Neural Network Inversion in Adversarial Setting via Background Knowledge Alignment

Ziqi Yang
National University of Singapore
yangziqi@comp.nus.edu.sg

Jiyi Zhang
National University of Singapore
jzhang93@comp.nus.edu.sg

Ee-Chien Chang
National University of Singapore
changec@comp.nus.edu.sg

Zhenkai Liang
National University of Singapore
liangzk@comp.nus.edu.sg

ABSTRACT

The wide application of deep learning technique has raised new security concerns about the training data and test data. In this work, we investigate the model inversion problem under 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 approach in various machine learning tasks and model architectures on multiple image datasets. We also confirm our results on Amazon Rekognition, a commercial prediction API that offers "machine learning as a service". We show that even with partial knowledge about the black-box 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.

CCS CONCEPTS

• **Security and privacy → Domain-specific security and privacy architectures**; • **Computing methodologies → Neural networks**.

KEYWORDS

neural networks; deep learning; model inversion; security; privacy

CCS '19, November 11–15, 2019, London, United Kingdom

ACM ISBN 978-1-4503-6747-9/19/11…$15.00
https://doi.org/10.1145/3319535.3354261

ACM Reference Format:
Ziqi Yang, Jiyi Zhang, Ee-Chien Chang, and Zhenkai Liang. 2019. Neural Network Inversion in Adversarial Setting via Background Knowledge Alignment. In *2019 ACM SIGSAC Conference on Computer and Communications Security (CCS '19), November 11–15, 2019, London, United Kingdom.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3319535.3354261

1 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 Amazon Rekognition APIs [2], provide scores of facial attributes, including emotion and eye-openness levels. There are also online services that evaluate users' face beauty and age [8, 50, 52]. 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. Security concerns arise in such applications.

Model inversion is a technique that aims to obtain information about the training data from the model's predictions. Research efforts of model inversion 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 [21, 30, 37, 39, 43, 44, 72]. We call this class of approach *optimization-based* approach. For example, model inversion attack (MIA) [21] 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) [28, 58, 65][1]. This is mainly because the optimization objective of optimization-based approach does not really capture the semantics of the data space, which we will further discuss in Section 2.2. The second class of approaches [10, 17, 18, 53] inverts 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 aim 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

[1] Our experiment also obtains similar result as shown in Figure 1 column (f).

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* [21, 22] 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 [21].

However, inversion in adversarial settings is different from existing model inversion settings. Specifically, in adversarial settings, the attacker does not have 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 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 [34] 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 [68]. 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 problem of model inversion under adversarial settings, and propose an effective approach for such attacks. 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. To address the problem of lacking access to training data, our approach draws the training data (of the inversion model) from a more generic data distribution based on the background knowledge. This set of *auxiliary samples* 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. The reconstructed facial images are recognizable.

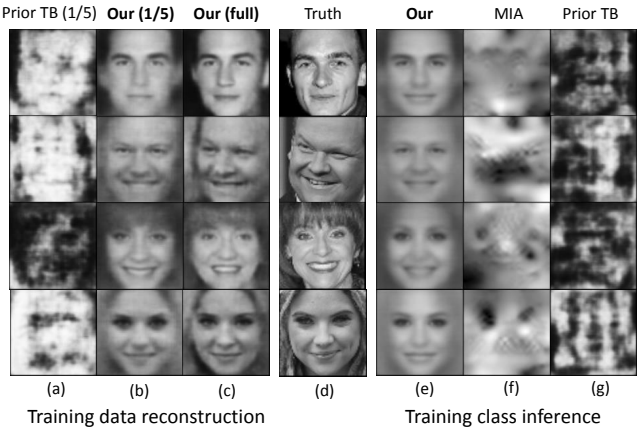

Figure 1: Training data reconstruction and training class inference of our and previous approaches against a facial recognition classifier. Column (a) and (b) compare our approach with a known training-based (TB) method when only 1/5 prediction results are available. Column (c) shows our result of inverting 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.

Figure 2: Inversion of commercial prediction API. (a) Recovered faces of victims. (b) Ground truth. The presented individuals are not seen to our inversion model during training.

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 [26]. 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, which clearly outperforms the result in column (a).

