Learning from Differentially Private Neural Activations with Edge Computing

Yunlong Mao
National Key Laboratory for
Novel Software Technology
Nanjing University
Nanjing, China
njucsmyl@163.com

Shanhe Yi
Department of Computer Science
College of William & Mary
Williamsburg, US
syi@cs.wm.edu

Qun Li
Department of Computer Science
College of William & Mary
Williamsburg, US
liqun@cs.wm.edu

Jinghao Feng
National Key Laboratory for
Novel Software Technology
Nanjing University
Nanjing, China
fengjinghao14@163.com

Fengyuan Xu
National Key Laboratory for
Novel Software Technology
Nanjing University
Nanjing, China
fengyuan.xu@nju.edu.cn

Sheng Zhong
National Key Laboratory for
Novel Software Technology
Nanjing University
Nanjing, China
sheng.zhong@gmail.com

***Abstract*—Deep convolutional neural networks (DNNs) have brought significant performance improvements to face recognition. However the training can hardly be carried out on mobile devices because the training of these models requires much computational power. An individual user with the demand of deriving DNN models from her own datasets usually has to outsource the training procedure onto an edge server. However this outsourcing method violates privacy because it exposes the users' data to curious service providers. In this paper, we utilize the differentially private mechanism to enable the privacy-preserving edge based training of DNN face recognition models. During the training, DNN is split between the user device and the edge server in a way that both private data and model parameters are protected, with only a small cost of local computations. We rigorously prove that our approach is privacy preserving. We finally show that our mechanism is capable of training models in different scenarios, e.g., from scratch, or through fine-tuning over existed models.**

***Index Terms*—Deep Learning, Convolutional Neural Networks, Privacy-Preserving, Differential Privacy**

I. INTRODUCTION

People with mobile devices such as smartphones, Google glasses or HoloLens can sense the environment and use collected sensitive data (image, sound, and more) to train a deep convolutional neural network (DNN) for various applications, e.g., face recognition of people met before. Usually, there are two ways for these mobile devices to train a DNN. The first one is to send all sensitive data to a central server (or cluster) which has enough computing power. The second one is to perform distributed training with a local DNN being trained on each device. Obviously, the first one is more suitable for mobile devices with limited computing resources. But user's private data will be violated seriously if the central server is untrusted [1]. The second one has higher requirements for devices' computing power. Assuming that it's possible for mobile devices to perform distributed training given powerful equipments like neural engines built in iPhone X and Tensorflow Lite training framework for mobile devices provided by Google, user's private data will still be violated by an active adversary even with efficient privacy-preserving schemes applied [2], [3]. To meet privacy and resource requirements, we offer a more suitable option for privacy-aware DNN training for mobile devices aided by edge computing. In particular, we will show how this can be done through the popular application of DNN based face recognition.

Training a multi-label DNN entirely on a mobile device is daunting due to resource limitations. Recently, many DNNs have been proposed for training fine-grained face recognition models, such as [4], [5], [6], [7], [8]. A notable feature that those networks have in common is the great depth of networks. The large amount of parameters to train makes both forward-passing convolutional computation and backward-passing weights updating very expensive. For example, a popular DNN VGG-16 [9], [5] needs 28 GB memory and several days for training with batch size 256[10]. Meanwhile, users' privacy cannot be guaranteed in client-server model. An untrusted server can peek at users' images containing confidential information. Even if cryptographic tools or obfuscation schemes are applied to protect images, membership inference attacks against a trained model could still be achieved [11], [12] even without training data present. There is still a huge gap between deploying fully functional DNNs on devices and reality.

Privacy issue in deep learning has been a hot topic recently because of severe privacy leakage results [11], [3], [12]. Several efficient privacy-aware DNN training mechanisms have been proposed. A differentially private (DP) gradients computing mechanism is designed in [13] to protect locally trained parameters. Privacy-preserving parameters aggregation for distributed learning has been studied in [2], [1]. A DP parameters updating mechanism is introduced in [2], while a secure parameters aggregation mechanism based on combing masking technique and threshold secret sharing is proposed in [1]. Targeting at privacy-preserving fine-tuning, [14] migrates the learning process from a client to a server after mixing basic features extracted by clients with noise. However this scheme focuses on fine-tuning only. None of these existed mechanisms can protect mobile user's private data in a client-server training model very well.

This paper tries to fill the gap by introducing a new client-server model based DNNs training scheme in privacy-preserving manners. Users within the scheme just do limited computation to train very deep neural networks on off-the-shelf devices and aided edge server with users' private data preserved. For example, smartphone users can train their own multi-label face recognition models on a edge server with their private images. Another possible application is to use smart cameras and edge computing server [15] to make real-time neighborhood surveillance with residents' privacy preserved. It will just take seconds to process a batch on mobile client. Edge server aided deep learning has attracted lots of attentions, such as [16], [17]. Our privacy-preserving deep learning scheme is more suitable in edge computing context than cloud computing environment. Because uploading the sanitized data to the server will cause a high network traffic demand. A server located in close geographical proximity to the user will have significant advantage to improve overall performance.

The basic idea of our scheme is based on an important observation, that a DNN can be split inside between two successive layers and deployed two partitions on different locations without hazarding the optimization. To minimize the cost of mobile users, we partition DNN after the first convolutional layer. Deploy the first part on user side while the second part on the edge server side. We keep the output of the user part privacy-preserving and feed the output of the user part as the input of the second part on server side. We avoid cryptographic tools so that we can keep user side lightweight. Meanwhile, we use the differential privacy to ensure a strong privacy guarantee for user's confidential datasets. We use VGG-Face network as a study example. Our scheme is based on VGG-16 architecture, but not limited to specific image processing techniques. In general, our contribution can be summarized as,

- We have proposed a privacy-preserving algorithm to calculate DP activations for convolutional layers. Based on this algorithm, we have designed a new privacy-preserving DNN training scheme for face recognition. We prove that our scheme has tighter privacy loss estimates than simply multiple compositions of differential privacy for both training and fine-tuning scenarios.
- By tracing privacy loss, noise level and training accuracy for different partitioning positions in DNNs, we give some inspirations for how to choose the optimal partitioning strategy depending on customized metrics (e.g. computing resources, privacy loss, training accuracy). We prove that partitioning at the first convolutional layer is the optimal solution for a trade-off between computing resources, privacy loss and training accuracy.
- We have implemented a privacy-preserving VGG-Face network for face recognition. We have also implemented a client demo on Android smartphone. We evaluate our scheme for training and fine-tuning tasks using public datasets. Evaluation results show that both privacy and accuracy are satisfactory.

II. Related Work

Recently, severe attacks against DNNs have been identified [12], [11], [3]. In [12], a model-inversion attack is identified against DNNs. An adversary may be able to extract parts of the training data from a well trained model. For example, an efficient model-inversion attack that recovers face images from a trained facial recognition system is demonstrated in [12]. Furthermore, a membership inference attack is reported in [11]. To address privacy issue in deep learning, Abadi et al. proposed a solution using differential privacy to protect model's parameters [13]. Core idea of Abadi's solution is to add Gaussian noise to the expected gradient in each iteration of neural network before applying weights updating algorithm. However, privacy threats of input data in client-server model is not considered in this work. A distributed client-server application with differential privacy is considered in [2], where each party keeps its input dataset private and jointly learns an accurate neural network model. To achieve this goal, Shokri et al. proposed a DP parameters updating algorithm to update the global model [2]. Specifically, each party locally runs a neural network algorithm on private dataset. Then all participants upload selected gradients to an aggregation server which will be in charge of global parameters updating. The selection of gradients includes how many parameters to update and perturbed values of gradients. This selection procedure follows differential privacy.

Within the same system model, Bonawitz et al. proposed a secure aggregation scheme for multi-party deep learning [1]. This paper uses masking technique and threshold secret sharing method rather than differential privacy to achieve secure aggregation of global parameters. This paper provides promising security. However this method cannot be applied to a single mobile device. In [18], Dowlin et al. reconstruct operators in neural network with primitive operations, such as addition and multiplication. Then authors use leveled homomorphic encryption to protect input datasets and intermediate results. Secure multi-party computation is a promising solution for general privacy-aware DNNs. But this general solution may suffer from heavy computation and complex communication protocols. It is not suitable for resources limited devices.

