A deep learning framework supporting model ownership protection and traitor tracing

Guowen XU
University of Electronic Science and Technology of China

Hongwei LI

Yuan ZHANG

Xiaodong LIN
University of Guelph

Robert H. DENG
Singapore Management University, robertdeng@smu.edu.sg

See next page for additional authors

Published in 2020 IEEE 26th International Conference on Parallel and Distributed Systems (ICPADS): Proceedings, pp. 438-446.
DOI: 10.1109/ICPADS51040.2020.00084.

A Deep Learning Framework Supporting Model Ownership Protection and Traitor Tracing

Guowen Xu[1], Hongwei Li[1](Corresponding author), Yuan Zhang[1], Xiaodong Lin[2], Robert H. Deng[3], and Xuemin (Sherman) Shen[4]
[1]University of Electronic Science and Technology of China, Chengdu 611731, China
[2] University of Guelph, 50 Stone Rd E, Guelph, ON N1G 2W1, Canada
[3]Singapore Management University, 178902 Singapore
[4]University of Waterloo, Waterloo Ontario N2L 3G1 Canada
Email: guowen.xu@foxmail.com; hongweili@uestc.edu.cn; zy_loye@126.com; xlin08@uoguelph.ca; robertdeng@smu.edu.sg; sshen@uwaterloo.ca

Abstract—**Cloud-based deep learning (DL) solutions have been widely used in applications ranging from image recognition to speech recognition. Meanwhile, as commercial software and services, such solutions have raised the need for intellectual property rights protection of the underlying DL models. Watermarking is the mainstream of existing solutions to address this concern, by primarily embedding pre-defined secrets in a model's training process. However, existing efforts almost exclusively focus on detecting whether a target model is pirated, without considering traitor tracing. In this paper, we present SecureMark_DL, which enables a model owner to embed a unique fingerprint for every customer within parameters of a DL model, extract and verify the fingerprint from a pirated model, and hence trace the rogue customer who illegally distributed his model for profits. We demonstrate that SecureMark_DL is robust against various attacks including fingerprints collusion and network transformation (e.g., model compression and model fine-tuning). Extensive experiments conducted on MNIST and CIFAR10 datasets, as well as various types of deep neural network show the superiority of SecureMark_DL in terms of training accuracy and robustness against various types of attacks.**

Index Terms—**Ownership Protection; Traitor Tracing; Watermarking; Deep Learning; Cloud Computing.**

I. Introduction

Deep neural networks (DNNs) such as the convolutional neural network (CNN), Residual Network, and recurrent neural network (RNN) have found numerous applications [1], [2]. To facilitate deployment of DNNs, many tech giants, such as Google, Amazon, and Microsoft, have offered Machine Learning as a Service (MLaaS). As a fast-growing business service, MLaaS provides professional, tailored, and satisfying deep learning models with a negligible price compared with training target models by the customers themselves.

However, MLaaS raises broad concerns of service providers about protection of their models' intellectual property rights. In fact, for a service provider, building a deep learning model typically incurs substantial costs to process large amount of training samples. As a consequence, these well-constructed models are considered commercial software with intellectual property rights, and should be properly protected to maintain the owner's competitive advantage in the market. On the other hand, a malicious customer may intentionally use the purchased model with some despicable purposes, for example, sell it for profits in the black market. Thus, it is crucial to have ownership protection mechanisms built in deep learning models before releasing them to customers.

Watermarking techniques have been applied in DNNs to provide verification channels for the models' intellectual property rights. For example, *Adi et al.* [3] design a robust watermarking algorithm by exploiting the potential vulnerabilities in DNNs, and utilize them as backdoors to embed watermarks. *Rouhani et al.* [4] present DeepSigns, an end-to-end watermark embedding framework to embed secrets in the probability density function of target layers. *Namba et al.* [5] also propose a robust watermarking of DNNs with exponential weighting, which exponentially increases the magnitude of the weights that affects the prediction result significantly during the watermark embedding process. By doing so, the resulting model becomes tolerant of both model modification and query modification without sacrificing prediction performance. Other works, like [6], [7], [8], [9], [10], achieve similar purposes with diverse techniques, such as adversarial examples [9], [7], regularization parameters [6], and backdoors based techniques [8], [10].

In general, the existing works exclusively focus on detecting whether a target model infringes copyright, and rarely consider tracking of traitors, i.e., rogue authorized customers who violate copyright protection policies by modifying and distributing pirate models for the money. In reality, a service provider may sell well-trained models to a large number of customers. Without embedding traitor tracing mechanisms, it will be extremely difficult for subsequent forensics when a dispute arises. However, designing a deep learning framework which enables end-to-end proof of ownership and unbiased traitor tracing is not a trivial task. First, watermarks should be robust against various watermarking removal attacks. Adversaries may attempt to invalidate the embedded watermark by modifying the model without significantly affecting its performance. Such network transformation attacks (e.g., model compression and model fine-tuning) have been demonstrated to be effective in breaking the integrity of watermarks [11], [12], [13]. Moreover, in contrast to embedding a uniform watermark for copyright protection, traitor tracing requires the service provider to embed a unique watermark (i.e., fingerprint) in DNNs for every customer. This distinction also spawns a new type of attacks called fingerprint collusion [14], in which multiple customers who have purchased the same model with different fingerprints may collude to generate an unmarked model. Although the fingerprint collusion attack is common in the multimedia field [15], [8], it poses new technical challenges in DNNs since we need to ensure the high accuracy of the model in the strategy to defend against such attacks.

In this paper, we present SecureMark_DL, a deep learning framework with end-to-end ownership protection. To provide proof of ownership as well as traitor tracing, we present an anti-collusion algorithm that embeds unique fingerprints for every customer within parameters (i.e., weights) of DNNs. The following summarizes the contributions of the paper.

- We embed a unique fingerprint for each customer into the target parameters of DNNs, by exploiting the high capacity of parameters in DNNs. In this way, SecureMark_DL provides a channel for ownership verification or traitor tracing while with only a slight impact on the prediction performance.
- We design a two-tier fingerprint scheme, which takes customers social networking attributes into consideration in the process of fingerprint generation. This feature facilitates a service provider to quickly find suspicious customer's groups and then trace the rogue customer.
- We conduct extensive experiments on real-world datasets to demonstrate that SecureMark_DL is robust against various attacks including fingerprint collusion and network transformation (e.g., model compression and model fine-tuning).

