Adversarial Neural Network Inversion via Auxiliary Knowledge Alignment

Ziqi Yang, Ee-Chien Chang, Zhenkai Liang
School of Computing, National University of Singapore
{yangziqi, changec, liangzk}@comp.nus.edu.sg

***Abstract*—The rise of deep learning technique has raised new privacy concerns about the training data and test data. In this work, we investigate the model inversion problem in the adversarial settings, where the adversary aims at inferring information about the target model's training data and test data from the model's prediction values. We develop a solution to train a second neural network that acts as the inverse of the target model to perform the inversion. The inversion model can be trained with black-box accesses to the target model. We propose two main techniques towards training the inversion model in the adversarial settings. First, we leverage the adversary's background knowledge to compose an auxiliary set to train the inversion model, which does not require access to the original training data. Second, we design a truncation-based technique to align the inversion model to enable effective inversion of the target model from partial predictions that the adversary obtains on victim user's data. We systematically evaluate our inversion approach in various machine learning tasks and model architectures on multiple image datasets. Our experimental results show that even with no full knowledge about the target model's training data, and with only partial prediction values, our inversion approach is still able to perform accurate inversion of the target model, and outperform previous approaches.**

I. Introduction

Machine learning (ML) models, especially deep neural networks, are becoming ubiquitous, powering an extremely wide variety of applications. The mass adoption of machine learning technology has increased the capacity of software systems in large amounts of complex data, enabling a wide range of applications. For example, facial recognition APIs, such as Apple's face-tracking APIs in the ARKit [1], provide scores of facial attributes, including emotion and eye-openness levels. There are also online services that evaluate users' face beauty and age [2], [3], [4]. Users often share their face-beauty scores on social media for fun. The prediction scores are definitely linked to the input face data, but it is not apparent to what level of accuracy the original data can be recovered. Privacy concerns arise in such applications.

There have been many research efforts in *model inversion*, which aims to obtain information about the training data from the model's predictions. They are largely divided into two classes of approaches. The first class inverts a model by making use of gradient-based optimization in the data space [5], [6], [7], [8], [9], [10], [11]. We call this class of approach *optimization-based* approach. For example, model inversion attack (MIA) [5] was proposed to infer training classes against neural networks by generating a representative sample for the target class. It casts the inversion task as an optimization problem to find the "optimal" data for a given class. MIA works for simple networks but is shown to be ineffective against complex neural networks such as convolutional neural network (CNN) [12], [13], [14]. Our experiment also obtains similar result as shown in Figure 1 column (f). This is mainly because the optimization objective of optimization-based approach does not really capture the semantics of the data space (see Section II-B). The second class of approach [15], [16], [17], [18] invert a model by learning a second model that acts as the inverse of the original one. We call this class of approach *training-based* approach. They target at reconstructing images from their computer vision features including activations in each layer of a neural network. Therefore, to maximally reconstruct the images, they train the second model using *full* prediction vectors on the *same* training data of the classifier.

In this work, we focus on the adversarial scenario, where an adversary is given the classifier and the prediction result, and aims to make sense of the input data and/or the semantics of the classification. Such problem is also known as the *inversion attack* [5], [19] wherein information about the input data could be inferred from its prediction values. There are two adversarial inversion settings, namely *data reconstruction* and *training class inference*.

Data reconstruction: In this attack, the adversary is asked to reconstruct unknown data given the classifier's prediction vector on it, which is exactly the inversion of the classifier. For example, in a facial recognition classifier that outputs the person's identity in a facial image, the attacker's goal is to reconstruct the facial image of a person.

Training class inference: This kind of attack aims at recovering a semantically meaningful data for each training class of a trained classifier. In the above-mentioned facial recognition example, the attack goal is to recover a recognizable facial image of an arbitrary person (class) in the training data. This attack can be achieved by inverting the classifier to generate a representative sample which is classified as the wanted class [5].

However, inversion in adversarial settings is different from existing model inversion settings. Specifically, in adversarial settings, the attacker has no knowledge about the classifier's training data. Thus, previously known training-based methods that use the same training data to train the "inversion" model cannot be directly applied. Existing optimization-based

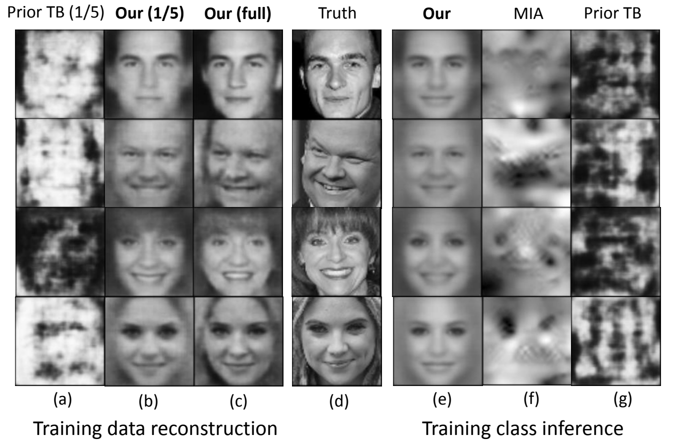


Fig. 1: Training data reconstruction and training class inference of our and previous approaches against a facial recognition classifier. Column (a) and (b) compare our approach with a known training-based (TB) method when only 1/5 prediction results are available. Column (c) shows our result of inverting full prediction vectors by training the inversion model on auxiliary samples. Column (e)-(g) compare our approach with MIA and the known prior TB method. Note that we use auxiliary samples for our approach in all experiments.

approaches require white-box accesses to the classifier to compute the gradients, and therefore they cannot work if the classifier is a blackbox. More importantly, the adversary might obtain only partial prediction results on victim data. For example, the ImageNet dataset [20] has over 20,000 classes. Since the majority values in the prediction vector are small, it is more practical to release only the top 5 to 10 predicted classes [21]. Victim users might also post partial predicted scores to the social media. The partial predictions largely limit the reconstruction ability of the inversion model. For example, Figure 1 column (a) shows the result of inverting a "truncated" prediction vector (i.e., keeping the 1/5 largest values while masking the rest to 0). The poor result is probably due to overfitting in the inversion model. Hence, although reconstruction is accurate on the full prediction vector, a slight deviation such as truncation in the prediction vector will lead to a dramatically different result.

In this paper, we formulate the adversarial settings, and propose an effective approach to invert neural networks in such settings. We adopt the above-mentioned training-based approach. That is, a second neural network (referred to as *inversion model*) is trained to perform the inversion. In contrast to previous work, instead of using the same distribution of the training data, in our setting, the training set (of the inversion model) is drawn from a more generic data distribution based on the background knowledge which is arguably much easier for the adversary to obtain. For instance, against a facial recognition classifier, the adversary could randomly crawl facial images from the Internet to compose an auxiliary set without knowing the exact training data (distribution). Figure 1 column (c) shows the reconstruction result of the inversion model trained on auxiliary samples, which clearly outperforms the result in column (a). We believe that this is because the auxiliary samples retain generic facial features (e.g., face edges, eyes and nose) and such information is sufficient to regularize the originally ill-posed inversion problem. Interestingly, our experiments show that even in cases where the actual training data is available, augmenting the training data with generic samples could improve inversion accuracy.

To make the inversion model also work when the adversary obtains only partial prediction result on victim user's data, we propose a truncation method of training the inversion model. The main idea is to feed the inversion model with truncated predictions (on the auxiliary samples) during training. Such truncation forces the inversion model to maximally reconstruct the samples based on the truncated predictions, and thus align the inversion model to the truncated values. Furthermore, truncation helps to reduce overfitting in a similar way of feature selection [22]. Figure 1 column (b) shows the result of inverting a partial prediction vector (i.e., keeping the 1/5 largest values while masking the rest to 0) using our truncation technique.

It turns out that the truncation technique is effective in our second problem on training class inference. To infer the training classes, we truncate the classifier's predictions on auxiliary samples to one-hot vectors. After the training is complete, it can produce a representative sample for each training class by taking one-hot vectors of the classes as input. Figure 1 column (e) shows the result of our training class inference. It significantly outperforms MIA (column (f)) and previous training-based approach (column (g)).

Depending on the adversary's role, we design two ways of training the inversion model. First, if the adversary is a user with no knowledge about the training data and has black-box access to the classifier (i.e., only prediction results are available to him), we construct the inversion model by training a separate model from sketch. Second, if the adversary is the developer who trains the classifier, the inversion model can be trained jointly with the classifier on the same training data, which derives a more precise inversion model. The objective is to force the classifier's predictions to also preserve essential information which helps the inversion model to reconstruct data. To this end, we use the reconstruction loss of the inversion model to regularize the training of the classifier.

We evaluate our approaches for various machine learning tasks and model architectures on multiple image datasets. The results show that our attack can precisely reconstruct data from partial predictions without knowing the training data distribution, and also outperforms previous approaches in training class inference. This work highlights that the rich information hidden in the model's prediction can be extracted, even if the adversary only has access to limited information. We hope for this work to contribute to strengthening user's awareness of handling derived data.

Contributions. In summary, we make the following contributions in this paper.

- We observe that even with partial knowledge of the training set, it is possible to exploit such background knowledge to regularize the originally ill-posed inversion problem.
- We formulate a practical scenario where the adversary obtains only partial prediction results, and propose a truncation method that aligns the inversion model to the space of the truncated predictions.
- Our inversion method can be adopted for training class inference attack. Experimental results show that our method outperforms existing work on training class inference.
- We observe that even if the classifier's training set is available to the adversary, augmenting it with generic samples could improve inversion accuracy of the inversion model.

II. Background

A. Machine Learning

In this paper, we focus on supervised learning, more specifically, on training classification models (classifiers) using neural networks [23]. The model is used to give predictions to input data. The lifecycle of a model typically consists of *training* phase (where the model is created) and *inference* phase (where the model is released for use).

Machine learning models. A machine learning classifier encodes a general hypothesis function F_w (with parameters w) which is learned from a training dataset with the goal of making predictions on unseen data. The input of the function F_w is a data point $\mathbf{x} \in \mathbb{R}^d$ drawn from a data distribution $p_x(\mathbf{x})$ in a d-dimensional space $\mathcal{X}$, where each dimension represents one attribute (feature) of the data point. The output of F_w is a predicted point $F_w(\mathbf{x})$ in a k-dimensional space $\mathcal{Y}$, where each dimension corresponds to a predefined class. The learning objective is to find the relation between each input data and the class as a function $F_w : \mathcal{X} \mapsto \mathcal{Y}$. A neural network (deep learning model) consists of multiple connected layers of basic non-linear activation functions (neurons) whose connections are weighted by the model parameter w, with a normalized exponential function $\mathsf{softmax}(\mathbf{z})_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$ added to the activation signals (referred to as *logits*) of the last layer. This function converts arbitrary values into a vector of real values in $[0, 1]$ that sum up to 1. Thus, the output could be interpreted as the probability that the input falls into each class.