Wang et al. has proposed a privacy-preserving deep learning framework with aided cloud server in [19], which describes how to inference with private data while the training phase is done with public data and generated noise. Client will use a lightweight DNN to extract raw features from private data for inference while training phase is mainly done in cloud server with public data. The most significant difference between their work and ours is that we preserve more precise features for both training and inference. On the other hand, client in our solution has less computing load but higher communication cost, when compared with [19].

Another research relevant to our work is [14], Osia et al. proposed a transfer learning based privacy-preserving DNN architecture. Low-level features are first extracted. Then principal component analysis is done before the client sends extracted features to the server. The way that client's privacy keeps preserved is to add noise to extracted features. Authors of [14] have done great work. But this scheme focuses on fine-tuning applications. Their architecture is not capable of training from scratch. In this paper, we propose a new scheme to address privacy issues in both training and fine-tuning scenarios for deep learning. Fine-tuning on well trained models is to train a small portion of DNN parameters while training from scratch will train all parameters. Taking VGG-16 as an example, the amount of parameters trained from scratch is about 33 times as much as fine-tuning parameters of the last one fully connected layer.

III. Preliminaries

In this section, we will briefly review the deep neural network architecture of our interest and the definition of non-interactive DP mechanism.

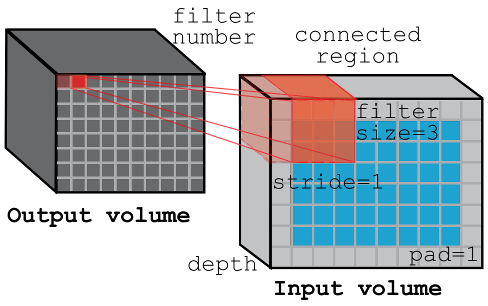


Fig. 1. Two neurons in a convolutional layer use one shared filter to compute output with their own locally connected region as input. Parameters including filter size, pad, stride, volume depth are also demonstrated here.

A. Deep Convolutional Neural Networks

DNNs, such as [20], [4], [5], [7] have similar architectures with other neural networks in appearance. What makes a convolutional neural network (CNN) mostly special is the **convolutional layer**. A convolutional layer is where filters convolve around input volume. Each neuron in the convolutional layer will compute a dot product between its weights and a locally square region that it is connected to in the input as corresponding output. If each neuron in one convolutional layer is connected with entire input volume, then this layer is also called fully connected layer. In order to extract features from images correctly and efficiently, locally connected convolutional layers are used as hidden layers. In short, locally connected convolutional layers treat each region of input as a feature dimension, while every local region of input is connected with one neuron. Instead of training each neuron individually, all neurons in one convolutional layer will share the same weight parameters of each filter. Generally, output volume size of a convolutional layer depends on input size, depth, kernel size, stride and padding style. Figure 1 shows how a convolutional layer works with these parameters.

As pointed out in [20], a Rectified Linear Unit (ReLU) right behind a convolutional layer can make training much faster than using a tanh unit. Thus, we assume the network we talk about uses **ReLU** to active all neurons' output. Then any neuron in a convolutional layer with ReLU applied will yield activation: $f(x) = \max(0, x)$, where x is the output of the neuron. A **pooling layer** in CNNs will summarize activations of neighbouring neurons in a convolutional layer and narrow down the input dimensions of the next convolutional layer. The most popular pooling method is MAX pooling, which will produce the maximum neuron among neighbouring neurons.

In the beginning of network workflow, the first convo-

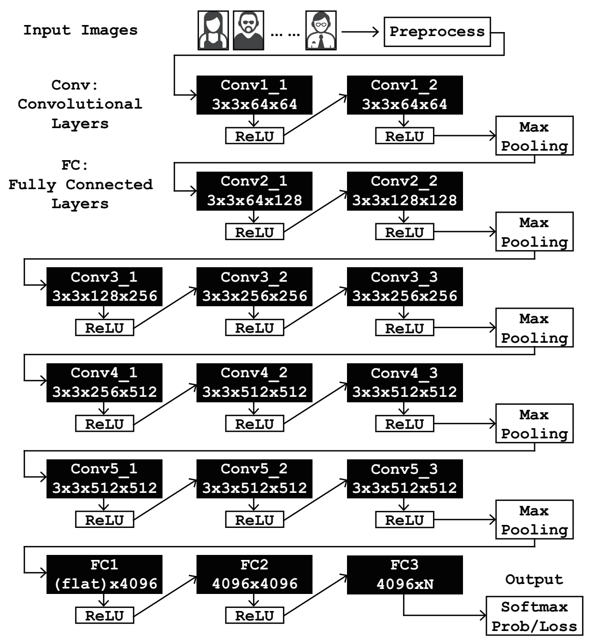


Fig. 2. Network architecture and workflow of VGG-Face. Dimensions and number of kernels in each convolutional layer are indicated. We will use "conv" with numbers to refer to some specific convolutional layer hereafter. Portraits in the figure are designed by Freepik

lutional layer will take face images as its input. Profiting from efficient algorithms, such as [21] and [22], human faces in images can be detected and aligned quickly and precisely. So, we assume that all face images in users' datasets are perfectly aligned. Note that our privacy-preserving scheme is designed for all DNN based face recognition. However we will use VGG-Face network in this paper as an example. Figure 2 demonstrates network architecture and workflow of VGG-Face.

B. Differentially Private Mechanism

Differential privacy [23] provides a promising foundation of privacy property. It is possible to implement a privacy-preserving mechanism for any specific application by using DP mechanism. In face recognition application, datasets for training and testing are collections of pairs of face images and labels. But we need more fine-grained datasets if we want to take a deep investigation of DNNs. The output of a convolutional layer with ReLU is volume of activations. These activations can be seen as outputs of specific query functions (or kernels). All regions extracted from all labeled face images will compose domain $\mathcal{D}$. All regions in the same position in all face images compose a dataset $d \in \mathcal{D}$. Thus an element in dataset d will be a region of pixels (include all color channels) in one image. An adjacent dataset d' is a dataset that differs in a single pair of pixel group and label with d.

Definition 1 (Differentially Private Mechanism)**.** *A random mechanism $M : \mathcal{D} \rightarrow \mathbb{R}$ with domain $\mathcal{D}$ and range $\mathbb{R}$ satisfies (ϵ, δ)-differential privacy if for any two adjacent inputs $d, d' \in \mathcal{D}$ and for any subsets of outputs $s \subset \mathbb{R}$ it holds that*

$$Pr[M(d) \in S] \leq e^{\epsilon} Pr[M(d') \in S] + \delta. \tag{1}$$

This definition [23] of differential privacy allows for that original ϵ-differential privacy can be broken with probability δ. To derive a (ϵ, δ)-DP mechanism from an original function $f : \mathcal{D} \rightarrow \mathbb{R}$, additive noise correlated to the sensitivity S_f of function f is needed. S_f can be defined as

$$S_f = \max_{d,d' \in \mathcal{D}} |f(d) - f(d')|. \tag{2}$$

Then if we want to construct a (ϵ, δ)-DP mechanism with Gaussian noise, it can be achieved by

$$M(d) \triangleq f(d) + \mathcal{N}(0, S_f^2 \cdot \sigma^2), \tag{3}$$

where $\mathcal{N}(0, S_f^2 \cdot \sigma^2)$ is a Gaussian distribution with mean 0 and standard deviation $S_f \sigma$. Following the theorem proposed in [23], this type of construction, corresponding to f and S_f, can guarantee that mechanism $M(d)$ satisfies (ϵ, δ)-differential privacy for multiple combinations of possible ϵ and δ.

IV. Threat Model

In a client-server model, the client is supposed to send training data to the server, and the server then performs training for the client. The privacy considered here is about client's confidential data. So the privacy property that we want to guarantee is, no semi-trusted server can tell whether a specific labeled face image is in client's datasets or not if the server has not seen this labeled image in any public dataset.

The semi-honest server is considered to be curious. This adversary type is similar to [2], [13]. However ours has a view of the whole learning procedure including input, output and parameters of every network layer which is on the server side, while [2] feeds the adversary with selected gradients and the adversary of [13] only has a view of the model's parameters. This means that our adversary is relatively powerful and hard to defend against.