The remainder of this paper is organized as follows. In Section II, we describe the background and problem statement. Then we present the technical details of SecureMark_DL in Section III. Performance evaluation is discussed in Section IV. Finally, we conclude the paper in Section V.

II. PROBLEM DEFINITION

In this section, we first introduce the basic concept of deep learning. Then we describe the problem statement in our scenario, which consists of deployed model, attack model, and desired goals.

A. The basics of Deep Learning

In general, a DNN usually consists of one input layer, one or more hidden layers, and one output layer. Fig. 1 shows a simple fully connected neural network (FN) containing one input layer, one hidden layer, and one output layer. The *fully connected* means that any two neurons between adjacent layers are connected with parameters (called set Θ) to each other. Here we use the FN as an example to review how a DNN is trained. Specifically, given an input $\boldsymbol{x} = \{x_1, x_2, x_3\}$, the output of the DNN can be denoted as $f_\Theta(\boldsymbol{x}) = \hat{\boldsymbol{y}} = \{\hat{y_1}, \hat{y_2}\}$, where $f_\Theta : \mathcal{X} \to \mathcal{Y}$ is the deep learning model parameterized

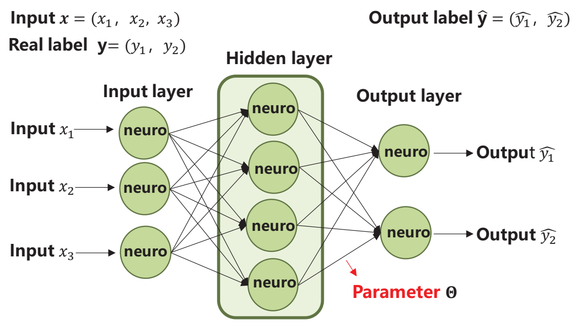

Fig. 1: An example of a fully connected neural network

by Θ. For classification problems, $\mathcal{X}$ represents the high-dimensional vector space, and $\mathcal{Y}$ is the space of the classes. Therefore, the goal of training a DNN is to find the optimal parameters that accurately reflect the relationship between $\boldsymbol{x}$ and $\boldsymbol{y}$, and ultimately make the DNN output $\hat{\boldsymbol{y}}$ infinitely close to the real label $\boldsymbol{y}$. To achieve this, assuming that the training set is $\mathcal{D} = \{(\boldsymbol{x_i}, \boldsymbol{y_i}); i = 1, 2, \cdots, V\}$, we first exploit a loss function $l(\boldsymbol{y}, f_\Theta(\boldsymbol{x}))$ to measure the difference between the real label $\boldsymbol{y}$ and predicted labels, typically, $l(\boldsymbol{y}, f_\Theta(\boldsymbol{x})) = ||\boldsymbol{y} - f_\Theta(\boldsymbol{x})||_2$, where $|| \cdot ||_2$ is the l_2 norm of a vector. Then, training a DNN is to minimize a optimization function as follows.

$$\min_{\Theta} \frac{1}{|\mathcal{D}|} \sum_{(\boldsymbol{x},\boldsymbol{y}) \in \mathcal{D}} l(\boldsymbol{y}, f_\Theta(\boldsymbol{x})) \tag{1}$$

The above function is usually optimized by the stochastic gradient descent (SGD)[16] algorithm as below.

$$\Theta^{j+1} \leftarrow \Theta^j - \eta \nabla l(\mathcal{D}, \Theta^j) \tag{2}$$

where Θ^j represents the parameters after the j-th iteration. η represents the learning rate, and $\nabla l(\mathcal{D}, \Theta^j)$ denotes the partial derivative of the parameter Θ^j on the set $\mathcal{D}$.

B. Deployed Model

As shown in Fig. 2, SecureMark_DL consists of *a cloud server* (also named model's owner or service provider) and M *customers*. Let Θ denote a DNN model that the cloud server wants to tailor for customer i, $i \in \{1, 2, \cdots M\}$. The server first initializes Θ and a unique fingerprint F_i for customer i. Then it trains the network Θ based on the criteria shown in Section II-A(a slightly different explained in the following Sections), and embeds the fingerprint F_i into the target parameters. Finally, the cloud server releases the model Θ to customer i once the training reaches the expected convergence conditions. In the event that a suspicious pirated model is spotted, the server can extract a fingerprint from the model, and verify the model's ownership or trace the traitor.

C. Attack Model

Consistent with most existing works [17], [4], we assume that customers are malicious in our system. In fact, when the

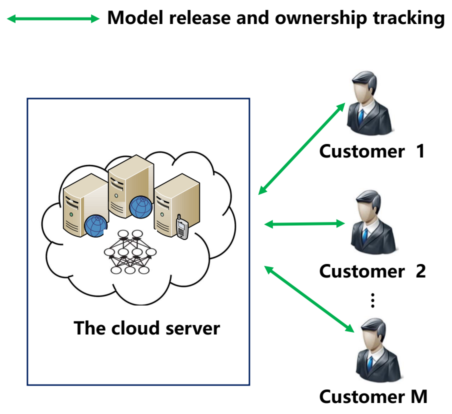

Fig. 2: Deployed Model

server publishes a trained model to an authorized customer, he may abuse the model and even collude with multiple customers in the attempt to erase the fingerprint embedded in the model. Specifically, we consider the following possible attacks launched by customers.

(1) *Model Fine-tuning*. It is usually cost-effective to fine-tune a pre-trained model so that it can be explored for similar applications, while learning a new model from scratch is often very time-consuming. Model fine-tuning can be performed by honest customers for model transformations, or adversaries for purposeful attacks. The parameters of the model usually change during this process.

(2) *Model Compression*. Model compression is also a standard optimization technique designed to reduce computational overhead by cropping a trained model, while not significantly reducing the accuracy of the compressed model. However, malicious customers may misuse optimization to remove fingerprints embedded in the parameters, e.g., to prune hidden layers with fingerprints in the model.

(3) *Collusion Attack*. Since traitor tracing requires the service provider to embed a unique fingerprint in a DNN model for each customer, malicious customers who have purchased the same model with different fingerprints may collude, such as taking the average of the parameters to generate an unmarked model.

D. Designed Goals

Based on the threat model, we formulate the design goals of SecureMark_DL as follows.

(1) *Utility*. Intuitively, we need to adjust the training criteria of a DNN model to cater for embedding of fingerprints, which inevitably has a negative impact on model accuracy. Hence, SecureMark_DL should guarantee that the trained model has a negligible reduction to accuracy.