Training phase. Let $\mathbf{x}$ represent the data drawn from the underlying data distribution $p_x(\mathbf{x})$, and $\mathbf{y}$ be the vectorized class of $\mathbf{x}$. The training goal is to find a function F_w to well approximate the mapping between every data point $(\mathbf{x}, \mathbf{y})$ in space $\mathcal{X} \times \mathcal{Y}$. To this end, we use a loss function $\mathcal{L}(F_w(\mathbf{x}), \mathbf{y})$ to measure the difference between the class $\mathbf{y}$ and the classifier's prediction $F_w(\mathbf{x})$. Formally, the training objective is to find a function F_w which minimizes the expected loss.

$$L(F_w) = \mathbb{E}_{\mathbf{x} \sim p_x}[\mathcal{L}(F_w(\mathbf{x}), \mathbf{y})] \tag{1}$$

The actual probability function $p_x(\mathbf{x})$ is intractable to accurately represent, but in practice, we can estimate it using samples drawn from it. These samples compose the training set $D \subset \mathcal{X}$. We predefine a class $\mathbf{y}$ for each data $\mathbf{x} \in D$ as supervision in the training process. Hence, we can train the model to minimize the empirical loss over the training set D.

$$L_D(F_w) = \frac{1}{|D|} \sum_{\mathbf{x} \in D} \mathcal{L}(F_w(\mathbf{x}), \mathbf{y}) \tag{2}$$

Nonetheless, this objective could lead to an overfitted model which attains a very low prediction error on its training data, but fails to generalize well for unseen data drawn from $p_x(\mathbf{x})$. A regularization term $R(F_w)$ is usually added to the loss $L_D(F_w)$ to prevent F_w from overfitting on the training data. In summary, the training process of a classifier is to find a model F_w that minimizes the following objective:

$$\mathcal{C}(F_w) = L_D(F_w) + \lambda R(F_w) \tag{3}$$

where the regularization factor λ controls the balance between the classification function and the regularization function.

Algorithms used for solving this optimization problem are variants of the gradient descent algorithm [24]. Stochastic gradient descent (SGD) [25] is a very efficient method that updates the parameters by gradually computing the average gradient on small randomly selected subsets (mini-batches) of the training data.

Inference phase. In the inference phase (or called testing phase), the model F_w is used to classify unseen data. Specifically, function F_w takes any data $\mathbf{x}$ drawn from the same data distribution $p_x(\mathbf{x})$ as input, and outputs a prediction vector $F_w(\mathbf{x}) = (F_w(\mathbf{x})_1, ..., F_w(\mathbf{x})_k)$, where $F_w(\mathbf{x})_i$ is the probability of the data $\mathbf{x}$ belonging to class i and $\sum_i F_w(\mathbf{x})_i = 1$.

B. Model Inversion

Our approach is related to many previous work on inverting neural networks from machine learning and computer vision communities. Inverting a neural network helps in understanding and interpreting the model's behavior and feature representations. For example, a typical inversion problem in computer vision is to reconstruct an image $\mathbf{x}$ from its computer vision features such as HOG [26] and SIFT [27], or from the activations in each layer of the network including the classifier's prediction $F_w(\mathbf{x})$ on it. In general, these inversion approaches fall into two categories: optimization-based inversion [5], [6], [7], [8], [9], [10], [11] and, most similar to our approach, training-based inversion [15], [16], [17], [18].

Optimization-based inversion. The basic idea of this branch of work is to apply gradient-based optimization in the input space $\mathcal{X}$ to find an image $\hat{\mathbf{x}}$ whose prediction approximates a given $F_w(\mathbf{x})$. To this end, the image $\hat{\mathbf{x}}$ should minimize some loss function between $F_w(\mathbf{x})$ and $F_w(\hat{\mathbf{x}})$. However, inverting the prediction of a neural network is actually a difficult ill-posed problem [17]. The optimization process tends to produce images that do not really resemble natural images especially for a large neural network [28]. To mitigate this issue, a series of studies have investigated using a natural image prior $P(\hat{\mathbf{x}})$ to regularize the optimization. The prior defines some statistics of the image. Formally, the inversion is to find an $\hat{\mathbf{x}}$ which minimizes the following loss function.

$$\mathcal{O}(\hat{\mathbf{x}}) = \mathcal{L}(F_w(\hat{\mathbf{x}}), F_w(\mathbf{x})) + P(\hat{\mathbf{x}}) \tag{4}$$

where $\mathcal{L}$ is some distance metric such as L2 distance. A special case of such inversion is to generate a representative image for some class y, by replacing $F_w(\mathbf{x})$ with vectorized y in this equation [5], [29]. Optimization-based inversion requires white-box access to the model to compute the gradients.

Various kinds of image priors have been investigated in the literature. For instances, a common prior is α-norm $P_\alpha(\mathbf{x}) = ||\mathbf{x}||_\alpha^\alpha$, which encourages the recovered image to have a small norm. Simonyan et al. [29] use L2 norm while Mahendran and Vedaldi [11] demonstrate that a relatively large α produces better result. They choose L6 norm in their experiments. Mahendran and Vedaldi [11] also investigate using total variation (TV) $P_{V^\beta}(\mathbf{x})$ for the image prior, which can encourage images to have piece-wise constant patches. It is defined in the following.

$$P_{V^\beta}(\mathbf{x}) = \sum_{i,j} \left((x_{i,j+1} - x_{i,j})^2 + (x_{i+1,j} - x_{i,j})^2 \right)^{\beta/2} \tag{5}$$

Besides, randomly shifting the input image before feeding it to the network is also used to regularize the optimization in recent work [30]. In addition, Yosinski et al. [28] investigate a combination of three other priors. Gaussian blur is used to penalize high frequency information in the image. They clip pixels with small norms to preserve only the main object in the image. Besides, they also clip pixels with small contribution to the activation. The contribution measures how much the activation increases or decreases when setting the pixel to 0. Fredrikson et al. [5] adopt denoising and sharpening filters as the prior in the model inversion attack (MIA). MIA targets at generating a representative image for training classes.

However, the simple hand-designed prior P in these approaches is limited. It cannot really capture the semantic information in the training data space, so the reconstruction quality especially against large networks is not satisfactory. Furthermore, this approach involves optimization at the test time because it requires computing gradients which makes it relatively slow (e.g., 6s per image on a GPU [11]).

Training-based inversion. This kind of inversion trains another neural network G_θ (referred to as *inversion model* in this paper) to invert the original one F_w. Specifically, given the *same* training set of images and their predictions $(F_w(\mathbf{x}), \mathbf{x})$, it learns a second neural network G_θ from sketch to well approximate the mapping between predictions and images (i.e., the inverse mapping of F_w). The inversion model G_θ takes the prediction $F_w(\mathbf{x})$ as input and outputs an image. Formally, this kind of inversion is to find a model g which minimizes the following objective.

$$\mathcal{C}(G_\theta) = \mathbb{E}_{\mathbf{x}\sim p_x}[\mathcal{R}(G_\theta(F_w(\mathbf{x})), \mathbf{x})] \tag{6}$$

where $\mathcal{R}$ is the image reconstruction loss such as L2 loss adopted in work [16]. Dosovitskiy and Brox [17] investigate using two additional loss items to regularize the training of the inversion model: loss in feature space and adversarial loss. Specifically, the loss in feature space encourages the reconstructed image to preserve perceptually important image features. The features, for example, can be the activations in some layer of the inversion model. The adversarial loss is added to keep the reconstructions realistic. Following the spirit of Generative Adversarial Networks (GANs) [31], it uses a discriminator D_ϕ to discriminate the reconstructed images from real ones, while the inversion model is trained to “fool” the discriminator into classifying the reconstructed images as real. The adversarial loss is defined in the following.

$$\begin{aligned} \mathcal{A}(G_\theta, D_\phi) =& \mathbb{E}_{\mathbf{x}\sim p_x}[\log(D_\phi(\mathbf{x}))] \\ &+ \mathbb{E}_{\mathbf{x}\sim p_x}[\log(1 - D_\phi(G_\theta(F_w(\mathbf{x}))))] \end{aligned} \tag{7}$$

In contrast to optimization-based inversion, training-based inversion is only costly during training the inversion model which is one-time effort. Reconstruction from a given prediction requires just one single forward pass through the network (e.g., 5ms per image on a GPU [16]).

III. Adversarial Model Inversion

The adversary could be either the user of a blackbox classifier F_w, or the developer of F_w. The adversary's capabilities and goals differ depending on the role and we consider three scenarios: (1) A curious user who intends to reconstruct the victim's input data from the victim's truncated predication vector; (2) A curious user who intends to infer F_w's functionality; (3) A malicious developer who intends to build a F_w, which subsequently could help in reconstructing the victims' input from their truncated predication vectors.

A. (Scenario 1) Data reconstruction with blackbox classifier

In this scenario, the adversary is a curious user who has black-box accesses to a classifier F_w. That is, the adversary can adaptively feed input to F_w and obtain the output. The adversary does not know the classifier's training data (distribution), architecture and parameters. However, the adversary has some background knowledge on F_w. Specifically, the adversary know the following:

- Although the adversary does not know the actual training data that is used to train F_w, the adversary can draw samples from a more “generic” distribution p_a of the training data. For instance, suppose F_w is a face recognition classifier trained on faces of a few individuals, although the adversary does not know the faces of those individuals, the adversary knows that the training data are facial images and thus can draw many samples from a large pool of facial images. Intuitively, a distribution p_a is more generic than the original one if p_a is the distribution after some dimension reductions are applied on the original.
- The adversary knows the input format of F_w, since he knows the distribution p_a. The adversary also knows the output format of F_w. That is, the adversary know the dimension of the predication vector. This assumption is

reasonable because even though F_w may return selected prediction values, the adversary can still estimate the dimension of the prediction vector. For example, he can query F_w with a set of input data and collect distinct classes in the returned predictions as the dimension.

The classifier F_w is also used by many other benign users. We assume that the adversary has the capability to obtain $\mathbf{f}$, a *m-truncated* predication vector of $F_w(\mathbf{x})$ where $\mathbf{x}$ is the input of a victim user, and m is a predefined parameter determined by the victim. Given a prediction vector $\mathbf{g}$, we say that $\mathbf{f}$ is m-truncated, denoted as $\mathsf{trunc}_m(\mathbf{g})$, when all m largest values of $\mathbf{g}$ remain in $\mathbf{f}$, while the rest are truncated to zeros. For instance,

$$\mathsf{trunc}_2((0.6, 0.05, 0.06, 0.2, 0.09)) = (0.6, 0, 0, 0.2, 0).$$

Now, given $\mathbf{f}$, black-box access to F_w, and samples from distribution p_a, the adversary wants to find a most probable data from the distribution p_a such that $\mathsf{trunc}(F_w(\mathbf{x})) = \mathbf{f}$. That is, ideally, the adversary wants to find a $\hat{\mathbf{x}}$ that satisfies the following:

$$\begin{aligned} \hat{\mathbf{x}} &= \arg\max_{\mathbf{x}\in X_{\mathbf{f}}} p_a(\mathbf{x}) \\ \text{subject to} \quad & X_{\mathbf{f}} = \{\mathbf{x} \in \mathcal{X} \mid \mathsf{trunc}(F_w(\mathbf{x})) = \mathbf{f}\} \end{aligned} \tag{8}$$

Let us call the problem of obtaining $\hat{\mathbf{x}}$ from F_w, $\mathbf{f}$ and p_a the *data reconstruction* problem.

