Theory-Oriented Deep Leakage from Gradients via Linear Equation Solver

Xudong Pan *[1], Mi Zhang[1], Yifan Yan[1], Jiaming Zhu[1], and Min Yang[1]

[1]Department of Computer Science, Fudan University

October 27, 2020

Abstract

In this paper, we take a theory-oriented approach to systematically study the privacy properties of gradients from a broad class of neural networks with rectified linear units (ReLU), probably the most popular activation function used in current deep learning practices. By utilizing some intrinsic properties of neural networks with ReLU, we prove the existence of exclusively activated neurons is critical to the separability of the activation patterns of different samples. Intuitively, an activation pattern is like the fingerprint of the corresponding sample during the training process. With the separated activation patterns, we for the first time show the equivalence of data reconstruction attacks with a sparse linear equation system.

In practice, we propose a novel data reconstruction attack on fully-connected neural networks and extend the attack to more commercial convolutional neural network architectures. Our systematic evaluations cover more than 10 representative neural network architectures (e.g., GoogLeNet, VGGNet and 6 more), on various real-world scenarios related with healthcare, medical imaging, location, face recognition and shopping behaviors. In the majority of test cases, our proposed attack is able to infer ground-truth labels in the training batch with near 100% accuracy, reconstruct the input data to fully-connected neural networks with lower than 10^{-6} MSE error, and provide better reconstruction results on both shallow and deep convolutional neural networks than previous attacks.

1 Introduction

In the past decade, deep learning starts to play a vital role in building modern intelligent systems on a wide range of critical application domains such

*xdpan18@fudan.edu.cn

as finance [HPW16], healthcare [EKN+17], surveillance [SD19], and security (e.g., [YLW+15, SYL+18, LXL+18]), exerting a far-reaching influence on the daily lives of common users. As is widely recognized, among all the enabling technology behind deep learning, *gradient-based optimization* is the one at the heart of the deep learning boom.

From G. Hinton's Turing-award-winning work on *backpropagation* in 1986 [RHW86] to modern optimizers standardized in popular deep learning libraries like Google's Tensorflow [ABC+16] and Facebook's PyTorch [PGM+19], a *gradient* plays a fundamental and ubiquitous role in the learning process of most deep learning models. Take the task of image classification for example. Intuitively, the gradient provides the image classifier a good direction to adapt its parameters in order to narrow the errors (i.e., the *loss function*) between the predictions and the ground-truth class labels. As an analogy of "mountain descent", during the learning process, as the model iteratively updates its parameters along the opposite direction of the gradient on different training samples, the loss function gradually decreases and the prediction of the learning model becomes more accurate.

Despite its fundamental role in deep learning systems, *the gradient is however born with a tell-tale heart.* Mathematically speaking, the gradient is no more magic but the parameter derivative of the loss function, which is explicitly calculated from the given training data and its ground-truth label. Consequently, an attacker may consider to extract sensitive information about the original training data from the captured gradients, as a detour to violate the privacy of the linked data owner. In the past three years, researches have demonstrated that an attacker who captures the gradient of a *single* training sample (for simplicity, we call it the *single-sample gradient*) can successfully infer its property [MSC+19], its label [ZMB20], its class representatives [HAPC17, WSZ+19] or the data input itself [ZLH19, ZMB20, GBD+20], with rather high accuracy.

Yet, instead of single-sample gradient, using *multi-sample gradient* for training is a more commonly adopted technical choice in practical deep learning systems considering the efficiency and the performance. As Fig. 1 shows, in a typical face recognition system, a mini-batch of training images are first input to the neural network classifier. The classifier then predicts the label, computes the average loss function, and uses back-propagation to derive the multi-sample gradient as the parameter derivative of the average loss. Equivalently, the above multi-sample gradient can also be viewed as the per-coordinate average of the single-sample gradients. Despite the wide usage of multi-sample gradient, existing researches on its privacy property is rarer. It is natural to ask, *whether multi-sample gradient is safer than single-sample gradient in terms of training data privacy?*

Our Work. In this paper, we investigate the above question for *data reconstruction attack* [ZLH19, ZMB20, GBD+20], a new and threatening privacy attack on learning systems where the model is accessed as a white-box (e.g.,

federated learning). As shown in the right part of Fig. 1, data reconstruction attack targets at reconstructing the training sample(s) from the corresponding gradient, which could pose huge threats to the confidentiality of private training data. At first glance, we may notice, to separate out the gradient of each single sample from their average is almost as ill-posed as recovering each summand from their summation [Wer87]. Hence, is it neither possible for an attacker to reconstruct multiple samples from the average gradient? *Surprisingly, such an attack is possible.* As accounted in [ZLH19], one of the earliest works "naively apply" the attack method they devised for reconstructing a single sample to the multi-sample case and they surprisingly observed the reconstruction is possible when the batch size is smaller than 8 on CIFAR-10. This preliminary work is further followed up by a few number of unpublished works [ZMB20, GBD+20] which made slight technical adjustments on the original attack method in [ZLH19] and report similar phenomenons.

Although previous works demonstrated the privacy risks of gradient even in the multi-sample case, most existing works evaluate their attacks only in a small number of scenarios, which could be insufficient for testing whether this privacy threat ubiquitously exists. Meanwhile, their attacks commonly rely on a kind of brute-force attack method, which mainly uses optimization techniques to search for a batch of samples that produces an average gradient as close as possible to the ground-truth gradient (i.e., *gradient matching*). On the one hand, such an approach rarely exploits intrinsic properties of neural networks, which we notice may leave much room for further improving the effectiveness and stability of existing attack. On the other hand, the optimization-based attack is unclear in the reason why it works, which hence provides our community with very limited insights on some fundamental yet unknown aspects of data reconstruction attack in the multi-sample case, including its underlying mechanism and other key factors which influences the reconstruction quality.

To bridge the gaps, we start with a thorough analytic study on the broad family of fully-connected neural networks equipped with rectified linear units (ReLU [GBC16]) to find out the enablers behind the curtain of data reconstruction attacks. ReLU, as probably the most popular activation function in deep learning practices [GBC16], behaves by letting nonnegative inputs pass through without modifications and blocking the negative inputs. As shown in Fig. 2, this special gate-like behavior of ReLU allows each input sample to hold its own set of activation paths, which form its *activation pattern* [MP+14,LvB18]. If we make an analogy between the activation pattern with a "thread" which fingerprints the training sample, when the gradients of single samples are averaged to produce the multi-sample gradient, the corresponding activation patterns are "knotted" together. As a key insight in our analysis, we find and prove the existence of at least one *exclusively activated* neuron at each layer (and, to be exact, at least two for the last-

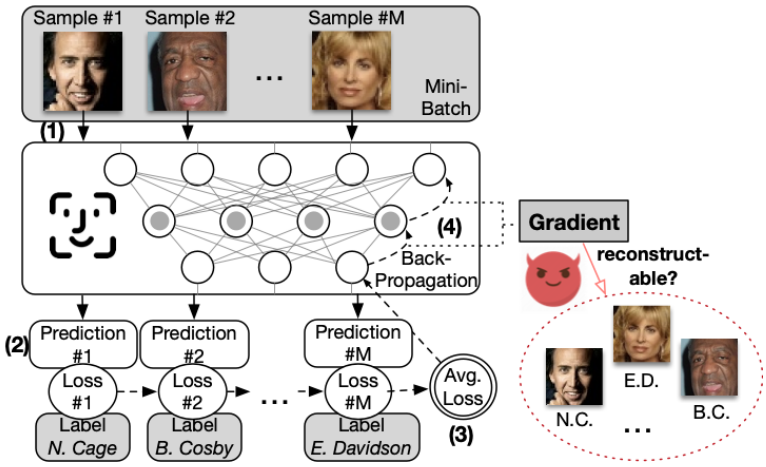

Figure 1: The information flow to produce multi-sample gradient in a face recognition model and data reconstruction attacks.

but-one layer) for every sample is sufficient and necessary for the attacker to "untie the knot", i.e., to separate out uniquely the respective activation pattern for every training sample in a batch (Proposition 2).

The separated activation patterns are critical to a successful data reconstruction attack. Based on the determined activation pattern of each sample, we further show the original gradient matching problem considered in previous attacks is naturally simplified to a sparse linear equation system called the *gradient equation*. At the practical level, we devise a novel data reconstruction attack which explicitly solves the linear gradient equation when the victim model is a fully-connected neural network. Furthermore, we extend this theory-oriented attack to cover more popular convolutional neural network architectures including GoogLeNet [SLJ+15] and VGGNet [SZ15]. As these architectures usually implement its last several layers as a fully-connected neural network, we hence propose a two-staged attack pipeline which first reconstructs almost every single feature map input to the fully-connected module and then reconstructs the raw data input from its corresponding feature map, by a deconvolution-based or a novel hybrid approach.

For experiments, we evaluate our newly designed attack algorithm on more than 10 representative neural network architectures including fully-connected neural networks, shallow (e.g., LeNet [LBB+98]) and deep convolutional neural networks widely used in commercial systems (including GoogLeNet, VGGNet and 6 more). Besides the standard image benchmarks which existing works mainly experiment with, we also design 5 more data-sensitive benchmarks on real-world datasets covering medical, e-commerce, face recognition and location-related scenarios. In the majority of test cases,

experimental results show our proposed data reconstruction attack is able to infer the ground-truth labels in the training batch with near 100% accuracy, reconstruct the input data to fully-connected neural networks with lower than 10^{-6} mean square error on average, and provide better reconstruction results on both shallow and deep convolutional networks than previous attacks. Besides, we also provide rich ablation studies in different configurations to demonstrate how the key factors of the deep learning system inherently influence the system's vulnerability against data reconstruction attacks.

In summary, we mainly make the following contributions.

- We provide the first comprehensive theory-oriented privacy analysis of the broad family of neural networks with ReLU activations against data reconstruction attacks, providing systematic evaluations of our proposed attack and previous attacks on over 10 representative popular neural network architectures in various data-sensitive application scenarios.
- We observe the separability of activation patterns as the key enabler of data reconstruction attack on multi-sample gradient. Moreover, we prove the existence of exclusively activated neurons as a necessary and sufficient condition to determine the activation patterns. With the determined activation patterns, we show the equivalence between the gradient matching problem and a sparse linear equation systems.
- We propose a novel data reconstruction attack on fully-connected neural networks which is based on explicitly solving the derived linear equation system from our analysis. We further extend the attack algorithm to convolution neural networks, which noticeably outperforms previous attacks in its effectiveness and its stability in the majority of test cases.

2 Related Work

Data Reconstruction Attacks. Different from previous attacks which mainly aim at reconstructing class representatives [FJR15, HAPC17], the primary goal of data reconstruction attack is to recover each single training sample behind the intermediate computation results accessed by the attacker. To the best of our knowledge, although it is perhaps Salem et al. first referred to such an attack class as data reconstruction attack [SBB+19], their work mainly studied reconstructing a batch of training samples from the changes of their outputs from an updated neural network, which however is not a common threat model in most distributed learning paradigms and thus is irrelevant to our work. Parallel to this work, Wang et al. improved [HAPC17] with a multi-task GAN to generate single samples by refining the recovered class representatives [WSZ+19], which however requires strong inner-similarity of samples in the same category. This limitation makes this attack not directly applicable to most of the scenarios and

datasets we consider.

Recently, starting from the preliminary study by Zhu et al. [ZLH19], a branch of researches [ZLH19, ZMB20, GBD+20] begin to explore a brute-force yet general approach towards successful data reconstruction attacks and presented meaningful empirical results on CIFAR-10 and ImageNet. These works are all based on the same learning-based framework: they initially set the batch of unknown training samples as variables, and search for the optimal training samples by minimizing the distance between the ground-truth gradient and the gradient computed from the variables. They mainly differ in the choice of the distance function to minimize (L2 distance in [ZLH19, ZMB20] and cosine distance in [GBD+20]). Although [ZMB20] uses the property of neural networks to recover the label of a single sample in prior before the learning-based attack, the trick only works for the single-sample gradient, which makes their method identical to [ZLH19] against the multi-sample case. However, all these attacks mainly stay at the empirical level and aim at showing the possibility of a successful data reconstruction attack from both single and multi-sample gradient, without demonstrating and understanding the underlying mechanisms of the data reconstruction attack. Meanwhile, their attacks are only conducted on a limited number of neural network architectures and standard benchmark datasets from the machine learning community, which is insufficient for evaluating whether the threats from data reconstruction attack do ubiquitously exist.