Generally, the curious server can violate client's privacy by copy-and-embezzling client's input images directly, or infering client's input images from learned model parameters [12]. We will deal with these two attacks in this paper. We assume that the server has unlimited computing power while communication channel between the server and any client is physically secure. Other kinds of attacks will be left for further study.

V. Differentially Private Deep Convolutional Neural Network

In neural networks, each hidden layer can be seen as a separate unit taking previous layer's output as its input. Input of the first layer in DNNs is special because it is original image data. It is not recommended to use high noise to perturb images directly because the utility of images will be damaged seriously [24]. However we consider to partition the network inside at some specific convolutional layer instead of the input layer because layers in DNNs are loosely coupled units. Once we select one layer to partition the whole network into two parts for the client and server, the client can hide all intermediate results and input from the server. All information that the server needs to know for the forward passing is the output of the last layer in the client's part. As for backward passing, the client can compute gradients and update parameters by following the chain rule as long as the server provides the partial of loss with respect to client's output.

To this end, we need to find out where should we partition the DNN and how can we preserve client's privacy while avoiding resource intensive computation at the same time. One important result of our study is that for any specific DNN, if all output activations of i-th layer are (ϵ, δ)-DP then weights updating mechanism will be ϵ_i-DP for each iteration, where ϵ_i is privacy budget for the i-th layer. And $\epsilon_i > \epsilon_j$, if $i < j$ for any two convolutional layers in the network. This means the deeper layer we interfere with, the more privacy is kept but the less recognition accuracy we obtain, if we use the same level noise. Without the loss of generality, we will introduce our scheme with a simple partitioning strategy where the client holds the first convolutional layer (with ReLU attached) and feeds the output to an untrusted server, who will succeed the client to finish following layers in a pre-designed DNN. In the next section, we will discuss how to make the optimal selection of partitioning position.

In our scheme, client sends output activations of the first convolutional layer instead of raw images to the edge server. Artificial noise will be added to output activations. The addition of artificial noise can prevent adversary edge server from reversing activations in case that the edge server learns parameters of the first convolutional layer somehow (e.g. if client does fine-tuning on any public pre-trained model). We are going to show that if we use DP Gaussian noise then model parameters' updating will also be privacy-preserving. A partitioning example for the VGG-Face network is illustrated in Figure 3.

A. Differentially Private Activations

As shown in Figure 3, when we partition VGG-Face network into two parts, except for the first convolutional layer on the client, all other layers being deployed on the edge server are the same as in original VGG-Face network. When the client communicates with the edge server, in client's view, server's partial network can be seen as a black-box function and vice versa. In forward passing, the first layer on the client can be seen as a composite function, whose input is client's datasets and output is volume of activations. Activation function denoted by f is actually the composition of kernels in convolutional layer, ReLU and local response normalization unit. In the first convolutional layer, we assume that there are m kernels with size $l \times l \times r$, where r is number of color channels. When one training example s in client's private datasets goes into $f^i, i \in [1, m]$, each activation is generated corresponding to one spatial position (e.g. (x, y)) on the surface of face image s. Then function f^i (f will be used equivalently if no ambiguity is caused.) can be defined as,

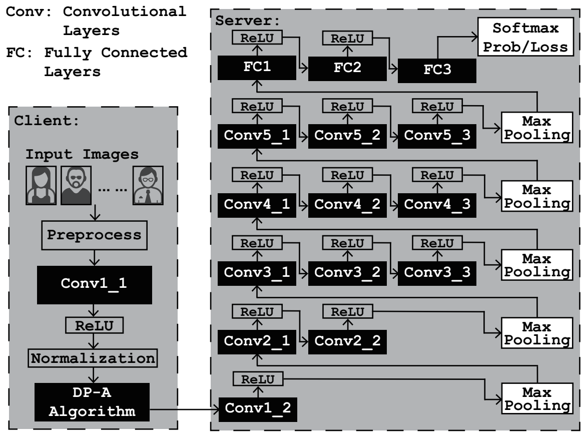


Fig. 3. After the partitioning, the first convolutional layer with our DP-algorithm is deployed on the client while the rest part of VGG-Face network will be deployed on an edge server.

$$f^i(s(x,y)) = a^i_{(x,y)} / (\gamma + \alpha \sum_{j=\max(0,i-\frac{u}{2})}^{\min(m-1,i+\frac{u}{2})} (a^j_{(x,y)})^2)^{\beta}, \tag{4}$$

where u, α, β, γ are constants which should be determined using a validation set empirically (Please refer to ImageNet paper [20] for more detailed information if interested). $a^i_{(x,y)}$ is the activation generated by i-th kernel at position (x, y).

To protect confidential datasets of the client, we need to guarantee that output activations of function f for every single image is privacy-preserving. Function f can be regarded as a specific query on client's datasets $d \in \mathcal{D}$. To construct a (ϵ, δ)-DP mechanism for f with Gaussian noise, sensitivity of f on adjacent datasets d, d' should be clarified. Based on the definition of function sensitivity [23] and ReLU's output activation $a^i_{(x,y)} \geq 0$,

TABLE I
NOTATION TABLE FOR QUICK REFERENCE

Notation	Comments
h, w, r	Image height h, width w, color channels r.
(x, y)	Pixel position offset, relative to top left of the image.
k_i	The i-th kernel in one convolutional layer, $i \in [1, m]$.
$\boldsymbol{K}_i$	Flattened vector of kernel k_i.
$g_{(x,y)}$	Group of pixels designated by a square region with length l, top left vertex at (x, y).
$\boldsymbol{G}_{(x,y)}$	Flattened vector of pixels group $g_{(x,y)}$.
$a^i_{(x,y)}$	Output activation of a neuron computed by applying kernel k_i at position (x, y) with the ReLU applied.
$c^i_{(x,y)}$	Output of $a^i_{(x,y)}$ with local response normalization applied.
$d^i_{(x,y)}$	Output of $c^i_{(x,y)}$ by applying DP mechanism.

we can define f's sensitivity S_f as,

$$\begin{aligned} S_f = \max_{d,d' \in \mathcal{D}} &|a^i_{(x,y)}(d)/(\gamma + \alpha \textstyle\sum_{j=\max(0,i-\frac{u}{2})}^{\min(m-1,i+\frac{u}{2})} (a^j_{(x,y)}(d))^2) \\ &- a^i_{(x,y)}(d')/(\gamma + \alpha \textstyle\sum_{j=\max(0,i-\frac{u}{2})}^{\min(m-1,i+\frac{u}{2})} (a^j_{(x,y)}(d'))^2)^\beta \\ \leq \max_{d \in \mathcal{D}} &a^i_{(x,y)}(d)/(\gamma + \alpha \textstyle\sum_{j=\max(0,i-\frac{u}{2})}^{\min(m-1,i+\frac{u}{2})} (a^j_{(x,y)}(d))^2)^\beta . \end{aligned}$$

Given sensitivity S_f, when f is applied at the same spatial position (x, y) on training face images, we can use $c^i_{(x,y)} + \mathcal{N}(0, S_f^2\sigma^2)$ to replace $c^i_{(x,y)}$ as output of f. Since $0 \leq a^i_{(x,y)}(s) < 1, \forall s \in d$ after we pre-process input images, we can have $0 \leq S_f < 1/\sqrt{2}$ when we select parameter $u = 5, \alpha = 1, \beta = 0.5, \gamma = 2$.

Parameters including stride, padding and kernel size should be determined before training session begins. With input images' size fixed, the shape of output volume of the neurons with m kernels should be,

$$pqm = (\lfloor \frac{w + 2pad - l}{stride} \rfloor + 1)(\lfloor \frac{h + 2pad - l}{stride} \rfloor + 1)m. \quad (5)$$

Although parameters including stride, padding and kernel size should usually be selected to give perfect alignment, we use rounding operator here just in case. In this way, we will have a (ϵ, δ)-DP mechanism for activations of convolutional layer. Detailed algorithm of DP activation mechanism named DP-A is shown in Algorithm.1. Note that input images should subtract their mean and then be divided by standard variance cross each mini-batch. Some notations used in our algorithm can be referred to Table.I.

