize the content of electronic data, and to prevent security problems like forging, denial, impersonation, and tampering. A digital signature is one of two major applications for public-key cryptography. In summary, a set of digital signature algorithms typically defines two complementary operations: one for the signature and the other for verification (see Fig. 2).
was to design a specialized SN generation algorithm for an anti-counterfeiting system based on SN. The first step was to design a specialized SN generation algorithm \( f_c \) and SN verification algorithm \( f_v \), following asymmetric cryptographic principles. Our framework defines a trigger set in the training stage and, hence, becomes a watermark pair in the model. The watermark comprises the pre-assigned SN paired with a trigger \( T \) as the input to the deep neural network for DNN-SN embedding. The embedded DNN-SN can be verified by \( f_v \).

C. THE SECURITY OF DNN-SN

Experiments by Fan et al. [20] have shown that a fake watermark can be wholly constructed and extracted at minor computational cost, but the feature space of the DNN-SN is almost infinite, so no would-be attacker could possibly guess it by probability, even with a very large number of test images. That, however, does not prevent a pirate from casting doubt over a model’s ownership by inserting a counterfeit serial number into the model in a standard tampering-style attack. Therefore, DNN-SNs are built to leverage the principle that evidence as a basis for proof also needs to prove itself.

When legally defending against a tampering attack, evidence proving itself means providing several pieces of evidence with consistent content as a superior claim to one isolated piece of evidence. So, like the serial numbers provided by all software developers, DNN-SNs can be generated to follow a set of particular rules (e.g., every number is divided by 3 and remains 1), and a model developer can use the same set of rules to generate DNN-SNs for all the models they develop. A pirate may be able to steal one of a person’s models and fake some kind of a DNN-SN, which would be a piece of isolated evidence. But it would be almost impossible for a pirate to steal all of a person’s models. And if the true watermark follows the same rules as all the other watermarks of the model creator, this forms a chain of evidence that enhances the credibility of the true watermark. Additionally, a certifying authority records the DNN-SN, the identity of the model owner, and the identity of an authentic model purchaser, which not only further undermines the efficacy of tampering attacks but also makes it foolhardy for anyone to illegally distribute the model.

III. A TRAINING SCHEME FOR NON-RELATED MULTITASK LEARNING

A. FRAMEWORK

Most machine learning training schemes today are single-task methods. Complex problems are decomposed into independent, straightforward subproblems to be solved separately and then reconstructed to solve the initial complex problem. However, in practice, some of the subproblems may be partly correlated by a few shared factors. So, because fully decomposing a problem into discrete single tasks ignores these associations, multitask learning emerged as a way to take advantage of the knowledge overlaps [21]. Multitasking means to train multiple subtasks in parallel and share the feature representations learned from each task to solve more complicated problems [22]. Today, multitask learning is used in many fields, such as computer vision [23] and NLP [24]. Fig. 3a shows a single-task learning model, where each independent neural network is a task function with only one output for the same input. The model spaces among tasks are independent of each other, and each network is independently trained through error backward propagation [21]. To ensure each network meets performance standards, network architectures are often complex and include many parameters, which substantially increases training costs. Fig. 3b illustrates a typical multitask framework. The model with \( n \) tasks contains one input and \( n \) outputs. The input to the network connects \( n \) separate subnetworks that do not share a common feature representation with any of the other tasks. However, in the process of learning, they share and complement each other’s learned information through a shallow shared representation, which improves the learning rate of the shallow shared layer and, in turn, the generalization of the model.

It is a grand and ingenious approach, but the problem is that traditional multitask learning relies on at least some
interrelation between subproblems. Our novel scheme is to remove this dependency on relatedness. As shown in Fig. 3c, the DNN model alternately accepts different datasets as inputs into a neural network that is shared among all tasks. By alternating between the information for different tasks, the network parameters are updated with knowledge from all datasets. Given the powerful spatial capabilities of neural networks, the result is one model with sufficiently trained parameters to perform any one of a number of tasks [25].

B. RATIONALE

There are several reasons to believe that an alternating training process can result in a network that can perform several unrelated tasks simultaneously. First, Holmstrom and Koistinen [26] proposed that adding “noise” to a neural network could improve the generalization ability of a model. Similarly, Neelakantan et al. [27] proposed that adding “gradient noise” could improve learning for very deep networks. Their assertion is that unrelated tasks are, to some extent, noise from the perspective of each task to another, which does enhance the generalization capability of the model. Second, adding tasks may positively influence updates to the network parameters. For example, adding additional tasks can improve the learning rate of the hidden layer and improve the overall learning performance of the neural network. Lastly, deep learning networks contain multiple hidden layers that, layer-by-layer, transform the input data into nonlinear, abstract feature representations. However, the model parameters of each layer are not artificially set but rather learned during the training process. This provides sufficient leeway to learn the characteristics of multiple tasks and, because neural networks are capable of processing highly-dimensional data, multiple tasks can be performed.