Privacy Attacks on Training Data and Beyond. As gradients can be more easily accessed in open-network distributed learning systems, a number of recent works begin to study various types of information leakage from gradients [MSC+19, NSH19, HAPC17]. For example, Melis et al. demonstrate the possibility of inferring from the gradient whether the training samples share certain properties (e.g., whether the faces all wear eyeglasses) [MSC+19], Hitaj et al. leverage a generative adversarial learning paradigm to infer the class representatives [HAPC17], while Nasr et al. exploit the gradient for membership inference [NSH19]. Different from these existing studies, in this paper, we are more curious about the limit of data reconstruction attacks, which can be usually considered as the worst-case privacy disclosure of training data from the gradient since, with each single sample recovered, the attacker can almost know everything about the training data, including the target property he/she is interested in.

Besides the gradient as the information source for attacks on training data privacy, researchers also explored, e.g., using the model parameters to infer the properties of training data [GWY+18, CLE+19], using the intermediate data representations to infer the sensitive attribute values of data samples [FLJ+14, FJR15, PZJY20], or using model explanations to reconstruct significant parts of the training set [SSZ19]. Aside from training data privacy, previous studies also cover many other aspects of machine learning privacy, such as the privacy risks of the data membership [SSS+17, SZH+19,

LF20], the parameters [TZJ+16], the hyper-parameters [WG18], the model architecture itself [DSR+18] or its functionality [OSF19, JCB+20].

3 Preliminary

3.1 Gradient in Deep Learning Systems.

Gradient plays an indispensable and ubiquitous role in modern deep learning systems, especially during the model training phase. In the following, we take the K-class classification task as an example, which covers many real-world use cases of deep learning. We denote a learning model as $f(\cdot; W)$, where W denotes its learnable parameters, and a training sample (X, Y), where X is called the data input and Y is the ground-truth label, ranging in $\{1, ..., K\}$. By convention, the learning model takes in the data input X and outputs a vector $f(X; W) \in \mathbb{R}^K$ (abbrev. f), where the c-th element of this vector after a softmax operation predicts the probability of X in class c, i.e., $p_c := [\text{softmax}(f(X; W))]_c = \frac{\exp f_c}{\sum_{c=1}^{K} \exp f_c}$.

With this prediction, the loss function $\ell(f(X; W), Y)$ (abbrev. ℓ) is usually calculated as the cross-entropy loss between the predicted probabilities and the ground-truth label, i.e., $\ell(f(X; W), Y) := -\log p_Y = -f_Y + \log \sum_{c=1}^{K} \exp f_c$.

To adapt the parameters to the given training sample, the gradient comes onto the stage, which calculates as $\overline{G}(X, Y; W) := \frac{\partial \ell(f(X;W),Y)}{\partial W}$. In other words, the gradient on the training sample is exactly the partial derivative of the loss function w.r.t. the model parameters. In many modern optimization algorithms (e.g., SGD [Rob07], Adam [KB15] and etc.), by updating the model parameter along or close to the opposite direction of the gradient with a prescribed step size, the loss function is guaranteed to decrease by iteration, which in turn means the learning model makes more accurate predictions. In terms of privacy, as the above definition of gradient shows, the gradient on a single-sample is an explicit function of the training sample and its label, which therefore makes it not hard to understand the success of existing privacy attacks on single-sample gradients [NSH19, MSC+19, ZMB20].

In practice, instead of the gradient calculated on a single sample, deep learning systems in practice mainly use the average gradient which is calculated on multiple training samples. Such a practice is usually more suitable for modern parallel computation devices, reduces the variance and bias in training data, and results in much faster convergence rate [Bub15]. Formally, given a mini-batch of M training samples $\{(X_m, Y_m)\}_{m=1}^{M}$, the multi-sample gradient is calculated as the coordinate-wise arithmetic average of the gradients for each single sample, which formally writes $\overline{G}(\{(X_m, Y_m)\}_{m=1}^{M}; W) := \frac{1}{M} \sum_{m=1}^{M} \overline{G}(X_i, Y_i; W)$.

Different from the case of single-sample gradients, each single gradient

which constitutes the average gradient seem hard to be separated out from the average gradient any longer. However, some previous works still find empirically that they could reconstruct almost each single training sample to a recognizable level [ZLH19,ZMB20,GBD$^+$20], although the effectiveness of these attacks rapidly deteriorates when the batch size M increases and the size of the learning model decreases. However, almost no previous works have analyzed why such an attack is possible. To bridge this gap, our work aims at unraveling these phenomenons by diving into the underlying mechanism of data reconstruction attacks.

3.2 Data Reconstruction Attacks and Gradient Equation

Existing data reconstruction attacks suppose the attacker has a white-box knowledge about the victim's learning model (i.e., the parameters and the architecture). In this setting, given the captured ground-truth gradient $\overline{G}$, previous attacks commonly adopt optimization-based techniques to solve the following *gradient matching* problem,

$$\min_{X_i,Y_i} D(\frac{1}{M}\sum_{i=1}^{M}\frac{\partial \ell(f(X_i;W),Y_i)}{\partial W}, \overline{G}) \tag{1}$$

where D measures the distance between the gradient produced by the variables under optimization and the ground-truth multi-sample gradient. For example, [ZLH19, ZMB20] implements D as L2 distance, while [GBD$^+$20] proposes to use cosine distance.

Equivalently, minimizing the above gradient-matching problem to reconstruct the training batch is only a practical implement of solving the *gradient equation*, which writes

$$\sum_{i=1}^{M}\frac{\partial \ell(f(X_i;W),Y_i)}{\partial W} = M\overline{G} \tag{2}$$

where $\{(X_i,Y_i)\}_{i=1}^{M}$ are the variables of the equation. With this equivalence, we may safely state that the plausibility of data reconstruction attacks is highly related with the solvability of the above gradient equation.

3.3 Neural Networks with ReLUs

Rectified Linear Units (ReLU). From fully-connected neural networks (FCN) and shallow convolutional neural networks (CNN) to deep CNNs like GoogLeNet and ResNet, a very broad class of popular neural network architectures is now implemented with the ReLU activation function (simply, ReLU). Concretely, a ReLU σ can viewed as a gate structure which allows non-negative values to pass through without any change and meanwhile

blocks negative values by outputting 0 instead. Such a behavior is formally written as $\sigma(x) = x$ if $x \geq 0$; $\sigma(x) = 0$ if $x < 0$.

In the theory part of our work, we mainly analyze the data reconstruction attacks against an $(H+1)$-layer FCN with ReLU activation. It is a common choice for a wide range of K-class classification tasks with flat feature vectors in e.g., $\mathbb{R}^{d_0}$. Such flat feature vectors may come from the dataset itself (e.g., a shopping record can be represented as a binary feature vector where each element denotes whether the corresponding item has been bought or not) or from the feature extraction part of most deep convolutional networks as in transfer learning [JZJ+18]. Formally, a $(H+1)$-layer ReLU FCN is represented as $f(X; W_0, W_1, ..., W_H) = W_H\sigma(W_{H-1}...W_1\sigma(W_0X)...)$, where $W_i \in \mathbb{R}^{d_{i+1}\times d_i}$ is the weight matrix at the i-th layer, the data input $X \in \mathbb{R}^{d_0}$, $d_{H+1} = K$ and σ is the ReLU activation function. In our analysis, we omit the bias terms in our analysis mainly for the simplification of notations. Without loss of generality, analytic results and attack algorithms in the remainder of this paper can be directly extended to ReLU networks with bias following the common practices in deep learning theory [GBC16], i.e., by adding a constant neuron to each layer.

Activation Pattern in a ReLU FCN. Intuitively, considering the gate-like behavior of ReLU, when a data sample is input to a neural network with ReLU, each coordinate of the data input selectively passes through a part of neurons at the current layer and meanwhile is blocked by the rest neurons due to the negativity or a vanishing weight of the neural connection. As Fig. 2, via layers of forwarding through the whole neural network, each sample presents a set of computation paths in the neural network, which forms its *activation pattern*. Below, we develop the idea of activation pattern in a more formal way.

When ReLU is applied to a vector as in the original definition of FCN, it is applied in a coordinate-wise way. For example, if we consider carefully the output of the first layer, that is $\sigma(W_0X)$, we can reformulate it as $\sigma(W_0X) = (\sigma(\sum_{j=1}^{d_0} W_0^{1j}X_j), ..., \sigma(\sum_{j=1}^{d_0} W_0^{d_1 j}X_j)) = \text{diag}(\mathbf{1}\{\sum_{j=1}^{d_0} W_0^{1j}X_j > 0\}, \ldots, \mathbf{1}\{\sum_{j=1}^{d_0} W_0^{d_1 j}X_j > 0\})W_0X := D_1(W_0, X)W_0X$ [LvB18]. For simplicity, we denote the last term as D_1W_0X.

As we can see, D_1 is a diagonal matrix of size $d_1 \times d_1$, where each entry D_1^{ii} exactly implies whether the output of the i-th neuron at the first layer can be passed on to the next layer as an input. We call such a matrix D_1 the *activation matrix* of the first layer. Similarly, we can reformulate the whole $(H+1)$-layer ReLU network as $f(X) = W_HD_HW_{H-1}...W_1D_1W_0X$ where the sequence of activation matrices $(D_1, \ldots, D_H)$ describes the *activation pattern* for the data input X.

Finally, we would like to mention a useful property of the activation pattern during the gradient back-propagation, that is, the activation matrix commutes with the derivative operation, i.e., $\nabla_{W_0} D_i(X)W_{i-1} \ldots W_0X =$

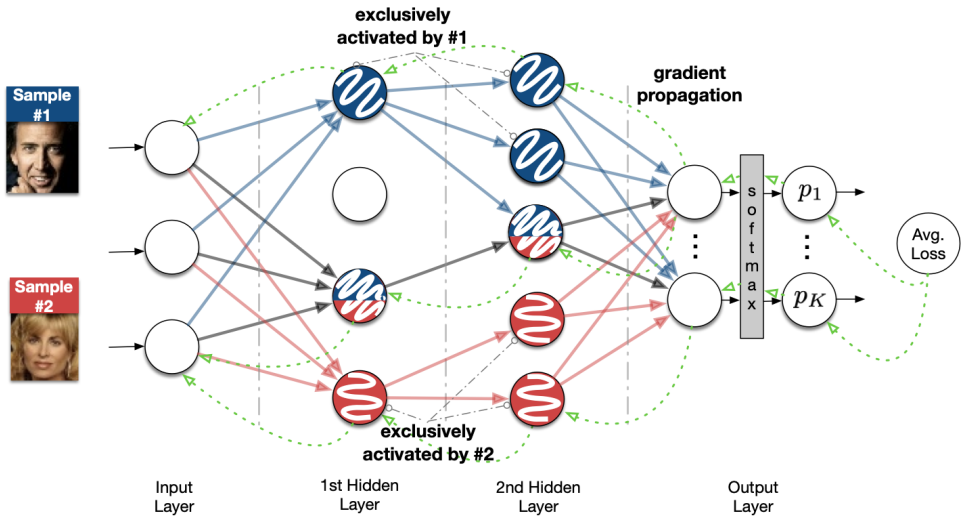

Figure 2: Illustration of two data samples as a batch input to a 4-layer FCN with ReLUs, where each data sample is forwarded through a set of computation paths which composes its *activation pattern* (D_1, D_2). For example, as the blue directed lines show, Sample #1 passes through the 1st and the 3rd neuron at the first hidden layer, which means its activation matrix at the first layer D_1 is $\text{diag}(1, 0, 1, 0)$. Similarly, at the second layer, its activation matrix D_2 is $\text{diag}(1, 1, 1, 0, 0)$. Moreover, we call a neuron which is only activated by one sample in an input batch as the *exclusively activated* neuron of the corresponding sample. For simplicity, the green dashed lines plot parts of the back-propagation paths. A key property is, the gradient is only produced along the same activation pattern in the forwarding phase (better viewed in color).

$D_i \nabla_{W_0} W_{i-1} \dots W_0 X$. In other words, *the gradient backpropagates along the same activated path as in the forwarding phase.* Fig. 2 illustrates the role of the activation pattern in the data forwarding and gradient backpropagation phase of an FCN with ReLUs, which plays a key position in our analysis.