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 who does not have the training data but 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 scratch. 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. In the later one, 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. We confirm our inversion technique on Amazon Rekognition API, which detects facial features such as emotions, beard and gender on users' uploaded images. We have no knowledge of the backend models used by the API. As shown in Figure 2, it is still possible to accurately reconstruct faces of victim users from such limited information. 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, augmenting it with generic samples could improve inversion accuracy.

2 BACKGROUND

2.1 Machine Learning

In this paper, we focus on supervised learning, more specifically, on training classification models (classifiers) using neural networks [36]. 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 $\text{softmax}(\mathbf{z})_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$ 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:

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

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 [5]. Stochastic gradient descent (SGD) [79] 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 $\sum_i F_w(\mathbf{x})_i = 1$.

2.2 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 [15] and SIFT [42], 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 [21, 30, 37, 39, 43, 44, 72] and, most similar to our approach, training-based inversion [10, 17, 18, 53].

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 $F_w(\mathbf{x})$. Such inversion can be also used to generate a representative image for some class y (i.e., training class inference), by replacing $F_w(\mathbf{x})$ with vectorized y [21, 66]. To this end, the image $\hat{\mathbf{x}}$ should minimize some loss function between $F_w(\mathbf{x})$ and $F_w(\hat{\mathbf{x}})$. Therefore, it requires white-box access to the model to compute the gradients. 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 [76]. 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 [44]). More discussions on optimization-based inversion are presented in Appendix A.

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 scratch 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_θ which minimizes the following objective.

$$C(G_\theta) = \mathbb{E}_{\mathbf{x}\sim p_x}[\mathcal{R}(G_\theta(F_w(\mathbf{x})), \mathbf{x})] \tag{4}$$

where $\mathcal{R}$ is the image reconstruction loss such as L2 loss adopted in work [18]. 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 [18]).

3 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.

3.1 (Scenario 1) Data Reconstruction with Blackbox Classifier

In this scenario, the adversary is a curious user who wants to infer a classifier F_w's training data or other victim users' input data (i.e., test data of F_w). The adversary has black-box accesses to the 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 knows 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 knows 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 $\text{trunc}_m(\mathbf{g})$, when all m largest values of $\mathbf{g}$ remain in $\mathbf{f}$, while the rest are truncated to zeros. For instance,

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

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 $\text{trunc}(F_w(\mathbf{x})) = \mathbf{f}$. That is, ideally, the adversary wants to find a $\hat{\mathbf{x}}$ that satisfies the following:

$$\begin{aligned} \hat{\mathbf{x}} &= \arg\max_{\mathbf{x}\in X_\mathbf{f}} p_a(\mathbf{x}) \\ \text{subject to} \quad X_\mathbf{f} &= \{\mathbf{x} \in \mathcal{X} \mid \text{trunc}(F_w(\mathbf{x})) = \mathbf{f}\} \end{aligned} \tag{5}$$

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

3.2 (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.

$$\begin{aligned} \hat{\mathbf{x}} &= \arg\max_{\mathbf{x} \in X_y} p_a(\mathbf{x}) \\ \text{subject to} \quad X_y &= \{\mathbf{x} \in \mathcal{X} \mid F_w(\mathbf{x})_y \text{ is high}\} \end{aligned} \tag{6}$$

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

3.3 (Scenario 3) Joint Classification 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.

3.4 What If 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 [28, 65]. Our experimental result, as shown in Figure 1 column (f), also reaches the same conclusion. Second, the adversary as a user does not know the classifier's training data, which makes it impossible for previous training-based approaches to train the inversion model on the same training data. Third, the adversary, as either a user or developer, might obtain truncated prediction results instead of the full 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).

4 APPROACH

Our approach adopts previously mentioned training-based strategy to invert the classifier. The overall framework is shown in Figure 3. 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 [7]

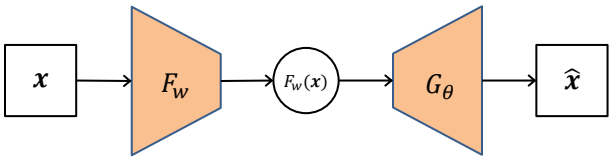

Figure 3: Framework of training-based inversion approach. The classifier F_w takes data x as input and produces a prediction vector $F_w(\mathbf{x})$. The inversion model G_θ outputs the reconstructed data $\hat{\mathbf{x}}$ with the prediction as input.

where F_w is the "encoder", G_θ is the "decoder", and the prediction can be thought of as the latent space.