(2) *Security*. First, for accurate tracking, the fingerprint embedded in each released model should be unique and easily retrieved by the server, which enables the cloud server to quickly track inappropriate use of published models conducted by any specific customer. Second, the proposed fingerprint embedding method should try to avoid false negatives, which means that SecureMark_DL can detect a traitor with a high probability. Third, the proposed fingerprint embedding method should try to avoid false positives, which means that the probability of accusing an innocent customer is very low.

(3) *Scalability*. To cope with the growing Internet user population, SecureMark_DL should be scalable in the number of customers it can support. This requires high efficiency of traitor tracking even if a model is released to a large number of customers. Also, to prevent fingerprints from being easily removed, the proposed fingerprint-embedding method should be able to embed a large amount of data in the target's DNN with slight loss of model accuracy.

(4) *Robustness*. SecureMark_DL should be robust against various attacks including fingerprints collusion and network transformation (i.e., model compression and model fine-tuning).

III. PROPOSED SCHEME

In this section, we describe the technical detail of SecureMark_DL. At a high-level view, we will describe how to generate a unique fingerprint for each customer, and the principle of embedding the fingerprint in the parameters of target DNN's layers. It is worth noting that each customer's fingerprint F_i consists of two parts, namely, F_i=*(community relationship code*$||$ *customer identity code)*. The former is mainly to improve the speed of finding suspicious pirates given an unknown model, while the latter is to correctly track traitors. Finally, we present the details of fingerprint extraction and verification.

A. Fingerprint Generating and Embedding

In social networks, *Apicella et al.* [18] have demonstrated that people in the same or similar communities are more likely to be connected to do the same thing, and the probability is very low for people in different communities. Based on this, we design a two-tier fingerprint structure, which means that each customer's fingerprint F_i consists of two parts, namely, F_i=($p_i||$ u_i). The former is a community relationship code represented the social network structure of customer i. Customers in the same or neighboring communities will be assigned similar community codes, and community codes between unrelated customers will be extremely different. This property will facilitate the server to quickly find suspicious customer groups. The latter is a customer identity code used to uniquely identify the customer i. We exploit the balanced incomplete block design (BIBD) [19] to build the anti-collusion identity code (ACC) for each customer, which guarantees that the composite code of any K or less user's identity codes is unique. This property allows the server to identify a group of K or fewer colluders accurately.

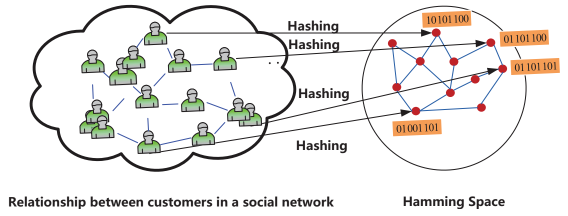

Fig. 3: Transforming the relationship among customers to Hamming space

1) Generating Community Relationship Code: The relationship among customers in a social network is usually high-dimensional, as the intimacy between them is determined by multiple factors such as geographic distance, social relationships, and blood relatives. If we directly use high-dimensional-vectors to represent the relationships between customers and embed them (as community codes) into the parameters of the DNN, excessive parameters will be required to embed fingerprint information, which inevitably reduces the accuracy of the original model. Moreover, it is also inefficient to search for similar community codes in high dimensional vectors. To address this problem, as shown in Fig. 3, we resort to the primitive Neighborhood Preserving Hashing (NPH) [20]. NPH can map high-dimensional objects to low-dimensional binary vectors lying on a Hamming space. Additionally, NPH preserves the neighborhood structure of objects. Since the converted community codes are binary, the Hamming distance between one code and any other code can be efficiently calculated by XOR and bit-count operation. Specifically, we first build an adjacency matrix to represent the social relationship between customers as below.

$$A_{ij} = \begin{cases} 1 : \text{if } \mathcal{O}_i \in \mathcal{C}(\mathcal{O}_j) \text{ or } \mathcal{O}_j \in \mathcal{C}(\mathcal{O}_i) \\ 0 : \text{otherwise} \end{cases} \tag{3}$$

where $i, j = 1, 2, \cdots M$ represents the number of customers. $\mathcal{C}(\mathcal{O}_j)$ is the set of customers who contact with j-th customer $\mathcal{O}_j$. A_{ij} can also be another positive value describing the intimacy between customers. In the following, we first introduce the underlying technologies of NPH, i.e., Non-negative Matrix Factorization (NMF) [21] and Locally Linear Embedding(LLE) [22], and then use them to generate the community relationship code for each customer based on the adjacency matrix.

NMF is used to approximately decompose a high-dimensional matrix into the product of two low-dimensional matrices. Specifically, given a target matrix $B = [b_1, b_2, \cdots b_M] \in \mathrm{R}^{E\times M}$, each column of B is a vector, the goal of NMF is to find two non-negative matrices $G = [g_1, g_2, \cdots, g_T] \in \mathrm{R}^{E\times T}$ and $P = [p_1, p_2, \cdots, p_M] \in \mathrm{R}^{T\times M}$ whose product is a good approximation of B. The specific optimization algorithm is as follows:

$$\begin{aligned} &\min_{G,P} ||B - GP||_F^2 \\ &\text{s.t.} \quad G \geq 0, \; P \geq 0 \end{aligned} \tag{4}$$

where $||\cdot||_F$ is the Frobenius norm of a matrix. In social networks, b_i can be seen as the properties of customer i, such as gender, age, and hobbies. G is a dictionary matrix, and p_i is a low-dimensional vector used to represent b_i.

Although NMF preserves semantic information when converting each b_i to p_i, it may lose the neighbor structure between different b_i. This means that customers from the same community may be far apart. We exploit the technology of Locally Linear Embedding(LLE) [22] to address this problem. LLE provides a way for an object to be reconstructed from its neighbors in a low-dimensional subspace, if it can be reconstructed by neighbors in the original high-dimensional space. Given a b_i, LLE first minimizes the reconstruction error as below.

$$\begin{aligned} &\min_{\omega} \sum_{i=1}^{i=M} |b_i - \sum_{b_j \in \mathcal{N}(b_i)} \omega_{ij} b_j|_2^2 \\ &\text{s.t.} \quad \sum_{b_j \in \mathcal{N}(b_i)} \omega_{ij} = 1 \end{aligned} \tag{5}$$

where $\mathcal{N}(b_i)$ denotes the k-nearest neighbors of b_i. Then, assuming that p_i is the representation of b_i in a low-dimensional space, in order to preserve the neighbor relationship of b_i, LLE minimizes the following function in the phase of selecting p_i.

$$\begin{aligned} &\min_{P} \sum_{i=1}^{i=M} |p_i - \sum_{i} \omega_{ij} p_j|_2^2 \\ &\text{s.t.} \quad P \geq 0 \end{aligned} \tag{6}$$

This ensures that if b_i and b_j are from the same or similar communities, their corresponding community relationship codes are also very close.