Gradient Equation of an FCN with ReLUs. As we have mentioned in the first part of this section, the loss function $\ell(f(X), Y)$ is usually implemented as the cross-entropy between the ground-truth label Y and the "softmax-ed" $f(X)$. With simple calculations, the gradient of the entropy loss on the i-th output of f(X) is in the following closed form

$$\frac{\partial \ell}{\partial f_c} = \overline{g}_c = -1 + p_c \text{ if } c = Y \text{ else } p_c \tag{3}$$

where p_c is the predicted probability for the sample X in class c. For convenience, we denote the *loss vector* $\overline{g} = (\overline{g}_1, ..., \overline{g}_K)$.

Therefore, for the parameter W_i at the i-th layer, the gradient is $\frac{\partial \ell}{\partial W_i} = \sum_{c=1}^{K} \overline{g}_c \frac{\partial f_c}{\partial W_i}$.

By replacing the left side as the captured gradient at the i-th layer, i.e., $\overline{G}_i$, we have the following gradient equation for W_i (more exactly, a matrix equation),

$$\overline{G}_i = \sum_{c=1}^{K} \overline{g}_c \frac{\partial f_c}{\partial W_i} \tag{4}$$

which in its current form provides a non-linear equation system for the attacker to solve in order to determine the value of X.

4 Behind the Curtain of Data Reconstruction Attacks

4.1 Overview

In Section 3.2, we have introduced that to conduct the data reconstruction attacks is in essence equivalent to find a solution to the gradient equation in Eq. 2. In this part, we first dig into the mechanism of data reconstruction attacks by using deep learning theory to analyze the solvability of the gradient equation of FCN. Then we propose our theory-oriented data reconstruction attack on FCN and its extension to convolutional neural networks. Below, we first formally define the threat model and the attack objective.

• **Threat Model.** We mainly follow the same threat model as in existing data reconstruction attacks [ZLH19, ZMB20, GBD+20], which requires the attacker to know

1. The ground-truth average gradient $\overline{G}$ produced by a mini-batch of M training samples.

2. The parameters W of the neural network w.r.t. which the gradient is calculated and the architecture of the neural network.

It is worth to notice, unlike previous attacks, we do not require the knowledge of the batch size M. In fact, our analysis below shows in most cases the attacker can even determine the batch size from the gradient only. Moreover, we require almost no prior knowledge about the victim's private dataset except the shape of the data input, which can usually be derived from the architecture information as well. Furthermore, unlike model inversion attacks [FJR15], our threat model does not require the parameters of the neural network to have any knowledge about the training set. In most of our experiments except the ablation study in Section 6.2, the victim neural network is untrained and has randomly-initialized weights.

• **Attack Objective.** In data reconstruction attacks, the attacker wants to reconstruct every single training sample, i.e., $\{(X_m, Y_m)\}_{m=1}^{M}$, including both the data input and the label. To measure the effectiveness of a data reconstruction attack, we in general measure the accuracy of the recovered labels and also the reconstruction error between each reconstructed input and its best-matching ground-truth data input. For more details on the concrete evaluation metrics commonly used for data reconstruction attacks, please see Section 5.3.

4.2 Attacker always prevails on single-sample gradients

As an appetizer, our analyses start from one of the most simplified cases where the gradient is only calculated on one training sample (X, Y) and the neural network f is an FCN with ReLU activations. In this configuration, we show analytically and algorithmically *an attacker can always exactly (within a tolerable numeric error) recover the label and the input of the single sample.*

First, we provide below the explicit form of the gradient equation in this case (cf. Eq. 4). For each layer index $i \in \{0, ..., H-1\}$, we have [Kaw16, Lemma 4.1]

$$\overline{G}_i = \sum_c \overline{g}_c (D_i W_{i-1} ... W_0 X)([W_H]_c^T D_H ... W_{i+1} D_{i+1}) \tag{5}$$

As we can see, the above equation is a little bit complicated due to the fact that both the loss vector $\overline{g}$ and the activation pattern $(D_1, \ldots, D_H)$ depend on the data input X in a non-linear manner. To some degree, this complexity probably makes the existing learning-based approach towards data reconstruction attacks become the most straightforward and currently, the seemingly only option for attacks. Below, we show the complexity can be eliminated by recovering the $\overline{g}$ and the activation pattern $(D_1, \ldots, D_H)$ in prior with some tricky properties of ReLU, which will finally reduce the original complicated gradient equation to a simplified linear form.

• **How to solve $\overline{g}_c$?** First, we prove the $\overline{g}_c$ can be determined from the equations w.r.t. the parameter W_H, which eases the downstream analysis. We directly calculate the derivative of the loss on each row of W_H (i.e., $[W_H]_i$ the i-th row) as $[\overline{G}_H]_i := \frac{\partial \ell}{\partial [W_H]_i} = \overline{g}_i f_{H-1}$, where $f_{H-1} \in \mathbb{R}^{d_H}$ is the output of the $(H-1)$-th layer. At the LHS of the above equation is the ground-truth gradient known to the attacker. By enumerating the index i in the above equation and forming the ratios, the attacker can get $K-1$ equations on the ratios $\overline{g}_K/\overline{g}_1, \overline{g}_{K-1}/\overline{g}_1$, ..., $\overline{g}_2/\overline{g}_1$. In other words, the variables $\{\overline{g}_c\}_{c=1}^K$ all depend on one free variable $\overline{g}_1$ by $\overline{g}_i = \frac{[\overline{G}_H]_i}{[\overline{G}_H]_1}\overline{g}_1$. We make two noteworthy remarks on the above observation.

Remark 1 (Exact Label Reconstruction)**.** *As we can see from Eq. 3, only if c hits the ground-truth label Y, then g_c is negative while the others are positive. As a result, by checking the sign of the recovered ratios, the attacker can easily determine the ground-truth label Y of the data X. For details, please see Algorithm 2.*

Remark 2 (Feasible Range of $\overline{g}_1$)**.** *Moreover, with the constraint that $\sum_{c\neq Y} \overline{g}_c = \sum_{c\neq Y} p_c \leq 1$, the attacker can determine the feasible range $[0, \delta]$ of $\overline{g}_1$, where δ is a rather small constant in practice. As a result, the attacker can choose a random value in the range or run repetitive attacks to get satisfying results. In the following, it is reasonable to assume $\overline{g_1}$ is known.*

• **How to solve D_i?** By checking the non-negative entries in the ground-truth gradient, the attacker can fully recover the activation pattern D_i by the following criterion.

Criterion 1. *If $[G_i]_{jk} \neq 0$, then $[D_{i+1}]_{kk} = 1$.*

To explain, this criterion utilizes the property of ReLU during the back-propagation phase we introduce in Section 3.3: *the gradient does not vanish only when it backpropagates through the ReLU gates that are activated during the forwarding phase.* In other words, for a parameter (i.e., a directed solid link between neurons in Fig. 2) which has a zero gradient, then the neuron at the end is not activated, which in turn means the corresponding entry in the activation matrix is zero. Intuitively, the criterion above can be readily checked with Fig. 2. The attacker can always check the position of the non-zero entries in $\overline{G}_i$, it is equivalent to say he/she has the knowledge of the green lines. Hence, the attacker can assert the ending nodes of each green line should be activated.

• **Reduction to a Linear Equation System.** With the $\overline{g}$ and the activation pattern determined, the gradient equation in Eq. 5 immediately degenerates to a linear equation system w.r.t. X, which can be solved by off-the-shelf numeric algorithms, e.g., LR decomposition.

Intuitively, the above result states, in the single-sample case of FCN, the gradient equation always has a unique solution, which is exactly the

ground-truth data input itself. As a result, by solving the linear equation system, the attacker is guaranteed to reconstruct the victim's private data exactly and hence prevails in the privacy game with a huge advantage.

• **Implement the Single-Sample Reconstruction Attack.** In summary, our theory-oriented data reconstruction algorithm against a single-sample gradient of an FCN follows the pipeline below: (1) determine $\overline{g}$ from the gradient equation of W_H; (2) determine the activation pattern with Criterion 1; (3) replace the $\overline{g}$ and $(D_1, \ldots, D_H)$ in Eq. 5 with the determined values and solve the resulting linear equation system with standard solvers like LU decomposition. For demonstration, Fig. 9 in Appendix 8.1 shows a visualization result of a celebrity face from the corresponding gradient. Compared with the ground-truth input, the reconstruction is exact within a tolerable numeric error.

4.3 The Mechanism of Data Reconstruction Attacks on Multi-Sample Gradient

Now, let us extend the single-sample case to the multi-sample case. By supposing the ground-truth gradient is calculated from an unknown mini-batch $\{(X_1, Y_1), \ldots, (X_M, Y_M)\}$, the gradient equation w.r.t. the parameter W_i can be written as $\overline{G}_i = \frac{1}{M} \sum_{m=1}^{M} \sum_{c=1}^{K} \overline{g}_c(X_m, Y_m) \frac{\partial f_c(X_m)}{\partial W_i}$.

As an analogy to the single-sample case, to determine the $\{\overline{g}_c(X_m, Y_m)\}_{m=1,c=1}^{M,K}$ and $\{D_i(X_m)\}_{m=1,i=1}^{M,H}$ in prior will reduce the gradient equation above to a system of linear scalar equations, which enables a successful data reconstruction attack by equation solving. Therefore, to understand the privacy risk of multi-sample gradient, we need to find under what conditions $\{\overline{g}_c(X_m, Y_m)\}_{m=1,c=1}^{M,K}$ and $\{D_i(X_m)\}_{m=1,i=1}^{M,H}$ can be uniquely determined.

Below, we first introduce the notion called the *exclusive activation* of a ReLU, which plays a key role in developing the following analysis.

Definition 1 (Exclusive Activation)**.** *Given a mini-batch of samples $\{(X_m, Y_m)\}_{m=1}^{M}$, we call the j-th neuron at the i-th layer is exclusively activated by one sample if $\sum_{m=1}^{M} [D_i(X_m)]_{jj} = 1$.*

Literally, we call a neuron is exclusively activated if it is only activated for only one sample in the mini-batch. For intuition, Fig. 2 illustrates two data samples and their corresponding exclusively activated neurons during their computation in a four-layer FCN.

• **When can we solve $\overline{g}_c^m$?** Inspired from the single-sample case, we again consider the gradient equation for W_H, that is, $[\overline{G}_H]_c = \frac{1}{M} \sum_{m=1}^{M} \overline{g}_c^m(X_m, Y_m) f_{H-1}^m(X_m)$.

For simplicity, we abbreviate $\overline{g}_c(X_m, Y_m), f_{H-1}(X_m)$ respectively as $\overline{g}_c^m$ and f^m. To recap, $f^m = D_H(X_m)W_{H-1}...D_1(X_m)W_0X_m \in \mathbb{R}^{d_{H-1}}$. Therefore, in general, if we form the ratio equations as in the single-sample case, the terms f^m could not canceled out, which makes it seemingly impossible to determine the ratios $\overline{g}_c^m$.

However, with the notion of exclusive activation, we observe the following sufficient condition for uniquely determining $\overline{g}_c^m$.

Proposition 1. *A sufficient requirement for determining the ratio of $\overline{g}_c^m$: each data sample has at least two exclusively activated neurons at the last but one layer.*

As a proof, we describe below the algorithm to determine the ratios $\{\overline{g}_c^m/\overline{g}_1^m\}_{m=1,c=2}^{M,K}$. For better intuition, we consider again the case in Fig. 2 where each sample X_m in a batch of size 2 has two exclusively activated neurons at the last layer and one commonly activated neuron (i.e., X_1 takes up the 1st and the 2nd neurons, and X_2 the 4th and 5th). In other words, both samples activates two different neurons at the last-but-one layer of the neural network. According to the gradient equation above, by forming the ratio vector $[\overline{G}_H]_4/[\overline{G}_H]_3$, we notice that for each exclusively activated neuron of the 1st sample (i.e., the 1st, 2nd neuron), the element $[[\overline{G}_H]_4/[\overline{G}_H]_3]_1 = [[\overline{G}_H]_4/[\overline{G}_H]_3]_2 = \overline{g}_4^1/\overline{g}_3^1$. We also provide a schematic proof of this property in Fig. 3. Based on this property, we can practically

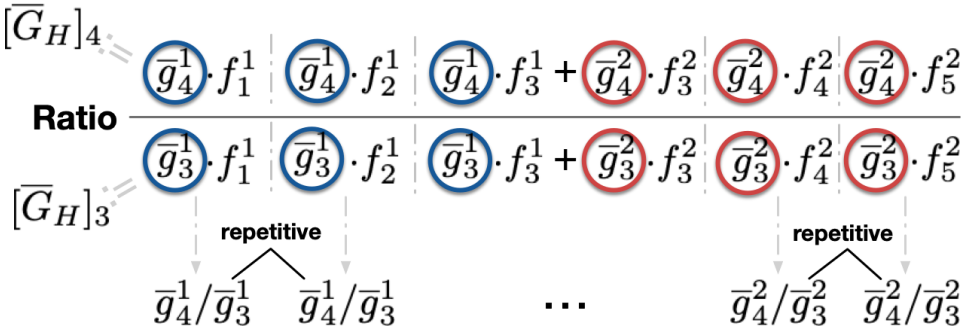

Figure 3: A schematic proof on the observation that exclusively activated neurons at the last-but-one layer helps solve the ratios among $\{g_c^m\}_{c=1}^K$ for each m.

detect the repetitive values in $[\overline{G}_H]_c/[\overline{G}_H]_1$ to determine the exclusively activated neurons for the m-th sample and then collect the value at the corresponding index of the ratio vector $[\overline{G}_H]_c/[\overline{G}_H]_1$ as the corresponding ratio $\overline{g}_c^m/\overline{g}_1^m$. Similarly, by enumerating the class index c, we can again reduce the $M \times K$ variables in $\{\overline{g}_c^m\}_{m=1,c=1}^{M,K}$ to K variables. For more details on the implementation, please refer to Algorithm 1 in Appendix 8.5.