B. Privacy-Preserving Weights Updating

After DP activations are transmitted to the edge server, client's task in forward passing is finished. To continue training, the edge server will run subsequent training process from the second convolutional layer with activations received as its input. There is no additional modification

Algorithm 1 DP activation mechanism DP-A

```
Require: Pre-processed training example $s(w, h, r)$,
    kernels and bias $\{k_i, b_i | 0 < i \leq m\}$, $stride$, $pad$.
Ensure: DP activations $\{d^i_1, d^i_2, \ldots, d^i_{p\times q}\}^m_{i=1}$.
 1: for all $k_i, i \in [1, m]$ do
 2:   In padded $s$, $x \leftarrow 0$, $y \leftarrow 0$
 3:   while $y + l < h + pad \times 2$ do
 4:     while $x + l < w + pad \times 2$ do
 5:       $a^i_{(x,y)} \leftarrow \max(0, < \boldsymbol{G}_{(x,y)}, \boldsymbol{K}_i > + b_i)$
 6:       $x \leftarrow x + stride$
 7:     end while
 8:     $y \leftarrow y + stride$
 9:   end while
10:   $x \leftarrow 0$, $y \leftarrow 0$
11:   while $y + l < h + pad \times 2$ do
12:     while $x + l < w + pad \times 2$ do
13:       $c^i_{(x,y)} \leftarrow \frac{a^i_{(x,y)}}{(\gamma + \alpha \sum_{j=\max(0,i-\frac{u}{2})}^{\min(m-1,i+\frac{u}{2})} (a^j_{(x,y)})^2)^\beta}$
14:       $d^i_{(x,y)} \leftarrow c^i_{(x,y)} + \mathcal{N}(0, S_f^2 \times \sigma^2)$
15:       $x \leftarrow x + stride$
16:     end while
17:     $y \leftarrow y + stride$
18:   end while
19: end for
```

needed for our scheme to be deployed on the edge server. However, it's important to ensure that when all activations generated by DP activation algorithm are compounded through server's convolutional layers, the output will still be privacy-preserving. The output of each layer can be seen as combination of DP activations. The total loss of network prediction can also be seen as a composed mechanism of multiple DP mechanisms. But the real challenge is how to guarantee the privacy loss of our composed mechanism can be tightly bounded instead of a simple composition of multiple DP mechanisms.

When multiple DP mechanisms are composed, the privacy guarantee normally goes down. Basically we encounter an adaptive composition situation suggested in Dwork's boosting theory [25]. If we directly follow the adaptive composition theory, the prediction mechanism of this deep learning network should be $(\epsilon', pq\delta + \delta')$-DP in adversary's view, where p, q are dimensions of activations for each kernel, $\epsilon' = \epsilon\sqrt{2pqln(\frac{1}{\delta'})} + pq\epsilon(e^\epsilon - 1), \delta' > 0$. We find that in DNN training, we can actually achieve a tighter bound than simple composition theory.

We use softmax log-loss function L to score the VGG-Face network's prediction. As to the gradient computation in backward passing, we use a popular method called mini-batch stochastic gradient descent (SGD). Simply, if we want to update parameters W in t-th iteration, we can use $W_t = W_{t-1} - lr * g_t$, where g_t is an average estimation cross the mini-batch to the gradient $\frac{\partial L}{\partial W}$, lr indicates learning rate. And this estimation g_t can be calculated by $g_t = \frac{1}{|S|}\sum_{s\in S}\frac{\partial L(s)}{\partial W}$, where S is a mini-batch randomly generated from dataset $d \in \mathcal{D}$. Loss L w.r.t s_i is $L(s_i) = -\log(\frac{e^{o_i}}{\sum_{j=1}^{N} e^{o_j}})$, where o_i is the prediction score of image sample s_i for one identification indexed by i among all N identifications.

Assume that we are looking at the first iteration of training session. All weight parameters in convolutional layers are initialized by sampling from normal distribution $N(0, 0.01)$. For prediction mechanism $M_p(S)$, the definition of adjacent datasets are still the same as before, but the element of dataset is one integral face image instead of just one group of pixels. This means we group dataset families for activating function into one new dataset family D. If we process two identical batch generated from D with M_p, then output loss for each sample should be the same. So is the gradient estimation. To show that M_p can be privacy-preserving, we need bound probabilistic differential when M_p applies to two adjacent datasets $d, d' \in D$.

Corollary 1. *Softmax log-loss L is ϵ_1-DP, where $\epsilon_1 = p\times q\times\epsilon - c$ and c is a small constant, if output activations of the first convolutional layer are all (ϵ, δ)-DP.*

Proof. Let d, d' be two adjacent batches where d has sample s' while d' has a faked record replacing s'. $|d| = |d'| = M$. For any $s_j \in d$, output of i-th convolutional layer with ReLU applied is denoted by $CL_i(s_j)$ with output value $l_j^{(i)}$, output of i-th convolutional layer before ReLU is denoted by $\hat{CL}_i(s_j)$ with possible value $\hat{l}_j^{(i)}$, where $j \in [1, M], s_j \in d$.

Assume fully connected layers don't apply dropout. If $l_j^{(15)}$ is the output of 15-th convolutional layer, then $Pr[M_p(s_j) = l|CL_{15}(s_j) = l_j^{(15)}] = 1$. But because $Pr[CL_{15}(s_j) = l_j^{(15)}|\hat{CL}_{15}(s_j) = \hat{l}_j^{(15)}]$ may not equal to 1, we assume that there are $n_j^{(15)}$ activations in $l_j^{(15)}$ are turned from negative to zero. Then $\hat{l}_j^{(15)}$ has a subspace $\Phi_j^{(15)}$ where $n_j^{(15)}$ activations before ReLU can have any negative values. Then, the privacy loss can be traced as,

$$
\begin{aligned}
&\frac{Pr[M_p(d) = L]}{Pr[M_p(d') = L]} \\
=&\frac{Pr[M_p(d) = L|CL_{15}(d) = l^{(15)}]Pr[CL_{15}(d) = l^{(15)}|}{Pr[M_p(d') = L|CL_{15}(d') = l^{(15)}]Pr[CL_{15}(d') = l^{(15)}|} \\
&\frac{\hat{l}^{(15)} \in \Phi^{(15)}]Pr[\hat{l}^{(15)} = \phi, \phi \in \Phi^{(15)}]}{\hat{l}^{(15)} \in \Phi'^{(15)}]Pr[\hat{l}^{(15)} = \phi, \phi \in \Phi'^{(15)}]}
\end{aligned}
$$

$$
\begin{aligned}
=&\frac{Pr[M_p(d) = L|CL_{15}(d) = l^{(15)}]Pr[CL_{15}(d) = l^{(15)}|}{Pr[M_p(d') = L|CL_{15}(d') = l^{(15)}]Pr[CL_{15}(d') = l^{(15)}|} \\
&\frac{\hat{l}^{(15)} \in \Phi^{(15)}]Pr[\hat{l}^{(15)} = \phi, \phi \in \Phi^{(15)}|CL_{14}(d) = l^{(14)}]}{\hat{l}^{(15)} \in \Phi'^{(15)}]Pr[\hat{l}^{(15)} = \phi, \phi \in \Phi'^{(15)}|CL_{14}(d') = l^{(14)}]} \\
&\frac{Pr[CL_{14}(d) = l^{(14)}|\hat{l}^{(14)} \in \Phi^{(14)}]Pr[\hat{l}^{(14)} = \phi, \phi \in \Phi^{(14)}]}{Pr[CL_{14}(d') = l^{(14)}|\hat{l}^{(14)} \in \Phi'^{(14)}]Pr[\hat{l}^{(14)} = \phi, \phi \in \Phi'^{(14)}]}.
\end{aligned}
$$

Since max pooling layers will take the maximal activation within each pooling frame, they reduce the output size of convolutional layers. But on the other hand, each max pooling layer helps to increase the possibility space Φ of the convolutional layer that is just in front of it.