B. (Scenario 2) Training class inference

Same as in scenario 1, in this scenario, the adversary is a curious user who has black-box accesses to a classifier F_w, and knows samples from a generic distribution p_a.

Instead of data reconstruction, the adversary wants to find a representative data of the training class. Given black-box access to a classifier F_w, and a target class y, the adversary wants to find a data $\hat{\mathbf{x}}$ that satisfies the following.

$$\begin{aligned} \hat{\mathbf{x}} &= \arg\max_{\mathbf{x}\in X_y} p_a(\mathbf{x}) \\ \text{subject to} \quad & X_y = \{\mathbf{x} \in \mathcal{X} \mid F_w(\mathbf{x})_y \text{ is high}\} \end{aligned} \tag{9}$$

where $F_w(\mathbf{x})_y$ is the confidence that $\mathbf{x}$ is classified by F_w as class y.

C. (Scenario 3) Joint classifier and model inversion

We also consider the scenario where the adversary is a malicious developer who trains the classifier F_w and sells/distributes it to users. Different from the previous two scenarios, here, the adversary has full knowledge about the classifier's training data, architecture and parameters, and has the freedom to decide what F_w would be. We assume that after F_w is released, the adversary is able to obtain truncated predication from the users, and the adversary wants to reconstruct the users' input.

Hence, in this scenario, the adversary goal is to train a classifier F_w that meets the accuracy requirement (w.r.t. the original classifier's task), while improving quality of data reconstruction.

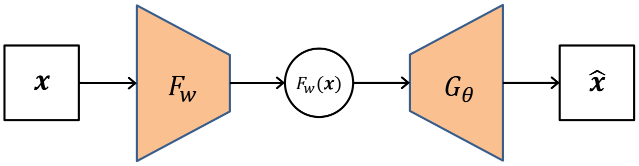


Fig. 2: Framework of training-based inversion approach. The classifier F_w takes data $\mathbf{x}$ as input and produces a prediction vector $F_w(\mathbf{x})$. The inversion model G_θ outputs the reconstructed data $\hat{\mathbf{x}}$ with the prediction as input.

D. Applying Prior Work in Adversarial Settings

Let us highlight the difference in the adversary's capabilities between our adversarial settings and previous inversion settings.

First, in our adversarial settings, when the adversary is a user, if he has only black-box accesses to the classifier, existing optimization-based inversion approaches including MIA do not work because they require white-box access to compute the gradients. Besides, many studies have shown that MIA against large neural networks tends to produce semantically meaningless images that do not even resemble natural images [13], [14]. Our experimental result, as shown in Figure 1 column (f), also reaches the same conclusion. Second, the adversary as a user has no knowledge about the classifier's training data (distribution), which makes previous training-based approaches impossible to train the inversion model on the same training data. Third, the adversary, as either a user or developer, might obtain truncated prediction results. This makes previous inversion models that are trained on full predictions ineffective in reconstructing data from partial predictions, as shown in Figure 1 column (a) and (g).

IV. Approach

Our approach adopts previously mentioned training-based strategy to invert the classifier. The overall framework is shown in Figure 2. Unlike the optimization-based approaches that invert a given prediction vector directly from F_w, here, an inversion model G_θ is first trained, which later takes the given prediction vector as input and outputs the reconstructed sample. This is similar to autoencoder [32] where F_w is the "encoder", G_θ is the "decoder", and the prediction is the latent space.

There are three aspects that are different from autoencoder, which we will elaborate in this section. (1) When F_w is fixed (Scenario 1&2), we need training data to train G_θ as we highlight earlier that the adversary does not have the data that trains F_w. Section IV-A describes how we obtain the training set for G_θ. (2) Since our adversary only obtains truncated prediction, we need a method to "realign" the latent space. Section IV-B gives the proposed truncation method. (3) In our Scenario 3, the F_w is not fixed and the adversary can decide F_w. In this case, this is a joint classifier and inversion model problem. Section IV-C gives our training method.

A. G_θ's Training Data: Auxiliary Set

The first important component in the construction of G_θ is its training set, which is referred to as *auxiliary set* in the rest of the paper. The auxiliary set should contain sufficient semantic information to regularize the ill-posed inversion problem.

We compose the auxiliary set by drawing samples from a more generic data distribution p_a than the original training data distribution p_x. For example, against a facial recognition classifier of 1,000 individuals, the auxiliary samples could be composed by collecting public facial images of random individuals from the Internet. These auxiliary samples still retain general facial features such as the face edges and locations of eyes and nose, which are shared semantic features of the original training data. We believe that such shared features provide sufficient information to regularize the originally ill-posed inversion task. To further improve the reconstruction quality of G_θ, we can specially choose the auxiliary set to better align the inversion model. For example, against a facial recognition classifier, we choose the dataset with mostly frontal faces as the auxiliary set, so as to align the inversion model to frontal faces.

Our experimental results demonstrate the effectiveness of sampling auxiliary set from a more generic data distribution, as shown in Section V-B. The inversion model can precisely reconstruct the training data points even if it never sees the training classes during the construction.

B. Truncation Method for Model Inversion

We propose a truncation method of training G_θ to make it aligned to the partial prediction results. Figure 3 shows the architecture of the classifier F_w and its inversion model G_θ. The idea is to truncate the classifier's predictions on auxiliary samples to the same dimension of the partial prediction vector on victim user's data, and use them as input features to train the inversion model, such that it is forced to maximally reconstruct input data from the truncated predictions. Formally, let $\mathbf{a}$ be a sample drawn from p_a, and $F_w(\mathbf{a})$ be the classifier's prediction. Let m be the dimension of the partial prediction vectors on victim's data. We truncate the prediction vector $F_w(\mathbf{a})$ to m dimensions (i.e., preserving the top m scores but setting the rest to 0). The inversion model G_θ is trained to minimize the following objective.

$$\mathcal{C}(G_\theta) = \mathbb{E}_{\mathbf{a}\sim p_a}[\mathcal{R}(G_\theta(\mathsf{trunc}_m(F_w(\mathbf{a}))), \mathbf{a})] \qquad (10)$$

where $\mathcal{R}$ is the loss function and we use L2 norm as $\mathcal{R}$ in this paper. Note that, it is also possible to add other loss functions such as adversarial loss in function 7 and TV loss in function 5 to make the generated data more realistic and natural. We leave it as future work. The truncation process can be understood as a similar way of feature selection [22] by removing unimportant classes in $F_w(\mathbf{a})$ (i.e., those with small confidence). It helps to reduce overfitting of G_θ such that it can still reconstruct the input data from the preserved important classes.

After G_θ is trained, the adversary can feed a truncated prediction $F_w(\mathbf{x})$ to G_θ and obtain the reconstructed $\mathbf{x}$.

Our inversion model G_θ can be adopted to perform training class inference (Scenario 2), which is the same adversarial goal in MIA [5]. Training class inference can be viewed as setting $m = 1$, which means the adversary knows only the class information and targets at generating a representative sample of each training class. MIA assumes that the adversary has a white-box access to F_w in the inference phase. Our method, on the contrary, works with a black-box access to F_w. Besides, our method can even work in the case that F_w releases only the largest predicted class with confidence value, because we can approximate the total number of training classes by collecting distinct classes from the classifier's predictions on our auxiliary set. This greatly reduces the requirement of the adversary's capability to infer training classes.

Our experimental results show that the truncation method of training the inversion model improves the reconstruction quality from truncated prediction. Previous training-based approach that directly inputs the partial prediction to the inversion model produces meaningless reconstruction results (see more evaluation details in Section V-C).

C. Joint Training of Classifier and Inversion Model

When the adversary is the developer of the classifier, he could alternatively jointly train the inversion model with the classifier, which leads to better inversion quality. In particular, let D be the classifier's training data. We regularize the classification loss $L_D(F_w)$ (Equation 2) with an additional reconstruction loss $R_D(F_w, G_\theta)$. Intuitively, this encourages the classifier's predictions to also preserve essential information of input data in the latent space, such that the inversion model can well decode it to recover input data. In this paper, we use L2 norm as the reconstruction loss $R_D(F_w, G_\theta)$.

$$R_D(F_w, G_\theta) = \frac{1}{|D|}\sum_{\mathbf{x}\in D} ||(G_\theta(\mathsf{truc}(F_w(\mathbf{x}))) - \mathbf{x}||_2^2 \qquad (11)$$

The joint training ensures that F_w gets updated to fit the classification task, and meanwhile, G_θ is optimized to reconstruct data from truncated prediction vectors.

It is worth noting that in training G_θ, we find that directly using prediction vector $F_w(\mathbf{x})$ does not produce optimal inversion results. This is because the logits $\mathbf{z}$ at the output layer are scaled in $[0, 1]$ (and sum up to 1) to form prediction $F_w(\mathbf{x})$ by the $\mathsf{softmax}$ function (see Section II-A), which actually weakens the activations of the output layer and thus loses some information for later decoding. To address this issue, we rescale the prediction $F_w(\mathbf{x})$ to its corresponding logits $\mathbf{z}$ in the following and use these $\mathbf{z}$ to replace the prediction $F_w(\mathbf{x})$ in our experiments.

$$\mathbf{z} = \log(F_w(\mathbf{x})) + c \qquad (12)$$

where c is a constant and is also optimized during training the inversion model.