After $\overline{g}_c^m$ is determined, similar to what we have mentioned in Remark 1, the attacker can further determine the labels of the samples in the mini-batch by checking the sign of the recovered $\{\overline{g}_c(X_m, Y_m)\}_{m=1,c=1}^{M,K}$. In fact, as a corollary from Proposition 1, according to the pigeonhole theorem, we

have the following theoretical bottleneck on the maximal number of reconstructable labels via our attack.

Corollary 1. *Our data reconstruction attacks on an FCN with ReLU can recover for sure the labels of at most $d_H/2$ samples from the averaged gradient.*

Although we still have not figured out whether this theoretical bottleneck on our algorithm is also applicable to any gradient-equation-based data reconstruction attacks, we do observe empirically in most cases our novel attack is the strongest in label reconstruction among existing attacks. Moreover, the proved bottleneck of our attack is already large. For example, for a typical FCN model on MNIST classification in Tensorflow's official tutorial, d_H is 128, which means our attack in the worst case can reconstruct the labels of 64 samples for sure from their average gradient, while, as far as we know, existing attacks only has theoretical guarantees on reconstructing the label of one single sample for sure [ZMB20].

• **When can we solve D_i^m?** Based on the knowledge of the exclusively activated neurons at the last-but-one layer, we present the following necessary and sufficient condition under which the attacker can uniquely determine the activation pattern $(D_i^m)_{i=1}^H$ for each data sample. For simplicity, we denote D_i^m for $D_i(X_m)$.

Proposition 2. *Given the knowledge on the exclusively activated neurons at the last-but-one layer, the attacker can determine $\{D_i^m\}_{i=1,m=1}^{H,M}$ with uniqueness, if and only if each data sample X_m has at least one exclusively activated neuron in D_i^m, $i \in \{1, ..., H-1\}$.*

Below, we provide a brief algorithmic proof on the sufficiency of the condition above for determining $\{D_i^m\}_{i=1,m=1}^{H-1,M}$. In general, the procedure of determining the activation patterns is recursively done from the last-but-one layer to the input layer. Initially, we have already recovered at least two exclusively activated neurons in D_H^m for each input X_m. Therefore, if we consider the j-th neuron as the exclusively activated one for X_m, then the j-th column of $\overline{G}_{H-1}$ only consists of the gradient w.r.t. X_m. Hence, by checking he non-zero poisitions of the j-th column, we immediately get the diagonal terms of D_{H-1}^m. Similarly, with the $H-1$ layer solved, the procedure can be done for the $(H-2)$-th layer, and so on, until the input layer. Readers may refer to Fig. 2 for better intuition. Meanwhile, the attacker can further determine the remaining non-exclusively activated neurons in D_H^m for each m-th sample by solving $\text{softmax}(W_H f_H^m) = p^m$ with optimization techniques, where p_c^m is known in prior from the loss vector $\overline{g}^m$ (Eq. 3). Details on the above algorithm can be found in Algorithm 3 in Appendix 8.5.

• **Implement Multi-Sample Reconstruction Attacks.** To summarize, our theory-oriented multi-sample data reconstruction attack in general

shares the same attack pipeline as its single-sample counterpart. A noteworthy technical detail is, in the multi-sample case, as the number of variables can be very large (e.g., about 1 million variables for 8 samples from ImageNet), classical general-purpose linear equation solvers like LU decomposition are hence intractable. By observing the sparsity of the resulting linear equation system (i.e., in each scalar equation, we observe on average over 95% variables have a vanishing coefficient), we leverage a special-purpose sparse linear equation solver called LSMR [CLS11] which has a desirable computation complexity for large-scale sparse linear equations. For visualization, Fig. 10 in Appendix 8.1 shows a mini-batch of skin cancer images reconstructed from the corresponding average gradient of 8 training samples from a face recognition system, when the plausibility condition is satisfied. Compared with the ground-truth input, the reconstruction from our algorithm is still of high quality.

• **Extension to Convolutional Neural Networks.** Finally, we extend our proposed data reconstruction attack algorithm above to image classifiers with convolutional neural networks. It is worth to notice, most of the shallow and deep convolution neural network (CNN) for classification usually implement its last several layers as an FCN with ReLUs, which on one hand facilitates transfer learning, but also on the other hand eases data reconstruction attacks [1]. We denote these CNN classsifiers explicitly as $f = h \circ g$, where h is the feature extraction part which are mainly composed of convolutional and pooling operations, and g is an FCN for classification. In detail, our attack pipeline on CNNs contains the following two stages,

1) At the first stage, we reconstruct the inputs to the fully connected component g. Based on our theory and algorithms on ReLU networks, it can be done with low reconstruction errors for both the single-sample and the multi-sample cases.

2) At the second stage, from the reconstructed last feature maps (i.e., the inputs to the FCN layer, denoted as $(F_m)_{m=1}^M$), we propose the following two different technical choices for reconstructing the training samples from the gradient.

• *Deconvolution-Based Approach:* Following the inverse order of the feature extraction part h, we correspondingly build a deconvolutional neural network by using a transposed convolution operation [NHH15], which shares the same filters with the original convolution operation, to invert the output of the convolution, and using an upsampling operation to invert the output of the pooling operation. We can directly input the reconstructed feature map into the built deconvolution network to get the reconstructed training samples. With evaluations, we find this approach is especially effective and

[1] Some architectures like ResNet would by default use only one linear layer as the classifier part instead of an FCN, while we find a theory-oriented reconstruction attack is still plausible. Details can be found in Appendix 8.6.

efficient for shallow CNNs like LeNet.

• *Hybrid Approach:* We propose to fuse the previous optimization-based approach with the recovered feature maps from our attack on the FCN part in the first stage, which corresponds to the following learning objective, $\min_{X_i} D(\frac{1}{M}\sum_{i=1}^{M}\nabla_W \ell(f(X_i;W),\overline{Y}_i),\overline{G})+\lambda\|h(x_m)-\overline{F}_m\|_1$, where the latter term aims at minimizing the L1 distance between the reconstructed input's feature map with the reconstructed ones, and the former term is to minimize the difference between the ground-truth gradient and the gradient computed from the variables. In practice, we implement the distance function D as the cosine distance between the flattened gradient at each layer. It is worth to notice, different from existing learning-based methods, our hybrid approach also utilizes the reconstructed labels $\overline{Y}_i$ with our proposed attack on the FCN part, which makes our approach highly stable. Via extensive evaluations in Section 6.1, we empirically prove that our hybrid approach noticeably improves the attack effectiveness of previous optimization-based attacks on deep CNNs in the majority of test cases.

5 Overview of Evaluations

5.1 Datasets

We provide an overview on the datasets, the corresponding learning task and the model architecture in Table 1. Based on considerations of research ethics, we choose public datasets to construct the listed data-sensitive scenarios for evaluations. Nevertheless, as our attack requires almost no prior knowledge about the dataset, we do think the reported results would truthfully reflect the potential threats to the confidentiality of private training data in the real world. For more details on each scenario, please refer to Appendix 8.2.

Table 1: Datasets and model architectures in experiments. *The letter in **bold** is used as an alias for the dataset, used in Table. 2

Domain	Dataset	Task	Input Size	Model
Benchmarks	**M**NIST	Hand-written Digits	$(1, 28, 28)$	FCN/Shallow CNN
	CIFAR-10	Objects	$(3, 32, 32)$	FCN/Shallow CNN
	ImageNet	Objects	$(3, 224, 224)$	Deep CNNs
Healthcare	**T**exas-100	Clinical Procedure	(6169)	FCN
	ISIC **S**kin Cancer	Lesion Diagnosis	$(3, 224, 224)$	Deep CNNs
Misc.	**F**aceScrub	Face Recognition	$(3, 224, 224)$	Shallow/Deep CNNs
	Purchase-100	Shopping Profile	(600)	FCN
	Location-100	Check-in Profile	(446)	FCN

5.2 Model Architectures

Fully-Connected Neural Network (FCN). As introduced in Section 3.3, FCN usually consists of multiple linear layers interleaved with ReLUs.

In the experiments, we implement the learning model as three-layer FCNs when the corresponding input are binary features or simple visual data like MNIST.

Shallow CNN. For shallow CNNs, we choose the famous LeNet architecture [LBB+98], which is commonly used on low-resolution images. A typical LeNet has its feature extraction module g that consists of: one input layer, one convolutional layer (kernel size 6×6, 6 output channels, 1 stride), one max-pooling layer of size 2×2, one convolutional layer (kernel size 5×5, 16 output channels, 1 stride), one max-pooling layer of size 2×2, and has its classifier module h as a 3-layer FCN.

Deep CNNs. Finally, to evaluate our methodology at a more practical level, we also provide extensive experiments on most popular deep CNNs of wide commercial usages, including ResNet-18 [HZR+16], VGG-11 [SZ15], DenseNet-121 [HLW17], AlexNet [Kri14], ShuffleNet-v2-x0-5 [MZZ+18], Inception-V3 [SVI+16], GoogLeNet [SLJ+15], MobileNet-V2 [SHZ+18]. For all these architectures except ResNet-18, we specify the classification module as a three-layer FCN which has a hidden layer of 4096 neurons. We leave the classification module of ResNet-18 by default as a linear layer as in [HZR+16] for validating our alternative reconstruction algorithm in Appendix 8.6.

5.3 Metrics

To evaluate the effectiveness of data reconstruction attacks, we measure the reconstruction error between pairs of reconstructed and ground-truth samples with the following metrics. Below, we denote the reconstructed (ground-truth) data input as $\hat{X}_m$ (X_m).

• **Mean Square Error (MSE)** measures the L2 difference between the reconstructed input and the ground-truth input, averaged over coordinates, used in [ZLH19,ZMB20]. Formally, the MSE metric writes $\text{MSE}(\hat{X}_m, X_m) = \frac{1}{\dim \mathcal{X}} \|\hat{X}_m - X_m\|_2$, where $\dim \mathcal{X}$ is the dimension of the input space. Intuitively, a lower MSE metric means a more effective attack.

• **Peak Signal-to-Noise Ratio (PSNR)** measures the ratio of the effective information and noises in the reconstructed images, which is also used in [GBD+20]. It formally computes as $\text{PSNR}(\hat{X}_m, X_m) = -10 \times \log_{10}(\text{MSE}(\hat{X}_m, X_m))$. It is worth to notice, although PSNR is a derived metric from MSE, it behaves slightly different when being averaged and provides a better perspective on comparing the recognizability of the reconstructed input, especially for the visual scenarios.

Remark 3 (PSNR and Recognizability)**.** *For better intuitions about the numeric results, based on our experimental observations, when the PSNR metric is over* 6*, the reconstructed image is largely correct in color distributions and has a certain level of recognizable details (e.g., Fig. 8 in Appendix 8.1). Meanwhile, a higher PSNR usually means more recognizable details.*

When PSNR is over 20 *(e.g., Fig. 10 in Appendix 8.1), the reconstruction results are highly recognizable for human observers. When PSNR is over* 100 *or even* ∞*, the reconstruction is considered to be exact (e.g., Fig. 9 in Appendix 8.1).*

For single-sample cases, we simply apply the above two metrics on the reconstructed data input and the ground-truth. For multi-sample cases, as the attacker aims at reconstructing the orderless batch of data samples, we first leverage the Hungarian algorithm [Kuh55] to find the best-matching pairs of reconstructed and ground-truth data inputs according to the pairwise MSE. Then we compute the average performance metrics over the best-matching pairs.

