Irene: Privacy-Preserving Face Tagging using Deep Learning

Yao Zheng, Ethan Gaebel, Wenjing Lou, Y. Thomas Hou
Virginia Tech, Blacksburg, USA
{zhengyao, egaebel, wjlou, thou}@vt.edu

***Abstract*—Face tagging is a photo management tool that utilizes facial recognition technologies to automatically identify faces in photos and make tagging suggestions. With its many utilities, the privacy controversy surrounding automatic face tagging is becoming a major concern for the general public. As a durable identifier, faceprints have long-term utility for identifying individuals. But current law affords little privacy protection to these data. In addition, the recent development of deep learning technologies motivates companies to collect and enroll a massive volume of users' photos and faceprints in order to improve the quality of their face tagging services. Such activities open the door to widespread tracking of the public, and facilitate stalking and harassment.**

To address this issue, we present a privacy-preserving face tagging framework that prohibits unconsented harvesting of users' photo and faceprint collections and is robust against unconsented identity tracking. The system is built based on a multi-task, distributed, collaborative deep learning system, which allows users to privately maintain their photos and create user-specific faceprints, while jointly improving the face tagging functionality. Our evaluations shows the performance of our framework is on par with the privacy-violating system, scoring a state-of-art tagging accuracy (92.1%) with all the privacy-preserving features.

I. Introduction

Face tagging is a photo management tool that utilizes facial recognition technologies to automatically identify faces in photos and make tagging suggestions. It is common seen as a feature integrated into social network products. For instance, companies such as Facebook, Google, and Apple all include face tagging features in their photo management products, which allow users to categorize, browse, and share their photos efficiently. Aside from enriching user experience, the usage of face tagging under social network context has also been stimulated by the advancements in artificial intelligence, in particular, the significant performance improvement due to the application of deep learning methods. For example, the most recent face recognition method by Google [1] using deep learning, was able to achieve a accuracy of 99.63%, which is 2.1% higher than human-level performance [2].

While it is exciting to enjoy this useful, time-saving feature, there are many privacy controversy over the adoption of automatic face tagging. The primary debate is whether companies are allowed to collect or store users faceprint without their consent. In order for the software to recognize a face, it enrolls the faceprint in a facial recognition database every time one of its users uploads a photo and tags someone. Currently, the legitimacy of this action remains ambiguous. In US, for instance, some states such as Illinois, has adopted the Biometric Information Privacy Act, which prohibits unconsented collection of users' biometric data, including faceprints. Other states are still governed by the Stored Communications Act, which generally consider meta-data, such as faceprints, to be non-communications contents and can be shared freely. In any case, once the faceprint are enrolled into the database, the companies can potentially tag this person on any photos they have. Such capability constitute a form of extra-judicial surveillance and is a severe breach of privacy.

Furthermore, many of the face tagging features rely on advanced deep learning models to make accurate suggestions. The training of these models usually requires a large collection of data. For instance, the FaceNet model from Google [1] was trained using 200 million photos and eight million unique identities. A central collection of millions of photos from millions of users pose additional privacy risks. First of all, a large collection of users' photos is an easy target for malicious activities. One of the most recent example is FindFace, which allows users to photograph people in a crowd and work out their identities. The data it uses are crawled from Vkontakte, a social network popular in Russia. Second of all, these photos are also subject to subpoenas and warrants. For instance, law enforcement agencies may issue warrant to social media companies like Facebook seeking copies of all photos users upload, along with all photos they are tagged in. Worse still, due to third-party doctrine, law enforcement agencies may even search and seizure these photos without probable cause or a judicial search warrant.

Our contributions. To address this conundrum, we design and implement a privacy-preserving face tagging framework. The conceptual innovations of the framework are threefold. To address the problem of unconsented harvesting of users' photos, we design a distributed, collaborative deep learning system for users to privately maintain their photo collections, while jointly train a deep learning model for face tagging. To address the problem of unconsented harvesting of faceprints, we design a multi-task deep learning model to create user-specific faceprint collections, and prohibit the aggregation of faceprints from multiple sources. To address the problem of unconsented identity tracking, we design a privacy-preserving training protocol, which prevents users from revealing personal identifiable information about their faceprint collections.

Our study goes beyond proof-of-concept and delves into the complicate deep neural network training for face tagging. Our key technical innovations include the design and integrate of various privacy-preserving features. To start, we make novel use of the random dropout method and elastic averaging

TABLE I. FREQUENTLY USED NOTIONS

ψ,ϕ,φ	The components of the deep learning model.
U,$\mathbf{b}$,V	Parameters of the deep learning model.
$\mathbf{x}$	The flattened vector of all parameters.
$\mathbf{g}(\mathbf{x})$	gradient vector.
τ, $\mathbf{t}$	Tags and their one-hot vector representations.
ℓ	A faceprint.
$\mathcal{T}$	The set of tagged faceprints.
W	Mahalanobis metric.
$d_W(\cdot,\cdot)$	Mahalanobis distance function parameterized by W.

stochastic gradient descent (EASGD) to design the distributed, collaborative learning system. We bisect the traditional deep learning model to allow each user to privately train a separate embedding metric to generate faceprints while collaborating on a shared deep feature extractor. We make novel use of noise-robust learning and multi-task large margin nearest neighbor (MT-LMNN) classifier for users to jointly bootstrap and fine-tune the deep learning model without revealing the identities of their faceprint collections. Finally, we adopt two differential privacy mechanisms, namely, addictive noise and sparse vector, for the bootstrapping and tuning stage, to allow users to train the deep learning model with differential privacy guarantee.

We evaluate our system with two real world face verification datasets, the Labeled Faces in the Wild (LFW) dataset [3] (for benchmarking), and the CASIA-WebFace dataset [4] (for training). The accuracy of the model produced by our privacy-preserving face tagging framework is on par with the privacy-violating system. Our evaluation shows that our framework can achieve state-of-art tagging accuracy (92.1%) with all the privacy-preserving features. The score is comparable to the highest accuracy achieved by `OpenFace` models trained with similar datasets.

II. IRENE: FRAMEWORK FOR PRIVACY-PRESERVING FACE TAGGING

In this section, we describe, Irene, our novel privacy-preserving face tagging framework, which is based on a multi-task, distributed, collaborative deep learning system. The frequently used notations are summarized in Table I. We use lower case letters for scalars, lower case bold letters for vectors, upper case letters for matrices, and upper calligraphic letters and blackboard bold letters for sets.

A. Problem Definition

We consider a cloud-based photo management service, where there are multiple users. The users have full control over their photo collections, *i.e.*, they can upload, maintain and delete them. There is a central cloud server belonging to the service provider, which stores all of the photos. In a typical photo management service, the photos stored in the server are encrypted, so that the service provider cannot peek into users photo albums. The users can access their photos on supported devices via the app developed the by the service provider. The main functionalities of the app include downloading and decrypting the photos for viewing, and basic editing capabilities such as cropping, toning, tagging, *etc.* For our study, we consider the app has a face tagging function based on a deep learning model.

A typical photo management service often maintains photos in their original resolution in the cloud, and transmits low resolution versions to the device to save space. For the simplicity of our study, we omit this feature. Note that our framework can be easily adjusted for a multi-resolution scenario by preprocessing the input photos [2], [5]–[9].

B. Functional Goal

The functional goal of Irene is to improve the app's face tagging accuracy based on the collective input of multiple users. Since all users use the same app with the same deep learning model for face tagging, we want them to collaborate and improve the quality of the model. For instance, when a user upload a photo, the app extract raw faces in the photo using traditional methods, such as cascading LBP or Haar classifiers [10], and use a deep learning model to extract faceprints. By comparing the faceprints with known ones, the app makes tagging suggestion to the users. The user tags these faces by accepting the right ones and correcting the wrong ones. These corrections can be used as feedbacks to update the deep learning model and improve its accuracy.