There are three aspects that are different from autoencoder. (1) In Scenario 1 and 2, unlike autoencoder, the F_w is given and fixed. Furthermore, the training data of F_w is not available to train G_θ. Section 4.1 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 4.2 gives the proposed truncation method. (3) In our Scenario 3, although F_w is not fixed, unlike autoencoder, there is an additional requirement on F_w accuracy. That is, it is a joint classifier and inversion model problem, where F_w is the classifier and G_θ is the inversion model. Section 4.3 gives our training method.

4.1 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 5.2. The inversion model can precisely reconstruct the training data points even if it never sees the training classes during the construction.

4.2 Truncation Method for Model Inversion

We propose a truncation method of training G_θ to make it aligned to the partial prediction results. Figure 4 shows the architecture of the classifier F_w and its inversion model G_θ. The idea is to

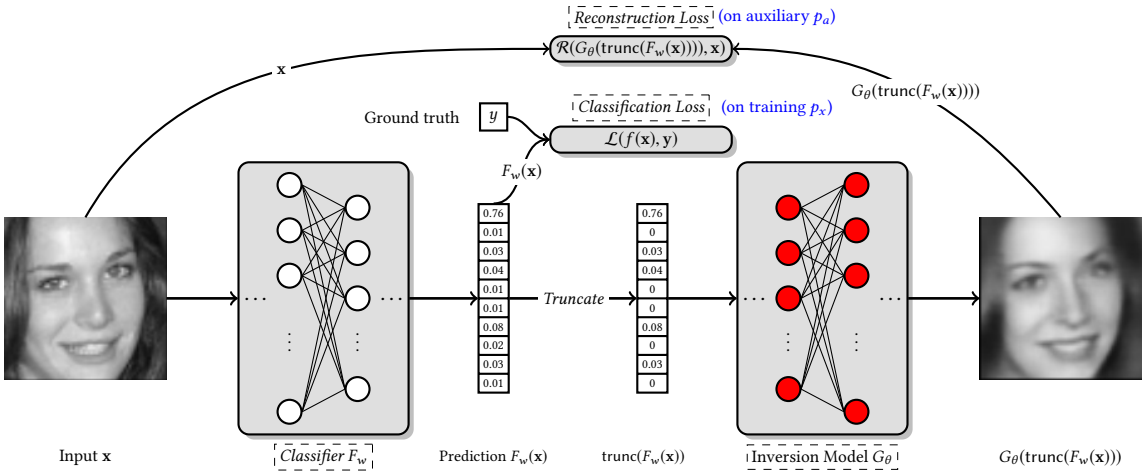

Figure 4: Architecture of the classifier and inversion model. The classifier F_w takes data $\mathbf{x}$ as input and produces a prediction $F_w(\mathbf{x})$. Such prediction vector is truncated (if necessary) to a new vector $\text{trunc}(F_w(\mathbf{x}))$. The inversion model G_θ takes the truncated prediction as input and outputs a reconstructed data $G_\theta(\text{trunc}(F_w(\mathbf{x})))$.

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.

$$C(G_\theta) = \mathbb{E}_{\mathbf{a}\sim p_a}[\mathcal{R}(G_\theta(\text{trunc}_m(F_w(\mathbf{a}))), \mathbf{a})] \tag{7}$$

where $\mathcal{R}$ is the loss function and we use L2 norm as $\mathcal{R}$ in this paper. The truncation process can be understood as a similar way of feature selection [26] 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 [21]. 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 5.3).

4.3 Joint Training of Classifier and Inversion Model

When the adversary is the developer of the classifier, he could 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 $R_D(F_w, G_\theta)$.

$$R_D(F_w, G_\theta) = \frac{1}{|D|}\sum_{\mathbf{x}\in D} ||(G_\theta(\text{trunc}(F_w(\mathbf{x}))) - \mathbf{x}||_2^2 \tag{8}$$

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 softmax function (see Section 2.1), 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.

$$\mathbf{z} = \log(F_w(\mathbf{x})) + c \tag{9}$$

Table 1: 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

where log is element-wise, and c is a scalar which is added to $\log(F_w(\mathbf{x}))$ element-wise and is also optimized during training the inversion model.

5 EXPERIMENTS

In this section, we evaluate the inversion performance of our method and compare it with existing work. We evaluate three factors: the choice of auxiliary set, the truncation method and the full prediction size. We also evaluate our approach on a commercial face recognition API provided by Amazon.