The possibility space of CL_2 in the chain to trigger $M_p = L$ has momentum $n^{(2)}$. Since $n^{(2)} \geq 1$, CL_1 can have at least one activation output to be turned. Recall that the output of the first convolutional layer $l^{(1)}$ consists of $p \times q$ activations corresponding to different subregions. Here we donate those activation elements of the first layer by Sub_i with value $l_i^{(1)}, i \in [1, p\times q]$. Noting that $Pr[Sub_i(d) = l_i^{(1)}] = e^{\epsilon}Pr[Sub_i(d') = l_i^{(1)}] + \delta$, we can revisit the privacy loss,

$$
\begin{aligned}
&\frac{Pr[M_p(d) = L]}{Pr[M_p(d') = L]} \\
\leq&\frac{\prod_{i=1}^{p\times q} Pr[Sub_i(d) = l_i^{(1)}]}{\prod_{i=1}^{p\times q} Pr[Sub_i(d') = l_i^{(1)}] + \sum_{j=1}^{p\times q}\prod_{i=1,i\neq j}^{p\times q}} \\
&\frac{}{Pr[Sub_i(d') = l_i^{(1)}]Pr[\hat{l}^{(2)} = \phi, \phi \in \Phi'^{(2)}} \\
&\frac{}{|Sub_j(d') \neq l_j^{(1)}]Pr[Sub_j(d') \neq l_j^{(1)}]} \\
\leq&\frac{1}{(\frac{1-\delta'}{e^{\epsilon}})^{p\times q} + \frac{p\times q\times(1+c_0)(1-\delta')^{p\times q}}{e^{p\times q\times\epsilon}}} \\
=&\frac{e^{p\times q\times\epsilon}}{(1-\delta')^{p\times q}(1+p\times q\times c_0)} \\
=&\exp(p\times q\times\epsilon - ln(1+p\times q\times c_0) - p\times q ln(1-\delta')),
\end{aligned}
$$

where $\delta' > 0, c_0 \in (0,1)$. If $\delta' \simeq 0$, then $ln(1-\delta') \simeq 0$. Since $ln(1+p\times q\times c_0) \leq p\times q\times c_0$ will always hold, we can ensure there exists real number c to satisfy $ln(1+p\times q(1+c_0)) + p\times q ln(1-\delta') \leq c$. Hence, $Pr[M_p(d) = L]/Pr[M_p(d') = L] \leq \exp(p\times q\times\epsilon - c)$. By symmetry, $Pr[M_p(d') = L]/Pr[M_p(d) = L] \geq \exp(-p\times q\times\epsilon + c)$. □

Based on this result, we can also count privacy loss in backward passing. One trip of backward passing mainly consists of gradient computing and variable updating. When we have gradient computed, variable updating will be trivial. So the part that really matters is gradient computing. To secure user's privacy against revealing from model's parameters, we here prove that our privacy-preserving weights updating satisfies differential privacy. Since we can control privacy loss with ϵ, the privacy level can be designed as a flexible hyperparamter for user to choose (within a feasible range).

Corollary 2. *Weights updating mechanism is* $(O(p_b \epsilon_0 \sqrt{T}), \delta_0)$*-DP for* T *iterations, where* $\epsilon_0 = c' + c$*, if output activations of the first convolutional layer are all* (ϵ, δ)*-DP.*

Proof. Given loss L, we can calculate gradient for weights of previous convolutional layer CL_{15} by,

$$\frac{\partial L}{\partial w_{ij}^{(15)}} = \sum_{x=1}^{w-l+1} \sum_{y=1}^{h-l+1} \frac{\partial L}{\partial \hat{l}_{(x,y)}^{(15)}} l_{(x+i,y+j)}^{(14)}. \tag{6}$$

The delta part on the right hand of this equation can be computed using chain rule,

$$\frac{\partial L}{\partial \hat{l}_{(x,y)}^{(15)}} = \frac{\partial L}{\partial l_{(x,y)}^{(15)}} \frac{\partial l_{(x,y)}^{(15)}}{\partial \hat{l}_{(x,y)}^{(15)}} = \frac{\partial L}{\partial l_{(x,y)}^{(15)}} f_a'(\hat{l}_{(x,y)}^{(15)}), \tag{7}$$

where f_a' is the derivative of activation function. When we want to compute gradient for any lower convolutional layer, we need the loss to be computed for that layer first. Loss for the u-th convolutional layer can be computed by,

$$\frac{\partial L}{\partial l_{(x,y)}^{(u)}} = \sum_{i=1}^{l} \sum_{j=1}^{l} \frac{\partial L}{\partial \hat{l}_{(x-i,y-j)}^{(u+1)}} w_{(i,j)}. \tag{8}$$

Given Equation (6-8), it can be inferred that when we want to compute gradient for the u-th convolutional layer, $l^{(u-1)}$, $\hat{l}^{(u)}$, $l^{(u)}$ are needed. Given loss L, probability of $l^{(u-1)}$, $\hat{l}^{(u)}$, $l^{(u)}$ happening can be calculated by conditional probability. Thus similar with proof of M_p, we can bound privacy loss for weights updating mechanism,

$$\begin{aligned}
&\frac{Pr[M_u(d) = g_t]}{Pr[M_u(d') = g_t]} \\
\leq &\frac{(\prod_{i=1}^{p \times q} Pr[Sub_i(d) = l_i^{(1)}])Pr[\hat{l}^{(u)} \in \Phi^{(u)}]/Pr[M_p(d) = L]}{(\prod_{i=1}^{p \times q} Pr[Sub_i(d') = l_i^{(1)}])Pr[\hat{l}^{(u)} \in \Phi'^{(u)}]/Pr[M_p(d') = L]} \\
\leq &e^{c'} \frac{(\prod_{i=1}^{p \times q} Pr[Sub_i(d) = l_i^{(1)}])Pr[M_p(d') = L]}{(\prod_{i=1}^{p \times q} Pr[Sub_i(d') = l_i^{(1)}])Pr[M_p(d) = L]} \\
= &\exp(c' + p \times q \times \epsilon - (p \times q \times \epsilon - c)),
\end{aligned}$$

where $e^{c'} \in (0, 1)$ depends on differential between momentums of $\Phi^{(u)}$ and $\Phi'^{(u)}$. This proof is for single iteration. Following the results that [13], [23] give, we can compose weights updating mechanisms with regarding to iterations as $(O(p_b \epsilon_0 \sqrt{T}), \delta_0)$-DP mechanism, where p_b is the sampling ratio of each batch, T is the total iteration number. □

C. Communication Complexity and Computation Complexity

In forward passing of training, the client should transmit activations volume and classification identifications to the server. Assume it will be E epochs when the server has done T iterations' training and each epoch consumes N images. Total activations the client needs to send is $ENpqm$. Assuming that activation type is *float32* and label type is *int32*, we can have client's transmission amount as $\Theta(4EN(pqm + 1))$ bytes for entire training session. In backward passing phase, the client need receive server's transmission for weights updating. Assuming that type of calculated loss for the first layer is *float32*, we can have client's reception amount as $\Theta(4T)$ bytes for entire training session.

Computation in client's side mainly contain three phases: pre-processing, activation computing, weights updating. Pre-processing can be bounded by $O(3EN)$ since the client needs to substract images' mean and divide images by standard deviation. Complexity of activation computing is actually the complexity of our DP-A algorithm, which can be bounded by $O(2pqmT)$ for T iterations. weights updating process can be bounded by $O(3pqmT)$ for T iterations. Hence, total computation complexity on client's side can be bounded by $O(3EN + 5pqmT)$ for the entire training session.

D. Fine-tuning

Many fine-grained, well-trained face recognizing models are public for use. Public face recognizing models such as [5], [7], [26], [8] have contributed a lot to accelerate development of both academia and industry. But how to make greater use of these public models is still an open question. One of the major concerns is that public models are pre-trained using fixed datasets. Datasets for training will never be big enough to satisfy all applications, especially when we want to recognize some identities belonging to private datasets. Also, some people may not want to train an entire DNN from scratch in practice, because it is difficult to get a proper dataset of sufficient volume and training procedure usually takes long time. Instead, it is a good idea to do fine-tuning [27] on a pre-trained DNN with one's own target datasets.