Through our system, we want the service provider to be able to collect this feedback and produces more accurate model. Note that the design of the collect-and-update protocol is non trivial. Because this feedback is associated to users' local faceprint collections, naively combining them often lead to a biased, mediocre model and have many privacy concerns. We will further discuss these challenges in section III.

C. Privacy Goals

The privacy goals of Irene is to preserve the privacy of users' photo and faceprint collections. As users interact with the service provider to improve the face tagging functionality, the service provider should not gain any direct and indirect knowledge about users' photo faceprint collections. We describe these goals from the following three aspects.

Prevent Unconsented Harvesting of Photos. Users should keep their collections of photos private from the service provider and other users. The service provider should not be able to obtain any photos as the system operates.

Prevent Unconsented Harvesting of Faceprints. Different users should extract different faceprints for the same person. This prevents multiple users from colluding and combine their faceprint collections for unsavory motives, such as extra-judicial surveillance.

Prevent Unconsented Identity Tracking The service provider should not learn any identifiable information about the faceprint collections through users' feedbacks. Specifically, the service provider should not know whether a particular faceprint is or is not used to compute a user's feedback.

In addition, the app should protect the detailed implementation of the deep learning model. Often, such a model is considered as proprietary to the company that creates it. Therefore, we assume the app operates as a blackbox. Users can input photos to the app and obtain faceprints, as well as tag the faceprints, but they do not have any knowledge of its internal workings.

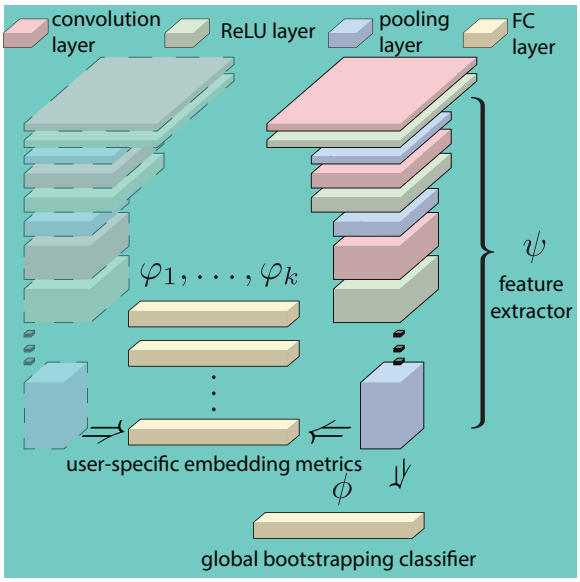

Fig. 1. Architecture of our deep learning model with one shared feature extractor, ψ, a global bootstrapping classifier, ϕ, and a user-specific embedding metric, φs.

D. Overview of Irene

Irene is a multi-task, distributed, collaborative deep learning system that relies on the following key observations: (i) *multi-task.* The deep learning model used for recognition can be divided into two modules: a feature extractor and a classifier. The former extracts useful, generic features from the input data. The latter converts them into domain-specific representations for the recognition task. This architecture allows users to obtain user-specific faceprint collections while collaborating on a shared feature extractor. (ii) *distributed.* A distributed system protects users' privacy by preventing a single party from holding all the photo and faceprint collections. Such system can be built through techniques such as random dropout [11] and EASGD [12]. (ii) *collaborative.* User collaboration boosts the diversity of the training data, and improves the quality of the deep learning model. Such collaboration can be carried out in a differentially private fashion via applications of differential privacy techniques, *i.e.*, additive noise and sparse vector [13], at various training stages [14]–[16]. We will elaborate each of these points in the following.

Architecture of the Multi-Task Model. Figure 1 shows the architecture of our deep learning model. The model consists of a convolutional neural network (CNN) feature extractor, ψ, a bootstrapping classifier, ϕ, and a user-specific embedding metric, φ. Our goal is to let users jointly bootstrap ψ through ϕ, and continue to improve ψ while training their individual φs for face tagging.

The feature extractor, ψ, is composed by cascading multiple linear and non-linear operator layers. It takes a raw face input, ℓ, and outputs a feature vector, $\psi(\ell) \in \mathbb{R}^d$.

The bootstrapping classifier, ϕ, transforms $\psi(\ell)$ into a classification vector $\phi(\psi(\ell)) = U\psi(\ell) + \mathbf{b} \in \mathbb{R}^n$, $U \in \mathbb{R}^{n \times d}$, $\mathbf{b} \in \mathbb{R}^n$, by means of a fully-connected layer containing n linear predictors. This classification vector, $\phi(\psi(\ell))$, is used to determine the likelihood of ℓ's class identity among n classes. The probability that ℓ belong to the ith class is computed through a softmax function,

$$p_i = \frac{e^{\phi(\psi(\ell))_i}}{\sum_{j=1}^{n} e^{\phi(\psi(\ell))_j}}. \quad (1)$$

The user-specific embedding metric, φ, generate the faceprint by normalizing $\psi(\ell)$ and projecting it to a $m \ll d$ dimensional embedding space using an affine projection $\varphi(\psi(\ell)) = V\psi(\ell)/\|\psi(\ell)\|_2$, $V \in \mathbb{R}^{m \times d}$.

Finally, face tags are generated through a nearest neighbor search in the embedding space, *i.e.*, $\tau(\ell) = \tau(\mathrm{NN}(\ell))$. The nearest neighbor function is defined as $\mathrm{NN}(\ell) = \arg\min_{x \in \mathcal{T}} \|\varphi(\psi(\ell)) - \varphi(\psi(x))\|_2$, where $\mathcal{T}$ refers to the set of tagged faceprints.

We use a embedding metric rather than a classifier for the final application due to robustness considerations [9]. For the classifier, the number of classes, n, is fixed. When a new identity appears, the classifier must be replaced with a new one with one more class. Comparing to that, a metric can incorporate the new identity without any adjustments to its dimensionality.

Distributed Learning System. Figure 2 shows the setup of our distributed learning system. We assume there are multiple users who have the app. Each app manages a local, private faceprint collection extracted from the user's photo collection, a local copy of ψ and ϕ, and an independent φ for face tagging. We further assume that, besides users' photos, the central server also serves as a parameters server and maintains the latest parameters of the shared components, ψ and ϕ. The idea is to let each user to download the latest parameters of ψ and ϕ, update them based on the local faceprint collections, and upload the new parameters to the central sever. This way, users avoid directly revealing their photo collections to the service provider or other users, but still provide information to the provider to improve the model.

There are two challenges to build such a system. First of all, the deep learning model contains many parameters. Communicating all of them between users and the server would incur a large overhead. Second of all, the training procedure, *i.e.* stochastic gradient descent (SGD), is highly sequential and requires a full view of the model. Naively applying it in a distributed setting will cause race conditions, *i.e.*, multiple users try to change the same parts of the deep learning model at the same time.

To cope with these problems, we adopt two methods. To reduce the communication overhead, we use a method known as random dropout [11] to distribute the parameters among different users. To reduce the race conditions, we rely on asynchronous EASGD [12] to prevent multiple users from clashing during parameter update.