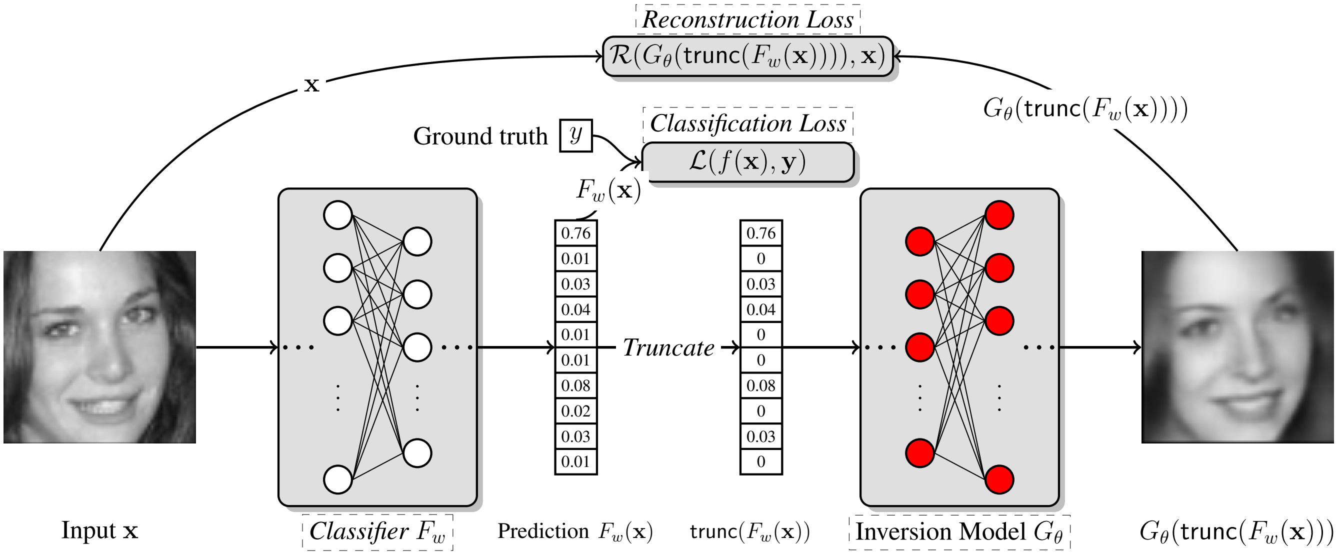

Reconstruction Loss
$\mathcal{R}(G_\theta(\mathsf{trunc}(F_w(\mathbf{x}))), \mathbf{x})$
$\mathbf{x}$
$G_\theta(\mathsf{trunc}(F_w(\mathbf{x})))$
Ground truth y
Classification Loss
$\mathcal{L}(f(\mathbf{x}), \mathbf{y})$
$F_w(\mathbf{x})$
0.76
0.01
0.03
0.04
0.01
0.01
0.08
0.02
0.03
0.01
Truncate
0.76
0
0.03
0.04
0
0
0.08
0
0.03
0
Input $\mathbf{x}$
Classifier F_w
Prediction $F_w(\mathbf{x})$
$\mathsf{trunc}(F_w(\mathbf{x}))$
Inversion Model G_θ
$G_\theta(\mathsf{trunc}(F_w(\mathbf{x})))$

Fig. 3: Architecture of the classifier and inversion model. The classifier F_w takes data $\mathbf{x}$ as input and produces a prediction $F_w(\mathbf{x})$. Such prediction vector is truncated (if necessary) to a new vector $\mathsf{trunc}(F_w(\mathbf{x}))$. The inversion model G_θ takes the truncated prediction as input and outputs a reconstructed data $G_\theta(\mathsf{trunc}(F_w(\mathbf{x})))$.

TABLE I: Data allocation of the classifier F_w and its inversion model G_θ.

Classifier F_w		Inversion Model G_θ	
Task	Data	Auxiliary Data	Distribution
FaceScrub	50% train, 50% test	FaceScrub 50% test data	Same
	80% train, 20% test	CelebA	Generic
	80% train, 20% test	CIFAR10	Distinct
MNIST	50% train, 50% test	MNIST 50% test data	Same
	80% train, 20% test (5 labels)	MNIST other 5 labels	Generic
	80% train, 20% test	CIFAR10	Distinct

V. Experiments

In this section, we evaluate the inversion performance of our method and compare it with existing work. We first introduce the experimental setup. We then evaluate three factors: the choice of auxiliary set, the truncation method and the full prediction size. We next evaluate our method in training class inference attack. Finally, we compare the inversion performance of the inversion model G_θ trained with the blackbox classifier F_w and G_θ trained jointly with F_w.

A. Experimental Setup

We perform evaluation on four benchmark image recognition datasets. For simplicity, all datasets are transformed to greyscale images with each pixel value in $[0, 1]$. We detail each dataset in the following.

- **FaceScrub [33].** A dataset of URLs for 100,000 images of 530 individuals. We were only able to download 48,579 images for 530 individuals because not all URLs are available during the period of writing. We extract the face of each image according to the official bounding box information. Each image is resized to 64×64.
- **CelebA [34].** A dataset with 202,599 images of 10,177 celebrities from the Internet. We remove 296 celebrities which are included in FaceScrub and eventally we obtain 195,727 images of 9,881 celebrities. This makes sure that our modified CelebA has no class intersection with FaceScrub. This large-scale facial image dataset can represent the generic data distribution of human faces. To extract the face of each image, we further crop the official align-cropped version (size 178×218) by width and height of 108 with upper left coordinate $(35, 70)$. Each image is resized to 64×64 and also resized to 32×32.
- **CIFAR10 [35].** A dataset consists of 60,000 images in 10 classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck). Each image is resized to 64×64 and also resized to 32×32.
- **MNIST [36].** A dataset composed of 70,000 handwritten digit images in 10 classes. Each image is resized to 32×32.

We use FaceScrub and MNIST to train the target classifier F_w. For each F_w that is trained using the dataset D, we separately train the inversion model G_θ using the same training set D, training set drawn from a more generic distribution of D, and training set drawn from distribution that is arguably semantically different from D. We call these three types of training data *same*, *generic* and *distinct* respectively. Table I presents the data allocation for each F_w and its corresponding G_θ. Note that the auxiliary set of generic data distribution has no class intersection with the training data of F_w (i.e., FaceScrub and MNIST). Specifically, we have cleaned CelebA by removing the celebrities which are also included in the FaceScrub dataset. We randomly select 5 classes of MNIST to train F_w and use the rest 5 classes to form the auxiliary set. The auxiliary set of distinct data distribution is composed of samples drawn from an explicitly different data distribution from the training data.

We use CNN to train F_w, and use transposed CNN to train

G_θ. The FaceScrub classifier includes 4 CNN blocks where each block consists of a convolutional layer followed by a batch normalization layer, a max-pooling layer and a ReLU activation layer. Two fully-connected layers are added after the CNN blocks, and Softmax function is added to the last layer to convert arbitrary neural signals into a vector of real values in $[0, 1]$ that sum up to 1. The FaceScrub inversion model consists of 5 transposed CNN blocks. The first 4 blocks each has a transposed convolutional layer succeeded by a batch normalization layer and a Tanh activation function. The last block has a transposed convolutional layer followed by a Sigmoid activation function which converts neural signals into real values in $[0, 1]$ which is the same range of auxiliary data. The MNIST classifier and inversion model have similar architecture as FaceScrub but with 3 CNN blocks in classifier and 4 transposed CNN blocks in inversion model. More details of model architectures are presented in Appendix A. The FaceScrub classifier and MNIST classifier achieve 85.7% and 99.6% accuracy on their test set respectively (80% train, 20% test), which are comparable to the state-of-the-art classification performance.

B. *Effect of Auxiliary Set*

The auxiliary set is an important factor that decides the inversion quality of G_θ. We evaluate our inversion model G_θ by training it on different auxiliary sets (i.e., same, generic and distinct) with the blackbox F_w. We use G_θ to perform data reconstruction of F_w's training data and test data (Scenario 1). Note that in this experiment, we assume that the adversary is given full prediction values.

We present the inversion results on randomly-chosen training data and test data of the F_w in Figure 4 and 5. It is clear to see that the closer the auxiliary set's data distribution is to the training data distribution, the better the inversion quality is. It is worth noting that the auxiliary set of the generic data distribution has no class intersection with the classifier's training data, which means the inversion model has never seen the training classes during its construction, but it can still accurately reconstruct the data from these classes. For example, digit 5,6,7 and 9 in MNIST are not included in the auxiliary set, yet G_θ can reconstruct these digits fairly accurately by capturing as many semantic features of these digits as possible such that they are visually like these digits. This further demonstrates that the generic background knowledge could be sufficient in regularizing the ill-posed inversion. However, if the auxiliary set's data distribution is too far from the generic data distribution, the inversion is not that satisfactory as shown in the 3rd row of Figure 4 and 5. We believe that it is because the classifier cannot fully extract the semantic features from these auxiliary samples such that its predictions on them do not provide sufficient information for the inversion model to decode.

Summary I: Even with no full knowledge about the classifier F_w's training data, accurate inversion is still possible by training G_θ using auxiliary samples drawn from a more generic distribution derived from background knowledge. Such auxiliary set is arguably easier to obtain.

C. *Effect of Truncation*

The truncation method of training the inversion model largely increases the inversion quality in cases where the adversary obtains only partial prediction results on victim user's data. Let us denote the dimension of the partial prediction that the adversary obtains on victim user's data as m. We perform the inversion attack (Scenario 1) with $m = 10, 50, 100$ and 530 against FaceScrub classifier and with $m = 3, 5$ and 10 against MNIST classifier. We present the inversion results of our truncation method and previous training-based method (i.e., without truncation) on FaceScrub classifier in Figure 6 and on MNIST classifier in Figure 7. The auxiliary set is composed of samples from the generic data distribution for FaceScrub classifier and from the same data distribution for MNIST classifier. The presented training data and test data are randomly chosen.

We can see that our approach outperforms previous approach especially when the adversary is given partial prediction results. Our approach can produce highly recognizable images with sufficient semantic information for all m. Previous approach, although can generate recognizable images when m is very large, produces meaningless result when m is small. For our approach, when m is relatively small, it appears that the inversion result is more a generic face of the target person with facial details not fully represented. For example, in the 7th column of Figure 6 where the ground truth is a side face, our inversion result is a frontal face when $m = 10$ and 50, and becomes a side face when $m = 100$. This result demonstrates that truncation indeed helps reduce overfitting by aligning G_θ to frontal facial images, and with smaller m, more generalization is observed.

Summary II: Our truncation method of training the inversion model G_θ makes it still possible to perform accurate inversion when the adversary is given only partial prediction results.

D. *Effect of Full Prediction Size*

Section V-C investigates the effect of m, the size of the truncated prediction. In this section, we investigate the effect of the size of the full prediction. Let k be the dimension of the full prediction. In our experiments, we randomly select k classes from the FaceScrub and MNIST datasets as the training data of F_w. We set $k = 10, 50, 100$ and 530 for FaceScrub and $k = 3, 5$ and 10 for MNIST. We present the inversion results against FaceScrub classifier in Figure 8 and against MNIST classifier in Figure 9. In all experiments, we assume that the

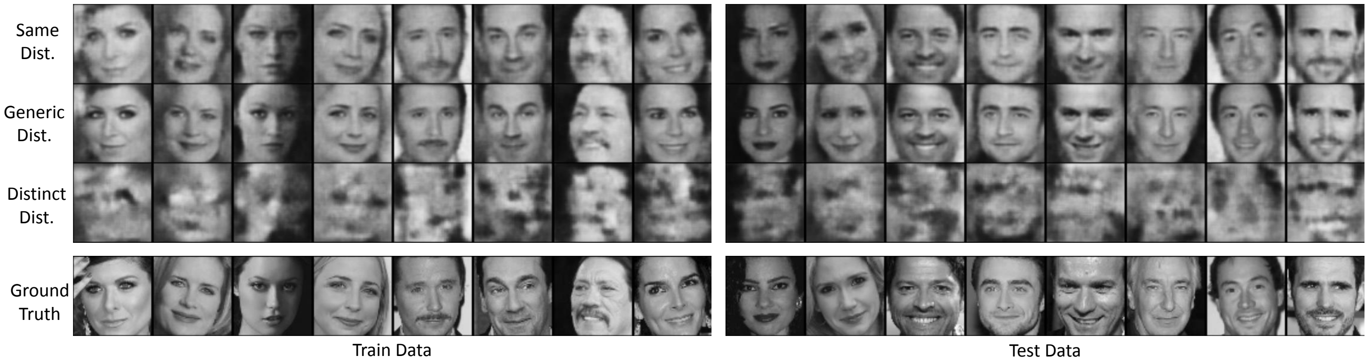


Fig. 4: Effect of the auxiliary set on the inversion quality. We use auxiliary set of the same (1st row), generic (2nd row) and distinct (3rd row) data distributions to train the inversion model. We perform inversion against the FaceScrub classifier F_w on the F_w's training data and test data.