The scheme we proposed is capable of privacy-preserving fine-tuning. Intuitively, fine-tuning with our scheme is a special case of training. Here we will focus on the situation where only output layer parameters will be tuned. Recall that each output activation of DP-A is (ϵ, δ)-DP. But in adversary's view, strength of privacy will degrade because images with the same noise distribution may appear in multiple rounds with regarding to one identity. With advanced composition theory in [23] applied directly, we know that output activations in T iterations will at least be $(\epsilon', p_b T \delta + \delta')$-DP where $\delta' \geq 0$,

$$\epsilon' = p_b \epsilon \sqrt{2Tln(1/\delta')} + p_b T \epsilon (e^{\epsilon} - 1). \tag{9}$$

Hence, to ensure $(\epsilon', p_b T \delta + \delta')$-differential privacy, ϵ in our original scheme should be decreased correspondingly. In our case it will be,

$$\epsilon = \epsilon' / (2 p_b \sqrt{2Tln(1/\delta')}). \tag{10}$$

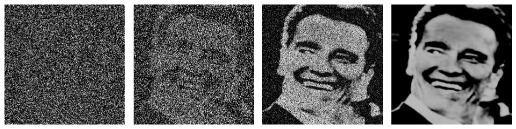

(a) $\epsilon = 1$ (b) $\epsilon = 5$ (c) $\epsilon = 20$ (d) w/o DP-A

Fig. 4. Output activations with DP-A and without DP-A when we fine-tune VGG-Face model on LFW dataset for $T = 10000$, $\delta = \delta' = 1e-5$, $u = 5, \alpha = 1, \beta = 0.5, \gamma = 2$.

We show some results of output activations of the first convolutional layer with different epsilon values in Figure 4. All activation volumes showed in this figure have been multiplied by 255 with negative values removed to make them more visual-friendly. We can tell from this figure when $\epsilon = 1$, utility of images is damaged significantly. When ϵ is around 5, input images can be well preserved. But when ϵ reaches $14 \sim 20$, privacy is almost gone.

VI. Optimal Selection of Partitioning Position

Since we have some helpful observations about privacy property on the first convolutional layer of DNNs, we can now investigate how to achieve the optimal selection of partitioning while taking privacy loss, computing resource and training accuracy into consideration. Assume that we will partition VGG-Face network at the i-th convolutional layer. This means that layers less or equal i will be in client's control, while layers greater than i will be deployed on the server. The client should generate noise which corresponds to function sensitivity of i-th convolutional layer output and then add this noise to the output.

In each convolutional layer, output volume should be compositions of its input. Instead, a max pooling layer and dropout operation will reduce dimensions of its input. Another fact is that the closer a convolutional layer is to final output (A latter convolutional layer will be used to refer to a convolutional layer which is closer to final output, when compared with another layer), the higher feature dimension of its output is. Because output activations of latter convolutional layer will contain compositions of local responses of earlier convolutional layers. This factor also makes output activations of a latter convolutional layer more sensitive than activations of earlier ones. Based on these two facts mentioned above, we can measure the privacy loss of total loss function when the DNN is partitioned at the i-th layer.

Corollary 3. *Weights updating mechanism is $(O(p_b\epsilon_i\sqrt{T})$, $\delta_i)$-DP, where $\epsilon_i = c' + c_i$, if the network is partitioned at the i-th convolutional layer and output activations of the i-th convolutional layer are all (ϵ, δ)-DP.*

Proof. Since the network is partitioned at the i-th convolutional layer, output activations of the i-th convolutional layer are all (ϵ, δ)-DP, which means that $Pr[Sub_j(d) = l_j^{(i)}] = e^{\epsilon} Pr[Sub_j(d') = l_j^{(i)}] + \delta, j \in [1, p \times q]$. Then similar as proof of Corollary.1, we have privacy loss,

$$\frac{Pr[M_p(d) = L]}{Pr[M_p(d') = L]}$$

$$\leq \frac{\prod_{j=1}^{p\times q} Pr[Sub_j(d) = l_j^{(i)}]}{\prod_{j=1}^{p\times q} Pr[Sub_j(d') = l_j^{(i)}] + \sum_{k=1}^{p\times q} \prod_{j=1, j\neq k}^{p\times q}}$$

$$\overline{Pr[Sub_j(d') = l_j^{(i)}] Pr[\hat{l}^{(i+1)} = \phi, \phi \in \Phi'^{(i+1)}|}$$

$$\overline{Sub_k(d') \neq l_k^{(i)}] Pr[Sub_k(d') \neq l_k^{(i)}]}$$

$$= \frac{e^{p\times q\times\epsilon}}{(1-\delta')^{p\times q}(1 + p \times q(1 + c_0'))},$$

where $\delta' > 0, c_0' \in (0, 1)$. There exits real number c_i to satisfy $ln(1 + p \times q(1 + c_0')) + p \times q ln(1 - \delta') \leq c_i$. Thus, loss function is $(p \times q \times \epsilon - c_i)$-DP. Possibility space of $(i+1)$-th convolutional layer $\Phi^{(i+1)}$ is smaller than $\Phi^{(2)}$ if $i > 1$. This leads to $c_0' < c_0$. Hence, $c_i < c$. Noting that weights of layers that are before the i-th convolutional layer should be updated by the client locally, we can revisit privacy property of weights updating mechanism for u-th layer which is after the i-th convolutional layer just like Corollary.2,

$$\frac{Pr[M_u(d) = g_t]}{Pr[M_u(d') = g_t]}$$

$$\leq e^{c'} \frac{(\prod_{j=1}^{p\times q} Pr[Sub_j(d) = l_j^{(i)}]) Pr[M_p(d') = L]}{(\prod_{j=1}^{p\times q} Pr[Sub_j(d') = l_j^{(i)}]) Pr[M_p(d) = L]}$$

$$= \exp(c' + p \times q \times \epsilon - (p \times q \times \epsilon - c_i)),$$

where c' depends on differential between momentums of $\Phi^{(u)}$ and $\Phi'^{(u)}$, which is irrelevant to the value of i. Thus, if we partition the network at the i-th convolutional layer, weights updating mechanism will be $(O(p_b\epsilon_i\sqrt{T}), \delta_0)$-DP, where $\epsilon_i = c' + c_i$. □

As mentioned in the proof, $c_0' < c_0$ because $\Phi^{(i+1)}$ is smaller than $\Phi^{(2)}$ for $i > 1$. For the same reason, we can tell $\Phi^{(i)}$ is smaller than $\Phi^{(j)}$ if $i > j$. Then we can directly infer that $c_i < c_j$, for $i > j$.

Corollary 4. *For any specific DNN, if all output activations of i-th layer are (ϵ, δ)-DP then weights updating mechanism will be ϵ_i-DP for each iteration. And $\epsilon_i < \epsilon_j$, if $i > j$ for any two convolutional layers in the network.*

This corollary directly reveal the correlation between privacy loss and partitioning position, given fixed noise level and unlimited computing resource. Corollary.4 can be seen as a derived result of Corollary.3. Since $c_i < c_j$ when $i > j$, $\epsilon_i = c' + c_i$ should be smaller than $\epsilon_j = c' + c_j$. Simply, Corollary.4 can be derived into a composing version where $(p_b\epsilon_i\sqrt{T})$ is smaller than $(p_b\epsilon_j\sqrt{T})$ for $i > j$.

Training accuracy of the model is difficult to be formalized. Because it involves too many undetermined factors during training. Besides, training accuracy may vary depending on the actual implementation. However, we can still give an empirical formula to predict training accuracy based on experiment results which are partially shown in the Evaluation section. With unlimited computing resource, training accuracy can be modeled as $1 - \frac{\alpha_a}{e^{\epsilon} + \beta_a}$, where α_a and β_a are empirical parameters. Taking our experiments as an example, α_a and β_a are fitted to be 2 and -1.

How much computing resource will be occupied mainly depends on the quantity of parameters. As for VGG-Face, parameter quantity for each layer can be found in Figure 2. Basically, a latter convolutional layer will get more parameters than a previous one. It can be seen as a linearly increasing tendency approximately. The computing resource needed by client side for partitioning at i-th layer can be depicted by the quantity of accumulated parameters, which will be $\sum_{j=1}^{i} o_j Q(j)$, where $Q(j)$ will return a normalized quantity of parameters of the j-th layer, o_j is 1 if the j-th layer is trainable and 0 if not.