Privacy-Preserving Collaborative Training. Like most deep learning based face recognition applications [1], [6], [8], [9], [17], the model is trained using two-stage training method, as shown in Figure 3. During the first stage, all users jointly bootstrap ϕ by considering a n-ways classification problem. Normally, this problem requires all users to agree on n unique identities appear in their photos. Such an agreement usually

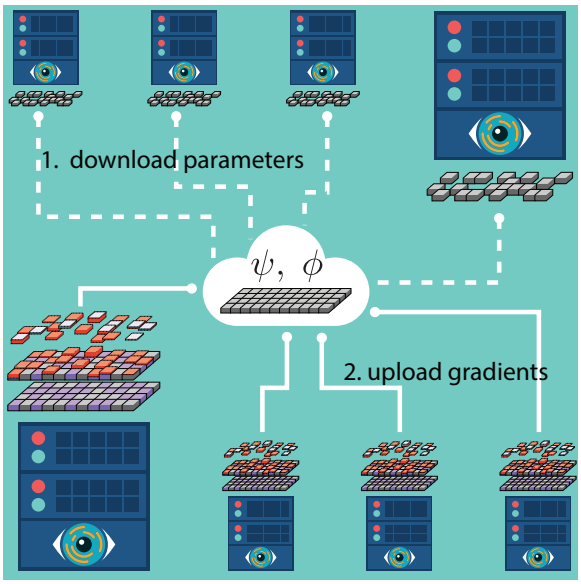

Fig. 2. Overview of our distributed learning with one parameter server and multiple users as participants. The shared parameters include ϕ, ψ_g, but not include ψ_l, which is unique for each users.

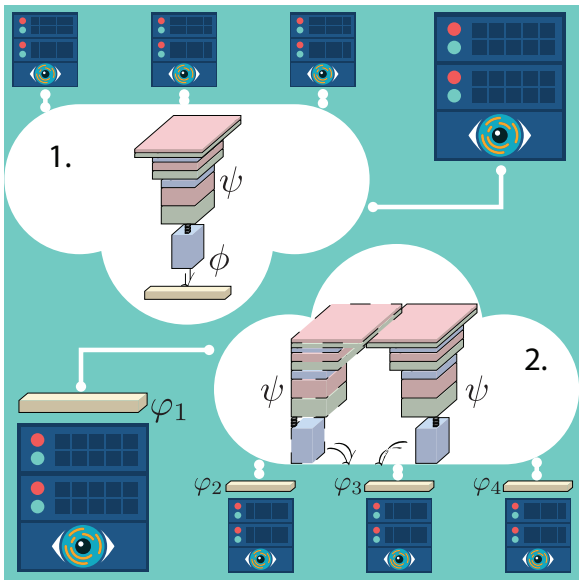

Fig. 3. Our two-stage, privacy-preserving collaborative training.

involves multiple rounds of communications between users, and risks exposing the identities of users' faceprint collections. Instead, we introduce a noisy bootstrapping scheme, which allows users to annotate faceprints without consensus; and compensate the noise by augmenting the learning objective with minimum entropy regularization [18].

During the second stage, each user privately trains a embedding metric, φ, to generate user-specific faceprint collections. To effectively train φ, we use a siamese network architecture, which applies the same ψ to pairs of faceprints, and compare the similarity between the outputs. Once φ is trained, users can fine-tune ψ by freezing φ and training ψ at a lower learning rate. To allow multiple users with different φs to tune ψ, we adopt the MT-LMNN to find a commonality embedding metric among users.

To prevent unconsented identity tracking, we apply SGD with differentially private updates [19] to ensure users' parameter updates do not depend on whether a particular faceprint is included in the computation. Note that, in our design, we only apply differential privacy mechanisms to update shared components, ψ and ϕ. The private metric, φ, is trained with a method that does not sacrifice accuracy for privacy.

III. Main Design Issues

In this section, we address several key design issues in Irene, all of which focus on enhancing the privacy-preserving property of the framework.

A. Privacy Enhancement with Noisy Bootstrapping

The first issue we address is to enhance the privacy protection during bootstrapping. As we show earlier, Bootstrapping is the first stage of the training, and a critical part to ease the subsequent metric learning. It is done by considering a n-ways classification problem. Each class represents a unique person appear in users faceprint collections. In a distributed setting, finding those classes usually involves multiple rounds of communications between users, and risks exposing the identities of users' faceprint collections. Observe that the communications between users is mainly for consistency check. Due to lack of coordination, users in a distributed setting might annotate the same person with different tags, or annotate different persons with the same tag. To completely remove inconsistent tags, users must perform cross examination between each other, which violates privacy. The cross-examination can be performed in a privacy-preserving fashion using methods such as secure multi-party computation (SMC) [20]. However, it often needs multiple rounds of communications between users and introduces a high computation cost.

Instead, we propose to skip the consistency check and handle the inconsistency during training. The method we use is a combination a noisy labeling scheme with a noise-robust learning objective. During bootstrapping, we first mitigate the consistency problem with usage of unique identity tags, such as persons' email addresses. Second, we divide the n classes among users, and allow each user to select $n' < n$ identity tags and encode them to one-hot vectors, $\mathbf{t} = \{1, 0\}^n$, $\sum t_i = 1$, where the position of '1' is determined using a pre-shared hash function $\mathrm{H}(\tau) \in [0, n]$. Due to human error and the possibility of hash collision, the tags generated this way are bound to be inconsistent, *i.e.*, the same tag might includes faceprints with different identities and vice versa.

To cope with the noise in the tag, we introduce an additional regularization term in the learning objective [18]. Specifically, we evaluate the following loss function,

$$\mathcal{L}(\mathbf{q}, \mathbf{t}) = \sum_{i=1}^{n} \alpha t_i \log q_i + (1 - \alpha) \log \max_i q_i, \qquad (2)$$

where q_i is described in Equation 1. In Equation 2, the first term is a regular cross-entropy loss that aims to make the predicted tag consistent with the tags provided by users. The second term computes the min-entropy of the prediction, which encourage the model to have a high confidence in predicting

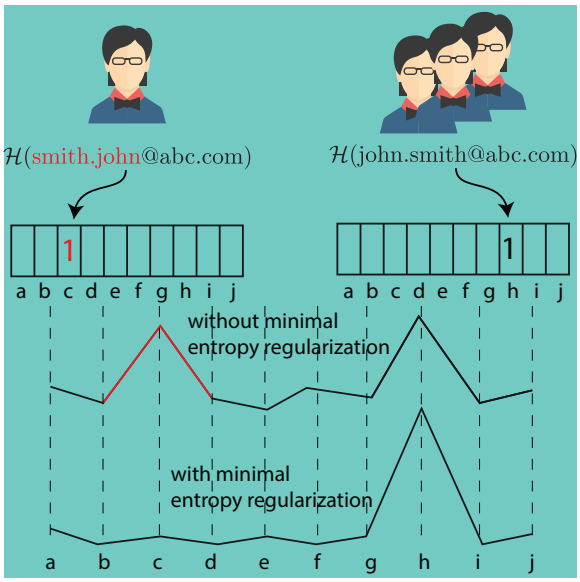

Fig. 4. Using minimum entropy regularization to reduce the model's sensitivity to tagging inconsistency.

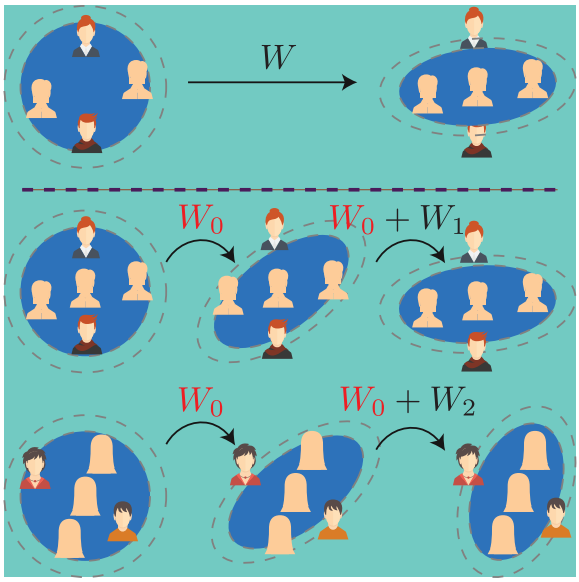

Fig. 5. Illustrations of LMNN and MT-LMNN, The former (above the dash line) uses a single metric to compute embeddings. The latter (below the dash line) uses a embedding metric, W_0, to capture the communality, and a idiosyncrasy metric, W_i for user-specific embedding projection.

tags regardless of $\mathbf{t}$. The effect, as shown in Figure 4, is that the model sacrifices less to match inconsistent tags and becomes more coherent.