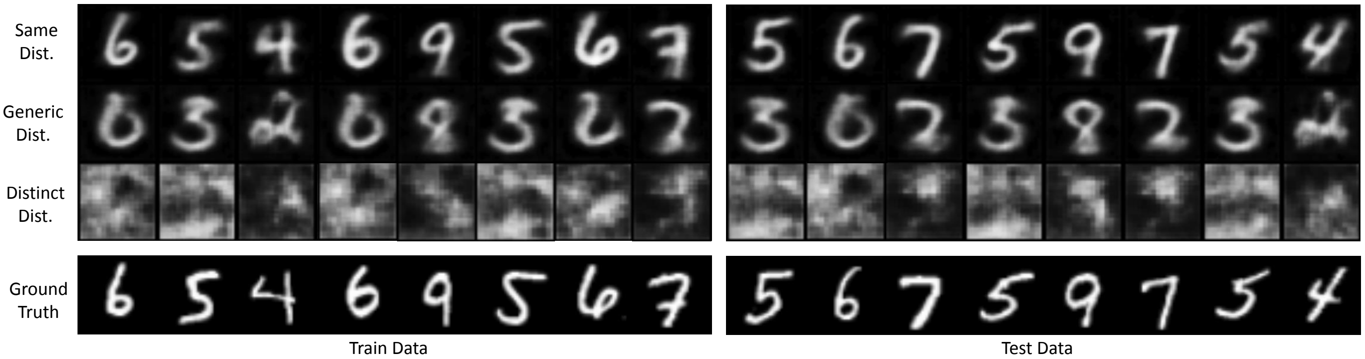


Fig. 5: Effect of the auxiliary set on the inversion quality. We use auxiliary set of the same (1st row), generic (2nd row) and distinct (3rd row) data distributions to train the inversion model. We perform inversion against the MNIST classifier F_w on the F_w's training data and test data.

adversary obtains the full prediction vectors (i.e., $m = k$). The auxiliary set is composed of samples from the generic data distribution for FaceScrub classifier and from the same data distribution for MNIST classifier. The presented training data and test data are randomly chosen.

Our experimental results show that the effect of k is similar with the effect of m that we have discussed. A larger k leads to a more accurate reconstruction of the target data and a small k leads to a semantically generic inversion of the corresponding class. This is because both k and m affect the amount of predicted information that the adversary can get. The factor k decides the size of the full prediction vector. The factor m decides how many predicted classes that the classifier releases from the k-dimension prediction vector.

E. Training Class Inference

This section evaluates our inversion model G_θ on training class inference attack (Scenario 2). We compare our method, MIA and a previous training-based approach (which has no truncation) against the same F_w trained on FaceScrub. For our attack, we use CelebA as the auxiliary set. It has no class intersection with FaceScrub. Hence, the inversion model G_θ trained on it has never seen the target training classes during its construction. In particular, we first use the auxiliary set to approximate the total number of training classes, and we obtain 530 classes which is actually the true number. During the training of the inversion model, we encode the classifier's prediction result on each auxiliary sample (i.e., the largest predicted class with confidence) to a 530-dimension vector by filling rest classes with zeros. We use the encoded predictions as features to train G_θ. After G_θ is trained, we convert each training class to a one-hot vector of 530 dimensions and feed the one-hot vectors to G_θ. The output of G_θ is the inferred image of the training class.

We present the inference results in Figure 10. We can see that the inferred images by MIA are semantically meaningless and do not even form recognizable faces. This is consistent with conclusions in previous research work [14], [13] where similar results were obtained. Our results demonstrate again that MIA is not effective when dealing with complicated network structures (i.e., CNN in our experiments). Previous training-based inversion also fail to produce recognizable facial images. Our method is able to generate highly human-recognizable faces for each person (class) by capturing se-

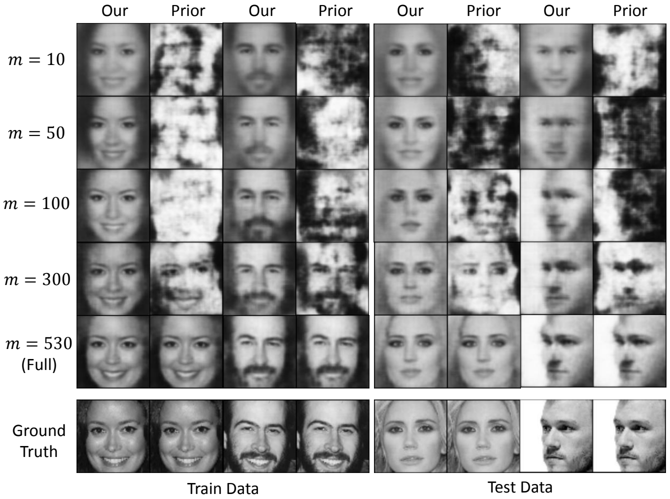


Fig. 6: Effect of truncation for the inversion model G_θ on the inversion quality. Inversion is performed when the adversary is given $10, 50, 100$ and 530 predicted classes (denoted as m in each row) from the classifier. Inversion results of our approach and previous approach are presented in odd and even columns respectively. We perform the inversion against FaceScrub classifier and present results on randomly-chosen training data and test data. The auxiliary set we use is drawn from generic data distribution.

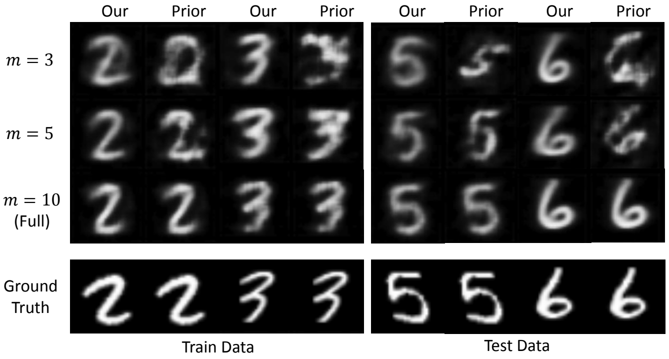


Fig. 7: Effect of truncation for the inversion model G_θ on the inversion quality. Inversion is performed when the adversary is given $3, 5$ and 10 predicted classes (denoted as m in each row) from the classifier. Inversion results of our approach and previous approach are presented in odd and even columns respectively. We perform the inversion against MNIST classifier and present results on randomly-chosen training data and test data. The auxiliary set we use is drawn from the same data distribution.

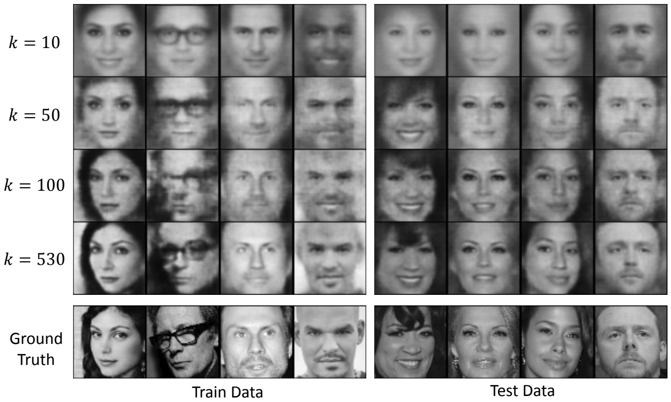


Fig. 8: Effect of the full prediction size on the inversion quality. Inversion is performed when the size of the full prediction vector is $10, 50, 100$ and 530 classes (denoted as k in each row). In each experiment, we assume the adversary obtains the full prediction vector. We perform the inversion against FaceScrub classifier and present results on randomly chosen training data and test data. The auxiliary set we use is drawn from generic data distribution.

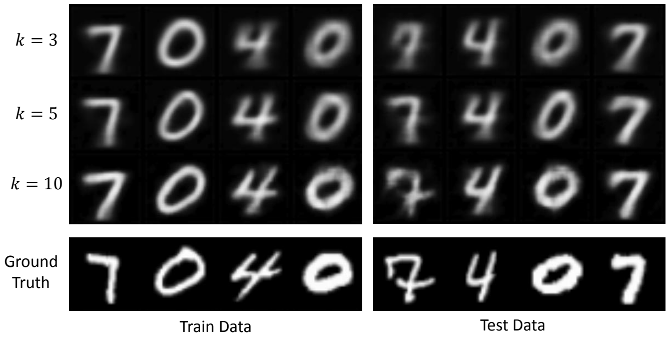


Fig. 9: Effect of the full prediction size on the inversion quality. Inversion is performed when the size of the full prediction vector is $3, 5$ and 10 classes (denoted as k in each row). In each experiment, we assume the adversary obtains the full prediction vector. We perform the inversion against MNIST classifier and present results on randomly chosen training data and test data. The auxiliary set we use is drawn from the same data distribution.

mantic features such as eyes, nose and beard. An interesting find is that our method consistently produces frontal faces even though the training facial images of F_w have various angles, expressions, background and even lightning. This demonstrates that the inversion model is well aligned to frontal faces and can capture the semantics of the training person's faces such that it can accurately infer the training classes.

Summary III: Our method outperforms previous methods in training class inference against complex neural networks. Our experimental results show that our method can generate highly recognizable and representative sample for training classes.

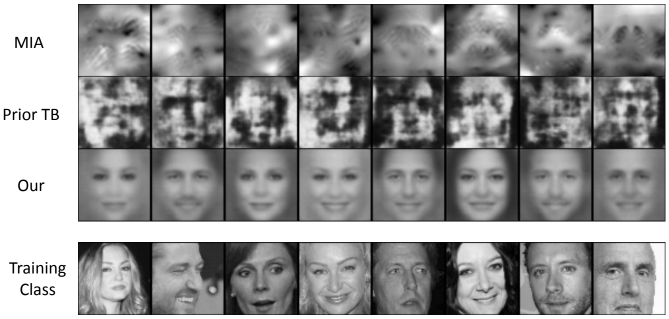


Fig. 10: Training class inference results. We present results of MIA (1st row), previous training-based (TB) inversion (2nd row) and our inversion (3rd row). Inversion is performed against FaceScrub classifier. We use auxiliary set of generic data distribution in our inversion.