5.1 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 [55].** 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 [41].** 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 [33].** 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 [35].** 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 1 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 (i.e., digit 0, 1, 2, 3, 8) of MNIST to train F_w and use the other 5 classes (i.e., digit 4, 5, 6, 7, 9) 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.

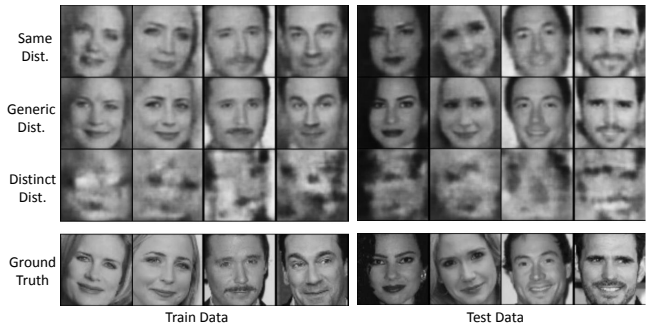

Figure 5: Effect of the auxiliary set on the inversion quality on FaceScrub F_w. We use auxiliary set of the same (1st row), generic (2nd row) and distinct (3rd row) data distributions to train the inversion model.

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 B. 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.

We train both F_w and G_θ on a workstation running Ubuntu 16.04 LTS equipped with two Intel Xeon CPUs E5-2620 V4 (2.1Ghz/8-core), 256GB RAM, and two NVIDIA Tesla P100 GPU cards. We use PyTorch [60] to implement neural networks. The number of queries to F_w during training G_θ is equivalent to the number of auxiliary samples. Roughly, it takes less than 12 hours to train G_θ for FaceScrub F_w, and less than 6 hours to train G_θ for MNIST F_w on our auxiliary sets. Once G_θ is trained, it performs inversion attack by a single forward pass through the neural network with the prediction values as input, which takes only a few milliseconds.

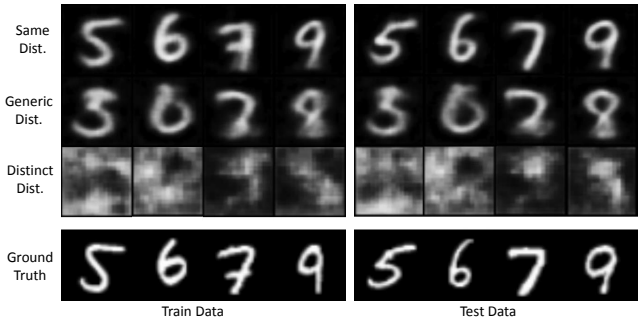

Figure 6: Effect of the auxiliary set on the inversion quality on MNIST F_w. We use auxiliary set of the same (1st row), generic (2nd row) and distinct (3rd row) data distributions to train the inversion model.

Table 2: Quantitative measurement (mean squared error) of the effect of the auxiliary set on the inversion quality.

Distribution	FaceScrub		MNIST	
Same Dist.	Train	0.0107	Train	0.0078
	Test	0.0074	Test	0.0069
Generic Dist.	Train	0.0114	Train	0.0153
	Test	0.0115	Test	0.0152
Distinct Dist.	Train	0.0315	Train	0.0415
	Test	0.0316	Test	0.0416

5.2 Effect of Auxiliary Set

Effect on inversion quality. We present the inversion results (Scenario 1) on randomly-chosen training data and test data of the F_w in Figure 5 and 6. We also present the mean squared error between the reconstructed images and the ground truth in Table 2. 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 (both visually and quantitatively). 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. 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 5 and 6.

> 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.

Effect on inversion alignment. We divide the cleaned CelebA dataset into 5 subsets of frontal, turn-left, turn-right, turn-down and turn-up faces. We use each subset as the auxiliary set to train G_θ, and test whether the specific auxiliary set can align the inversion result. We use OpenCV library [12] for Python3 and FAN network [13] to detect the facial orientation (See Appendix C for more details), and eventually get 109,105 frontal faces, 48,137 turn-left faces, 33,632 turn-right faces, 4,278 turn-down faces and 2,304 turn-up faces. We present the inversion results on randomly-chosen training data and test data of FaceScrub classifier in Figure 7. It is clear to see that the specific auxiliary set does align G_θ's inversion result to the corresponding facial orientation. It is worth noting that, in this experiment, the auxiliary set is drawn from a more "specialized" distribution that contains only one type of pose, in contrast to the original training data which contains all 5 types of pose.