Given all these constraints, we can now have our object function to minimize cost for the client as,

$$\begin{aligned} \min_{\epsilon_i} \quad & w_1\epsilon_i + w_2 \sum_{j=1}^{i} o_j Q(j) - w_3(1 - \frac{\alpha_a}{e^{\epsilon_i} + \beta_a}), \\ \text{s. t.} \quad & w_1, w_2, w_3 > 0, i \in [1, 15]. \end{aligned}$$

w_1, w_2, w_3 are weights for different factors. Generally, we can assign them all as 1 since we take these factors equally. The convexity of the cost function can be easily calculated by any optimization tool. It can be proved that the cost function will get its minimum value when $i = 1$ in this case. This means partitioning at the first convolutional layer is the best choice when we take multiple factors into our consideration. Although the optimal partitioning position will vary when specific factors are valued more, it can always be calculated as long as following our cost function.

VII. Evaluation

We have implemented our privacy-preserving DNN for face recognition with TensorFlow framework, based on VGG-Face network, whose architecture is shown in Figure 1. The dataset we use is Labeled Face in the Wild dataset (LFW) [28], which has been regarded as a standard benchmark for unconstrained face recognition. According to [5], training data for published model does not contain LFW dataset. So some subsets of LFW dataset will be used to perform training and fine-tuning. Face images alignment is performed using MTCNN tool provided by [21]. To allow further comparison with other work, we use the same training parameters as described in original VGG-Face training process [5]. All convolutional layers have $stride = 1, pad = 1$. Max pooling size is 2×2. Mini-batch size is 64. Momentum coefficient is 0.9. Learning rate is initialized with 0.01 and exponentially decayed with factor 0.1. Besides, for the local normalization unit that we use in the partitioning layer, $u = 5, \alpha = 1, \beta = 0.5, \gamma = 2$.

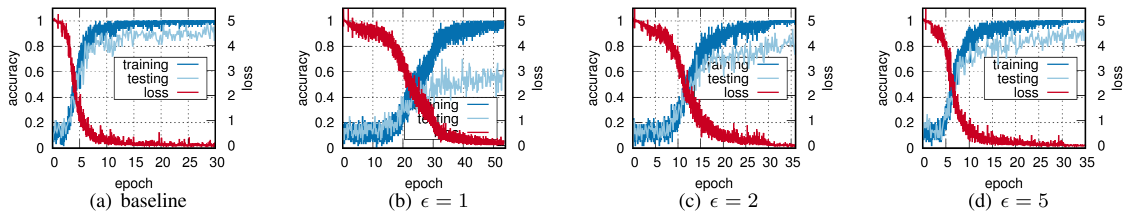
Since LFW dataset is unbalanced in images per person, we use some subsets of LFW dataset to train a new model or to fine-tune on a existing model. In order to take more advantages of public pre-trained models and to save training time, a well trained VGG-Face caffe model provided by [5] is used to initialize weight parameters. This caffe model is converted into a tensorflow model using a popular tool [29]. We first perform experiments with the first convolutional layer as partitioning position. Then we will show how different partitioning positions affect training process exclusively. To verify the feasibility of our privacy-preserving training architecture, we also implement a client demo on Android smartphones. We have measured computation overhead and memory consumption to evaluate our scheme's performance on client side.

A. Training Results

We use Amazon Web Service EC2 P2 instance to perform training experiments. Gaussian noise and local response normalization are added to the output of ReLU in the first convolutional layer. We filter LFW dataset by choosing persons with no less than 10 images. This leads to a subset which contains 158 identities, 4324 labeled face images. We split images of each person in a 9:1 ratio to define training set and testing set. For each iteration, images in training set are randomly sampled to compose a batch. To evaluate training progress, we record output of loss function, training accuracy and testing accuracy. In Figure 5, training results of different epsilon values are shown. Training session with no noise added is recorded as baseline. The smaller the epsilon is, the higher the noise is. It is obvious that small epsilon makes training process more unstable. It will take more epochs to train with higher noise than lower noise to achieve the same training accuracy or the same loss. When $\epsilon = 2 \sim 5$, our scheme can achieve strong privacy and high accuracy. In evaluation, we have verified testing performance in the same client-server model as training, which means testing procedure will use the same level noise as training. Thus, when noise is very high ($\epsilon \leq 1$), classification accuracy will be affected.

B. Fine-tuning Results

To perform fine-tuning on pre-trained VGG-Face model, we use a LFW subset where each person has at least 50 images. We split images of each person in a



(a) baseline (b) $\epsilon = 1$ (c) $\epsilon = 2$ (d) $\epsilon = 5$

Fig. 5. Training results of accuracy and loss with regard to different epsilons.

9:1 ratio to define training set and testing set. Weights in pre-trained VGG-Face model are loaded before training. During training, weights in fully connected layers will be tuned while all the other parameters keeps immovable. Learning rate will be initialized by $1e-3$ and then decreased to $1e-4$. The network is still partitioned at the first convolutional layer. Fine-tuning result in the same setting with no noise added is seen as our baseline of tuning. Results of tuning with different epsilons are shown in Figure 6. High accuracy can be achieved in early learning stage. But adding noises to activations can affect learning speed. Especially when $\epsilon \leq 1$, noise is too large for the network to minimize its loss because trainable parameters are limited in tuning cases. However, when $epsilon = 2 \sim 5$, we can still get high accuracy and strong privacy after slightly more epochs. Specifically, it takes no more than 5 epochs for tuning with $\epsilon = 3$ to achieve similar accuracy as the baseline.

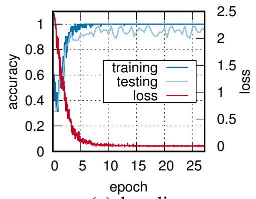
C. Selection of Partitioning

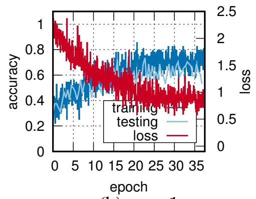
To investigate how partitioning position would affect training, we perform multiple training sessions with different partitioning positions. The same level noises are added to partitioning layers, which is equivalent to $\epsilon = 5$ in the first convolutional layer. Public model parameters are loaded before training to lead to a quicker convergence. Also, we use a LFW subset where each person has at least 20 images. Images of each person are split in a 9:1 ratio to define training set and testing set. All other hyper-parameters are the same as previous training evaluation. We generally choose four positions in the network, their relative distances can be measured in Figure 2. As shown in Figure 7, in the same level noise situation, the latter the partitioning layer is, the harder the training process is. When the network is partitioned at "conv`5_1`" layer, activations are so sensitive to the noise that loss cannot be reduced in early stage. This directly depicts how training accuracy is affected by partitioning position.

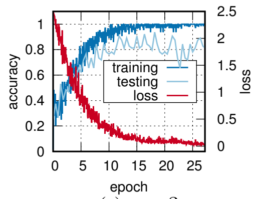
D. Mobile Application Evaluations

To understand the impact of our client-server model in terms of mobile client time cost, memory consumption, energy cost, network transmission cost, we have implemented and evaluated our client application. The biggest challenge of training model on smartphone is the substantial memory consumption, which makes training full model on smartphone unaffordable. To address this challenges, we train only one or two layers at the mobile client with carefully tuned parameters. Our implementation is based on ND4J, a multi-dimensional array based Java library. Our device model is Huawei Nexus 6P (2GHz Qualcomm Snapdragon 810 processor), with a non-removable Li-Po 3450 mAh battery.

As shown in Figure 8, we find that the mobile client is efficient at small batch size and the time cost of backward passing scales against the batch size. The client can also afford to run two layers with no batching within the memory limits. Figure 9 provides the breakdown of the client's memory consumption which consists of the JVM heap and the native allocated memory. The JVM heap size is measured in the Android device monitor, while the native allocated memory is estimated by polling the `/proc/meminfo`. We first make sure that the Android phone is in idle state and record the available memory as a baseline. Then we keep polling the available memory while running our application. We observe that the memory cost scales linearly against the batch size, and the memory cost of addition layer depends on the layer parameters. We coarsely measure the energy cost under different layer configurations and batch sizes through a twenty-minute battery pressure test. During the test, we keep the training running and log the battery consumption every 5 minutes. The results shown in Figure 10 indicate that the energy consumption is acceptable given that the device we are using has a 3450 mAh battery. The network transmission cost, illustrated in Figure 11, is straight-forward, as the client needs to send the forward propagation result after the activation to the remote server and will receive the backward-propagated partial of loss with respect to client's output. The network transmission cost is moderate as the the first two convolution layers have a small depth (both 64). The transmission cost grows linearly against the batch size and it is determined by the parameter of boundary layer.