Training under this new objective can be done by using an expectationmaximization (EM)-like algorithm [18]. The E-step corrects the noisy tags by randomly choose between noisy tags and the predictions of the current model with probability $(\alpha, 1-\alpha)$. The M-step uses SGD to update the model parameters to better predict the corrected tags. the hyper-parameter, α, can be found through cross-validation.

This method effectively omits consistency check between users and prevent the users from revealing the identities of their faceprint collections. However, even with the min-entropy regularization, the model obtained through the noisy bootstrapping is bound to be sub-optimal, which makes the subsequent metric learning and fine-tuning more critical.

B. Privacy Enhancement with User-Specific Metrics

The private metric learning is the second stage of the collaborative training, during which each user trains a separate φ to generate faceprints. This can be accomplished using large margin nearest neighbor (LMNN) classification along with a siamese network architecture. Specifically, we temporarily freeze ψ, and let each user create a siamese network with two identical branches. Each branch include a ψ followed by a φ. For a pair of images, (ℓ_i, ℓ_j), the network outputs a pair of vectors, $(V\hat{\psi}(\ell_i), V\hat{\psi}(\ell_j))$, where $\hat{\psi}(\ell) = \psi(\ell)/\|\psi(\ell)\|_2$. The Mahalanobis distance between $V\hat{\psi}(\ell_i)$ and $V\hat{\psi}(\ell_j)$ is defined as,

$$\begin{aligned} d_W(\ell_i, \ell_j) &= \|V\hat{\psi}(\ell_i) - V\hat{\psi}(\ell_j)\|_2 \\ &= \sqrt{(\hat{\psi}(\ell_i) - \hat{\psi}(\ell_j))^T W (\hat{\psi}(\ell_i) - \hat{\psi}(\ell_j))}, \end{aligned} \tag{3}$$

where $W = V^T V$ is the Mahalanobis metric. Our training objective can be described as the following optimization problem. Consider a faceprint triplet (ℓ_a, ℓ_p, ℓ_n) containing an anchor faceprint, ℓ_a, a positive faceprint ℓ_p, $\mathcal{T}(\ell_p) = \mathcal{T}(\ell_a)$, and a negative faceprint ℓ_n, $\mathcal{T}(\ell_n) \neq \mathcal{T}(\ell_a)$. We want to learn a metric, W, that keeps each positive faceprint, ℓ_p, closer to its anchor than other negative faceprints — by a margin, *i.e.*,

$$d^2_W(\ell_a, \ell_n) - d^2_W(\ell_a, \ell_p) \geq 1 \tag{4}$$

The metric, W, can be found by solving the following optimization problem,

$$\begin{aligned} \text{minimize} \quad & \sum_{a,p} d^2_W(\ell_a, \ell_p) + \sum_{a,p,n} \xi_{apn} \\ \text{subject to} \quad & d^2_W(\ell_a, \ell_n) - d^2_W(\ell_a, \ell_p) \geq 1 - \xi_{apn} \\ & \xi_{apn} > 0 \\ & W \succ 0, \end{aligned} \tag{5}$$

where ξ_{apn} is the slack variable penalizing violations of the constraint in Equation 4. This is a semidefinite programming (SDP) problem and can be solved using a standard solver [21].

Once users find their own φs, they can perform face tagging through nearest neighbor search. At this point, users have two choices: (1) complete the training and quit the collaboration. In the future, if ψ is updated, these users can download the new model and separately update their own φs. But they will not contribute and improve ψ. (2) continue to collaborate and fine-tune (train at a lower learning rate) ψ based on current φ.

For the second scenario, the problem becomes slightly complicated. Because different users now have different φs, each user will in fact tune ψ with a different loss function, and the collaboration will fail. To cope with this problem, we aim to find a commonality embedding metric shared between various users. For the kth user, we define

$$\begin{aligned} d_{W_0, W_k}(\ell_i, \ell_j) = \\ \sqrt{(\hat{\psi}(\ell_i) - \hat{\psi}(\ell_j))^T (W_0 + W_k)(\hat{\psi}(\ell_i) - \hat{\psi}(\ell_j))}, \end{aligned} \tag{6}$$

where $W_0 \succ 0$ is the commonality embedding metric and $W_k \succ 0$ is a user-specific idiosyncrasy embedding metric.

Intuitively, W_0 represents the general trend shared by all, and W_k specializes a metric further for each user, as shown in Figure 5. These metrics can be found by solving a MT-LMNN problem [22],

$$\begin{aligned} \text{minimize} \quad & \|W_0 - I\|_F^2 + \sum_{k=1}^{K} \Big(\|W_k\|_F^2 + \\ & \sum_{a,p} d^2_{W_0,W_k}(\ell_a, \ell_p) + \sum_{a,p,n} \xi_{apn} \Big) \\ \text{subject to} \quad & d^2_{W_0,W_k}(\ell_a, \ell_n) - d^2_{W_0,W_k}(\ell_a, \ell_p) \geq 1 - \xi_{apn} \\ & \xi_{apn} > 0 \\ & W_0, W_1 \ldots, W_K \succ 0. \end{aligned} \tag{7}$$

This problem can be solved in a distributed fashion using dual decomposition [23]. During the process, users can keep the W_k private and only exchange W_0.

With the previous adjustment, users can collaboratively tune ψ by substituting φs with the commonality embedding metric, and converting the convex formulation of LMNN into an empirical triple loss

$$\begin{aligned} \mathcal{L}(\psi) = \sum_{a,p,n} \max\{0, \; & 1 - \|V_0\hat{\psi}(\ell_a) - V_0\hat{\psi}(\ell_n)\|_2^2 + \\ & \|V_0\hat{\psi}(\ell_a) - V_0\hat{\psi}(\ell_p)\|_2^2\}. \end{aligned} \tag{8}$$

The new loss function allows users to tune ψ using SGD.

C. Enable Distributed Training

To facilitate collaboration, we develop a two-pronged approach to distribute the training among multiple users. The approach aims to address the two challenges raised in our distributed, collaborative setting. First, due to the large number of parameters in a deep learning model, naively communicating all of them between users and the parameter server would incur a large overhead. Second, the training procedure, *i.e.* SGD, is highly sequential and requires a full view of the model. Naively applying it in a distributed setting will cause race conditions.

To reduce the communication overhead, we employ the random dropout to "thin" the deep learning model. [11]. The method was originally designed to reduce neurons co-adaptation and prevent overfitting. As illustrated in Figure 6, during each training epoch, dropout randomly removes neurons (along with their connections) from the deep learning model with probability, p, and train a "thinned" deep learning model. By doing so, it makes the presence of any particular neuron unreliable and breaks up the co-adaptations.

We extend this idea to distribute reduce the number parameters users communicates with the parameter server. When users request model parameters, the parameter server produces a "thinned" model using random dropout. Users can evaluate the "thinned" model with their private faceprint collections and upload the gradients of the parameters to the server. This approach greatly reduces the workload at users' sides, with each user updates a smaller set of parameters.

Not that the random dropout is only applied to the parameter-heavy layers, *i.e.*, fully-connected layers, where each neuron has a separate set of parameters. In convolution layers, all neurons shared a set of parameters, and the benefit of random dropout diminishes.