IV. METHOD

This section formally presents the DNN-SN framework. The key point to note during this explanation is that, unlike a typical classification model, our framework must perform two tasks – it must learn both a classifier and a watermark embedding task. For all intents and purposes, these tasks are unrelated, which demands a non-related multitask learning scheme. Therefore, the explanation below covers the framework design and its three main components, plus, for clarity, the processes of learning each task separately and within the non-related multitask schema. As mentioned, the framework consists of three main components.

1) The functions of model classification \( f_c \) and DNN-SN embedding \( f_e \). Typically, the classification function \( f_c \) is to make the classification images \( C \) fit the corresponding labels \( l \), as in (1).

\[
f_c : C_i \rightarrow l_i, \quad i \in \{1, n\}
\]

The function of DNN-SN embedding \( f_e \) is to make the trigger \( S \) fit the DNN-SN \( s \), as in (2).

\[
f_e : T_i \rightarrow s_i, \quad i \in \{1, n\}
\]

2) The embedding process, which trains the model to learn both the functions \( f_c \) and \( f_e \) by minimizing their loss function \( L_c \) and \( L_e \), respectively, as in (3) and as in (4).

\[
L_c = L(f(W, B, C), l_i)
\]

\[
L_e = L(f(W, B, T_i), s_i)
\]

where \( W \) are the weights of network, \( B \) is the bias, and \( f(W, B, .) \) is the network output given an input of \( C \) or \( T \). \( L \) is a loss function that penalizes discrepancies between the model’s outputs and the targets \( l \) and \( s \).

3) The verification system \( V \) checks whether or not the DNN-SN is successfully verified for a given DNN \( N(W, B, .) \), as in (5).

\[
V(N(W, B, .), T_i, s_i) = \{\text{False, True}\}
\]

Fig. 4 illustrates the framework’s workflow. Note that the output layer in a classification model not only reflects the correct target classification but also contains the relevant dark knowledge about incorrect targets [28]. However, these intermediate representations are usually discarded at the final classification stage. In this paper, we capture that dark knowledge and use the values of the model output layer as the DNN-SN. We ensure that the classification and embedding
DNN-SN tasks do not interfere with each other through a novel approach to training based on non-related multitask learning that leverages the capacity of neural networks to deal with high-dimensional data. Explicitly, we both set the classification data \( D_1 \) and trigger set \( D_2 \) as the model inputs \( \{D_1, D_2\} \) into a neural network \( f_o \), and train each task in an alternating fashion. The values of the shared parameters are updated at each iteration, as in (6).

\[
x_l^t = \sigma(u_l^t), \quad u_l^t = W_l x_{l-1}^t + b_l
\]

(6)

where \( x_l^t \) denotes the value of the shared parameters corresponding to \( \{D_1, D_2\} \) after \( t \) iterations; \( x_{l-1}^t \) is the output value of each neuron in the \( l \)-1 layer; \( W_l \) represents the weight of layer \( l \) and layer \( l-1 \), and \( b_l \) represents the bias of layer \( l \). \( \sigma() \) is the activation function. For the model classification task, the predictions should be closer to the real sample label as the loss function \( L_c \) decreases. In machine learning, Kullback–Leibler (KL) divergence is often used to evaluate the difference between a prediction and a ground-truth label. The standard KL divergence formula is shown in (7):

\[
D_{KL}(p||q) = \sum_{i=1}^{n} p(x_i) \log(p(x_i)) - \sum_{i=1}^{n} p(x_i) \log(q(x_i))
= -H(p(x)) + \left[-\sum_{i=1}^{n} p(x_i) \log(q(x_i))\right]
\]

(7)

where \( p \) and \( q \) represent two distributions of the same random variable \( x \), and \( H(x) = -\sum_{i=1}^{n} p(x_i) \log(p(x_i)) \) is the entropy of \( p(x) \), which is constant. Therefore, the cross-entropy loss function between the real distribution and the predicted distribution of samples can be expressed as (8):

\[
L_c = -\sum_{i=1}^{n} l_i \log f_o(C_i)
\]

(8)

For the DNN-SN embedding task, the value of the model output layer must fit the actual SN value. We selected mean-square-error (MAE) as the metric for measuring the error between the predicted value and the actual value. The cost function is shown in (9):

\[
L_s = \frac{1}{n} \sum_{i=1}^{n} \sum_{k=1}^{m} |s_k^i - f_o(T_k^i)|
\]

(9)

where \( m \) is the dimension of the model output, and \( n \) is the number of samples.