We now introduce the technical details of NPH to find the community relationship code for each customer. Concretely, we first rewrite Eqn.(4) as below.

$$\begin{aligned} &\min_{G,P} \sum_{i=1}^{i=M} |b_i - Gp_i|^2 \\ &\text{s.t.} \quad G \geq 0, \; P \geq 0 \end{aligned} \tag{7}$$

Then, by introducing the idea of LLE, Eqn.(7) is modified to

$$\begin{aligned} &\min_{G,P} \sum_{i=1}^{i=M} |b_i - \sum_{j} \omega_{ij} G p_i|^2 \\ &\text{s.t.} \quad G \geq 0, \; P \geq 0 \end{aligned} \tag{8}$$

where j satisfies the condition of $b_j \in \mathcal{N}(b_i)$. We further reformulate the above formula as follow.

$$\begin{aligned} &\min_{G,P} ||B - GPW^T||_F^2 \\ &\text{s.t.} \quad G \geq 0, \; P \geq 0 \end{aligned} \tag{9}$$

By solving Eqn.(9), we can get the binary vector p_i to represent b_i and preserve the neighbor relationship of the original data.

2) Solving Algorithm: It is known from the optimization theory that Eqn.(9) is not a convex function under the domain consisting of G and P together. However, Eqn.(9) is alternatively convex on G and P, thereby many numerical optimization methods such as gradient descent and projected gradient [23] can be used to seek local minima. In this paper, we adopt projected gradient to solve the above objective function because of its fast convergence characteristics. Specifically, let $\mathcal{O} = ||B - GPW^T||_F^2$, we have

$$\begin{aligned}\mathcal{O} &= tr[(B - GPW^T)(B - GPW^T)^T] \\ &= tr(BB^T) - 2tr(BWP^TG^T) + tr(GPW^TWP^T)\end{aligned} \tag{10}$$

where $tr(\cdot)$ is the trace of a matrix. Based on the natures of $tr(AB) = tr(BA)$ and $tr(A) = tr(A^T)$, we can compute the partial derivative of the objective function for G and P as below.

$$\begin{aligned}\frac{\partial \mathcal{O}}{\partial G} &= 2GPW^TWP^T - 2BWP^T \\ \frac{\partial \mathcal{O}}{\partial P} &= 2G^TGPW^TW - 2G^TBW\end{aligned} \tag{11}$$

Then, we consider the objective function as two separate sub-problems, and update one of the variables while fixing the value of the other one. Specifically, given $G_{ij}^1 \geq 0$, $P_{ij}^1 \geq 0$, for any i, j, the two sub-problems are shown as follows.

$$\widetilde{f}(P^{k+1}) = \arg\min_{P\geq 0} f(G^k, P^k) \tag{12}$$

$$\widetilde{f}(G^{k+1}) = \arg\min_{G\geq 0} f(G^k, P^{k+1}) \tag{13}$$

Here we take Eqn.(9) as an example to show how to solve the above problems. We first rewrite Eqn.(9) as

$$\begin{aligned}&\min_P \widetilde{f}(P) = ||B - GPW^T||_F^2 \\ &\text{s.t.} \quad P_{i,j} \geq 0, \forall i, j\end{aligned} \tag{14}$$

The update of current solution p_i^k to p_i^{k+1} with projected gradient is shown as follows.

$$p_i^{k+1} = P[p_i^k - \alpha^k \nabla \widetilde{f}(p_i^k)] \tag{15}$$

where p_i, $i \in [1, M]$ is a column of P, and we have

$$P_{ij} = \begin{cases} 0 : p_{ij} \leq 0 \\ p_{ij} : \text{otherwise} \end{cases} \tag{16}$$

In addition, for any α, $p_i^{k+1} = P[p_i^k - \alpha^k \nabla \widetilde{f}(p_i^k)]$ should satisfy the restriction

$$\widetilde{f}(p_i^{k+1}) - \widetilde{f}(p_i^k) = \sigma \nabla \widetilde{f}(p_i^k)^T (p_i^{k+1} - p_i^k) \tag{17}$$

where σ is a small positive constant. Based on the Taylor approximation of $\widetilde{f}$ at p_i^k up to second order. i.e.,

$$\widetilde{f}(p_i^{k+1}) \approx \widetilde{f}(p_i^k) + \nabla \widetilde{f}(p_i^k)^T \upsilon + \frac{1}{2}\upsilon^T \nabla^2 \widetilde{f}(p_i^k)\upsilon \tag{18}$$

where $\upsilon = (p_i^k)^T(p_i^{k+1} - p_i^k)$, and $\nabla^2 \widetilde{f}(\cdot)$ is the Hessian matrix of $\widetilde{f}(P)$. Eqn.(17) can be modified as

$$(1 - \sigma)\nabla \widetilde{f}(p_i^k)^T \upsilon + \frac{1}{2}\upsilon^T \nabla^2 \widetilde{f}(p_i^k)\upsilon \leq 0 \tag{19}$$

Once the optimal α is obtained, we use Eqn.(15) to iteratively update P until a stationary P is obtained. Then, we can find the optimal G with a similar method as follows.

$$\begin{aligned}&\min_G \widetilde{f}(G) = ||B^T - WP^TG^T||_F^2 \\ &\text{s.t.} \quad G_{i,j} \geq 0, \forall i, j\end{aligned} \tag{20}$$

3) Kernel Trick: The above problem is easily solved when given B. However, in reality, it is difficult for the server to obtain the properties of each customer (i.e., $B = [b_1, b_2, \cdots b_M]$), but the connection relationship between different customers (i.e., the adjacency matrix A) can be relatively easily obtained. Based on this, we introduce the kernel trick to solve this problem.