5.3 Effect of Truncation

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 results of our truncation method and previous training-based method (i.e., without truncation) on FaceScrub classifier in Figure 9 and on MNIST classifier in Figure 10. 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. Our approach can produce highly recognizable images 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 9 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. We also perform quantitative measurement of the inversion quality as shown in Figure 8. The mean squared error of our approach is much smaller than that of the prior approach, especially when m is small.

> 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.

5.4 Effect of Full Prediction Size

Section 5.3 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 $k = 10, 50, 100$

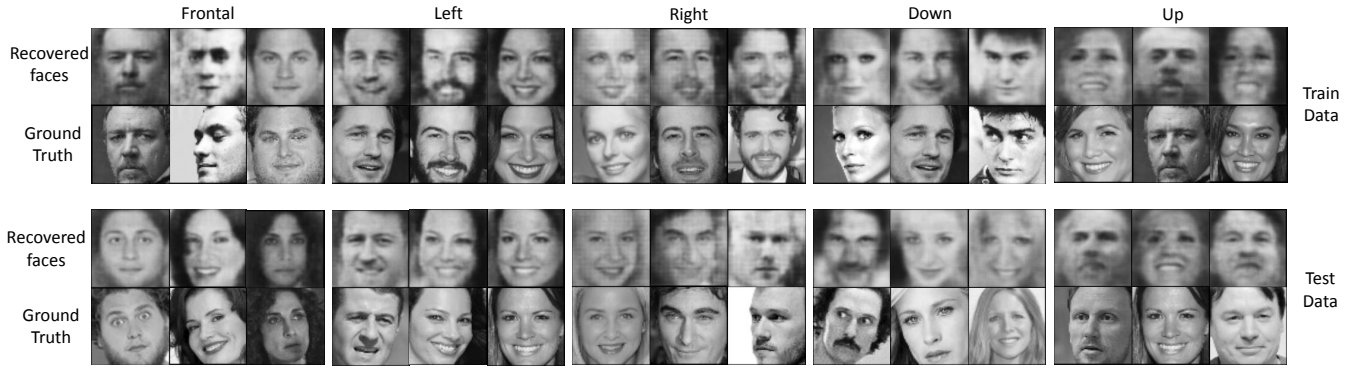

Figure 7: Effect of the auxiliary set on the inversion alignment. We use auxiliary set of frontal, left, right, down and up faces of celebA to train the inversion model. We present result on FaceScrub classifier's randomly-chosen training data and test data.

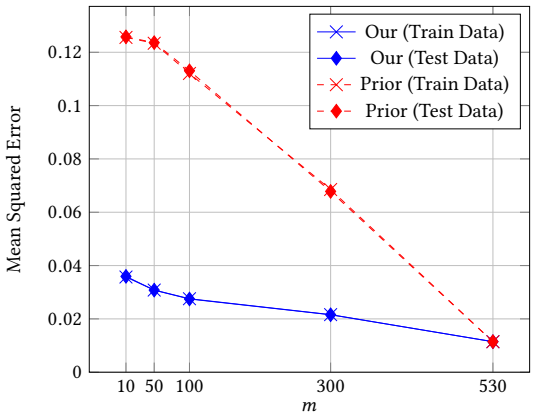

Figure 8: Quantitative measurement of the effect of truncation (m) for G_θ on the inversion quality on FaceScrub F_w. The x-axis is the m, and the y-axis is mean squared error.

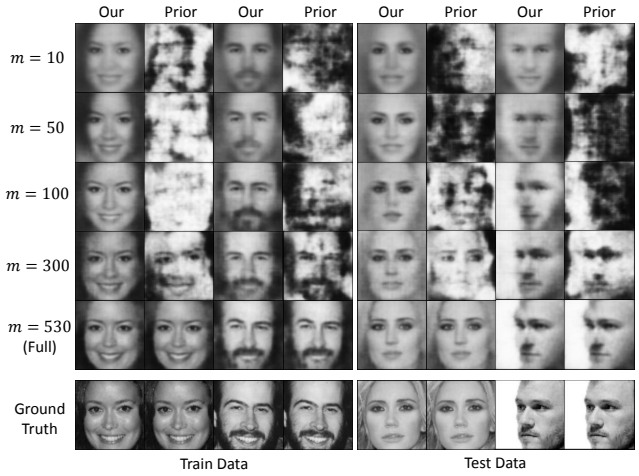

Figure 9: Effect of truncation (m) for G_θ on the inversion quality on FaceScrub F_w. Results of our and previous approach are presented in odd and even columns respectively.