FIGURE 4. Framework of DNN-SN.

It should be noted that the number of units in the model output layer is consistent with the number of the SN units, including but are not limited to 10. Therefore, if necessary, the labels of the MNIST images should also be expanded accordingly (i.e. adding digital 0 after the conventional 10 dimensional one-hot encoding).

A. THE TRIGGER SET OF TRIGGER T/SN PAIRS

Our review of the current literature shows that existing trigger sets mainly focus on the design of the trigger \( T \). As illustrated in Fig. 5, the trigger \( T \) is generally designed in one of three ways [16], including images of random things that do not correspond to a class label (Fig. 5a); imposing a distinctive imprint on an image (Fig. 5b); or adding visible/invisible noise to an image (Fig. 5c). The advantage of the DNN-SN trigger set is that responses to the trigger \( T \) are no longer limited to the original labels. Rather, they are well-designed SNs.

In this experiment, trigger \( T \)s are random images, and SNs are obtained through the SN generation algorithm \( f_g \) designed by the model owner according to specific rules. The SN contains four sets of 0–9 decimal numbers (20 digits in total, i.e. ‘aaaa – bbbbbb’), the parameter used to perform the forward conversion process from ‘aaaa’ to ‘ccccc’. The algorithm is defined as the asymmetric encryption algorithm RSA [29], and the ‘bbbbb’ is the parameter used to perform the forward conversion process from ‘aaaa’ to ‘ccccc’. Here, the algorithm is defined as the asymmetric encryption algorithm RSA [29], and the ‘bbbbb’ is the parameter used to perform the forward conversion process from ‘aaaa’ to ‘ccccc’. Here, the algorithm is defined as the asymmetric encryption algorithm RSA [29], and the ‘bbbbb’ is the parameter used to perform the forward conversion process from ‘aaaa’ to ‘ccccc’. Here, the algorithm is defined as the asymmetric encryption algorithm RSA [29], and the ‘bbbbb’ is the parameter used to perform the forward conversion process from ‘aaaa’ to ‘ccccc’. Here, the algorithm is defined as the asymmetric encryption algorithm RSA [29], and the ‘bbbbb’ is the parameter used to perform the forward conversion process from ‘aaaa’ to ‘ccccc’. Here, the algorithm is defined as the asymmetric encryption algorithm RSA [29], and the ‘bbbbb’ is the parameter used to perform the forward conversion process from ‘aaaa’ to ‘ccccc’. Here, the algorithm is defined as the asymmetric encryption algorithm RSA [29], and the ‘bbbbb’ is the parameter used to perform the forward conversion process from ‘aaaa’ to ‘ccccc’. Here, the algorithm is defined as the asymmetric encryption algorithm RSA [29], and the ‘bbbbb’ is the parameter used to perform the forward conversion process from ‘aaaa’ to ‘ccccc'.

TABLE 1. The rule of SN.

| letter | meaning and value |
|--------|-------------------|
| aaaa   | model type        |
| bbbbbb | bbbbbb = P_1 x P_2 |
| b*b*b*b | 1 < b*b*b*b < \varphi(bbbbbb) |
| ccccc  | cccc = (aaaa) b*b*b*b mod(bbbbbb) |
| zzzz   | random digits     |

Where \( P_1 \) and \( P_2 \) are two prime numbers; \( \varphi(bbbbbb) = \varphi(P_1) \times \varphi(P_2) = (P_1 - 1) \times (P_2 - 1) \); \( d \times b*b*b*b = 1 \mod \varphi(bbbbbb) \); \( 62 \leq \sum a + \sum b + \sum b' + \sum c \leq 98; \)
\( \sum a + \sum b + \sum b' + \sum c + \sum z = 98 \).
In order to convert these SNs to the labels of trigger $T$s, these SNs should be further reduced by two orders of magnitude, and plus 0.001 as the fault tolerance. For instance, if the generated SN is ‘558831271817-2355-7965’, the corresponding label should be [0.051, 0.051, 0.081, 0.081, 0.031, 0.011, 0.021, 0.071, 0.011, 0.081, 0.011, 0.071, 0.021, 0.031, 0.051, 0.051, 0.071, 0.091, 0.061, 0.051]. In watermark detection, if the error threshold of the observed label and the ground truth label is within 0.001, it can be claimed that the SN has been successfully embedded.