Suppose a map here maps Euclidean space $\mathtt{R}^M$ to Hilbert space $\mathtt{Z}$, i.e., $\phi : \mathtt{R}^M \rightarrow \mathtt{Z}$. Let $\phi(B)$ represent the matrix over the Hilbert space, where $\phi(B) = [\phi(b_1), \phi(b_2), \cdots, \phi(b_M)]$. Then, the objective function Eqn.(9) can be rewritten as below.

$$\begin{aligned}&\mathcal{O} = \min_{G,P} ||\phi(B) - GPW^T||_F^2 \\ &\text{s.t.} \quad G \geq 0, \ P \geq 0\end{aligned} \tag{21}$$

As described before, Eqn.(21) can be rewritten as

$$\begin{aligned}\mathcal{O} &= tr[(\phi(B) - GPW^T)(\phi(B) - GPW^T)^T] \\ &= tr(Q) - 2tr(\phi(B)WP^TG^T) + tr(GPW^TWP^T)\end{aligned} \tag{22}$$

where $Q = \phi(B)^T\phi(B)$ represents the kernel matrix. Then, to generalize NPH, we first define the dot product in the Hilbert space as follows:

$$Q(b_i, b_j) = (\phi(b_i) \cdot \phi(b_j)) = \phi(b_i)^T\phi(b_j)$$

Considering that G can be regarded as a linear combination of $\phi(b_1)$, $\phi(b_2)$, $\cdots$, $\phi(b_M)$, hence G can be denoted as $G = \phi(B)C^T$, where C is a coefficient matrix. We use $\phi(B)C^T$ instead of G in Eqn.(22), and we can get

$$\begin{aligned}&\mathcal{O} = \arg\min_{C,P}(tr(Q) - 2tr(QC^TPW) + tr(WP^TCQC^TPW^T)) \\ &\quad \text{s.t.} \quad C \geq 0, \ P \geq 0\end{aligned} \tag{23}$$

As described in Eqn.(23), we can observe that B is not required to solve the above objective function. In fact, the kernel matrix Q constructed on matrix B represents the similarity among data. Therefore, it can be thought of as a black box function on B to generate a similarity matrix. Since the adjacency matrix A is also used to represent the similarity between data, we can use the adjacency matrix instead of the kernel function.

4) Generating Customer Identity Code: Due to the collision of community relationship codes, each customer cannot be uniquely represented by this type of code alone. We assign each customer an additional unique customer identity code. Specifically, the technology of balanced incomplete block design (BIBD) [19] is adopted to build the anti-collusion (ACC) identity code for each customer, which guarantees that the composite code of any K or less user identity codes is unique. This property allows the server to identify a group of K or fewer colluders accurately. In brief, a (g, k, γ)-BIBD is a pair $(\mathcal{F}, \mathcal{A})$ where $\mathcal{F}$ is a set consisting of g-dimensional elements. $\mathcal{A}$ is the collection of k-element subsets (blocks) which satisfies that each pair of elements of $\mathcal{F}$ appear together exactly γ times in the subsets. The (g, k, γ)-BIBD has a total of $t = \gamma(g^2 - g)/(k^2 - k)$ blocks (k is the block sizes) which can be represented by its corresponding incidence matrix $\mathtt{C}_{g\times t}$.

Concretely, the elements in the incidence matrix have the following connections:

$$c_{ij} = \begin{cases} 1 : \text{if } i^{th} \textit{ value occurs in } j^{th} \textit{ block} \\ 0 : \text{otherwise} \end{cases} \tag{24}$$

If we set $\gamma = 1$ and assign the bit complement of the columns of incidence matrix as the code-vectors, the resulting $(g, k, 1)$-BIBD code is $(k-1)$-resilient and supports up to $M = t$ customers. Hence, given the designed incidence matrix $\mathtt{C}_{g\times M}$ and an orthogonal matrix $\mathcal{F}_{g\times g} = (\mathtt{f}_1, \mathtt{f}_2, \cdots, \mathtt{f}_g)$, we first compute a coefficient matrix $\mathcal{E}_{g\times M}$ with the linear mapping $e_{ij} = 2c_{ij} - 1$. Then, the customer identity code u_j of each customer $j, j = 1, 2, \cdots M$ is generated as below.

$$u_j = \sum_{i=1}^{i=g} e_{ij}\mathtt{f}_j \tag{25}$$

5) Embedding Fingerprint: We have introduced how to generate a community relationship code p_j and a customer identify code u_j for each customer j. Given a fingerprint F_j=($p_j||\ u_j$), we now present the technical details of embedding F_j into the parameters of Θ. As described before, existing works exclusively focus on ownership proof, and rarely consider tracking of traitors. To combat this, we embed F_j into the parameters of Θ using a customized regularization loss during DNN training. The intuition behind is that there are abundant redundancies existing in DNNs' parameters due to its high dimensionality. We use this redundancy and make embedded fingerprints as an auxiliary task during the training process. Therefore, the embedded fingerprint is integrated as an integral part of the parameters, which ensures that adversaries cannot remove fingerprints without dramatically compromising the performance of the labeled model. Concretely, we first select some hidden layers (i.e., target layers) in Θ that are suitable for embedding fingerprints, where the parameters belonging to the target layers are represented as $\theta^{sub} \in \Theta$. As a result, we add a new term $\mu MSE(F_j - \mathbf{X}\Theta^{sub})$ into the original objective function (refer to Eqn.(1)), and embed the F_j by requiring the server to train the following objective function.

$$\min_{\Theta} \frac{1}{|\mathcal{D}|} \sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}} l(\boldsymbol{y}, f_\Theta(\boldsymbol{x})) + \mu MSE(F_j - \mathbf{X}\Theta^{sub}) \tag{26}$$

where MSE is the mean square error function, and μ represents the embedding weight. $\mathbf{X}$ is the secret random projection matrix generated by the server, and Θ^{sub} is the flattened averaged parameters of θ^{sub} for embedding F_j.

As a proof-of-concept analysis, we embed the F_j into the target layers of Θ. Specifically, we know that the parameters θ^{sub} is 3D tensor, i.e., $\theta^{sub} \in \mathrm{R}^{S\times\delta\times\eta}$, where S is the length of the input of Θ, δ is the kernel size and η is the number of channels used in Θ. If the parameters in the continuous layers are rearranged accordingly, the order of the filtering channels will not change the output of the neural network. Hence, we take the average of θ^{sub} over all channels and stretch the resulting tensor into a vector $\Theta^{sub} \in \mathrm{R}^{\beta}$, where $\beta = S \times \delta$. Θ^{sub} is multiplied with a secret random matrix $\mathbf{X} \in \mathrm{R}^{(T+g)\times\beta}$ for comparing with the $(T+g)$-dimensional binary vector F_j. To embed the F_j into the target layers, we add the additional term $\mu MSE(F_j - \mathbf{X}\Theta^{sub})$ into the original objective function, whose goal is enforcing the difference between F_j and $\mathbf{X}\Theta^{sub}$ to be minimized during training.