and 530 for FaceScrub and $k = 3, 5$ and 10 for MNIST. We present the inversion results against FaceScrub classifier in Figure 11 and against MNIST classifier in Figure 12. In all experiments, we assume that the 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. Our quantitative measurement of the inversion quality under different k (as shown in Figure 13) also shows that the mean squared error is inversely proportional to k. 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.

5.5 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

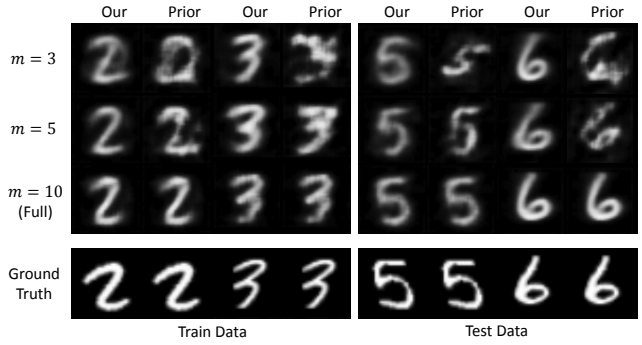

Figure 10: Effect of truncation (m) for G_θ on the inversion quality on MNIST F_w. Results of our and previous approach are presented in odd and even columns respectively.

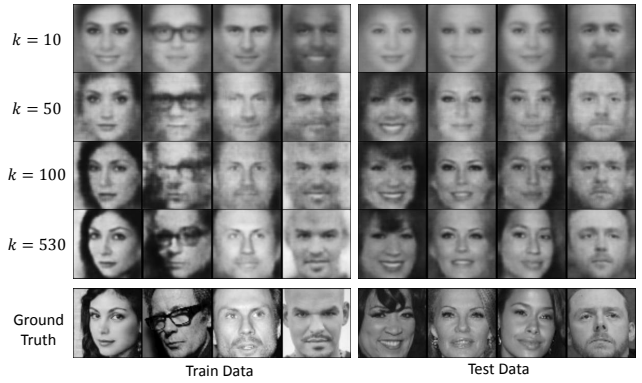

Figure 11: Effect of the full prediction size (k) on the inversion quality on FaceScrub F_w.

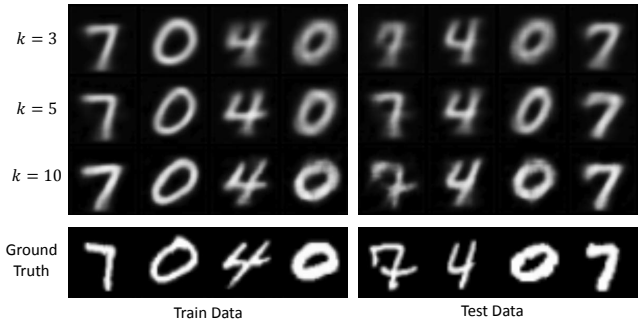

Figure 12: Effect of the full prediction size (k) on the inversion quality on MNIST F_w.

vectors to G_θ. The output of G_θ is the inferred image of the training class.

We implement the prior MIA attack by following the Algorithm 1 of [21]. Similarly, we set AuxTerm($\mathbf{x}$) = 0 for all $\mathbf{x}$ because the adversary has no other information except the target label. We also

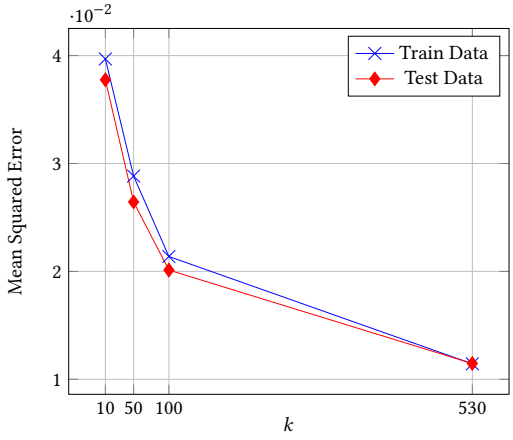

Figure 13: Quantitative measurement of the effect of the full prediction size (k) on the inversion quality on FaceScrub F_w. The x-axis is the k, and the y-axis is mean squared error.