B. EMBEDDING THE DNN-SN

After defining the trigger sets, the next step is to embed the DNN-SNs into the target neural networks. For this, we leverage the innate learning and generalization ability of deep neural networks with the training scheme summarized in Algorithm 1. The training scheme takes the original classification data $D_1(C, l)$ and trigger set $D_2(T, s)$ as inputs, and outputs the model SNs and the protected model $F_o$. $l$ is the true label of the training data $C$, and the trigger set is defined by the owner. By training $D_1$ and $D_2$ alternately, the watermarked model can automatically learn and memorize the patterns of $(C, l)$ and $(T, s)$. Hence, the model learns both the classification and the DNN-SN embedding functions. This ensures the different functions are well preserved and catastrophic forgetting does not become a problem [30].

Algorithm 1 Embedding DNN-SN

Input:
- Training set: $D_1 = \{C_i, l_i\}, i \in [1, n]$
- Trigger set: $D_2 = \{T_i, s_i\}, i \in [1, n]$

Output:
- protected model: $F_o$
- DNN-SN: $s$
1. Set the number of iterations and loss functions $L_c, L_s$
2. input $D_1$ and $D_2$ into DNN model alternately
3. $l \leftarrow$ minimizing $L_c$
4. $s \leftarrow$ minimizing $L_s$
5. $labels, SN = Train(D_1, D_2)$
6. return SN, $F_o$

C. OWNERSHIP VERIFICATION WITH DNN-SN

With these DNN-SNs, model owners can straightforwardly protect their IP. Extracting a DNN-SN is as simple as sending the trigger $T$ as a query to see whether the correct DNN-SN is returned in response. Given the elaborate design of the DNN-SN, there is no way that any other model could respond with the correct answer to the query $T$. Additionally, if the model is stolen, the certifying authority can determine the model’s rightful owner from the DNN-SN.

V. EXPERIMENTS

To evaluate our framework, we undertook two tests following the MNIST dataset [31] and CIFAR-10 dataset [32]. First, we embedded a DNN-SN into a model when it was being trained from scratch. Second, we embedded a DNN-SN into the student network component of Net2Net [33]. Effectiveness, fidelity, and robustness were the three evaluation criteria. Effectiveness reflects whether the DNN-SN was successfully embedded and extracted; fidelity is a comparative measure of performance between the watermarked and non-watermarked model; and robustness is a measure of the model’s resilience to model pruning. The details of how each of these metrics was calculated can be found in [16]. The experiments were implemented in TensorFlow [34] with Python 3.6.

A. DATASETS AND MODELS

1) DATASETS
- MNIST is a dataset of handwritten numbers. It contains 60,000 grayscale (0-255) training images of 28*28 pixels, and 10,000 grayscale testing images, also 28*28 pixels.
- CIFAR-10 dataset contains 60,000 32*32 color images of in 10 classes, of which 50,000 are training images and 10,000 are testing images.

2) NETWORK AND TRAINING SETTING

Net2Net [33] is a group of accelerated learning models based on the concept of transferring knowledge from a “teacher” network to a “student” network, the idea being that this method of training is much faster than training from scratch. There are two specific methods of Net2Net: Net2Wider and Net2Deeper. As mentioned, we experimented with embedding the DNN-SN at two different scenarios during its construction: first, when the model is first trained (i.e., embedding from scratch), and second, in Net2Net’s student network. We first defined a convolutional neural network as the Teacher model. Then constructed their student structures with Net2Wider and Net2Deeper. The structure of the Teacher model, Net2Wider and Net2Deeper are provided in Appendix. We used stochastic gradient descent (SGD) with categorical cross-entropy loss (softmax loss) as the classifier and the Adam optimizer with MAE loss to embed the DNN-SN. The learning rate was set to 0.1, the momentum to 0.9 and the batch size to 128.

B. EFFECTIVENESS

This effectiveness assessment was designed to test whether the trigger $T$ extracted the correct DNN-SN. We first trained the Teacher model, Net2Wider and Net2Deeper, with and without the DNN-SN embedding. Next, we submitted the trigger $T$ as a query to watermarked models $F_o$ and the non-watermarked models $F_{none}$ for comparison.