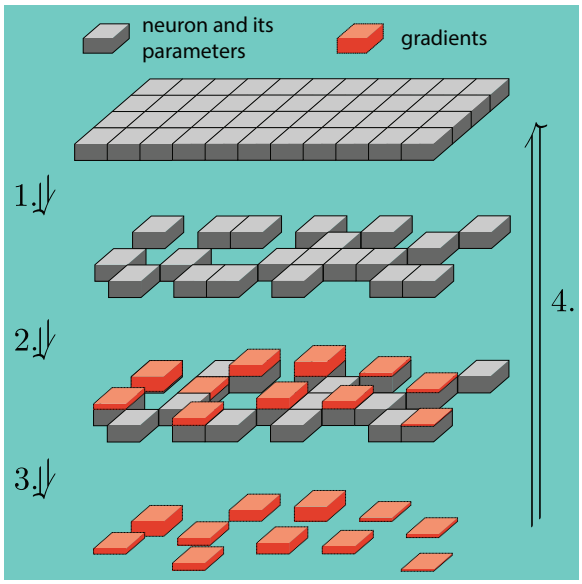

Fig. 6. Illustration of random dropout.

To mitigate the race condition problem, we adopt asynchronous EASGD, a parallel SGD algorithm that provides fast convergent minimization [12]. The algorithm, as shown in algorithm 1, works in an asynchronous, round-robin fashion. Each user maintains a local clock, i. At each clock tick, users update the local parameters using SGD. Whenever λ divides the local clock of the kth user, i^k, the kth user communicates with the parameter server and requests the current value of the center parameters $\tilde{\mathbf{x}}$. The user then computes the elastic difference $\beta(\mathbf{x} - \tilde{\mathbf{x}})$ and sends it back the parameter server who updates $\tilde{\mathbf{x}}$.

Comparing to regular SGD, EASGD allows users to perform more perform more exploration by fluctuating the local parameters further from the center parameters. This additional exploration is shown to be useful [12] in a deep learning setting, due to the existence of many local optima solutions.

Algorithm 1: asynchronous EASGD: Processing by user k and the parameter server

Input: learning rate η, moving rate β, communication period $\lambda \in \mathbb{N}$.
Initialize: $\tilde{\mathbf{x}}$ is download from the parameter server, $\mathbf{x}^k = \tilde{\mathbf{x}}$, $i^k = 0$.
repeat
  $\mathbf{x} \leftarrow \mathbf{x}^k$;
  if λ divides i^k **then**
    $\mathbf{x}^k \leftarrow \mathbf{x}^k - \beta(\mathbf{x} - \tilde{\mathbf{x}})$;
    $\tilde{\mathbf{x}} \leftarrow \tilde{\mathbf{x}} + \beta(\mathbf{x} - \tilde{\mathbf{x}})$;
  end
  $\mathbf{x}^k \leftarrow \mathbf{x}^k - \eta \mathbf{g}^{i^k}(\mathbf{x}^k)$;
  $i^k \leftarrow i^k + 1$;
until forever;

D. Incorporate Differential Privacy

Finally, we use differentially private SGD [15], [19] to prevent the service provider from tracking certain faceprints

via users' parameter updates. The idea of differential privacy is to randomize users' parameter updates, *i.e.*, the gradients, so that they are independent of whether or not a particular faceprint is included in the computation.

We employ two techniques to achieve differential privacy. Each of them is used during a particular training stage. During bootstrapping, we employ the additive noise technique to add noise to the gradients. The amount of noise is calibrated to the gradients' sensitivity, which is defined by the maximum difference caused by any adjacent inputs, *i.e.*, two batches that differ in only one faceprint. In a deep learning setting, there is no obvious bound on gradient's sensitivity. A single change in the batch can cause a dramatic change in the gradients. To cope with this problem, we artificially introduce a *posterior* bound using the norm clipping [15]. We clip each gradient in l_2 norm by replacing the gradient vector with $\mathbf{g}/\max(1, \frac{\|\mathbf{g}\|_2}{\gamma_1})$, where γ_1 is the clipping threshold.

With Norm clipping, we can substitute $\mathbf{g}(\mathbf{x})$ in algorithm 1 with its differentially private counterpart, as shown in algorithm 2. The method is (ϵ, δ)-differentially private with respect to the batch [15].

Algorithm 2: Differentially private $\mathbf{g}(\mathbf{x})$ using additive noise

Input: faceprint batch $\mathcal{B} = \{\ell_1, \ldots, \ell_B\}$, loss function $\mathcal{L}(\mathbf{x}, \ell)$, parameters $\mathbf{x}$,
approximate factor $\delta \neq 0$,
indistinguishability factor ϵ,
clipping threshold γ_1.

Initialize: **Let** $\sigma(\epsilon) = \gamma_1 \frac{\sqrt{2 \log \frac{1.25}{\delta}}}{\epsilon}$,
Let $\text{clip}(x) = \frac{x}{\max\{1, \frac{\|x\|_2}{c}\}}$.

$\mathbf{g}(\mathbf{x}) \leftarrow \frac{1}{B}\left(\sum_{\ell \in \mathcal{B}} \text{clip}(\nabla \mathcal{L}(\mathbf{x}, \ell)) + \mathcal{N}(0, \sigma(\epsilon)\mathbf{I})\right)$

During fine-tuning, we employ the sparse vector technique [13] to achieve differential privacy. The method is used to for privately reporting the outcomes of a potentially very large number of computations, provided that only a few are "significant." Comparing to the additive noise mechanism which add noise to all gradients, the sparse vector method only process gradients that are larger than certain value. The algorithm is shown in algorithm 3, which is also (ϵ, δ)-differentially private with respect to the batch [13].

Both methods can be useful during the collaboration. The additive noise technique introduce extra randomness in the gradients, which is helpful during initial exploration. The spare vector technique focus on the large gradients, which is useful during final convergence. Therefore, users can choose between these two mechanisms based on the stage of the collaboration and enjoy the benefit of both methods.

IV. Experiments and Results

In this section, we report the performance of Irene based on two benchmark face verification datasets, the LFW dataset [3], and the CASIA-WebFace dataset [4]. We first test a few setups of Irene by changing the hyper-parameters, α, β, and γs. Then we evaluate the privacy/utility trade-off using the best setup and varying the number of users, k, and the privacy loss (ϵ, δ).

Algorithm 3: Differentially private $\mathbf{g}(\mathbf{x})$ using sparse vector

Input: faceprint batch $\mathcal{B} = \{\ell_1, \ldots, \ell_B\}$, loss function $\mathcal{L}(\mathbf{x}, \ell)$, parameters $\mathbf{x}$,
approximate factor $\delta \neq 0$,
indistinguishability factor ϵ,
clipping threshold γ_2, selection threshold γ_3,
sparsity threshold γ_4.

Initialize: **Let** $\epsilon_1 = \frac{\sqrt{512}}{\sqrt{512}+1}\epsilon$, $\epsilon_2 = \frac{2}{\sqrt{512}+1}\epsilon$,
$\sigma(\epsilon) = \frac{\sqrt{32c \ln \frac{2}{\delta}}}{\epsilon}$.
Let $\gamma_5 = \gamma_3 + \text{Lap}(\sigma(\epsilon_1))$.
Let count $= 0$.
Let $\text{clip}(x) = \frac{x}{\max\{1, \frac{\|x\|_2}{c}\}}$.

$\mathbf{g}(\mathbf{x}) \leftarrow \mathbf{0}$;
tmp $\leftarrow \frac{1}{B}\left(\sum_{\ell \in \mathcal{B}} \nabla \mathcal{L}(\mathbf{x}, \ell)\right)$;
repeat
  randomly sample a gradient **tmp**$_i$;
  if $\text{clip}(tmp_i) + \text{Lap}(2\sigma(\epsilon_1)) \geq \gamma_5$ **then**
    $g_i(\mathbf{x}) \leftarrow \text{clip}(tmp_i + \text{Lap}(\sigma(\epsilon_2)))$;
    $\gamma_5 = \gamma_3 + \text{Lap}(\sigma(\epsilon_1))$;
    count $\leftarrow$ count $+ 1$
  end
until count $\geq \gamma_4$;

A. Dataset

We conduct our experiments on two benchmark datasets. The first one is the LFW dataset. It contains 13,233 images with 5,749 identities, and is the standard benchmark for automatic face verification. The second one, CASIA-WebFace [4], is a larger dataset containing 494,414 images with 10,575 identities. To make a fair comparison with other privacy-violating methods, we follow the standard evaluation protocol in unrestricted setting: Using the CASIA-WebFace dataset to train the model and use the LFW dataset to test performance. We use the full CASIA-WebFace dataset for tuning, but only select 3,000 identities for bootstrapping to prevent the vanishing gradient problem.