(a) baseline


(b) $\epsilon = 1$


(c) $\epsilon = 3$

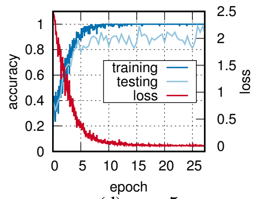
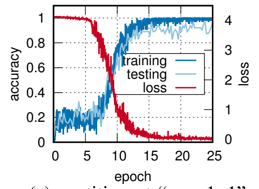
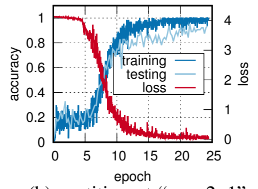
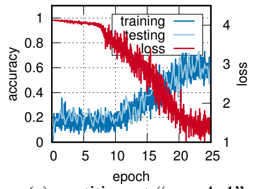

(d) $\epsilon = 5$

Fig. 6. Fine-tuning results of accuracy and loss with regard to different epsilons.


(a) partition at "conv1_1"


(b) partition at "conv2_1"


(c) partition at "conv4_1"

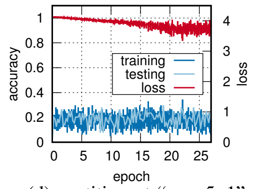

(d) partition at "conv5_1"

Fig. 7. Training results of accuracy with regard to different partitioning selections.

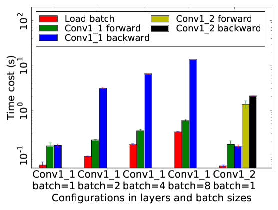


Fig. 8. Time cost on mobile client.

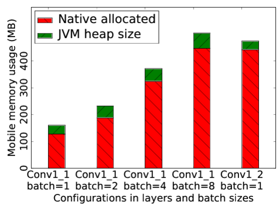


Fig. 9. Memory cost on mobile client.

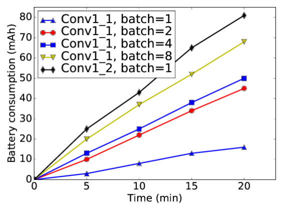


Fig. 10. Energy cost on mobile client.

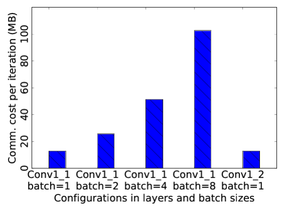


Fig. 11. Network transmission cost on mobile client.

VIII. Conclusion

We have proposed a new edge computing based DNN training architecture with DP mechanism to protect private data. Corollaries and evaluation results obtained in this paper ensure that applying DP mechanism on activations is a feasible solution for outsourcing training tasks of DNNs to untrusted edge servers. Our observation of partitioning the network shows that training accuracy is more sensitive to deeper convolutional layer's noise. Taking client's computing load and accuracy into consideration, we recommend to keep just the first convolutional layer with ReLU and local response normalization applied on the client. Since there are many other DNNs except for VGG networks, verifying similar corollaries and observation in other deep neural networks may be our future work.

Acknowledgement

The authors would like to thank all the reviewers for their helpful comments. This project was supported in part by US National Science Foundation grant CNS-1816399, NSFC-61321491, NSFC-61425024, NSFC-61872176, and NSFC-61872179.

References

[1] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. ACM, 2017, pp. 1175–1191.

[2] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '15. ACM, 2015, pp. 1310–1321.

[3] B. Hitaj, G. Ateniese, and F. Perez-Cruz, "Deep models under the gan: Information leakage from collaborative deep learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. New York, NY, USA: ACM, 2017, pp. 603–618.

[4] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "Deepface: Closing the gap to human-level performance in face verification," in *IEEE CVPR*, 2014, pp. 1701–1708.

[5] O. M. Parkhi, A. Vedaldi, and A. Zisserman, "Deep face recognition," in *Proceedings of the British Machine Vision Conference*, 2015, pp. 41.1–41.12.

[6] Y. Sun, X. Wang, and X. Tang, "Deep learning face representation from predicting 10,000 classes," in *IEEE CVPR*, 2014, pp. 1891–1898.

[7] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *IEEE CVPR*, 2015, pp. 815–823.

[8] Y. Wen, K. Zhang, Z. Li, and Y. Qiao, "A discriminative feature

learning approach for deep face recognition," in *ECCV*, 2016, pp. 499–515.

[9] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *ArXiv e-prints*, 2014.

[10] M. Rhu, N. Gimelshein, J. Clemons, A. Zulfiqar, and S. W. Keckler, "vdnn: Virtualized deep neural networks for scalable, memory-efficient neural network design," in *49th Annual IEEE/ACM International Symposium on Microarchitecture*, 2016, pp. 1–13.

[11] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *2017 IEEE Symposium on Security and Privacy (SP)*, May 2017, pp. 3–18.

[12] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '15. ACM, 2015, pp. 1322–1333.

[13] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. ACM, 2016, pp. 308–318.

[14] S. A. Osia, A. Shahin Shamsabadi, A. Taheri, H. R. Rabiee, N. D. Lane, and H. Haddadi, "A hybrid deep learning architecture for privacy-preserving mobile analytics," *ArXiv e-prints*, 2017.

[15] S. Yi, C. Li, and Q. Li, "A survey of fog computing: Concepts, applications and issues," in *Workshop on Mobile Big Data*, 2015, pp. 37–42.

[16] Z. Tao and Q. Li, "esgd: Communication efficient distributed deep learning on the edge," in *USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18)*. Boston, MA: USENIX Association, 2018. [Online]. Available: https://www.usenix.org/conference/hotedge18/presentation/tao

[17] Y. Mao, S. Yi, Q. Li, J. Feng, F. Xu, and S. Zhong, "A privacy-preserving deep learning approach for face recognition with edge computing," in *USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18)*. Boston, MA: USENIX Association, 2018. [Online]. Available: https://www.usenix.org/conference/hotedge18/presentation/mao

[18] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. E. Lauter, M. Naehrig, and J. Wernsing, "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy," in *ICML*, 2016, pp. 201–210.

[19] J. Wang, J. Zhang, W. Bao, X. Zhu, B. Cao, and P. S. Yu, "Not just privacy: Improving performance of private deep learning in mobile cloud," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery &#38; Data Mining*, ser. KDD '18. ACM, 2018, pp. 2407–2416.

[20] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NIPS*, 2012, pp. 1097–1105.

[21] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao, "Joint face detection and alignment using multitask cascaded convolutional networks," *IEEE Signal Processing Letters*, vol. 23, no. 10, pp. 1499–1503, 2016.

[22] G. Huang, M. Mattar, H. Lee, and E. G. Learned-miller, "Learning to align from scratch," in *NIPS*, 2012, pp. 764–772.

[23] C. Dwork and A. Roth, "The algorithmic foundations of differential privacy," *Found. Trends Theor. Comput. Sci.*, vol. 9, pp. 211–407, 2014.

[24] N. Mohammed, R. Chen, B. C. Fung, and P. S. Yu, "Differentially private data release for data mining," in *ACM KDD*, 2011, pp. 493–501.

[25] C. Dwork, G. N. Rothblum, and S. Vadhan, "Boosting and differential privacy," in *IEEE FOCS*, 2010, pp. 51–60.

[26] B. Amos, B. Ludwiczuk, and M. Satyanarayanan, "Openface: A general-purpose face recognition library with mobile applications," CMU-CS-16-118, CMU School of Computer Science, Tech. Rep., 2016.

[27] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" in *NIPS*, 2014, pp. 3320–3328.

[28] G. B. Huang and E. Learned-Miller, "Labeled faces in the wild: Updates and new reporting procedures," Tech. Rep., 2014.

[29] S. Dasgupta, "Caffe to tensorflow," https://github.com/ethereon/caffe-tensorflow, accessed April 22, 2017.