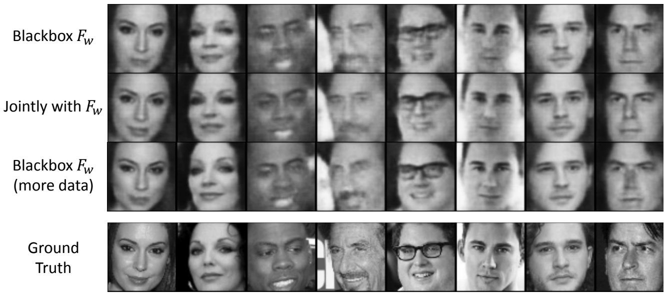


Fig. 11: Inversion results of the inversion model G_θ: trained with blackbox F_w vs trained jointly with F_w. The target F_w is FaceScrub. We perform the inversion on the test data. For training G_θ with the blackbox F_w, we use the same training data of F_w as the auxiliary set (1st row) and also use augmented training data with CelebA as the auxiliary set (3rd row). All the other training details of the two construction methods are the same.

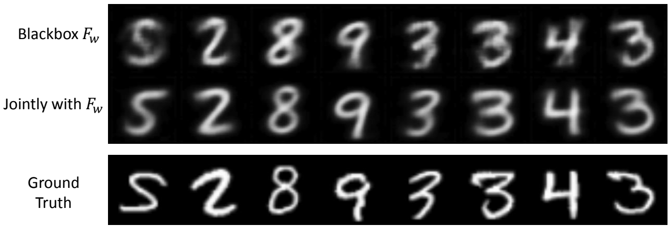


Fig. 12: Inversion results of the inversion model G_θ: trained with blackbox F_w vs trained jointly with F_w. The target F_w is MNIST. We perform the inversion on the test data. For training G_θ with the blackbox F_w, we use the same training data of F_w as the auxiliary set. All the other training details of the two construction methods are the same.

TABLE II: Classification accuracy of the classifier F_w and average reconstruction loss of the inversion model G_θ. Results are reported on test set.

Classifier	Acc.[1]	Acc.[2]	Acc.[3]	Recon.[1]	Recon.[2]	Recon.[3]
FaceScrub	78.3%	76.8%	78.3%	0.1072	0.0085	0.0083
MNIST	99.4%	99.2%	-	0.3120	0.0197	-

[1] We train F_w first and then train G_θ with black-box accesses to F_w.
[2] We jointly train F_w and G_θ
[3] We train F_w first and then train G_θ with black-box accesses to F_w. The training data is augmented with CelebA.

F. Inversion Model Construction: Trained with Blackbox F_w vs Trained Jointly with F_w

The inversion model G_θ could be trained with a fixed given blackbox F_w (as in Scenario 1&2) or trained jointly with F_w (as in Scenario 3). We evaluate and compare the inversion quality of G_θ and classification accuracy of F_w between the two construction methods. To make the construction method of G_θ as the only factor that affects the inversion performance, we use the same training data of F_w as the auxiliary set for the construction with blackbox F_w. All the other training details (e.g., epochs, batch size, optimizer) of the two construction methods are the same. We use G_θ to perform test data reconstruction against the FaceScrub and MNIST classifiers. In particular, we split a classification dataset to 50% as training data and the rest 50% as test data.

We present the inversion results of the two construction methods against FaceScrub and MNIST classifiers in Figure 11 and 12. We can see that constructing G_θ by jointly training it with F_w leads to a more accurate reconstruction of the test data than that trained with the blackbox F_w. To quantify the reconstruction quality, we present the average reconstruction loss of G_θ constructed by the two methods in the 5th and 6th columns of Table II. We can see that the joint training actually leads to a lower reconstruction loss on the test set. This result is intuitive, because the joint training can make F_w's prediction vectors also preserve essential information about the input data along with the classification information such that G_θ can well decode it for reconstruction. However, the better reconstruction quality is achieved at the cost of lower classification accuracy. We present the classification accuracy in the 2nd and 3rd columns of Table II. We can see that the classification accuracy does drop because of the joint training, but it is still within an acceptable range (0.2%-1.5%). Note that the classification accuracy, especially of FaceScrub (78.3%), is a little bit lower than the accuracy (85.7%) in earlier experiments because here we use 50% of the original set to train F_w which is less than previously 80% of the dataset.

Certainly, the adversary (who is the malicious developer in Scenario 3) can choose to abandon the joint training. That is, F_w is first trained to achieve high accuracy, and then the adversary trains the G_θ. Interestingly, the adversary can choose to use two different training sets, one for F_w and a different auxiliary set for G_θ. In this experiment, we augment the original training set with CelebA as the auxiliary set. We present the result in the 3rd row of Figure 11 and present the average reconstruction loss on the test set in the last column in Table II. We can see that by augmenting the training set with generic data, the inversion model G_θ trained with the blackbox F_w can achieve comparable reconstruction quality to G_θ trained jointly with F_w.

Summary IV: Inversion model G_θ trained jointly with the classifier F_w outperforms G_θ trained with blackbox F_w in inversion quality but at the cost of acceptable accuracy loss. Augmenting the auxiliary set with more generic data can improve G_θ trained with the blackbox F_w to be comparable to G_θ trained jointly with F_w, and maintain the accuracy.

VI. Related Work

While deep learning based systems are reported to attain very high accuracy in various applications [37], [38], there remains some limitations that hinder wide adoption of these systems. On the one hand, various studies have demonstrated that, similar to other data-driven applications, neural networks pose a threat to privacy [39], [13], [19], [12]. Besides, the fragility of neural networks is also reported in the presence of adversarial attacks [40], [41]. On the other hand, an important drawback of neural networks is their lack of explanation capability [42]. Thus, motivated either by security or interpretation reasons, many research efforts have been proposed in the literature.

A. ML Privacy

Researchers have strongly suggested that ML models suffer from various privacy threats to its training data [39], [13], [19], [12], [43]. For instance, given access to an ML model, an adversary can infer non-trivial and useful information about its training set [39].

Shokri et al. [13] study membership inference attacks against ML models, wherein an adversary attempts to learn if a record of his choice is part of the private training sets. Fredrikson et al. [5] study a model inversion attack to infer a representative sample for a target training class. Fredrikson et al. [19] also propose another attack to infer sensitive attributes of training data from a released model. Hidano et al. [44] further propose to infer sensitive attributes in a scenario where knowledge of the non-sensitive attributes is not necessarily provided. Wu et al. [45] propose a methodology to formalize such attribute inference attacks. Giuseppe et al. [46] show that an adversary could infer the statistical information about the training dataset in the inference phase of the target model. Similarly, Hitaj et al. [14] investigate information leakage from collaborative learning but in the training phase. Melis et al. [47] investigate membership inference and property inference against collaborative ML during its training. Wang et al. [48] study user-level privacy leakage against collaborative ML also during its training process. Song et al. [49] embed sensitive information about several training data points into neural networks in its training phase by exploiting the large capacity of neural networks or massive unused capacity of model's parameter space. This enables the adversary to extract specific data points in the latter inference phase.

Some of the above mentioned attacks on training data privacy infer sensitive attributes or statistical information about the training data. Others can extract training data points but have to manipulate the training process of the model. Membership inference attack works in the inference phase but requires the data is given. Our work, on the other hands, examines how one can reconstruct either specific training data points or test data points from their prediction results in the inference phase.

B. ML in Adversarial Settings

Deep learning techniques, while achieving utmost accuracy in various application domains [38], [37], were not originally designed with built-in security. However, recent years have witnessed an ever increasing adoption of DL models for security-sensitive tasks, causing the accuracy of the models' predictions to have significant implications on the security of the host tasks. Various works have suggested that ML models are likely vulnerable in adversarial settings [50], [51], [40]. In particular, an adversary could force a victim model to deviate from its intended task and behave erratically according to the adversary's wish. The adversary can stage these attacks by corrupting the training phase (e.g., poisoning the training set with adversarial data augmentation [52], [53], employing adversarial loss function [51]), maliciously modifying the victim model [40], or feeding the victim model with adversarially crafted samples [54], [41].

In particular, Szegedy et al. [41] leverage constrained optimization algorithm to propose an evasion technique that confuses the victim model into misclassifying an adversarial sample which is obtained by introducing a minimal and visually imperceptible perturbation to a clean sample. Papernot et al. [55] further show that an adversarial sample that is crafted to target one victim model can also be effective in evading another victim model, suggesting a robustness of adversarial samples against different model configurations. While there exists various attempts to combat against adversarial samples (e.g., defensive distillation [56] or denoising autoencoder [57]), their effectiveness continue to be challenged by different attacks [40].

On an alternative direction, Liu et al. [53] study a trojaning attack on neural networks in which an adversary modifies a pre-trained model into a trojaned model. The trojaned model has similar performance with the pre-trained model on normal data, but will behave maliciously when presented with specific trojan trigger, which is a small stamp that is to be pasted on a clean sample. A related attack is introduced in BadNets [51], wherein an adversary poisons the training set with specially crafted malicious samples so as to "backdoor" the output model. The backdoored model attains good accuracy on clean data, but would behave badly when presented with specific attacker-chosen inputs.

Our focus, on the contrary, is not to deviate ML models from their intended tasks, but rather to investigate an possibility of reconstructing input data from the model's predictions on them.

C. Secure & Privacy-Preserving ML

In the wake of security and privacy threats posed to ML techniques, much research has been devoted to provisioning secure and privacy-preserving training of ML models [58], [59], [60], [61], [62]. For instances, Abadi et al. study a framework to train deep learning models that enjoy differential privacy. Shokri et al. [59] propose a protocol for privacy-preserving collaborative deep learning, enabling participants to jointly train a model without revealing their private training data. Following this work, Phong et al. [63] apply homomorphic encryption to further ensure the data privacy against an honest-but-curious server. Bonawitz et al. [64] present a solution for secure aggregation of high-dimensional data. In addition, systems for oblivious multi-party machine learning has also been built using trusted hardware primitive [60].

The threat models assumed by these techniques are to protect privacy of users' data contributed to the training set. Our work studies a different threat model wherein the adversary targets at reconstructing user data given the model's prediction results on them.

Some research work on protecting the predictions of ML models has also been proposed recently [61], [62]. For examples, Dwork and Feldman [61] study a method to make the model's predictions achieve differential privacy with respect to the training data. Our work, on the other hand, studies the mapping between the prediction vectors and the input data by training another inversion model. Juvekar et al. [62] propose Gazelle as a framework for secure neural network inference in the "machine-learning-as-a-service" setting using cryptographic tools such that the server cannot obtain the predictions. However, our work studies a setting where the user posts their prediction results (e.g., predicted scores of face beauty and dress sense) to social media which causes an exposure of the model's predictions.