Besides, we also want to measure the effectiveness of label reconstruction. For this purpose, we use

- **Label Reconstruction Accuracy (lAcc)** to measure the hit ratio of the reconstructed labels. Formally, lAcc computes the ratio between the number of the labels present in both the ground-truth and the reconstructed label sets [2] and with the ground-truth batch size.

For more details on other common settings, please refer to Appendix 8.4.

6 Results & Analysis

6.1 Attack Effectiveness

Attacks on Various Scenarios. We compare the effectiveness of our proposed data reconstruction attack with the previous attacks, including DLG [ZLH19], iDLG [ZMB20], and Inverting [GBD$^+$20], on each scenario we describe in Section 5.1. All baseline attacks are based on solving the optimization problem like Eq. 1 to find approximate solutions to the gradient equation. For more details on baselines, please see Section 2. In this part, we consider the situations when the batch size M is 1 and 8 respectively for evaluating the single-sample and multi-sample cases. It is worth to notice, iDLG only differs from DLG by inferring the exact label in prior when the batch size is 1, and is identical to DLG for the multi-sample case. Hence, we omit reporting iDLG's result for $M = 8$. Besides, we also provide a tentative comparison of the time cost for different attack approaches in Appendix 8.3.

Results & Analysis. Table 2 compares the performance of different attacks on different scenarios. In summary, these results strongly implies that the threats of data reconstruction attacks may ubiquitously exist on a wide range of intelligent systems and popular neural network architectures.

(1) Risks of Label Leakage. First, in terms of the lAcc metric, in 54 out of 62 test cases, our attack reaches 100% accuracy whenever reconstructing

[2] Technically, multiset, which caters for repetitive labels in the same batch.

Table 2: Reconstruction Metrics of our proposed attack and previous attacks on different scenarios. All statistics are averaged on 10 controlled repetitive tests (Details in Appendix 8.4). The best metrics in each case are marked in **bold**.

			DLG [ZLH19]			iDLG [ZMB20]			Inverting [GBD+20]			Ours		
			lAcc.	MSE	PSNR	lAcc.	MSE	PSNR	lAcc.	MSE	PSNR	lAcc.	MSE	PSNR
FCN [GBC16]	M. [LBB+98]	$M=1$	0.200	1.865	1.300	1.000	2.082	−2.811	1.000	0.853	0.690	**1.000**	**0.000**	**152.267**
		$M=8$	0.500	2.176	−3.284	-	-	-	0.275	2.323	−3.477	**1.000**	**0.207**	**32.069**
	P. [SSS+17]	$M=1$	0.000	NaN	−63.498	1.000	0.000	88.423	1.000	0.297	24.675	**1.000**	**0.000**	∞
		$M=8$	0.275	33.977	−2.927	-	-	-	0.900	7.681	−8.135	**1.000**	**0.000**	∞
	T. [SSS+17]	$M=1$	0.000	1.050	−0.214	1.000	0.001	49.645	1.000	45.507	−16.304	**1.000**	**0.000**	∞
		$M=8$	0.563	45.701	−27.366	-	-	-	0.100	24.378	−13.446	**1.000**	**0.000**	∞
	L. [SSS+17]	$M=1$	0.700	0.329	73.612	1.000	0.002	69.070	1.000	0.017	34.987	**1.000**	**0.000**	∞
		$M=8$	0.000	18.912	−2.078	-	-	-	0.775	6.532	−7.508	**1.000**	**0.000**	∞
LeNet [LBB+98]	M.	$M=1$	0.400	1.575	−1.419	1.000	1.065	−0.269	1.000	1.054	−0.219	**1.000**	**0.757**	**1.231**
		$M=8$	0.375	1.226	−0.834	-	-	-	0.600	1.083	−0.310	**1.000**	**0.857**	**0.684**
	C. [Kri09]	$M=1$	0.000	0.519	4.952	1.000	0.545	3.178	1.000	**0.187**	**8.462**	**1.000**	0.202	7.269
		$M=8$	0.350	0.867	1.426	-	-	-	0.800	**0.191**	**7.576**	**1.000**	0.309	5.927
	F. [NW14]	$M=1$	0.000	0.263	6.098	1.000	0.199	9.755	1.000	**0.159**	**10.471**	**1.000**	0.163	7.994
		$M=8$	0.700	0.253	6.660	-	-	-	0.775	**0.181**	**7.998**	**1.000**	0.206	7.427
ResNet [HZR+16]	I. [RDS+15]	$M=1$	0.000	0.481	3.443	1.000	0.492	3.104	1.000	0.362	4.470	**1.000**	**0.145**	**9.415**
		$M=8$	0.125	0.375	4.581	-	-	-	0.917	0.296	5.718	**1.000**	**0.206**	**7.329**
	F.	$M=1$	0.000	0.503	3.746	1.000	0.464	3.687	1.000	0.226	6.630	**1.000**	**0.018**	**18.056**
		$M=8$	0.300	0.571	2.731	-	-	-	0.781	0.244	6.616	**0.958**	**0.188**	**7.903**
	S. [GCC+18]	$M=1$	0.400	0.371	5.530	1.000	0.276	6.788	1.000	0.071	**12.484**	**1.000**	**0.060**	12.206
		$M=8$	0.375	0.464	3.791	-	-	-	0.792	0.217	7.289	**0.875**	**0.127**	**10.008**
DenseNet [HLW17]	I.	$M=1$	0.000	0.207	6.842	1.000	0.428	3.690	1.000	0.202	6.940	**1.000**	**0.202**	**6.947**
		$M=8$	0.125	0.277	5.686	-	-	-	0.625	0.274	5.722	**1.000**	**0.231**	**6.725**
	F.	$M=1$	1.000	0.303	5.186	1.000	0.328	4.836	1.000	0.268	5.724	**1.000**	**0.053**	**12.750**
		$M=8$	0.250	0.275	6.068	-	-	-	0.625	0.290	5.845	**1.000**	**0.109**	**9.817**
	S.	$M=1$	1.000	0.022	16.512	1.000	0.030	15.271	1.000	**0.015**	**18.264**	**1.000**	0.020	16.919
		$M=8$	0.250	0.115	9.816	-	-	-	0.750	**0.109**	9.990	**0.825**	0.122	**10.130**
ShuffleNet [MZZ+18]	I.	$M=1$	1.000	0.582	2.354	1.000	0.576	2.394	1.000	0.552	2.280	**1.000**	**0.134**	**8.738**
		$M=8$	0.875	0.593	5.180	-	-	-	0.375	**0.310**	**5.180**	**1.000**	0.366	4.811
	F.	$M=1$	1.000	0.532	2.742	1.000	0.531	2.746	**1.000**	**0.325**	**4.884**	1.000	0.572	2.428
		$M=8$	**1.000**	0.518	**3.359**	-	-	-	0.375	**0.255**	0.254	0.812	0.694	1.753
	S.	$M=1$	1.000	0.194	7.119	1.000	0.195	7.103	1.000	**0.127**	**8.956**	**1.000**	0.346	4.614
		$M=8$	0.625	0.243	6.203	-	-	-	0.625	**0.117**	**9.549**	**0.875**	0.297	5.475
Inception [SVI+16]	I.	$M=1$	0.000	0.506	2.957	1.000	1.567	−1.951	1.000	0.305	5.164	**1.000**	**0.144**	**8.407**
		$M=8$	0.000	1.462	−1.494	-	-	-	0.125	0.425	**4.104**	**1.000**	**0.420**	4.077
	F.	$M=1$	0.000	0.759	1.196	1.000	1.273	−1.050	1.000	0.487	3.126	**1.000**	**0.242**	**6.165**
		$M=8$	0.125	1.188	−0.123	-	-	-	0.625	0.690	2.234	**0.887**	**0.240**	**6.319**
	S.	$M=1$	0.000	0.368	4.341	1.000	**0.046**	**13.408**	1.000	0.318	4.970	**1.000**	0.063	12.011
		$M=8$	0.000	0.258	5.934	-	-	-	0.500	**0.133**	**9.306**	**0.838**	0.170	8.376
GoogLeNet [SLJ+15]	I.	$M=1$	1.000	0.519	3.349	1.000	0.576	2.394	1.000	0.276	5.590	**1.000**	**0.202**	**6.942**
		$M=8$	0.000	0.593	2.370	-	-	-	0.375	0.450	3.631	**1.000**	**0.384**	**4.540**
	F.	$M=1$	1.000	0.532	2.742	1.000	0.531	2.746	1.000	0.517	2.866	**1.000**	**0.268**	**5.719**
		$M=8$	1.000	0.519	3.349	-	-	-	0.375	0.329	**5.797**	**1.000**	**0.298**	5.703
	S.	$M=1$	1.000	0.194	7.119	1.000	0.195	7.103	1.000	**0.030**	**15.289**	**1.000**	0.045	13.505
		$M=8$	**1.000**	0.253	6.023	-	-	-	0.750	**0.071**	**11.798**	0.875	0.178	9.016
VGGNet [SZ15]	I.	$M=1$	0.000	0.213	6.720	1.000	0.256	5.924	1.000	0.203	6.921	**1.000**	**0.191**	**7.179**
		$M=8$	0.000	0.272	5.744	-	-	-	0.875	0.260	5.929	**1.000**	**0.203**	**6.425**
	F.	$M=1$	1.000	0.347	4.599	1.000	0.520	2.842	1.000	0.264	5.790	**1.000**	**0.132**	**8.801**
		$M=8$	0.375	0.227	6.872	-	-	-	0.875	0.234	6.774	**1.000**	**0.057**	**12.570**
	S.	$M=1$	0.000	0.272	5.744	1.000	**0.018**	**17.556**	1.000	0.025	16.066	**1.000**	0.024	16.180
		$M=8$	0.625	0.083	11.055	-	-	-	0.750	0.076	11.459	**1.000**	**0.058**	**13.515**
MobileNet [SHZ+18]	I.	$M=1$	1.000	0.582	2.354	1.000	0.576	2.394	1.000	0.978	0.095	**1.000**	**0.132**	**8.790**
		$M=8$	0.000	0.593	2.370	-	-	-	0.750	0.786	1.196	**1.000**	**0.274**	**5.725**
	F.	$M=1$	1.000	0.532	2.742	1.000	0.531	2.746	1.000	0.728	1.380	**1.000**	**0.267**	**5.728**
		$M=8$	0.625	0.519	3.349	-	-	-	0.625	0.349	5.602	**1.000**	**0.312**	**5.731**
	S.	$M=1$	1.000	**0.194**	**7.119**	1.000	0.195	7.103	1.000	0.316	5.007	**1.000**	0.216	6.665
		$M=8$	0.000	0.253	6.023	-	-	-	0.625	**0.092**	**10.816**	**0.812**	0.271	6.569
AlexNet [Kri14]	I.	$M=1$	1.000	0.368	0.325	1.000	0.283	5.482	1.000	0.237	6.260	**1.000**	**0.060**	**12.205**
		$M=8$	0.375	0.480	3.355	-	-	-	0.750	0.256	6.137	**1.000**	**0.113**	**9.743**
	F.	$M=1$	0.000	0.413	3.840	1.000	0.254	5.950	1.000	0.192	7.162	**1.000**	**0.009**	**20.096**
		$M=8$	0.750	0.325	5.278	-	-	-	0.875	0.148	8.708	**1.000**	**0.051**	**13.220**
	S.	$M=1$	0.000	0.159	7.995	1.000	0.025	28.037	1.000	0.007	21.583	**1.000**	**0.002**	**27.280**
		$M=8$	0.375	0.358	4.790	-	-	-	**1.000**	0.140	9.303	0.875	**0.044**	**14.425**

the label(s) of a single sample or multiple samples [3], much better than the performance of each baseline. In other words, it means, with our attack, the adversary is very likely to be able to conduct an exact label reconstruction only if the exclusive activation condition in Proposition 1 is satisfied with the given training batch. In fact, we also do statistics on the proportion of training batches which satisfy the exclusive activation condition in three typical scenarios, i.e., FCN, LeNet and VGGNet with the last-but-one fully-connected layer of dimension 512. The results are shown in Fig. 4, along 10 training epochs. As we can see, when the batch size is smaller than a certain

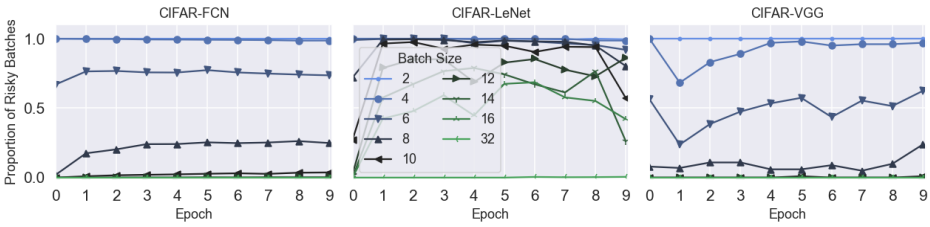

Figure 4: Proportions of mini-batches satisfying the exclusive activation condition in CIFAR-10 over 10 epochs of training.

threshold (about $8-10$ in these cases), the exclusive activation condition is satisfied in over 50% of the batches. Strikingly, for LeNet, the threshold on the batch size doubles to 16 and almost 80% of the batches satisfy the exclusive activation condition when the batch size is smaller. Meanwhile, the proportion of risky batches even increases when the learning process goes on. These above phenomenons alarm the privacy risks of label leakage when the training batch behind the gradient is potentially separable at the granularity of the exclusively activated neurons. It may be inspiring for future researches to consider to check the number of exclusively activated neurons in a batch for alerting the potential label leakage.