B. Model Configurations

Table II shows the configuration of ψ, ϕ, and φ. The name is formatted as "layer type-kernel size-input dimensions-output dimensions". We omit the Rectified Linear Unit (ReLU) layer for simplicity. For ψ, we use a 15-layer (not counting the maxpooling layer) VGG network [9], [24]. The model has approximately 147 millions parameters. For ϕ, we use a fully-connected layer with 3,000 dimensions, followed by a softmax classifier with minimum entropy regularization. For φ, we use a fully-connected layer with 128 dimensions, followed by the MT-LMNN classification or the triple loss.

C. Implementation Details

Our implementation is based on `torch` [25] linked against the `NVIDIA CuDNN` libraries to accelerate training. The distributed learning framework and EASGD implementation is based on the `torch-distlearn` library [26]. The MT-LMNN problem is solved through the `Gurobi solver`

TABLE II. MODEL CONFIGURATIONS.

ψ			
0	input 224 × 224 RGB images		
1	conv-3 × 3-3-64	11	conv-3 × 3-256-512
2	conv-3 × 3-64-64	12	conv-3 × 3-512-512
3	maxpool-2 × 2-64-64	13	conv-3 × 3-512-512
4	conv-3 × 3-64-128	14	maxpool-2 × 2-512-512
5	conv-3 × 3-128-128	15	conv-3 × 3-512-512
6	maxpool-2 × 2-128-128	16	conv-3 × 3-512-512
7	conv-3 × 3-128-256	17	conv-3 × 3-512-512
8	conv-3 × 3-256-256	18	maxpool-2 × 2-512-512
9	conv-3 × 3-256-256	19	fc-7 × 7-512-4096
10	maxpool-2 × 2-256-256	20	fc-1 × 1-4096-4096
ϕ		φ	
1	fc-1 × 1-4096-3000	1	fc-1 × 1-4096-128
2	Equation 2	2	Equation 7 or Equation 8

[27] (binded to torch via the `torch.gurobi` library [28]). The neural network training code is based on `OpenFace 0.2.0` [17]. All our experiments were carried on an Amazon g2.8xlarge instance with 4 NVIDIA GRID GPUs, each with 1,536 CUDA cores and 4GB of video memory, and 32 vCPUs with 60GB of memory.

We follow the same procedure described in [9] to pre-process the datasets. faceprints are located using pre-trained models from `dlib`. We sample each faceprint by cropping four 224×224 pixel patches from the four corners and the center with horizontal flip (*i.e.* 10 in total, as in [29] and [9]), and average the feature vectors.

In our default setup, we fix the number of users at 8. For bootstrapping, we set learning rate to 10^{-2}, the privacy loss at $(0.5,\ 10^{-3})$ per batch with batch size of 94, which is roughly the square root of the number of faceprints each user has. For tuning, we set the learning rate to 10^{-3}, the privacy loss at $(1,\ 10^{-3})$ per batch with a batch size of 250. The increase in ϵ is due to the increase of batch size. Due to the memory limitation, each batch is further divided into mini-batches with a size of 25. To reduce experiment time, we start with a pre-trained 11-layer VGG network and initialize the additional layers using Xavier initialization[1]. For SGD, we set the momentum coefficient to 0.9 and weight decay to 5×10^{-4}. The setups for experiment-specific parameters are further explained in the corresponding subsections.

D. Results of Noisy Bootstrapping

To test the performance of our noisy bootstrapping scheme, we introduce artificial tagging noise by replacing the correct tags with random samples from the tag list. The fractions of noise range from 30% to 50% with a step of 2%. The hyper-parameters, α ranges from 0.8 to 1.0 with a step of 0.05. The hash function is constructed using a SHA-1" hash.

The result is shown in Figure 7. The min-entropy regularization gives signification benefit in the case of inconsistent tags. With the permuted tags we introduce, the regularization method is shown to be effective to improve classification accuracy, and it achieves that without explicitly modeling the tag noise distribution. In our empirical result, the regularization method is able to keep the classification accuracy above 90% optimal with as many as 44% inconsistent tags. Consider the fact that human facial recognition accuracy is around 97.25%

[1]It is done by random sampling from a Gaussian distribution with 0 mean and 10^{-2} standard deviation.

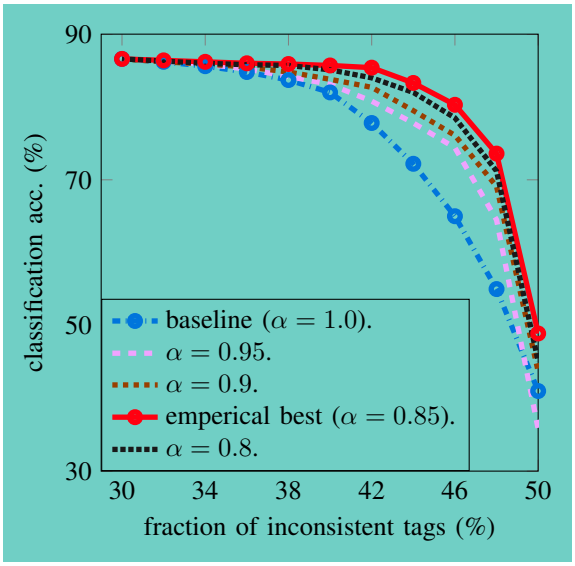

Fig. 7. Users independently tag their faceprint collection to protect privacy, and use entropy minimization regularization to compensate the tag inconsistency. The baseline $\alpha = 1.0$ is the normal softmax classifier. The empirical best $\alpha = 0.85$ achieves the best classification result with inconsistent tags.

[2]. This allows 8 users to each create at most $n' = 530$ tags for the collaborative bootstrapping with $n = 3000$ classes.

E. Results of User-Specific Metric Learning

To show that our method is robust against unconsented faceprint harvesting, we generate user-specific embedding vectors for faceprints with the same tag and measure the similarity scores between them. Intuitively, we want different users to obtain different embedding vectors for the same input. If these vectors are sufficiently separated, each user cannot compare her faceprint collection with the collections of others. This prevents multiple users to collude and combine their faceprint collections for unsavory motives, such as extra-judicial surveillance.

The result is shown in Figure 8. In our implementation, the embedding vectors are on the unit hypersphere, and the scores is computed based on their l2 distance. The range of the scores is between 0 to 4. The embedding metrics φs are trained with MT-LMNN classification. The upper figure shows that users obtains different φs that project the same input to different locations in the embedding space. The lower left figure shows the distribution of similarity scores between embedding vectors generated by the same user. Based on the distribution, a user can use a threshold score of 0.99 to distinguish different people. The lower right figure shows the distribution of similarity scores between embedding vectors generated by different users. Based on this distribution, if two user collude and compare their embedding vectors use the same threshold, the nearest neighbor search will generate the wrong tags with a probability of 93.4%.