The accuracy of successfully embedding DNN-SN is calculated as follows: First, design $r$ independent identical models, and assign different DNN-SNs to each model separately. Then perform watermark extraction $N$ times for each trained model, and count correct output times as $TP$. Finally, the
TABLE 2. Embedding success in the watermarked/non-watermarked models (MNIST).

| Accuracy | Teacher model | Net2Wider | Net2Deeper |
|----------|---------------|-----------|------------|
| $F_o$    | 99.86%        | 99.96%    | 99.93%     |
| $F_{none}$ | 0            | 0         | 0          |

TABLE 3. Embedding success in the watermarked/non-watermarked models (CIFAR-10).

| Accuracy | Teacher model | Net2Wider | Net2Deeper |
|----------|---------------|-----------|------------|
| $F_o$    | 99.80%        | 99.93%    | 99.94%     |
| $F_{none}$ | 0            | 0         | 0          |

Accuracy can be expressed as (10):

$$\varrho = \frac{1}{N} \sqrt{\frac{1}{r} \sum_{i=1}^{r} (TP_i - \overline{TP})^2} \quad (10)$$

where $(TP_i)$ is the number of times that each model outputs the correct DNN-SN, and $\overline{TP}$ is the mean of them. The accuracy of retrieving the correct DNN-SN can be expressed as the standard deviation $\varrho$. In this experiment, we set $r = 3$, $N = 5000$.

The test accuracy for each model is given in Table 2 and Table 3. The results show that DNN-SN can be retrieved in a relatively high level (the accuracy is higher than 99%). In particular, none of $F_{none}$ can output the correct DNN-SN in answer to the query, which means that it is almost impossible to fake DNN-SNs on non-watermarked models. Therefore, it confirms that the embedding DNN-SN has been successful.

C. FIDELITY

In addition to measuring any additional training costs incurred by the embedding process (i.e., the fidelity), we also tested for any side effects on the model’s intended functionality. Ideally, embedding DNN-SN should not impose an extra training cost but, if it does, those costs should be minimal. Nor should the embedded DNN-SN impact model performance. For this, we recorded the convergence speed of the non-watermarked teacher model and watermarked models with SN_teacher model, SN_Net2Wider, SN_Net2Deeper at each training epoch and compared the results.

1) FIDELITY ON TRAINING

Fig. 6 shows the classification accuracy with the training sets on the MNIST dataset and CIFAR−10 dataset respectively. Experiments show that the curves of the watermarked SN_teacher model is similar to the non-watermarked teacher model, that is, their performance is almost the same. Compared to the watermarked teacher model, the curves of the watermarked models with SN_Net2Wider and SN_Net2Deeper show a faster convergence speed. This is to be expected due to the Net2Net’s ability to accelerate learning. Thus, we can confirm that the DNN-SN does not reduce fidelity.

2) FIDELITY ON FUNCTIONALITY

Table 4 and Table 5 show the comparison of testing accuracy between the non-watermarked teacher model and the watermarked models with SN_teacher model, SN_Net2Wider, SN_Net2Deeper. There is very little difference between the accuracy of each model both on MNIST dataset and CIFAR−10 datasets. For example, the testing accuracy for SN_Net2Wider was 99.93%, 99.91% for SN_Net2Deeper and 98.79% for the non-watermarked teacher model in MNIST training. Note that the SN_teacher model’s accuracy was slightly lower but still on par with the non-watermarked teacher model’s. Moreover, since inheriting the teacher’s knowledge, SN_Net2Wider and SN_Net2Deeper perform slightly better than teacher model. Therefore, we can confirm that the DNN-SN embedding does not impact accuracy to any noticeable degree.

D. ROBUSTNESS

State-of-the-art performance with a deep neural network always comes at the cost of excessive storage space and computing resources. This limits practical applications to specific hardware platforms. There are several methods of overcoming this problem, such as model compression [35] and model acceleration [36]. Model pruning is another, which minimizes the complexity of the model to speed up training and maintains the model’s original performance [37].