D. ML Inversion for Interpretation

Although deep neural networks have demonstrated impressive performance in various applications, it remains not clear why they perform so well. Much research has been working on interpreting and understanding neural networks [28], [65], [29], [66], [67], [68]. Inverting a neural network is one important method to understand the representations learned by neural networks.

Simonyan et al [29] visualize a classification model by generating an image which maximizes the class score, and computing a class saliency map for a given image and class. Both methods are based on computing the gradient of the class score with respect to the input image. Zeiler and Fergus [69] propose the DeConvNet method to backtrack the network computations to identify parts of the image responsible for certain neural activations. They attach a deconvnet to each of the convnet layers so as to provide a propagation back to image pixels. Du et al. [70] explain a DNN-based prediction by a guided feature inversion framework to identify the contribution of each feature in the input as well as to inspect the information employed by the DNN for the prediction task. Their experimental results suggest that the higher layers of DNN do capture the high-level features of the input. Jacobsen et al. [71] propose an invertible neural network that preserves information about the input features in each of their intermediate representations at each layer. They replace the non-invertible components of the RevNet [72] by invertible ones. Gillbert et al. [73] focus on a theoretical explanation of the invertibility of CNN and propose a mathematical model to recover images from their sparse representations. However, they make several model assumptions for mathematical analysis, and there is still a gap between the mathematical model and the CNN used in practical scenarios. Mash et al. [67] train generative inversion models to interpret features learned by neural networks. They make use of autoregressive neural density models to express a distribution over input features given an intermediate representation.

The above mentioned research work inverts neural networks in order to understand and interpret them. Thus, the inversion can leverage all possible information of the model and the training data. Our work, on the contrary, inverts a neural network in the adversarial settings where the adversary's capability is limited.

VII. Conclusion

We proposed an effective model inversion approach in the adversary setting based on training an inversion model that acts as an inverse of the original model. We observed that even with no full knowledge about the original training data, an accurate inversion is still possible by training the inversion model on auxiliary samples drawn from a more generic data distribution. We proposed a truncation method of training the inversion model by using truncated predictions as input to it. The truncation helps align the inversion model to the partial predictions that the adversary might obtain on victim user's data. Our experimental results show that our approach can achieve accurate inversion in adversarial settings and outperform previous approaches.

The seemingly coarse information carried by the prediction results might lead to causal sharing of such information by users and developers. Our work, on the other hand, shows the surprising reconstruction accuracy of the inversion model when applied in adversarial settings. It is interesting in the future work to investigate how other loss functions, and other generative techniques such as Generative Adversarial Network can be incorporated in the adversarial inversion problem.

References

[1] Apple, "ARFaceAnchor.BlendShapeLocation," https://developer.apple.com/documentation/arkit/arfaceanchor/blendshapelocation.

[2] Microsoft, "How-Old.net," https://www.how-old.net/.

[3] Beauty.AI, "Beauty.AI," http://beauty.ai/.

[4] Microsoft, "Xiaoice," https://kan.msxiaobing.com/ImageGame/Portal.

[5] M. Fredrikson, S. Jha, and T. Ristenpart, "Model Inversion Attacks that Exploit Confidence Information and Basic Countermeasures," in *Proceedings of the 22nd {ACM} {SIGSAC} Conference on Computer and Communications Security, Denver, CO, USA, October 12-6, 2015*, 2015, pp. 1322–1333. [Online]. Available: http://doi.acm.org/10.1145/2810103.2813677

[6] S. Lee and R. M. Kil, "Inverse mapping of continuous functions using local and global information," *IEEE Transactions on Neural Networks*, vol. 5, no. 3, pp. 409–423, May 1994.

[7] Linden and Kindermann, "Inversion of multilayer nets," in *International 1989 Joint Conference on Neural Networks*, 1989, pp. 425–430 vol.2.

[8] B.-L. Lu, H. Kita, and Y. Nishikawa, "Inverting feedforward neural networks using linear and nonlinear programming," *IEEE Transactions on Neural Networks*, vol. 10, no. 6, pp. 1271–1290, Nov 1999.

[9] C. A. Jensen, R. D. Reed, R. J. Marks, M. A. El-Sharkawi, J.-B. Jung, R. T. Miyamoto, G. M. Anderson, and C. J. Eggen, "Inversion of feedforward neural networks: algorithms and applications," *Proceedings of the IEEE*, vol. 87, no. 9, pp. 1536–1549, sep 1999.

[10] A. R. Várkonyi-Kóczy, *Observer-Based Iterative Fuzzy and Neural Network Model Inversion for Measurement and Control Applications*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 681–702. [Online]. Available: https://doi.org/10.1007/978-3-642-03737-5{_}49

[11] A. Mahendran and A. Vedaldi, "Understanding deep image representations by inverting them," in *{IEEE} Conference on Computer Vision and Pattern Recognition, {CVPR} 2015, Boston, MA, USA, June 7-12, 2015*, 2015, pp. 5188–5196. [Online]. Available: https://doi.org/10.1109/CVPR.2015.7299155

[12] N. Papernot, P. Mcdaniel, and P. State, "Security and Privacy in Machine Learning," in *2018 IEEE European Symposium on Security and Privacy (EuroS\&P)*, vol. 392001, no. December, 2018, pp. 1–16. [Online]. Available: https://project.inria.fr/wifs2017/files/2017/12/WIFS{_}T2{_}Papernot.pdf

[13] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership Inference Attacks Against Machine Learning Models," in *2017 {IEEE} Symposium on Security and Privacy, {SP} 2017, San Jose, CA, USA, May 22-26, 2017*, 2017, pp. 3–18. [Online]. Available: https://doi.org/10.1109/SP.2017.41

[14] B. Hitaj, G. Ateniese, and F. Perez-Cruz, "Deep Models Under the GAN: Information Leakage from Collaborative Deep Learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, vol. 1, 2017, pp. 603–618. [Online]. Available: http://arxiv.org/abs/1702.07464

[15] C. M. Bishop, *Neural Networks for Pattern Recognition*. New York, NY, USA: Oxford University Press, Inc., 1995.

[16] A. Dosovitskiy and T. Brox, "Inverting Visual Representations with Convolutional Networks," in *2016 {IEEE} Conference on Computer Vision and Pattern Recognition, {CVPR} 2016, Las Vegas, NV, USA, June 27-30, 2016*, 2016, pp. 4829–4837. [Online]. Available: https://doi.org/10.1109/CVPR.2016.522

[17] ——, "Generating Images with Perceptual Similarity Metrics based on Deep Networks," in *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds. Curran Associates, Inc., 2016, pp. 658–666.

[18] C. Nash, N. Kushman, and C. K. Williams, "Inverting supervised representations with autoregressive neural density models," *arXiv preprint arXiv:1806.00400*, 2018.

[19] M. Fredrikson, E. Lantz, S. Jha, S. Lin, D. Page, and T. Ristenpart, "Privacy in pharmacogenetics: An end-to-end case study of personalized warfarin dosing." in *USENIX Security Symposium*, 2014, pp. 17–32.

[20] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Advances In Neural Information Processing Systems*, pp. 1–9, 2012.

[21] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Training very deep networks," in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'15. Cambridge, MA, USA: MIT Press, 2015, pp. 2377–2385. [Online]. Available: http://dl.acm.org/citation.cfm?id=2969442.2969505

[22] M. A. Hall and L. A. Smith, "Feature selection for machine learning: Comparing a correlation-based filter approach to the wrapper," in *Proceedings of the Twelfth International Florida Artificial Intelligence Research Society Conference*. AAAI Press, 1999, pp. 235–239. [Online]. Available: http://dl.acm.org/citation.cfm?id=646812.707499

[23] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, 2015.

[24] M. Avriel, *Nonlinear programming: analysis and methods*. Courier Corporation, 2003.

[25] T. Zhang, "Solving large scale linear prediction problems using stochastic gradient descent algorithms," in *Proceedings of the twenty-first international conference on Machine learning*. ACM, 2004, p. 116.

[26] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, June 2005, pp. 886–893 vol. 1.

[27] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vision*, vol. 60, no. 2, pp. 91–110, Nov. 2004. [Online]. Available: https://doi.org/10.1023/B:VISI.0000029664.99615.94

[28] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson, "Understanding neural networks through deep visualization," in *Deep Learning Workshop, ICML*, 2015.

[29] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," in *ICLR workshop track*, 2014.

[30] A. Mahendran and A. Vedaldi, "Visualizing deep convolutional neural networks using natural pre-images," *International Journal of Computer Vision*, no. Springer, 2016.

[31] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[32] P. Baldi, "Autoencoders, unsupervised learning and deep architectures," in *Proceedings of the 2011 International Conference on Unsupervised and Transfer Learning Workshop - Volume 27*, ser. UTLW'11. JMLR.org, 2011, pp. 37–50. [Online]. Available: http://dl.acm.org/citation.cfm?id=3045796.3045801

[33] H.-W. Ng and S. Winkler, "A data-driven approach to cleaning large face datasets," in *IEEE International Conference on Image Processing (ICIP)*. IEEE, 2014, pp. 343–347.

[34] Z. Liu, P. Luo, X. Wang, and X. Tang, "Deep learning face attributes in the wild," in *Proceedings of International Conference on Computer Vision (ICCV)*, 2015.

[35] A. Krizhevsky, V. Nair, and G. Hinton, "The cifar-10 dataset," *online: http://www. cs. toronto. edu/kriz/cifar. html*, 2014.

[36] Y. LeCun, "The mnist database of handwritten digits," *http://yann. lecun. com/exdb/mnist/*, 1998.

[37] D. Silver *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, 2016.

[38] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *IEEE International Conference on Computer Vision (ICCV)*, 2015.

[39] G. Ateniese, L. V. Mancini, A. Spognardi, A. Villani, D. Vitali, and G. Felici, "Hacking smart machines with smarter ones: How to extract meaningful data from machine learning classifiers," *International Journal of Security and Networks*, vol. 10, no. 3, pp. 137–150, 2015.

[40] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *IEEE Symposium on Security and Privacy (SP)*, 2017.

[41] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," in *International Conference on Learning Representations*, 2014.

[42] R. Andrews, J. Diederich, and A. B. Tickle, "Survey and critique of techniques for extracting rules from trained artificial neural networks," *Know.-Based Syst.*, vol. 8, no. 6, pp. 373–389, Dec. 1995. [Online]. Available: https://doi.org/10.1016/0950-7051(96)81920-4

[43] J. Jia and N. Z. Gong, "AttriGuard: A Practical Defense Against Attribute Inference Attacks via Adversarial Machine Learning," in *27th {USENIX} Security Symposium ({USENIX} Security 18)*. Baltimore, MD: {USENIX} Association, 2018, pp. 513–529. [Online]. Available: https://www.usenix.org/conference/usenixsecurity18/presentation/jia-jinyuan

[44] S. Hidano, T. Murakami, S. Katsumata, S. Kiyomoto, and G. Hanaoka, "Model Inversion Attacks for Prediction Systems : Without Knowledge of Non-Sensitive Attributes," in *2017 15th Annual Conference on Privacy, Security and Trust (PST)*, 2017.