F. Results of Collaborative Fine-Tuning

We evaluate the performance of the collaborative fine-tuning through the convergence rates and improvements of verification accuracy. We start with a ψ bootstrapped through

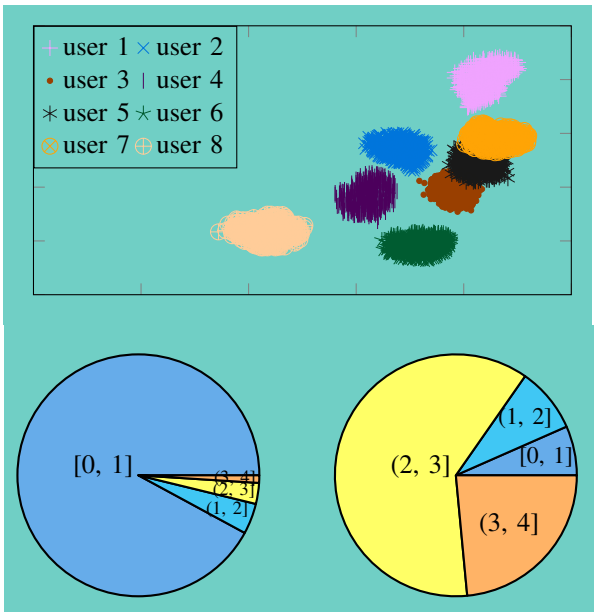

Fig. 8. (Upper) different users' clusters of the largest class in the LFW dataset with 513 images. (Lower, right) the distribution of similarity scores between embedding vectors generated by the same user. (Lower, left) distribution of similarity scores between embedding vectors generated by the different users.

faceprint collections that contains 30% of noise. We compare the tuning using two types of embedding metrics. The first type is the idiosyncrasy embedding metric trained with LMNN, and the second type is the commonality embedding metric trained with MT-LMNN. We set the hyper-parameter, β, to 0.001, and examine the communication periods from the following set, $\lambda = \{16, 32, 64\}$.

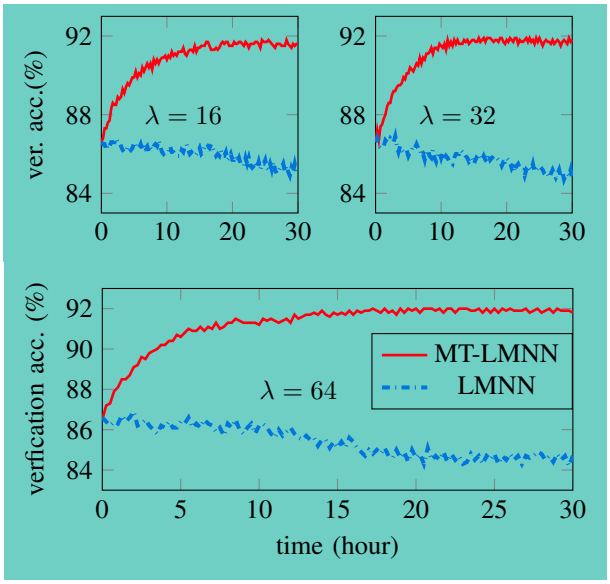

Fig. 9. Tuning the deep neural network with the idiosyncrasy embedding metrics (learned through LMNN and the commonality embedding metrics (learned through MT-LMNN).

The result is shown in Figure 8. Tuning with the idiosyncrasy embedding metrics performs badly on all counts. The verification accuracy degenerates from 86.6% to around 84%. The reason is because each user tries to tune the network with different embedding metrics, and the collective effort does not point to an optimal solution. Due to the small learning rate, the damage is contained. But tuning with the commonality embedding metric shows promising results on all counts. The verification accuracy boosts from 86.6% to around 92.1%, which is comparable to the best accuracy achieved by current `OpenFace` models. This, however, does not realize the full potential of the model. In [9], similar model architecture was able to achieve 97.27% accuracy on the LFW dataset after tuning. But it was trained in a privacy-violating fashion and with a different dataset. Increasing the communication period does not affect the final verification accuracy. But it does slightly increase the convergence rate due to the additional exploration introduced by EASGD.

TABLE III. NUMBER OF PARAMETERS AFTER DROPOUT

dropout rate	num. of param. (bootstrap)	num. of param. (tuning)
0	147 million	147 million
0.3	108 million	108 million
0.4	94 million	94 million
0.5	81 million	81 million

G. Effects of Distributed Learning

We evaluate the effects of parallelism by measure the convergence rates during bootstrap and tuning. We set $\beta = 0.001$, $\lambda = 64$, and examine the dropout rate from the following set, $\{0.3, 0.4, 0.5\}$. We measure the convergence rate by arbitrarily choosing a fixed test accuracy (85% for bootstrapping, 91% for tuning), and measuring the time the algorithm takes to reach that accuracy.

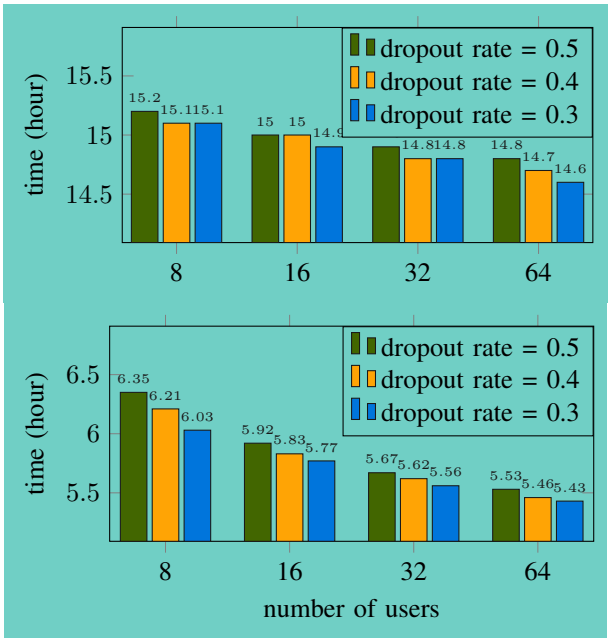

Fig. 10. (Upper) time to reach 85% accuracy during bootstrapping, with different dropout rates. (Lower) time to reach 91% accuracy during fine-tuning, with different dropout rates.

The result is shown in Figure 10. Using random dropout in combination with EASGD enables multiple users to train

the deep learning model in parallel. In fact, as the number of user doubles, the time to reach convergence accelerates by approximately 1%. In addition, because approximately 80% of the parameters come from the fully-connected layers. The dropout method can significantly reduce to number of parameters exchanged between users and the parameter server, as shown in Table III. However, increasing the dropout rate can also delay the time to reach convergence. In our case, increase the dropout rate by 0.1 will delay the convergence time by 2%. Therefore, if the users computation and communication budget permit, a smaller dropout rate is preferred to shorten the training time.

H. Effects of Differential Privacy

Finally, we evaluate the impact of differentially private parameter update. As we discussed earlier, incorporating differential privacy mechanism into the learning process inevitably sacrifices model quality for privacy. Here we perform a detailed evaluation by varying the privacy loss, (ϵ, δ). We set the clipping thresholds, γ_1 and γ_2, to 0.01, the selection threshold, γ_3, to 0.005, and the sparsity threshold, γ_4, to 10,000. We vary the value of ϵ between 0.2 to 1, and vary the value of δ from the following set, $\{10^{-3}, 10^{-4}, 10^{-5}\}$.

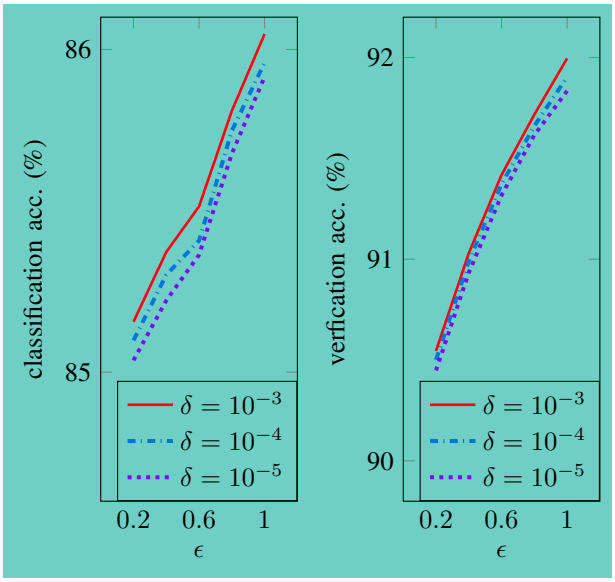

Fig. 11. Evaluation of accuracy with different differential privacy loss (ϵ,δ). Each curve corresponds to a different δ value.