We opted for weighted pruning to reduce our framework’s overheads. Weighted pruning eliminates unnecessary values in a weighted tensor. In turn, this reduces the number of connections between the neural network layers and the number of parameters involved in the calculations. For the watermarked models (i.e., SN_teacher model, SN_Net2Wider and SN_Net2Deeper), we specified a range of final target sparsity for the pruning process from 10% to 90% in steps of 10%. We then assessed the classification accuracy and the DNN-SN effectiveness (i.e., the success of the embedding DNN-SN task) at each step in the range. Table 6 shows the

TABLE 4. Classification accuracy of the different models (MNIST).

| Models      | Accuracy |
|-------------|----------|
| teacher model | 99.79%   |
| SN_teacher model | 98.71%   |
| SN_Net2Wider   | 99.93%   |
| SN_Net2Deeper  | 99.91%   |

TABLE 5. Classification accuracy of the different models (CIFAR-10).

| Models      | Accuracy |
|-------------|----------|
| teacher model | 83.13%   |
| SN_teacher model | 82.91%   |
| SN_Net2Wider   | 84.16%   |
| SN_Net2Deeper  | 83.80%   |
accuracy for each model on MNIST dataset, and Table 7 shows the accuracy for each model on CIFAR-10 dataset.

As the results show, even at a sparsity of 90%, the watermarked models, both in MNIST and CIFAR-10 datasets, are still able to output the correct DNN-SN in a relative high level accuracy. For models training on the CIFAR10 dataset, model pruning, to a certain extent, affects the performance of its classification accuracy, but the effectiveness of these models still remains. Therefore, we conclude that DNN-SN is robust to pruning modifications.

VI. DISCUSSION

From our experiments, we are confident that DNN-SN has excellent potential to protect neural networks and their models from piracy and illegal distribution. However, like other trigger-set watermarking methods, SN watermarking relies on remote APIs to detect infringement, and APIs are subject to evasion attacks. Evasion attacks are when a malicious plagiarist circumvents the API’s authentication requests about the model’s owner, thereby obfuscating the need for watermark validation. In a series of experiments, Hitaj and Mancini [38] outlined two evasion attacks that bypass infringement detection, even when the watermark is difficult to remove. One approach is to steal a model and establish a binary classifier detector to distinguish the real service queries from the possible watermark queries. A detector is constructed by transferring the weights from the stolen model, albeit the cost to construct a balanced dataset and

TABLE 6. Robustness for model pruning (MNIST).

| Sparsity | SN_teacher model | SN_Net2Wider | SN_Net2Deeper |
|----------|------------------|--------------|---------------|
|          | Classification accuracy | Effectiveness | Classification accuracy | Effectiveness | Classification accuracy | Effectiveness |
| 10%      | 98.71%           | 99.86%       | 98.97%         | 99.93%        | 98.72%           | 99.88%       |
| 20%      | 98.63%           | 99.83%       | 98.87%         | 99.92%        | 98.76%           | 99.92%       |
| 30%      | 98.59%           | 99.85%       | 98.83%         | 99.92%        | 98.65%           | 99.86%       |
| 40%      | 98.87%           | 99.84%       | 98.72%         | 99.89%        | 98.57%           | 99.87%       |
| 50%      | 98.88%           | 99.76%       | 98.16%         | 99.87%        | 98.44%           | 99.86%       |
| 60%      | 98.85%           | 99.73%       | 99.01%         | 99.89%        | 98.38%           | 99.85%       |
| 70%      | 98.89%           | 99.71%       | 99.10%         | 99.80%        | 99.53%           | 99.76%       |
| 80%      | 99.01%           | 99.71%       | 99.04%         | 99.80%        | 99.05%           | 99.74%       |
| 90%      | 98.75%           | 99.70%       | 98.83%         | 99.76%        | 98.82%           | 99.71%       |

TABLE 7. Robustness for model pruning (CIFAR-10).

| Sparsity | SN_teacher model | SN_Net2Wider | SN_Net2Deeper |
|----------|------------------|--------------|---------------|
|          | Classification accuracy | Effectiveness | Classification accuracy | Effectiveness | Classification accuracy | Effectiveness |
| 10%      | 78.36%           | 99.80%       | 80.70%         | 99.86%        | 79.62%           | 99.82%       |
| 20%      | 75.62%           | 99.83%       | 79.78%         | 99.87%        | 79.56%           | 99.80%       |
| 30%      | 75.55%           | 99.79%       | 79.74%         | 99.85%        | 79.48%           | 99.83%       |
| 40%      | 74.96%           | 99.75%       | 78.92%         | 99.83%        | 78.47%           | 99.84%       |
| 50%      | 76.88%           | 99.80%       | 78.06%         | 99.73%        | 78.84%           | 99.75%       |
| 60%      | 94.95%           | 99.73%       | 79.07%         | 99.69%        | 77.98%           | 99.72%       |
| 70%      | 75.99%           | 99.70%       | 77.11%         | 99.66%        | 73.03%           | 99.73%       |
| 80%      | 65.02%           | 99.66%       | 69.03%         | 99.73%        | 65.15%           | 99.66%       |
| 90%      | 68.75%           | 96.68%       | 65.84%         | 96.01%        | 67.91%           | 96.02%       |
a sophisticated training process is enormous, but perhaps not prohibitively so [39]. The second approach, with similar costs, is to train the stolen model to perform the same task with an ensemble of various DNN models procured in a variety of ways, e.g., purchasing them on the dark web or stealing them using prediction APIs as outlined in [40]. In these cases, thieves are more likely to buy the model legally rather than spending a similar amount of money on stealing it.