(2) Risks of Data Reconstruction. In terms of the MSE and PSNR metrics, our attack algorithm still shows a more desirable reconstruction quality than baselines in the majority of scenarios (43 out 62 test cases). Noteworthily, on all four FCN cases, the reported reconstruction MSE of our attack is very close or equal to zero. Especially on the three cases with binary inputs (i.e., Purchase-, Texas-, Location-100), our attack yields exact reconstructions of the data inputs, while previous attacks could only reconstruct more noises from the gradients than the inputs itself (as their negative PSNR shows). Next, results on other CNN architectures show, although our

[3] When we check the cases that fail to achieve 100% lAcc, we find it is mainly because the calculated ratio vector has numeric errors, which makes the detection of repetitive values not fully exact.

deconvolution-based attack on LeNets presents a slightly lower performance than the *Inverting* baseline, our hybrid approach does outperform the baseline methods with a non-trivial margin. For example, against all the three scenarios with AlexNet, the average improvement in PSNR of our methods over the best baseline is about 78%. This noticeable improvement empirically demonstrates the benefits of fusing the reconstructed feature maps into the conventional learning-based attacks. Moreover, among the commercial deep CNNs, we find almost all neural network architectures would suffer from at least one data reconstruction attacks with PSNR higher than 7, which usually indicates a recognizable reconstruction. Visualizations are provided in Fig. 8 in Appendix 8.5. In summary, combined with risks of label leakage, we find the threats of data reconstruction attacks ubiquitously exist in a wide range of application scenarios and neural network architectures. For discrete data inputs and FCNs, the attack is proved to be more threatening at both theoretical and empirical levels. As FCNs remain a popular neural network architecture in current practices of deep learning, we hope our analysis on the mechanism of data reconstruction attack against FCN could help our community develop effective mitigation in the future.

6.2 Ablation Studies

In this part, we provide more empirical evidence on several key factors which influence the effectiveness of data reconstruction attacks. We mainly conduct our attack and the baselines with three neural network architectures, i.e., a three-layer FCN, LeNet and VGG, on CIFAR-10. For better comparison, we implement the classification module of the latter two model also as three-layer FCNs, which consist of a 3024-dim input layer, a d-dim hidden layer (i.e., the last-but-one layer) and a 10-dim output layer.

6.2.1 Impact of Batch Size

First, we vary the size of the victim batch from 2 to 16 with a stride 2, together with 32. Fig. 5 reports the corresponding attack performance in terms of PSNR and lAcc. As we can see, most PSNR curves show decreases when the batch size increases. From our perspective, this phenomenon is mainly because: for a larger batch, the corresponding gradient equation system is more likely to be underdetermined, which, if the attacker instead searches for the least-square-error solution, would result in a larger reconstruction error. Moreover, on the FCN and VGGNet, we notice the PSNR curves and lAcc bars of all attacks decrease more radically when the batch size is below 10, and stay stable afterwards. Such a phenomenon is more clearly observed on our attack. In fact, according to Fig. 4, when the batch size is over 10 for FCN and VGG, the probability of a batch to satisfy the exclusive activation is close to zero, which makes an attacker less likely to

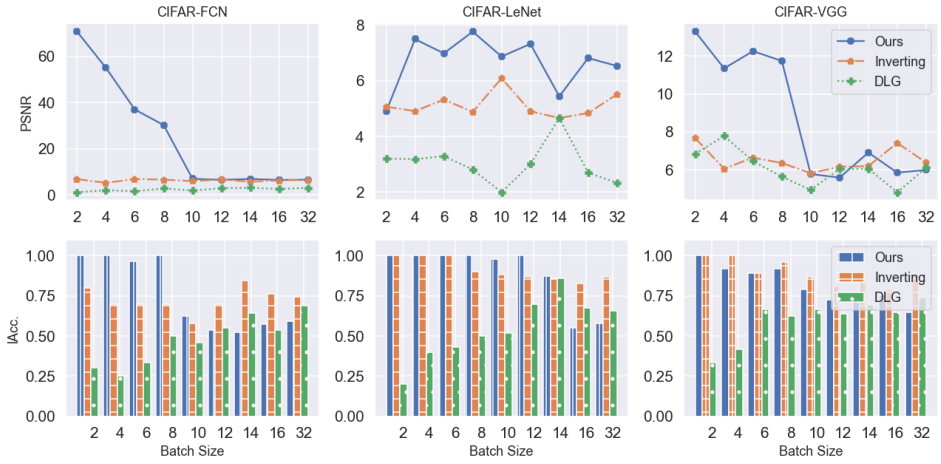

Figure 5: Effectiveness of data reconstruction attacks when the batch size is varied.

separate out the activation pattern of each single sample from the average gradient. As a result, the performance of our attack degrades to a similar level as the baselines when the batch size goes over the threshold. Similarly, as the threshold for LeNet is over 16 (as shown in Fig. 4), the corresponding PSNR curves in Fig. 4 show a low variance. In conclusion, a larger batch size usually indicates less threats from data reconstruction attacks, while, when the batch size increases over a certain threshold, the risk level becomes stable.

6.2.2 Impact of Layer Width

Next, we vary the width d of the last-but-one layer of the FCN parts of all three architectures in $64, 128, 256, 512, 1024$. Fig. 6 shows the corresponding attack performance when the batch size is 4 and the model is untrained, in terms of PSNR and lAcc. As we can see, except the unstable performance of DLG, the effectiveness of both our attack and Inverting in most cases increases when the layer width is increased (on FCN, as both the PSNR and lAcc of our attack are high, the trend is hence less obvious). Based on our theory, a wider last-but-one layer imposes the following two-sided impacts: (1) It provides the attacker more equations to determine the data input and (2) increases the possibility of batches to satisfy the exclusive activation condition, which further enlarges the risks of exact label leakage (as shown by the almost 100% label reconstruction accuracy of our attack in Fig. 7). As a suggestion for intelligent system builders, when the model architecture satisfies the utility requirement, a smaller last-but-one-layer can to some

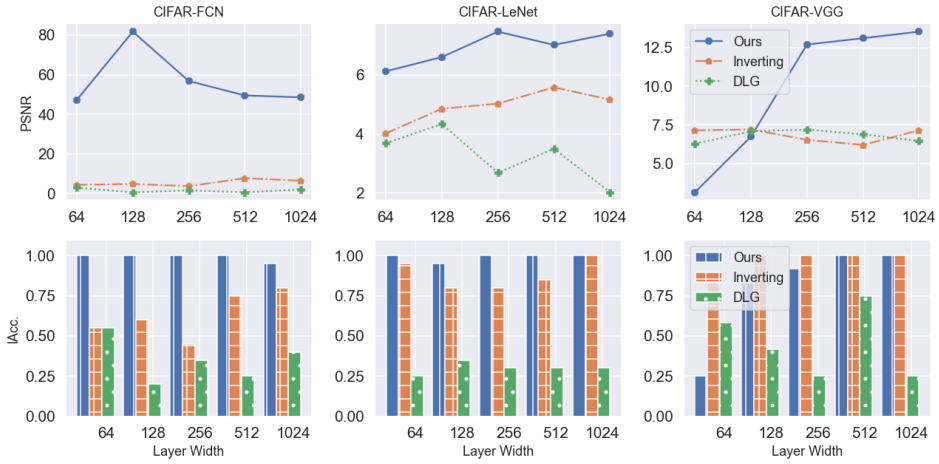

Figure 6: Effectiveness of data reconstruction attacks when the width of the last-but-one fully-connected layer is varied.

degree alleviate the risk of gradient leakage.

6.2.3 Impact of Training Phase

In the third part, we study how the attack effectiveness changes along the learning process. We train each model for 10 epochs and conduct data reconstruction attacks on 10 randomly sampled training batches at the beginning of each epoch. All the models converge after 10 training epochs. The upper part of Fig. 7 presents the box-plot of the attack effectiveness in terms of PSNR between every single sample and its best-matching reconstruction. As we can see, with the learning progressing, the median values of the PSNRs show a trend of decline. According to our analysis, behind the decline is the decreasing number of non-vanishing scalar gradients and the resulting underdetermined gradient equation. To validate this point, we collect and plot the non-sparsity of gradients (i.e., the ratio of gradient coordinates larger than 10^{-7}). As is shown in the lower part of Fig. 7, on all three scenarios, the change of the reconstruction quality roughly conforms to that of the gradient sparsity. From this experiment, an attacker seems to gain more advantages if he/she chooses the attack timing as the initial stage of the learning process, while system builders may adopt a stricter privacy mechanism on the gradient at the beginning of the learning process and gradually relax the protection mechanism later on for better efficiency and utility on the primary learning task.

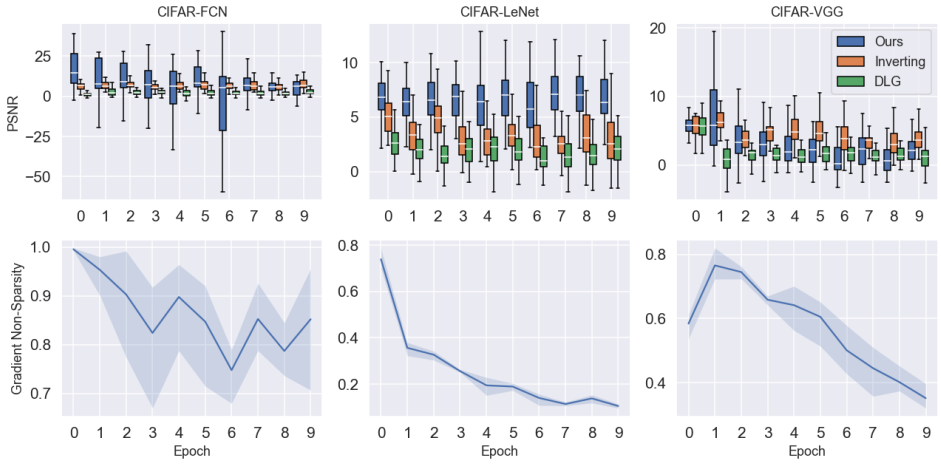

Figure 7: Effectiveness of data reconstruction attacks and the non-sparsity of captured gradients in 10 training epochs.

7 Conclusion

In this paper, we provide the first systematic study on the threat of data reconstruction attacks on gradients of deep learning systems. To shed light on future mitigation studies, we present the first linear-equation-solver interpretation on the underlying mechanism of this new attack class, propose a novel attack method which outperforms existing attacks on the majority of test cases in terms of both attack effectiveness and stability, and provide extensive evaluations on a number of real-world application scenarios and many popular neural network architectures to alert system builders on potential data leakage from gradients. We hope our study can arouse more research interests and efforts from our community on further investigating and strengthening the privacy properties of gradients, in order to build more secure and private-preserving intelligent systems.

References

[ABC+16] M. Abadi, P. Barham, J. Chen, et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, 2016.

[Bub15] Sébastien Bubeck. Convex optimization: Algorithms and complexity. *Found. Trends Mach. Learn.*, 8:231–357, 2015.

[CLE+19] N. Carlini, C. Liu, Úlfar Erlingsson, et al. The secret sharer: Evaluating and testing unintended memorization in neural networks. In *Security*, 2019.