B. Fingerprint Extraction and Verification

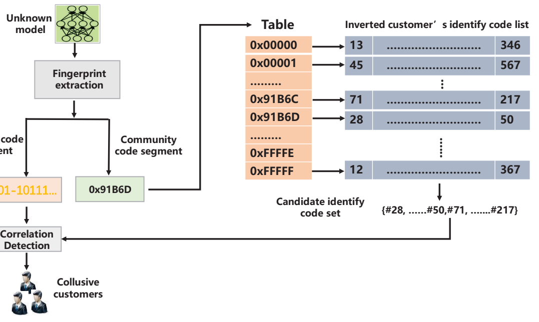

Fig. 4: Example of fingerprint extraction and verification

For model ownership verification, the server needs to perform fingerprint extraction and verification on the suspicious model to discover possible piracy. Generally speaking, there are two methods of fingerprint extraction, one is blind extraction and the other is non-blind extraction. Our SecureMark_DL uses the latter because it has a higher fingerprint detection rate. Reviewing the fingerprint embedding criteria in our SecureMark_DL, i.e., enforcing the difference between F_j and $\mathbf{X}\Theta^{sub}$ to be minimized during training. The server only needs to obtain the parameters θ^{sub} of the target layers during the extraction process, and computes the flattened averaged version Θ^{sub}. As a result, the fingerprint F_j is recovered with $F_j = \mathbf{X}\Theta^{sub}$, where $\mathbf{X}$ is the random projection matrix selected by the server. As described before, each customer's fingerprint F_i consists of community relationship code and customer identity code, namely, F_j=($p_j||\ u_j$). In order to quickly find traitors, as shown in Fig. 4, we first pick up the community relationship code fragments of F_j and compare it with the hash table to find similar communities. Since customers in the same or neighboring communities will be assigned similar community codes, the community code synthesized by multiple malicious customers is still small. Then, we treat all customers in selected communities as potential traitors, and exploit the property of customer identity code to find the collusive customers.

IV. PERFORMANCE EVALUATION

In this section, we conduct experiments to evaluate the performance of our SecureMar_DL. In our experiments, the Cloud is simulated with a Lenovo server which has Intel(R) Xeon(R)E5-2620 2.10GHZ CPU, 16GB RAM, 256SSD, 1TB mechanical hard disk and runs on the Ubuntu 18.04 operating system. Besides, each customer is simulated by one

TABLE I: Classification Accuracy

Dataset	CIC-length	CRC-length	Baseline	Accuracy
MNIST	49	20	97.63%	97.62%
		30		97.62%
		40		97.61%
		50		97.59%
		60		97.59%
MNIST	121	20	97.63%	97.57%
		30		97.55%
		40		97.55%
		50		97.53%
		60		97.53%
CIFAR-10	49	20	91.43%	91.42%
		30		91.42%
		40		91.41%
		50		91.41%
		60		91.40%
CIFAR-10	121	20	91.43%	91.40%
		30		91.40%
		40		91.39%
		50		91.39%
		60		91.38%

HP i7 8550u notebook, which has one 1.8GHZ CPU, 8G flash memory, 256SSD and 500G mechanical hard disk, and assigned with 64-bit Windows 10 operating system. We train SecureMar_DL with two different datasets, i.e., MNIST[1] and CIFAR-10[2], where we adopt a five-layer ReLU-based fully-connected feed-forward neural network with 900 hidden units as the training model in the MNIST experiment, and the VGG-19 [24] with batch-normalization is adopted as the training model in the CIFAR-10 experiment.

A. Classification Accuracy

We first analyze the classification accuracy of our SecureMar_DL. As discussed before, we need to adjust the training criteria of a DNN model to cater for embedding of fingerprints, which inevitably has a negative impact on model accuracy. To quantitatively analyze it, we train SecureMar_DL under two datasets (i.e., MNIST and CIFAR-10) with fingerprints of different lengths, where the length of CIC (Customer Identity Code) is 49 and 121 bits, respectively, and the length of CRC (Community Relationship Code) is selected from 20 to 60 bits. The MNIST dataset contains 60000 images of handwritten digits, where 50000 samples are training samples and the rest are test samples. CIFAR-10 is a benchmark dataset for object recognition which contains 10000 test images and 50000 training images.

As shown in TABLE I, we observe that SecureMar_DL still maintains excellent classification accuracy (>91.35%), even if we embed a 121-bit CIC and a 60-bit CRC in the model. Compared with the baseline (i.e, training an identical model without embedding fingerprints), as the length of the fingerprint increases, the classification accuracy of SecureMar_DL only slightly degrades. Such results are mainly derived from the high capacity of DNN parameters. In DNN, the redundancy and high dimensionality of parameters make many parameters have a slight impact on the classification of the final model. Based on this, we use this redundancy and make embedded fingerprints as an auxiliary task during the training process. As a result, the embedded fingerprint is integrated as an integral part of the parameters, which ensures the robustness of fingerprints against various attacks while not significantly changing the classification accuracy of the original model.

[1]http://yann.lecun.com/exdb/mnist/

[2]https://www.cs.toronto.edu/ kriz/cifar.html

B. Performance of Fingerprint

In this section, we conduct experiments to discuss the performance of the proposed fingerprint. We first discuss the property of community relationship code of preserving neighborhood structure, and then demonstrate the robustness of customer identity code to various attacks.

1) Analysis of Community Relationship Code: As described before, community relationship codes preserve the neighborhood structure of customers. This property will facilitate the server to quickly find suspicious customer groups. To demonstrate this, we conduct experiments over three public social network dataset, i.e., Facebook, Twitter and Google, all of them are downloaded from the Stanford Network Analysis Platform (SNAP)[3]. Each dataset is an adjacency graph, where each node represents a user (that is, the customer in our experiment), and the edge between two points indicates that there is a connection between the two. The specific information of each selected dataset is as follows.

- `Facebook` is an undirected graph, which contains 4039 nodes and 88,234 edges. These nodes (i.e., customers) belong to 10 communities.
- `Twitter` is a directed graph with 81306 customers and 1768149 edges. These customers belong to 1000 communities.
- `Google` is a directed graph with 107614 customers and 13673453 edges. These customers belong to 133 communities.