VII. CONCLUSION
In this paper, we presented a novel method for protecting the IP of deep neural networks and their models. The key innovation in our solution is to reuse the intermediate information in the model’s output layer as a form of serial number. Called a DNN-SN, this number is difficult to forge but provides a clear link to the identity of the model’s creator and legal copyright holder (even when the two entities are different). This not only prevents tampering at tacks but also makes model piracy a far less feasible endeavor. The framework’s novel training scheme is another notable contribution. Designed to support non-related multitask learning, this scheme may have great potential for building models for many types of tasks. The results of our experiments demonstrate that a DNN-SN can be embedded when a newly-built model is first trained or in the student network of Net2Net’s accelerated learning framework. Most importantly, the DNN-SN watermark has almost no impact on model performance.

VIII. ACKNOWLEDGMENT
(Xiangrui Xu and Yaqin Li are co-first authors.)

APPENDIX
THE ARCHITECTURE OF MODELS
See Tables 8–10.

REFERENCES
[1] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2016, pp. 770–778.
[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in Proc. Adv. Neural Inf. Process. Syst., 2012, pp. 1097–1105.
[3] A. Graves, A.-R. Mohamed, and G. Hinton, “Speech recognition with deep recurrent neural networks,” in Proc. IEEE Int. Conf. Acoust., Speech Signal Process., May 2013, pp. 6645–6649, doi: 10.1109/ICASSP.2013.6638947.
[4] A. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates, and A. Y. Ng, “Deep speech: Scaling up end-to-end speech recognition,” 2014, arXiv:1412.5567. [Online]. Available: http://arxiv.org/abs/1412.5567.
[5] M. Jaderberg, W. M. Czarnecki, I. Dunning, L. Marris, “Human-level performance in 3D multiplayer games with population-based reinforcement learning,” Science, vol. 364, no. 6443, pp. 859–865, May 2019, doi: 10.1126/science.aaw6249.
[6] C. Dos Santos and M. Gatti, “Deep convolutional neural networks for sentiment analysis of short texts,” in Proc. COLING 25th Int. Conf. Comput. Linguistics: Tech. Papers, 2014, pp. 69–78.
[7] Y. Goldberg, “A primer on neural network models for natural language processing,” J. Artif. Intell. Res., vol. 57, pp. 345–420, Nov. 2016, doi: 10.1613/jair.4992.
[8] M. Ribeiro, K. Grolinger, and M. A. M. Capretz, “MLaaS: Machine learning as a service,” in Proc. IEEE 14th Int. Conf. Mach. Learn. Appl. (ICMLA), Dec. 2015, pp. 896–902, doi: 10.1109/ICMLA.2015.152.
[9] A. Salem, Y. Zhang, M. Humbert, P. Berrang, M. Fritz, and M. Backes, “ML-leaks: Model and data independent membership inference attacks and defenses on machine learning models,” 2018, arXiv:1806.01246. [Online]. Available: http://arxiv.org/abs/1806.01246.
[10] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” 2014, arXiv:1409.1556. [Online]. Available: http://arxiv.org/abs/1409.1556.
Y. Uchida, Y. Nagai, S. Sakazawa, and S. Satoh, “Embedding watermarks into deep neural networks,” in Proc. ACM Int. Conf. Multimedia Retr., New York, NY, USA, Jun. 2017, pp. 269–277, doi: 10.1145/3078971.3078974.

Y. Adi, C. Baum, M. Cisse, B. Pinkas, and J. Keshet, “Turning your weakness into a strength: Watermarking deep neural networks by backdooring,” in Proc. 27th USENIX Secur. Symp., 2018, pp. 1615–1631.

E. Le Merrer, P. Perez, and G. Trédan, “Adversarial frontier stitching for remote neural network watermarking,” 2017, arXiv:1711.01894. [Online]. Available: http://arxiv.org/abs/1711.01894.