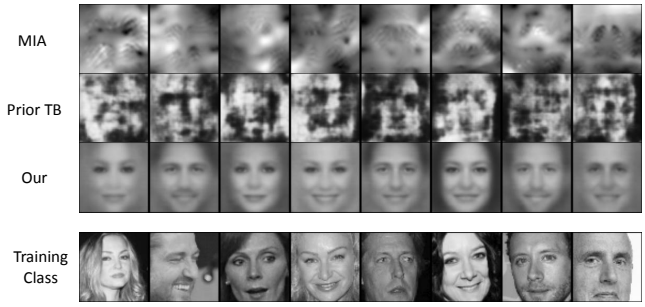

Figure 14: Training class inference results. We present results of MIA (1st row), previous training-based (TB) inversion (2nd row) and our inversion (3rd row).

set the Process function to be a NLMeans denoising filter followed by a Gaussian sharpening filter.

We compare the inversion results of our method and previous methods in Figure 14. We can see that the inferred images by MIA are semantically meaningless and do not even form recognizable faces when MIA is applied to CNNs. This is consistent with conclusions in previous research work [28, 65] where similar unsatisfactory inversion results were obtained. 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 semantic 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. A possible reason is that the auxiliary data consists of majority frontal faces. Since the training aligns the inversion model to the training distribution, thus recognizable frontal faces are generated.

Figure 15: Inversion results of G_θ: trained with blackbox FaceScrub F_w vs trained jointly with FaceScrub F_w. We use augmented training data with CelebA as the auxiliary set for the 3rd row.

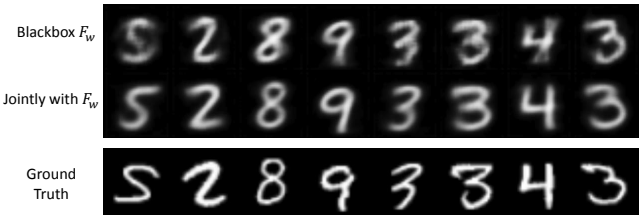

Figure 16: Inversion results of G_θ: trained with blackbox MNIST F_w vs trained jointly with MNIST F_w.

Table 3: Classification accuracy of the classifier F_w and average reconstruction loss (MSE) of the inversion model G_θ. Results are reported on test set.

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

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

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.

5.6 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 15 and 16. 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, namely mean squared error (MSE), of G_θ constructed by the two methods in the 5th and 6th columns of Table 3. 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 3. 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 15 and present the average reconstruction loss on the test set in the last column in Table 3. 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.

5.7 Experiment on Commercial Prediction API

We evaluate our approach on the commercial Amazon Rekognition API [2], which detects facial features of users' uploaded images. We have no knowledge of the backend models used by the API. We query the API with our auxiliary dataset and use the predictions (facial features) as input to train inversion models. We evaluate the

Table 4: Quantitative measurement (mean squared error) of the inversion on Amazon Rekognition API.

Features	Unknown individuals	Known individuals but unknown images
Remove Landmark & Pose	0.0472	0.0469
Remove Landmark	0.0470	0.0462
Round(1)	0.0454	0.0443
Round(3)	0.0437	0.0438
Round(5)	0.0437	0.0438
No round (80 features)	0.0437	0.0438

inversion models on reconstructing unknown individuals' images and known individuals' unknown images.

We use 80% of the cleaned CelebA dataset as the auxiliary set, and leave the remaining 20% as victim images of known individuals. We use the FaceScrub dataset as victim images of unknown individuals. Note that, we query the API with the original full-size CelebA images to obtain the accurate predictions, but train the inversion model (whose architecture is presented in Figure 19) using our resized 64×64 images. In summary, we send 156,582 queries (auxiliary samples) to Amazon Rekognition API to train G_θ, and send 87,724 queries (victim images) to test G_θ's ability of reconstructing victim data. The training time of G_θ is also roughly less than 12 hours.