We evaluate the performance of community relationship code over the three datasets, where the length of code is ranging from 15 to 60 bits. Specifically, given the adjacent matrix A, we first generate the community relationship code for each customer exploiting the primitive Neighborhood Preserving Hashing (NPH) [20]. Then, we construct a matrix P to record the Hamming distance between customers, where p_{ij} denotes the Hamming distance between customer i's community relationship code and customer j's community relationship code. We set $p_{ij} = 1$ if p_{ij} is less than a given threshold distance $dist \in [0, 3]$, otherwise, $p_{ij} = 0$. As a result, by comparing P with the adjacent matrix A, we can obtain the accuracy of preserving the neighborhood structure of community relationship code under different social networks. The results are shown in Figure 5. It is clear that community relationship code can effectively represent the social network between customers and ensure the neighborhood structure between different customers. In addition, the tendency of accuracy increases as the length of the code increases.

[3]http://snap.stanford.edu/

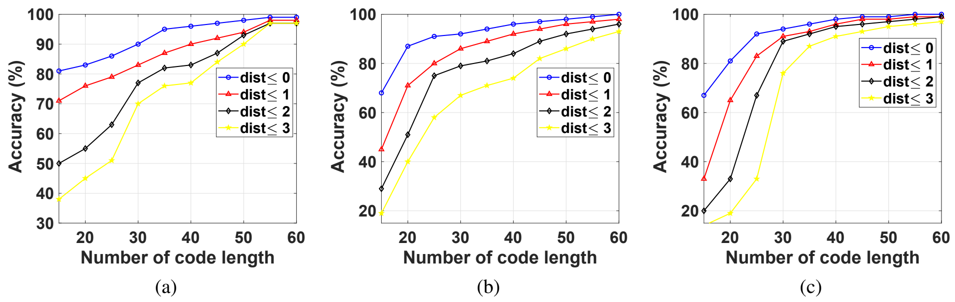

Fig. 5: The accuracy of preserving the neighborhood structure of Community Relationship Code under different social networks. (a) Facebook. (b) Twitter. (c) Google

2) Analysis of Customer Identity Code: We adopt two separate $(121, 11, 1)$-BIBD AND-ACC codebook and assign the concatenation of one codebook column to the other codebook column as a code-vector for each customer. As a result, such codebooks can accommodate a total of $N = \frac{121(120)}{11(10)} \times \frac{121(120)}{11(10)} = 17421$ customers and resist up to $k = 11 - 1 = 10$ colluders. We embed the fingerprint with the MNIST and CIFAR-10 dataset, respectively, where the weight μ of the embedded fingerprint is set to 0.2. The threshold τ used in extraction is set to 0.85.

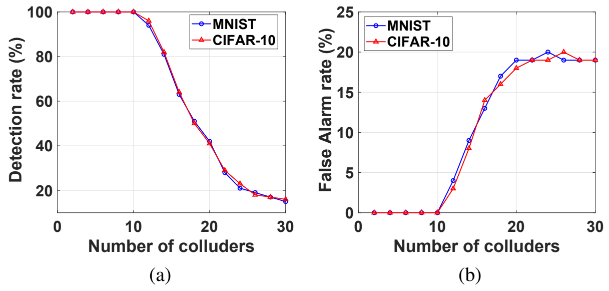

Fig. 6: The performance of $(121, 11, 1)$-BIBD Customer Identity Code with collusion.

Collusion Attack: Fig. 6a shows the detection rate of $(121, 11, 1)$-BIBD customer identity code under two data sets with different numbers of colluders. We can observe that the detection rate remains 100% when the number of colluders is 10 or less, which means that the maximum number of colluders that $(121, 11, 1)$-BIBD customer identity code can resist is consistent with its own nature. For the sake of comprehensive analysis, we also record the false alarm rate on both datasets. As shown in Fig. 6b, it is clear that the error rate is always maintained at 0% when the number of colluders does not exceed 10, which is consistent with the previous maximum detectable number $k = 10$.

Model Fine-tuning Attack: In order to demonstrate the robustness of our customer identity code to model fine-tuning attack, we retrained the fingerprinted model with the original optimization function (shown in Eqn.(1)). The results are shown in Fig. 7a and Fig. 7b. We can see that the detection rate remains 100%, and the error rate is always maintained at 0% when the number of colluders does not exceed 10, which is consistent with the previous experiments shown in Fig. 6. Therefore, our customer identity code is robust against model fine-tuning attacks.

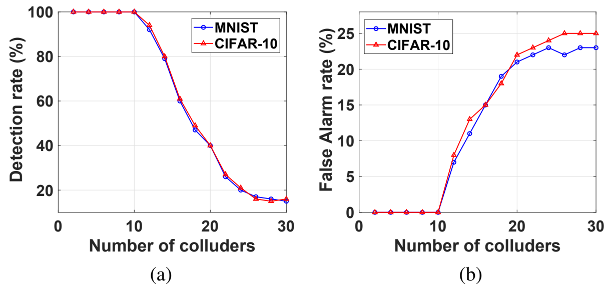

Fig. 7: The performance of $(121, 11, 1)$-BIBD Customer Identity Code with model fine-tuning.

Model Compression Attack: In the end, we analyze the effectiveness of our customer identity code for model compression attacks. Here we consider a very common model compression method, namely parameter pruning. Specifically, we perform parameter pruning of the trained model with different rates, and record the performance of the customer identity code against the collusion attack after pruning. Fig. 8 shows the results under MNIST and CIFAR-10 datasets. Indeed, as the proportion of pruned parameters increases, the accuracy of detection and false alarm rate will deteriorate slightly. However, the performance of our customer identity code is still consistent with previous experiments when the number of colluders does not exceed 10. Therefore, our customer identity code is robust against the model compression attack.

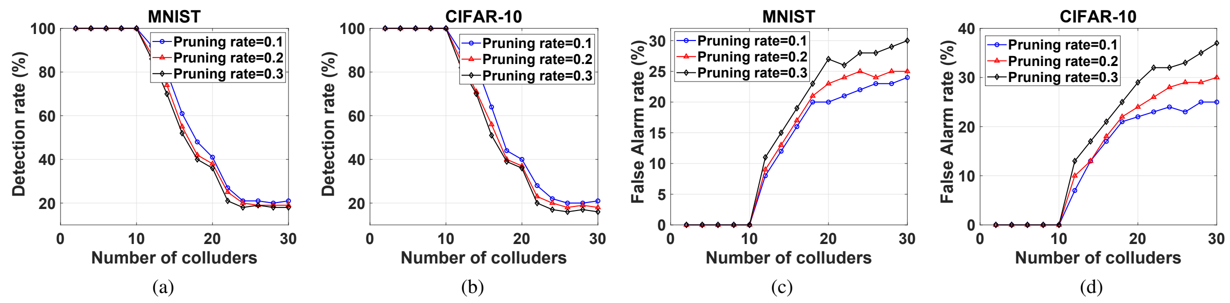

Fig. 8: The performance of $(121, 11, 1)$-BIBD Customer Identity Code with parameter pruning.