[45] X. Wu, M. Fredrikson, S. Jha, and J. F. Naughton, "A Methodology for Formalizing Model-Inversion Attacks," in *2016 IEEE 29th Computer Security Foundations Symposium (CSF)*, jun 2016, pp. 355–370.

[46] G. Ateniese, L. V. Mancini, A. Spognardi, A. Villani, D. Vitali, and G. Felici, "Hacking Smart Machines with Smarter Ones: How to Extract Meaningful Data from Machine Learning Classifiers," *International Journal of Security and Networks*, vol. 10, no. 3, pp. 137–150, sep 2015. [Online]. Available: http://dx.doi.org/10.1504/IJSN.2015.071829

[47] L. Melis, C. Song, E. De Cristofaro, and V. Shmatikov, "Exploiting Unintended Feature Leakage in Collaborative LearningâĹŮ," in *Proceedings of 40th IEEE Symposium on Security & Privacy (S&P 2019)*, 2018. [Online]. Available: http://arxiv.org/abs/1805.04049

[48] Z. Wang, M. Song, Z. Zhang, Y. Song, Q. Wang, and H. Qi, "Beyond Inferring Class Representatives : User-Level Privacy Leakage From Federated Learning," in *The 38th Annual IEEE International Conference on Computer Communications (INFOCOM 2019)*, 2019.

[49] C. Song, T. Ristenpart, and V. Shmatikov, "Machine Learning Models that Remember Too Much," in *Proceedings of the 2017 {ACM} {SIGSAC} Conference on Computer and Communications Security, {CCS} 2017, Dallas, TX, USA, October 30 - November 03, 2017*, 2017, pp. 587–601. [Online]. Available: http://doi.acm.org/10.1145/3133956.3134077

[50] H. Dang, Y. Huang, and E.-C. Chang, "Evading classifiers by morphing in the dark," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017.

[51] T. Gu, B. Dolan-Gavitt, and S. Garg, "Badnets: Identifying vulnerabilities in the machine learning model supply chain," *arXiv preprint arXiv:1708.06733*, 2017.

[52] B. Biggio, B. Nelson, and P. Laskov, "Poisoning attacks against support vector machines," in *Proceedings of the 29th International Coference on International Conference on Machine Learning*. Omnipress, 2012.

[53] Y. Liu, S. Ma, Y. Aafer, W.-C. Lee, J. Zhai, W. Wang, and X. Zhang, "Trojaning attack on neural networks," in *25nd Annual Network and Distributed System Security Symposium (NDSS)*, 2018.

[54] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[55] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *ASIACCS*, 2017.

[56] N. Papernot, P. D. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a Defense to Adversarial Perturbations Against Deep Neural Networks," in *{IEEE} Symposium on Security and Privacy, {SP} 2016, San Jose, CA, USA, May 22-26, 2016*, 2016, pp. 582–597. [Online]. Available: https://doi.org/10.1109/SP.2016.41

[57] D. Meng and H. Chen, "Magnet: a two-pronged defense against adversarial examples," *arXiv preprint arXiv:1705.09064*, 2017.

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

[59] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*. ACM, 2015, pp. 1310–1321.

[60] O. Ohrimenko, F. Schuster, C. Fournet, A. Mehta, S. Nowozin, K. Vaswani, and M. Costa, "Oblivious multi-party machine learning on trusted processors." in *USENIX Security Symposium*, 2016, pp. 619–636.

[61] C. Dwork and V. Feldman, "Privacy-preserving Prediction," in *Proceedings of the 31st Conference On Learning Theory*, ser. Proceedings of Machine Learning Research, S. Bubeck, V. Perchet, and P. Rigollet, Eds., vol. 75. PMLR, 2018, pp. 1693–1702. [Online]. Available: http://proceedings.mlr.press/v75/dwork18a.html

[62] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "Gazelle: A Low Latency Framework for Secure Neural Network Inference," in *27th {USENIX} Security Symposium ({USENIX} Security 18)*, 2018. [Online]. Available: http://arxiv.org/abs/1801.05507

[63] L. T. Phong, Y. Aono, T. Hayashi, L. Wang, and S. Moriai, "Privacy-Preserving Deep Learning via Additively Homomorphic Encryption," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 5, pp. 1333–1345, 2018.

[64] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy preserving machine learning." *IACR Cryptology ePrint Archive*, vol. 2017, p. 281, 2017.

[65] E. W. Saad and D. C. Wunsch, II, "Neural network explanation using inversion," *Neural Netw.*, vol. 20, no. 1, pp. 78–93, Jan. 2007. [Online]. Available: http://dx.doi.org/10.1016/j.neunet.2006.07.005

[66] A. Mahendran and A. Vedaldi, "Visualizing deep convolutional neural networks using natural pre-images," *International Journal of Computer Vision*, vol. 120, no. 3, pp. 233–255, 2016.

[67] C. Nash, N. Kushman, and C. K. Williams, "Inverting supervised representations with autoregressive neural density models," *arXiv preprint arXiv:1806.00400*, 2018.

[68] Q. Zhang, Y. N. Wu, and S.-C. Zhu, "Interpretable Convolutional Neural Networks," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. [Online]. Available: http://arxiv.org/abs/1710.00935

[69] M. D. Zeiler and R. Fergus, "Visualizing and Understanding Convolutional Networks," in *Computer Vision - {ECCV} 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part {I}*, 2014, pp. 818–833. [Online]. Available: https://doi.org/10.1007/978-3-319-10590-1{_}53

[70] M. Du, N. Liu, Q. Song, and X. Hu, "Towards explanation of dnn-based prediction with guided feature inversion," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery &#38; Data Mining*, ser. KDD '18. New York, NY, USA: ACM, 2018, pp. 1358–1367. [Online]. Available: http://doi.acm.org/10.1145/3219819.3220099

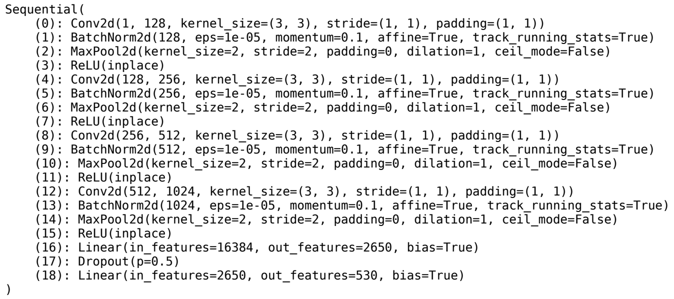
[71] J.-H. Jacobsen, A. W. Smeulders, and E. Oyallon, "i-revnet: Deep invertible networks," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=HJsjkMb0Z

[72] A. N. Gomez, M. Ren, R. Urtasun, and R. B. Grosse, "The Reversible Residual Network : Backpropagation Without Storing Activations," in *Advances in Neural Information Processing Systems*, vol. 1, no. Nips, 2017, pp. 1–11.

[73] A. C. Gilbert, Y. Zhang, K. Lee, Y. Zhang, and H. Lee, "Towards understanding the invertibility of convolutional neural networks," *IJCAI International Joint Conference on Artificial Intelligence*, pp. 1703–1710, 2017.

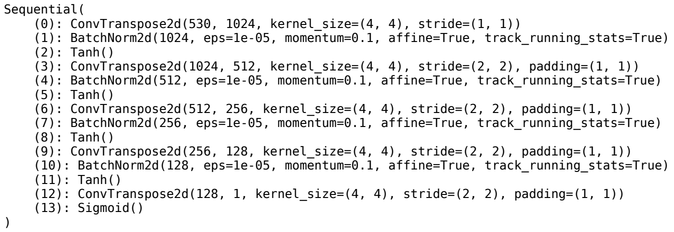
Appendix

A. Model Architecture



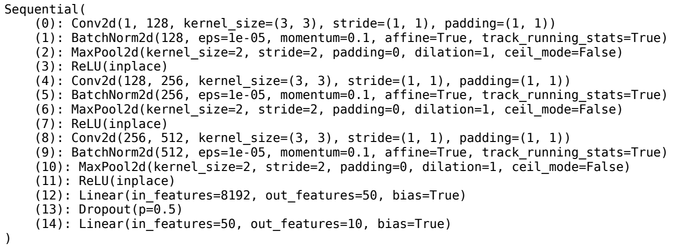
```
Sequential(
    (0): Conv2d(1, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): ReLU(inplace)
    (4): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (5): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (7): ReLU(inplace)
    (8): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (10): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (11): ReLU(inplace)
    (12): Conv2d(512, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (14): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (15): ReLU(inplace)
    (16): Linear(in_features=16384, out_features=2650, bias=True)
    (17): Dropout(p=0.5)
    (18): Linear(in_features=2650, out_features=530, bias=True)
)
```

Fig. 13: FaceScrub classifier architecture.



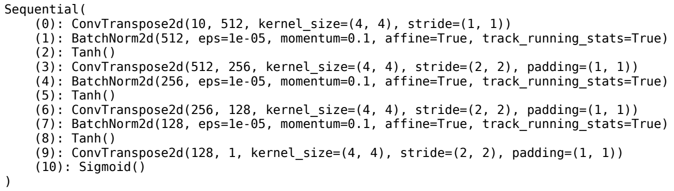
```
Sequential(
    (0): ConvTranspose2d(530, 1024, kernel_size=(4, 4), stride=(1, 1))
    (1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): Tanh()
    (3): ConvTranspose2d(1024, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (4): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): Tanh()
    (6): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (7): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (8): Tanh()
    (9): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (10): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (11): Tanh()
    (12): ConvTranspose2d(128, 1, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (13): Sigmoid()
)
```

Fig. 14: FaceScrub inversion model architecture.



```
Sequential(
    (0): Conv2d(1, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): ReLU(inplace)
    (4): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (5): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (7): ReLU(inplace)
    (8): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (10): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (11): ReLU(inplace)
    (12): Linear(in_features=8192, out_features=50, bias=True)
    (13): Dropout(p=0.5)
    (14): Linear(in_features=50, out_features=10, bias=True)
)
```

Fig. 15: MNIST classifier architecture.



```
Sequential(
    (0): ConvTranspose2d(10, 512, kernel_size=(4, 4), stride=(1, 1))
    (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): Tanh()
    (3): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): Tanh()
    (6): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (7): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (8): Tanh()
    (9): ConvTranspose2d(128, 1, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (10): Sigmoid()
)
```

Fig. 16: MNIST inversion model architecture.