B. Darvish Rouhani, H. Chen, and F. Koushanfar, “DeepSigns: A generic watermarking framework for IP protection of deep learning models,” 2018, arXiv:1804.00750. [Online]. Available: http://arxiv.org/abs/1804.00750.

J. Guo and M. Potkonjak, “Watermarking deep neural networks for embedded systems,” in Proc. Int. Conf. Computer-Aided Design, Nov. 2018, pp. 1–8.

J. Zhang, Z. Gu, J. Jang, H. Wu, M. P. Stoecklin, H. Huang, and I. Molloy, “Protecting intellectual property of deep neural networks with watermarking,” in Proc. Asia Conf. Comput. Commun. Secur. (ASIACCS), 2018, pp. 159–172.

F. Güneş, R. Wolfinger, and P.-Y. Tan, “Stacked ensemble models for improved prediction accuracy,” in Proc. Static Anal. Symp., 2017, pp. 1–19.

X. Cao, J. Jia, and N. Zhengjiang Gong, “IPGuard: Protecting intellectual property of deep neural networks via fingerprinting the classification boundary,” 2019, arXiv:1910.12903. [Online]. Available: http://arxiv.org/abs/arXiv:1910.12903.

R. Kaur and A. Kaur, “Digital signature,” in Proc. Int. Conf. Comput. Sci., Sep. 2012, pp. 295–301, doi: 10.1109/ICCSC.2012.25.

R. He, W. Sun Lee, H. Tou Ng, and D. Dahlmeier, “Exploiting document knowledge for aspect-level sentiment classification,” 2018, arXiv:1806.04346. [Online]. Available: http://arxiv.org/abs/1806.04346.

R. He, W. Sun Lee, H. Tou Ng, and D. Dahlmeier, “An interactive multi-task learning network for end-to-end aspect-based sentiment analysis,” 2019, arXiv:1906.06906. [Online]. Available: http://arxiv.org/abs/arXiv:1906.06906.

L. Holmstrom and P. Koistinen, “Using additive noise in back-propagation training,” IEEE Trans. Neural Netw., vol. 3, no. 1, pp. 24–38, Jan. 1992.

A. Neelakantan, L. Vilnis, Q. V. Le, I. Sutskever, L. Kaiser, K. Kurach, and J. Martens, “Adding gradient noise improves learning for very deep networks,” 2015, arXiv:1511.06807. [Online]. Available: http://arxiv.org/abs/arXiv:1511.06807.

G. Hinton, O. Vinyals, and J. Dean, “Distilling the knowledge in a neural network,” 2015, arXiv:1503.02531. [Online]. Available: http://arxiv.org/abs/arXiv:1503.02531.

S. Robinson, “Still guarding secrets after years of attacks, RSA earns accolades for its founders,” Siam News vol. 36, no. 5, pp. 1–4, 2003.

K. James, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, and K. Milan, “Overcoming catastrophic forgetting in neural networks,” Proc. Nat. Acad. Sci. USA, vol. 114, no. 13, pp. 3521–3526, Mar. 2017.

Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” Proc. IEEE, vol. 86, no. 11, pp. 2278–2344, Nov. 1998.

A. Krizhevsky et al., Learning Multiple Layers of Features From Tiny Images. Citeseer, 2009.

T. Chen, I. Goodfellow, and J. Shlens, “Net2Net: Accelerating learning via knowledge transfer,” 2015, arXiv:1511.05641. [Online]. Available: http://arxiv.org/abs/arXiv:1511.05641.

M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, and M. Devin, “Tensorflow: A system for large-scale machine learning,” in Proc. 12th USENIX Symp. Operating Syst. Design Implement. (OSDI), 2016, pp. 265–283.

X. Xu et al.: “Identity Bracelets” for DNNs

Yaqin Li received the B.E. and M.E. degrees in computer science and the Ph.D. degree in control science and engineering from the Huazhong University of Science and Technology, in 2002, 2007, and 2012, respectively.

She joined the College of Mathematics and Computer Science, Wuhan Polytechnic University, Wuhan, China, where she is currently an Associate Professor. Her research interests include pattern recognition, machine learning, and image processing.

Cao Yuan received the B.E. and M.E. degrees in computer science and the Ph.D. degree in radio physics from Central China Normal University, in 2002, 2006, and 2009, respectively.

After that, he joined the College of Mathematics and Computer Science, Wuhan Polytechnic University, Wuhan, China, where he is currently an Associate Professor. His research interests include pattern recognition, machine learning, and image processing.

---