[CLS11] FongDavid Chin-Lung and SaundersMichael. Lsmr: An iterative algorithm for sparse least-squares problems. *SIAM Journal on Scientific Computing*, 2011.

[DSR+18] Vasisht Duddu, Debasis Samanta, D. Vijay Rao, et al. Stealing neural networks via timing side channels. *ArXiv*, abs/1812.11720, 2018.

[EKN+17] A. Esteva, B. Kuprel, R. Novoa, et al. Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542:115–118, 2017.

[FJR15] Matt Fredrikson, S. Jha, and T. Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *CCS*, 2015.

[FLJ+14] Matt Fredrikson, E. Lantz, S. Jha, et al. Privacy in pharmacogenetics: An end-to-end case study of personalized warfarin dosing. In *Security*, pages 17–32, 2014.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[GBD+20] Jonas Geiping, Hartmut Bauermeister, Hannah Dröge, et al. Inverting gradients - how easy is it to break privacy in federated learning? *ArXiv*, abs/2003.14053, 2020.

[GCC+18] D. Gutman, Noel C. F. Codella, M. E. Celebi, et al. Skin lesion analysis toward melanoma detection: A challenge at the 2017 international symposium on biomedical imaging (isbi), hosted by the international skin imaging collaboration (isic). *ISBI*, pages 168–172, 2018.

[GWY+18] Karan Ganju, Qi Wang, Wei Yang, et al. Property inference attacks on fully connected neural networks using permutation invariant representations. *CCS*, 2018.

[HAPC17] Briland Hitaj, Giuseppe Ateniese, and Fernando Pérez-Cruz. Deep models under the gan: Information leakage from collaborative deep learning. *CCS*, 2017.

[HLW17] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. *CVPR*, pages 2261–2269, 2017.

[HPW16] J. B. Heaton, Nicholas G. Polson, and J. H. Witte. Deep learning in finance. *ArXiv*, abs/1602.06561, 2016.

[HZR+16] Kaiming He, X. Zhang, Shaoqing Ren, et al. Deep residual learning for image recognition. *CVPR*, pages 770–778, 2016.

[JCB+20] Matthew Jagielski, N. Carlini, David Berthelot, Alex Kurakin, et al. High accuracy and high fidelity extraction of neural networks. In *Security*, 2020.

[JZJ+18] Yujie Ji, Xinyang Zhang, S. Ji, X. Luo, et al. Model-reuse attacks on deep learning systems. *CCS*, 2018.

[Kaw16] Kenji Kawaguchi. Deep learning without poor local minima. In *NIPS*, 2016.

[KB15] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *ArXiv*, abs/1412.6980, 2015.

[Kri09] A. Krizhevsky. Learning multiple layers of features from tiny images. 2009.

[Kri14] A. Krizhevsky. One weird trick for parallelizing convolutional neural networks. *ArXiv*, abs/1404.5997, 2014.

[Kuh55] H. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.

[LBB+98] Yann LeCun, Léon Bottou, Yoshua Bengio, et al. Gradient-based learning applied to document recognition. 1998.

[LF20] Klas Leino and Matt Fredrikson. Stolen memories: Leveraging model memorization for calibrated white-box membership inference. In *Security*, 2020.

[LvB18] Thomas Laurent and James von Brecht. The multilinear structure of relu networks. In *ICML*, 2018.

[LXL+18] Huichen Li, X. Xu, Chang Liu, Teng Ren, et al. A machine learning approach to prevent malicious calls over telephony networks. *S&P*, pages 53–69, 2018.

[MP+14] Guido Montúfar, Razvan Pascanu, et al. On the number of linear regions of deep neural networks. *ArXiv*, abs/1402.1869, 2014.

[MSC+19] Luca Melis, Congzheng Song, Emiliano De Cristofaro, et al. Exploiting unintended feature leakage in collaborative learning. *S&P*, pages 691–706, 2019.

[MZZ+18] Ningning Ma, X. Zhang, Hai-Tao Zheng, et al. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *ECCV*, 2018.

[NHH15] Hyeonwoo Noh, Seunghoon Hong, and B. Han. Learning deconvolution network for semantic segmentation. *ICCV*, pages 1520–1528, 2015.

[NSH19] Milad Nasr, Reza Shokri, and Amir Houmansadr. Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. *S&P*, pages 739–753, 2019.

[NW14] Hongwei Ng and S. Winkler. A data-driven approach to cleaning large face datasets. *ICIP*, pages 343–347, 2014.

[OSF19] Tribhuvanesh Orekondy, Bernt Schiele, and Mario Fritz. Knockoff nets: Stealing functionality of black-box models. *CVPR*, pages 4949–4958, 2019.

[PGM+19] Adam Paszke, Sam Gross, Francisco Massa, et al. Pytorch: An imperative style, high-performance deep learning library. In *NIPS*, pages 8024–8035. 2019.

[PZJY20] Xudong Pan, Mi Zhang, Shouling Ji, and Min Yang. Privacy risks of general-purpose language models. *S&P*, pages 1314–1331, 2020.

[RDS+15] Olga Russakovsky, J. Deng, H. Su, et al. Imagenet large scale visual recognition challenge. *IJCV*, 115:211–252, 2015.

[RHW86] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[Rob07] H. Robbins. A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407, 2007.

[SBB+19] Ahmed Salem, Apratim Bhattacharyya, Michael Backes, et al. Updates-leak: Data set inference and reconstruction attacks in online learning. *ArXiv*, abs/1904.01067, 2019.

[SD19] G. Sreenu and M. A. Saleem Durai. Intelligent video surveillance: a review through deep learning techniques for crowd analysis. *J. Big Data*, 6:1–27, 2019.

[SHZ+18] Mark Sandler, A. Howard, Menglong Zhu, A. Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. *CVPR*, pages 4510–4520, 2018.

[SLJ+15] Christian Szegedy, Wei Liu, Yangqing Jia, , et al. Going deeper with convolutions. *CVPR*, pages 1–9, 2015.

[SSS+17] R. Shokri, Marco Stronati, Congzheng Song, et al. Membership inference attacks against machine learning models. *S&P*, pages 3–18, 2017.

[SSZ19] Reza Shokri, Martin Strobel, and Yair Zick. Privacy risks of explaining machine learning models. *ArXiv*, abs/1907.00164, 2019.

[SVI+16] Christian Szegedy, V. Vanhoucke, S. Ioffe, et al. Rethinking the inception architecture for computer vision. *CVPR*, pages 2818–2826, 2016.

[SYL+18] W. Song, Heng Yin, Chang Liu, et al. Deepmem: Learning graph neural network models for fast and robust memory forensic analysis. *CCS*, 2018.

[SZ15] K. Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *ArXiv*, abs/1409.1556, 2015.

[SZH+19] Ahmed Salem, Yibao Zhang, Mathias Humbert, et al. Ml-leaks: Model and data independent membership inference attacks and defenses on machine learning models. *NDSS*, 2019.

[TZJ+16] Florian Tramèr, F. Zhang, A. Juels, et al. Stealing machine learning models via prediction apis. In *Security*, 2016.

[Wer87] A. Werschulz. What is the complexity of ill-posed problems? *Numer. Funct. Anal. And Optimiz.*, 1987.

[WG18] Binghui Wang and N. Gong. Stealing hyperparameters in machine learning. *S&P*, pages 36–52, 2018.

[WSZ+19] Zhibo Wang, Mengkai Song, Zhifei Zhang, et al. Beyond inferring class representatives: User-level privacy leakage from federated learning. *IEEE Conference on Computer Communications*, pages 2512–2520, 2019.

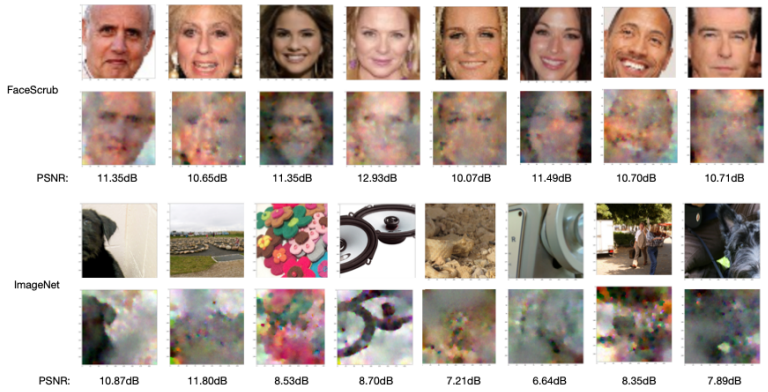

Figure 8: Reconstruction results of a random batch of 8 224×224-resolution samples from the gradient of an untrained AlexNet, with the corresponding PSNRs reported.

[YLW+15] Zhenlong Yuan, Yongqiang Lu, Zhaoguo Wang, et al. Droid-sec: deep learning in android malware detection. In *SIGCOMM*, 2015.

[ZLH19] Ligeng Zhu, Zhijian Liu, and Song Han. Deep leakage from gradients. *ArXiv*, abs/1906.08935, 2019.

[ZMB20] Bo Zhao, Konda Reddy Mopuri, and Hakan Bilen. idlg: Improved deep leakage from gradients. *ArXiv*, abs/2001.02610, 2020.

8 Appendix

8.1 Proof Sketches for Analytic Results

• *Proof Sketch for the Sufficiency Part of Proposition 2.* For convenience, we denote the j-th element of D_i^m as $\alpha_{i,j}^m$, i.e., the activation state of the j-th neuron at the i-th layer when X_m is the input. Formally, the exclusive activation of a neuron is expressed as $\alpha_{i,j}^m$ takes the value 1 for and only for a certain sample X_m. For intuition, readers may refer to Fig. 2 as an illustrative example.

• *Initial Step*: As a by-product of solving $\overline{g}_c^m$ and the assumed exclusivity, we already recovered at least two exclusive elements in D_H^m for each input X_m.

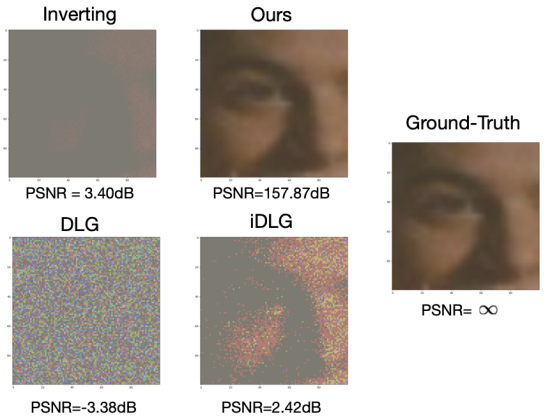

Figure 9: Reconstruction results of a celebrity face from its gradient of a $30000-500-20$ FCN on FaceScrub dataset, with the reported peak signal-to-noise. Compared with the experimental results of three existing attacks, our novel attack algorithm yields an almost perfect reconstruction quality.

- *Recurrent Step*: Next, we consider the gradient equation w.r.t. W_{H-1}.

$$\overline{G}_{H-1} = \frac{1}{M}\sum_{m=1}^{M}\sum_{c=1}^{K}\overline{g}_c(D_{H-1}^m ... W_0 X_m)([W_H]_c^T D_H^m) \tag{6}$$

  Then, we expand it explicitly to individual scalar equations.

$$\begin{aligned} M[\overline{G}_{H-1}]_{ij} &= \sum_{m=1}^{M}\sum_{c=1}^{K}\overline{g}_c\alpha_{H-1,i}^m f_{H-2,i}^m [W_H]_{jc}\alpha_{H,j}^m \\ &:= \sum_{m=1}^{M} C_{ij}^m \alpha_{H-1,i}^m \alpha_{H,j}^m \end{aligned} \tag{7}$$

  In the last line, we use the C_{ij}^m to replace the multiplier (which we assume is always non-zero). The following is the key of the recurrent step. As $\{D_H^m\}_{m=1}^M$ has at least one exclusive nonzero position to each other, the terms in the summation above therefore has at most one non-vanishing term for this exclusively activated neuron, indexed by e.g., j, which can be found based on the knowledge of $\{D_H^m\}_{m=1}^M$. In fact, the j-th column of $\overline{G}_{H-1}$, i.e., $[C_{ij}^m \alpha_{H-1,i}^m]$, immediately gives the diagonal terms of D_{H-1}^m, if we simply check the non-zero positions of $[\overline{G}_{H-1}]_{:,j}$. Similarly, with the solved $\{D_{H-1}^m\}_{m=1}^M$, the procedure can be done for the $(H-2)$-th layer, and so on, until the input layer.