V. Conclusion

In this paper, we have proposed SecureMark_DL, a deep learning framework with end-to-end ownership protection. SecureMark_DL is robust against various attacks including fingerprints collusion and network transformation. Extensive experiments conducted on MNIST and CIFAR-10 datasets show the superiority of SecureMark_DL in terms of training accuracy and robustness against various types of attacks. In the future, we will focus on increasing the level of privacy protection, and improving the robustness of proposed fingerprint to other emerging attacks.

Acknowledgment

This work is supported by the National Key R&D Program of China under Grants 2017YFB0802300 and 2017YF-B0802000, the National Natural Science Foundation of China under Grants 62020106013, 61972454, 61802051, 61772121, and 61728102, Sichuan Science and Technology Program under Grants 2020JDTD0007 and 2020YFG0298, the Peng Cheng Laboratory Project of Guangdong Province PCL2018KP004.

References

[1] H. Li, Q. Chen, H. Zhu, D. Ma, H. Wen, and X. S. Shen, "Privacy leakage via de-anonymization and aggregation in heterogeneous social networks," *IEEE Transactions on Dependable and Secure Computing*, vol. 17, no. 2, pp. 350–362, 2020.

[2] G. Xu, H. Li, S. Liu, K. Yang, and X. Lin, "Verifynet: Secure and verifiable federated learning," *IEEE Transactions on Information Forensics and Security*, vol. 15, no. 1, pp. 911–926, 2020.

[3] Y. Adi, C. Baum, M. Cisse, B. Pinkas, and J. Keshet, "Turning your weakness into a strength: Watermarking deep neural networks by backdooring," in *Proceedings of USENIX Security*, 2018, pp. 1615–1631.

[4] B. Darvish Rouhani, H. Chen, and F. Koushanfar, "Deepsigns: An end-to-end watermarking framework for ownership protection of deep neural networks," in *Proceedings of ACM ASPLOS*, 2019, pp. 485–497.

[5] R. Namba and J. Sakuma, "Robust watermarking of neural network with exponential weighting," in *Proceedings of ACM AsiaCCS*, 2019, pp. 228–240.

[6] Z. Yang, H. Dang, and E.-C. Chang, "Effectiveness of distillation attack and countermeasure on neural network watermarking," *arXiv preprint arXiv:1906.06046*, 2019.

[7] S. Szyller, B. G. Atli, S. Marchal, and N. Asokan, "Dawn: Dynamic adversarial watermarking of neural networks," *arXiv preprint arXiv:1906.00830*, 2019.

[8] J. Guo and M. Potkonjak, "Watermarking deep neural networks for embedded systems," in *Proceedings of IEEE/ACM ICCAD*, 2018, pp. 1–8.

[9] E. Le Merrer, P. Perez, and G. Trédan, "Adversarial frontier stitching for remote neural network watermarking," *Neural Computing and Applications*, vol. 32, no. 13, pp. 9233–9244, 2020.

[10] 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 *Proceedings of ACM AsiaCCS*, 2018, pp. 159–172.

[11] H. Wu, C. Wang, J. Yin, K. Lu, and L. Zhu, "Sharing deep neural network models with interpretation," in *Proceedings of the Web Conference*, 2018, pp. 177–186.

[12] D. Hitaj and L. V. Mancini, "Have you stolen my model? evasion attacks against deep neural network watermarking techniques," *arXiv preprint arXiv:1809.00615*, 2018.

[13] G. Xu, H. Li, H. Ren, K. Yang, and R. H. Deng, "Data security issues in deep learning: Attacks, countermeasures and opportunities," *IEEE Communications Magazine*, vol. 57, no. 11, pp. 116–123, 2019.

[14] M. Wu, W. Trappe, Z. J. Wang, and K. R. Liu, "Collusion-resistant fingerprinting for multimedia," *IEEE Signal Processing Magazine*, vol. 21, no. 2, pp. 15–27, 2004.

[15] Y. Meng, W. Zhang, H. Zhu, and X. S. Shen, "Securing consumer iot in the smart home: Architecture, challenges, and countermeasures," *IEEE Wireless Communications*, vol. 25, no. 6, pp. 53–59, 2018.

[16] G. Xu, H. Li, Y. Zhang, S. Xu, J. Ning, and R. Deng, "Privacy-preserving federated deep learning with irregular users," *IEEE Transactions on Dependable and Secure Computing*, 2020, DOI:10.1109/TDSC.2020.3005909.

[17] H. Chen, B. D. Rouhani, C. Fu, J. Zhao, and F. Koushanfar, "Deepmarks: A secure fingerprinting framework for digital rights management of deep learning models," in *Proceedings of ACM ICMR*, 2019, pp. 105–113.

[18] C. L. Apicella, F. W. Marlowe, J. H. Fowler, and N. A. Christakis, "Social networks and cooperation in hunter-gatherers," *Nature*, vol. 481, no. 7382, p. 497, 2012.

[19] L. A. Bassalygo and V. A. Zinoviev, "Remark on balanced incomplete block designs, near-resolvable block designs, and q-ary constant-weight codes," *Problems of Information Transmission*, vol. 53, no. 1, pp. 51–54, 2017.

[20] C. Liu, H. Ling, F. Zou, and L. Yan, "Neighborhood preserving hashing for fast similarity search," in *Proceedings of ACM MM*, 2012, pp. 945–948.

[21] X. Fu, K. Huang, N. D. Sidiropoulos, and W.-K. Ma, "Nonnegative matrix factorization for signal and data analytics: Identifiability, algorithms, and applications." *IEEE Signal Process Magazine*, vol. 36, no. 2, pp. 59–80, 2019.

[22] B.-Y. Sun, X.-M. Zhang, J. Li, and X.-M. Mao, "Feature fusion using locally linear embedding for classification," *IEEE Transactions on Neural Networks*, vol. 21, no. 1, pp. 163–168, 2009.

[23] C.-J. Lin, "Projected gradient methods for nonnegative matrix factorization," *Neural Computation*, vol. 19, no. 10, pp. 2756–2779, 2007.

[24] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.