The result is shown in Figure 11. Each curve corresponds to the best accuracy achieved with a fixed δ. Since the noise level σ is a sub linear function of $\delta^{-}1$ (square root of its logarithm), lowering δ (*i.e.*, stronger differential privacy guarantees) slightly decreases the test accuracy for bootstrapping and tuning. In our evaluation, the changes are within 0.05% when δ increases or decreases by a factor of 10. For a fixed δ, changing the value of ϵ has a much larger impact on the test accuracy. Since the noise level σ is a linear function of $\epsilon^{-}1$, lowering ϵ (*i.e.*, stronger differential privacy guarantees) affects the test accuracy almost linearly. In our evaluation, the test accuracies drop 0.003 and 0.005 for bootstrapping and tuning respectively when lowering ϵ by 0.2.

V. Related Work

The existing literature on privacy protection in machine learning mostly targets conventional machine learning algorithms and addresses three objectives: privacy of the data used for learning a model or as input to an existing model, privacy of the model, and privacy of the models output. Techniques based on secure multi-party computation (SMC) can help protect intermediate steps of the computation when multiple parties perform collaborative machine learning on their proprietary inputs. SMC has been used for learning decision trees, linear regression functions, association rules, Naive Bayes classifiers, and k-means clustering. In general, SMC techniques impose non-trivial performance overheads and their application to privacy-preserving deep learning remains an open problem.

Techniques that protect privacy of the model include privacy-preserving probabilistic inference, privacy-preserving speaker identification, and computing on encrypted data. By contrast, our objective is to collaboratively train a neural network that can be used privately and independently by each participant. Differential privacy is a popular approach to privacy-preserving machine learning. It has been applied to boosting, principal component analysis, linear and logistic regression, support vector machines, risk minimization, and continuous data processing. Recent results show that a noisy variant of stochastic gradient descent achieves optimal error for minimizing Lipschitz convex functions over l2-bounded sets, and that randomized dropout, used to prevent overfitting, cal also strengthen the privacy guarantee in a simple 1-layer neural network. To the best of our knowledge, none of the previous work addressed the problem of collaborative deep learning with multiple participants using distributed stochastic gradient descent. Aggregation of independently trained neural networks using differential privacy and secure multi-party computation is suggested in. Unfortunately, averaging neural-network parameters does not necessarily result in a better model. Unlike previously proposed techniques, our system achieves all three privacy objectives in the context of collaborative neural-network training: it protects privacy of the training data, enables participants to control the learning objective and how much to reveal about their individual models, and lets them apply the jointly learned model to their own inputs without revealing the inputs or the outputs. Our system achieves this at a much lower performance cost than cryptographic techniques such as secure multi-party computation or homomorphic encryption and is suitable for deployment in modern large-scale deep learning.

VI. Conclusion

This work is our first attempt to address the potential privacy violation caused by the face tagging features offered by a majority of cloud-based photo management software. We proposed a privacy-preserving face tagging framework based on a multi-task, distributed, collaborative deep learning system. The framework prevents service providers from collecting and storing user's faceprints data without permission, yet still allow multiple users to jointly improve the face tagging functionality. Our preliminary evaluations shows that our framework can achieve state-of-art tagging accuracy (92.1%) with provable differential privacy guarantee.

ACKNOWLEDGMENT

REFERENCES

[1] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 815–823.

[2] L. Wolf, "Deepface: Closing the gap to human-level performance in face verification," in *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on. IEEE*, 2014.

[3] G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller, "Labeled faces in the wild: A database for studying face recognition in unconstrained environments," Technical Report 07-49, University of Massachusetts, Amherst, Tech. Rep., 2007.

[4] D. Yi, Z. Lei, S. Liao, and S. Z. Li, "Learning face representation from scratch," *ArXiv preprint arXiv:1411.7923*, 2014.

[5] Y. Sun, Y. Chen, X. Wang, and X. Tang, "Deep learning face representation by joint identification-verification," in *Advances in Neural Information Processing Systems*, 2014, pp. 1988–1996.

[6] Y. Sun, X. Wang, and X. Tang, "Deep learning face representation from predicting 10,000 classes," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 1891–1898.

[7] ——, "Deeply learned face representations are sparse, selective, and robust," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 2892–2900.

[8] Y. Sun, D. Liang, X. Wang, and X. Tang, "Deepid3: Face recognition with very deep neural networks," *ArXiv preprint arXiv:1502.00873*, 2015.

[9] O. M. Parkhi, A. Vedaldi, and A. Zisserman, "Deep face recognition," in *British Machine Vision Conference*, vol. 1, 2015, p. 6.

[10] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, IEEE, vol. 1, 2001, pp. I–511.

[11] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting.," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[12] S. Zhang, A. E. Choromanska, and Y. LeCun, "Deep learning with elastic averaging sgd," in *Advances in Neural Information Processing Systems*, 2015, pp. 685–693.

[13] C. Dwork, A. Roth, *et al.*, "The algorithmic foundations of differential privacy," *Foundations and Trends in Theoretical Computer Science*, vol. 9, no. 3-4, pp. 211–407, 2014.

[14] 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," *ArXiv preprint arXiv:1511.06807*, 2015.

[15] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," *ArXiv preprint arXiv:1607.00133*, 2016.

[16] S. Sukhbaatar, J. Bruna, M. Paluri, L. Bourdev, and R. Fergus, "Training convolutional networks with noisy labels," *ArXiv preprint arXiv:1406.2080*, 2014.

[17] *Openface*, Available: https://cmusatyalab.github.io/openface/, 2016.

[18] S. Reed, H. Lee, D. Anguelov, C. Szegedy, D. Erhan, and A. Rabinovich, "Training deep neural networks on noisy labels with bootstrapping," *ArXiv preprint arXiv:1412.6596*, 2014.

[19] S. Song, K. Chaudhuri, and A. D. Sarwate, "Stochastic gradient descent with differentially private updates," in *Global Conference on Signal and Information Processing (GlobalSIP), 2013 IEEE*, IEEE, 2013, pp. 245–248.

[20] W. Du and M. J. Atallah, "Secure multi-party computation problems and their applications: A review and open problems," in *Proceedings of the 2001 workshop on New security paradigms*, ACM, 2001, pp. 13–22.

[21] K. Q. Weinberger, J. Blitzer, and L. K. Saul, "Distance metric learning for large margin nearest neighbor classification," in *Advances in neural information processing systems*, 2005, pp. 1473–1480.

[22] S. Parameswaran and K. Q. Weinberger, "Large margin multi-task metric learning," in *Advances in neural information processing systems*, 2010, pp. 1867–1875.

[23] S. Samar, S. Boyd, and D. Gorinevsky, "Distributed estimation via dual decomposition," in *Control Conference (ECC), 2007 European*, IEEE, 2007, pp. 1511–1516.

[24] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *International Conference on Learning Representations*, 2015.

[25] R. Collobert, K. Kavukcuoglu, and C. Farabet, "Torch7: A matlab-like environment for machine learning," in *BigLearn, NIPS Workshop*, 2011.

[26] *Torch-distlearn*, Available: https://github.com/twitter/torch-distlearn.git, 2016.

[27] Gurobi Optimization, Inc., *Gurobi optimizer reference manual, version 6.5*, Available: http://www.gurobi.com/documentation, 2016.

[28] *Gurobi.torch*, Available: https://github.com/bamos/gurobi.torch.git, 2016.

[29] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman, "Return of the devil in the details: Delving deep into convolutional nets," 2014.