- *Proof Sketch for the Necessity Part of Proposition 2.* In this part, we prove by contradiction this condition is also necessary for uniquely determining the

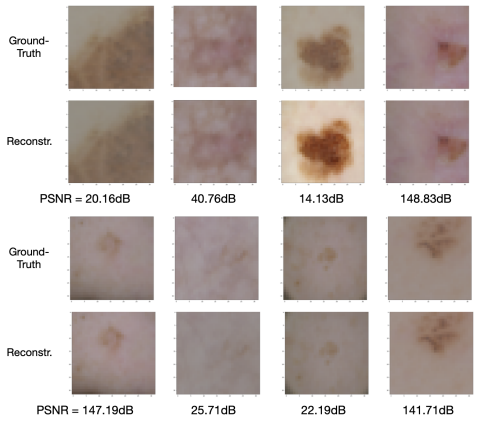

Figure 10: Reconstruction results of a batch of skin cancer imaging (of size 8) from its gradient of a $(3072 - 500 - 9)$ FCN on ISIC skin cancer dataset, with the PSNR metric reported below each best-matching sample pair.

activation patterns. Assume otherwise there exist a batch of training samples where the m-th sample X_m has its activation pattern $(D_i^m)_{i=1}^H$ that are not exclusively activated at the i-th layer (for a certain $i \in \{1, \ldots, H-1\}$) yet the activation patterns of each data sample in this batch are uniquely determined by the ground-truth gradient $(\overline{G}_i)_{i=0}^H$ (recall $\overline{G}_i$ denotes the gradient of the loss function w.r.t. the weight matrix W_i, which connects the i-th and the $(i+1)$-th layers).

Denote the indices of the samples which activate the j-th neuron at the i-th layer as $\mathcal{I}(j)$. Following the above assumption, we have $|\mathcal{I}(j)| \geq 2$ for all the j-th neuron activated by X_m. Then, it is easy to ascertain that, for each j-th neuron, the cooridinatewise *or* of the diagonal entries α_i^m of the boolean activation matrices of the samples indexed by $m \in \mathcal{I}(j)$ satisfy the following equation,

$$\bigvee_{m \in \mathcal{I}(j)} \alpha_i^m = \delta_{ij} \tag{8}$$

where, similar to Criterion 1, the d_{i+1}-dim vector δ_{ij} tests whether the element in the j-th row of G_i is non-vanishing or not. Formally, $(\delta_{ij})_k = 0$ if $(G_i)_{jk} = 0$, while otherwise $(\delta_{ij})_k = 1$. By collecting the equations as in Eq. 8 for each j, we have an equation system on the activation matrices at the i-th layer, which however is ill-posed and has many different solutions (mainly because the *or* operation is commutable). Therefore, this gives a contradiction and we prove the necessity of the existence of at least one exclusively activated neuron for the first $H-1$ layers.

8.2 Details of Scenarios

Academic Benchmarks. For the first group of scenarios, we study three standard benchmark image datasets, i.e., MNIST [LBB+98], CIFAR-10 [Kri09] and ImageNet [RDS+15], which are considered in previous data reconstruction attacks. These three datasets originate from the machine learning community and are widely used as computer vision benchmarks for image classification and many other tasks. These three datasets mainly cover daily objects (e.g., hand-written digits, pets, etc.) and show incremental complexity in various aspects (e.g., total pixels, color channels, class number). We accompany these datasets with all the three classes of neural network architectures we focus on, namely, fully-connected neural networks (FCN), shallow CNNs (LeNet) and deep CNNs (with details later). In Table 2, the FCN architecture on MNIST is $(768-200-10)$. The FCN part of the LeNet for CIFAR-10 has a hidden layer of 200 units.

Healthcare-related. For the second group of scenarios, we consider healthcare-related deep learning systems based on two real-world medical datasets: Texas-100 [SSS+17] and ISIC skin cancer imaging [GCC+18]. Texas-100 includes millions of hospital stay records as a part of the AMS medical datasets and can be used to build prediagnosis systems for determining the medical procedures the patient should take. In Texas-100, each record consists 6169-dimensional discrete attributes, which correspond to the corresponding patient's private profiles, including physiological, demographic and economic features. As a common practice, we implement the learning model as an FCN, which has the $(6169-500-100)$ architecture. As a comparison to discrete healthcare attributes, ISIC skin cancer datasets are composed of medical imaging of patients' lesions and their corresponding categories (e.g., benign, malignant and 7 more). According to a recent progress of applying Inception-V3 to dermatology [EKN+17], we implement this task as a 9-class lesion classification task with different types of Deep CNNs.

Miscellaneous. In the last group, we choose other reprensentative privacy-critical scenarios for evaluation purposes, which covers identity-related, location-related, shopping-behavioral aspects of common users' private personal profile. For the identity-related task, we consider a face recognition system built with a subset of the Facescrub dataset [NW14], which consists of portraits of 20 celebrities randomly selected from the full dataset. We implement the underlying model with both shallow and deep CNNs. In Table 2, the FCN part of the LeNet model has a hidden layer of 500 units. For location-related and shopping-behavioral tasks, we use two real-world datasets, Location-100 (i.e, check-in records originally from the Foursquare website) and Purchase-100 (i.e., shopping records originally from a Kaggle challenge). Both Location-100 and Purchase-100 are preprocessed and made publicly-available by [SSS+17], where each record is represented respectively as 600-dim. and 446-dim. binary features, characterizing the correspond-

ing user's purchased items or visited locations. The FCN architectures are respectively $(600 - 128 - 100)$, $(446 - 200 - 100)$.

8.3 Comparison of Time Complexity

For our attack, we report the average time of accomplishing the attack once. For the optimization-based baselines, we report the average time cost until the loss function of each attack becomes stable within a 10^{-5} variance, which is the same termination condition for experiments on the attack effectiveness. Table 3 below lists the average time costs over 10 repetitive tests when the batch size is 8, within the same environment described in the next section. As we can see, the time complexity of our attack is at a similar level as previous attacks and can be slightly more efficient for FCNs and shallow CNNs.

Table 3: Time costs of different attacks on CIFAR-10 with three typical architectures (sec.).

	DLG	Inverting	Ours
CIFAR-FCN	112.3	93.7	60.7
CIFAR-LeNet	167.2	55.3	8.7
CIFAR-VGGNet	240.0	212.0	227.7

8.4 More Details on Experimental Settings

For all the experiments compared with the previous attacks, we always fix the same victim configuration for each attack method, which includes but is not limited to the batch of samples, the model parameter and the captured gradients. We run 10 repetitive tests for each case and report the average attack performance. For experiments on attack effectiveness in Table 2, we randomly sample the victim batch repetitively until the batch satisfies the condition that there are at least two exclusively activated neurons at the last-but-one layer and at least one exclusively activated neurons at other layers, or such a batch is not found in 1000 trials. We leverage the deconvolution-based approach in attacks on LeNet and use the hybrid approach in attacks on deep CNNs, where the regularization coefficient λ in the hybrid approach is set as 0.01. As a complement, in the ablation study part, we vary the fixed configurations above by providing systematic evaluations on the impact of different architectures, the batch sizes and different learning phases on the resulting gradient leakage risk. For our attack via the hybrid approach on deep CNNs and other baselines, we optimize the learning objective until the loss function becomes stable within a 10^{-5} variance.

Experiment Environments. All the experiments are implemented with PyTorch [PGM+19], which is an open-source software framework for numeric computation and deep learning. All our experiments are conducted on a Linux server running Ubuntu 16.04, one AMD Ryzen Threadripper 2990WX 32-core processor and 1 NVIDIA GTX RTX2080 GPU.

8.5 Algorithm Details

In this part, we provide the algorithmic descriptions of the key procedures in our proposed data reconstruction attacks on FCNs.

Algorithm 1 Determine $\{\overline{g}_c^m\}_{c=1,m=1}^{K,M}$.

Input: The gradient of the last layer $\overline{G}_H$.
Output: Reconstructed labels $\{Y_1, \ldots, Y_M\}$ and loss vectors $(\overline{g}_c^m)_{c=1,m=1}^{K,M}$.
Compute $r_c := [\overline{G}_H]_c / [\overline{G}_H]_1$ for every c in $1, \ldots, K$.
Find all the disjoint index groups $(\mathcal{I}^m)_{m=1}^M$ where $(r_2)_j$ is constant whenever $j \in \mathcal{I}^m$. ▷ M is hence the inferred batch size and $\mathcal{I}^m$ is the index set of the exclusively activated neurons at the last-but-one layer.
for all c in $1, \ldots, K$ **do**
    for all m in $1, \ldots, M$ **do**
        Select an arbitrary index j from $\mathcal{I}^m$.
        $\overline{g}_c^m / \overline{g}_1^m \leftarrow (r_c)_j$.
    end for
end for
for all m in $1, \ldots, M$ **do**
    $Y_m \leftarrow$ Apply Algorithm 2 to $(\overline{g}_c^m)_{c=1}^K$.
    Estimate the upper bound of feasible range of $\overline{g}_1^m$ as $\delta_m \leftarrow \overline{g}_1^m / \overline{g}_{Y_m}^m$
    Fix $\overline{g}_1^m = 2 \times \delta_m / 3$. ▷ This is practiced in all our experiments.
    Calculate each $\overline{g}_c^m$ according to the ratio.
end for

Algorithm 2 Exact Label reconstruction from the loss vector.

Input: The loss vector for the m-th sample $(\overline{g}_c^m)_{c=1}^K$.
Output: Reconstructed label Y_m.
if $(\overline{g}_c^m)_{c=1}^K$ has one negative element **then**
return $Y_m \leftarrow$ The index of the negative element
else
return $Y_m \leftarrow 1$
end if

Algorithm 3 Determine activation patterns $(D_i^m)_{i=1,m=1}^{H,M}$.

Input: The gradients $(\overline{G}_i)_{i=0}^{H}$ at each layer and the index sets $(\mathcal{I}_H^m)_{m=1}^{M}$ of exclusively activated neurons at the last-but-one layer.
Output: Reconstructed activation patterns $(D_i^m)_{i=1,m=1}^{H,M}$.
$\mathcal{I}_{\text{cur}} \leftarrow (\mathcal{I}_H^m)_{m=1}^{M}$.
for all i in $H-1,\ldots,1$ **do**
    for all m in $1,\ldots,M$ **do**
        Select an arbitrary index j from $\mathcal{I}_{\text{cur}}^m$.
        $\text{diag}(D_i^m) \leftarrow ([\overline{G}_i]_{:,j} \neq 0)$
    end for
    Construct the index sets $(\mathcal{I}_i^m)_{m=1}^{M}$ of exclusively activated neurons at the i-th layer from $(D_i^m)_{m=1}^{M}$.
    $\mathcal{I}_{\text{cur}} \leftarrow (\mathcal{I}_i^m)_{m=1}^{M}$.
end for
for all m in $1,\ldots,M$ **do**
    Solve f_H^m from the equation $\text{softmax}(W_H f_H^m) = p_c^m$.
    $\text{diag}(D_H^m) \leftarrow (f_H^m \neq 0)$.
end for

8.6 Approximate Label Reconstruction and Linear Layer Reconstruction

When an FCN fails to satisfy the exclusive activation condition, or when the classification module of some deep CNNs consists of only one linear layer, our exact label reconstruction algorithm in Algorithm 2 could not be directly applied. Yet, we find the attacker can still exploit the following risky property of neural networks with ReLUs, which enables approximate label reconstruction attacks. We consider again the gradient of the last-but-one layer, which writes $\sum_{m=1}^{M} \overline{g}_c^m f_{H-1}^m$. Note f_{H-1}^m is the output of the $(H-1)$-th layer, which, according to the definition of ReLU, has all of its element as nonnegative. Moreover, to recap , $\overline{g}_c^m$ is negative only if c is the ground-truth label Y_m. As a result, once $[\overline{G}_H]_c$ has any non-negative value, then c always occurs as a ground-truth label in the victim's training batch. Such an approximate label reconstruction attack can achieve 100% accuracy when the training batch contains no data samples which share the identical labels. Otherwise, if the attacker has an expected batch size of the victim's training batch, he/she can guess sufficient labels randomly from all the cs that satisfy the above criterion. Moreover, once the labels are determined, the attacker can minimize optimization-based objective similar to Eq. 1 to reconstruct the feature maps which are input to a classification module implemented with only one linear layer. Then, he/she can leverage our hybrid approach again for attacks on deep CNNs. For the demonstration purpose, we specify the classification module of ResNet-18 in Table 2 as one linear layer.