At the time of writing, Amazon Rekognition API produces features including emotions (7), smile (1), facial attributes (9), bounding box (4), facial landmarks (60), pose (3), quality (2) and confidence of face (1). We discard the bounding box, quality and confidence of face and use the remaining 80 features. We standardize the feature values to [0, 1]. The facial landmark coordinates are converted to the ratio of their distance to the bounding box boundaries to the box width/height, with the upper left corner of the box as the origin. Figure 22 in Appendix D presents an example of preprocessed predicted facial features before rounding.

We train three inversion models on three different truncated feature sizes respectively: **80** features; **20** features by removing landmarks; and **17** features by removing both landmarks and pose. To further test the robustness of our approach, we round the original feature values of victim images to **1**, **3** and **5** decimals respectively. We present the result in Figure 17. We can see that if the predicted 80 features of the victim images are all input to the inversion model, it can accurately reconstruct these images (Row 6). Besides, it can capture more information than what the API tells. For example, it also recovers information about cheeks, teeth and hair length. Rounding the predicted feature values does not have significant effect on the inversion (Row 3-5). Even though the landmark information is removed from the inversion model's input, it can still recover recognizable faces (Row 2). If both the landmarks and pose information are removed, the recovered faces are not that satisfactory, but can capture facial features such as sunglasses, beard, age range and gender which are content of the API output (Row 1). We further perform quantitative measurement of the inversion quality and present the result in Table 4. We can confirm our conclusion both visually and quantitatively.

Figure 17: Inversion on Amazon Rekognition API. The 1st row is the result of G_θ on 17 features by removing landmarks and pose from the API output on victim images. The 2nd row is the result of G_θ on 20 features by removing landmarks. The 3rd-6th rows are results of G_θ on 80 features.

6 DISCUSSION

Prior work has established the connection between overfitting and training data inference. For example, Yeom et al. [75] show that prior model inversion attack [21, 22] and membership inference attack [65] are deeply related and are both sensitive to overfitting of the target model. Recent work [67] also leverages overfitting to embed private information about training data into the model.

Our work, on the contrary, leverages generalization to "transfer" information from the auxiliary dataset to the task of training G_θ. Generalization enables a model to adapt properly to new, previously unseen data, based on which transfer learning can apply the knowledge gained while solving one problem to a different but related problem. Using auxiliary data to train G_θ is related to transfer learning. Specifically, the soft predictions of a well-generalized F_w on auxiliary data drawn from a generic data distribution encode rich information between classes. The G_θ will learn to capture such hidden useful information to reconstruct input data. Our truncation method further helps G_θ to focus on the most important features. Hence, an extremely overfitting model will provide no useful information in soft predictions, which, however, is usually not the case for a working model.

There is a trend of users publishing derived prediction results directly or indirectly. For examples, iOS has a built-in functionality that allows users to share AR emojis which are derived from their faces [62]. Many users share their face-age scores on social media for fun [71]. Some sharing platform even asks users for their prediction results [70]. Fortunately, these prediction values are still coarse at present, which might make it difficult to fully invert the model. However, the potential enhanced versions of these prediction services could provide more accurate values in the future, which may lead to more accurate inversion attacks. Besides, MLaaS companies often state in the privacy policy that they do not store or abuse users' uploaded data [6, 51], but they seldom mention how they deal with the derived prediction results. Hopefully, our work can

help encourage the companies to update the privacy policy (e.g., explicitly stating how the derived prediction results are handled).

The model inversion attack leaks useful information to malicious parties. For example, in the case of facial recognition systems, adversaries can recover a representative face of individuals whose facial images are used as the training data [21, 65]. However, there are opposing views on whether such attack should be classified as a privacy attack [47, 65].

7 RELATED WORK

ML Privacy. Researchers have strongly suggested that ML models suffer from various privacy threats to its training data [3, 22, 31, 58, 65]. For instance, given access to an ML model, an adversary can infer non-trivial and useful information about its training set [3].

Membership inference attack [48, 65] enables an adversary to learn if a record of his choice is part of the private training sets. Model inversion attack [21, 22] tries to infer a representative sample or sensitive attributes of training data. Some work further improves sensitive attribute inference either by formalizing it [74] or without knowing the non-sensitive attributes [27]. Giuseppe et al. [4] 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. [28] investigate information leakage from collaborative learning but in the training phase. Recent work also studies privacy issues in collaborative learning [48, 73] and ML-as-a-service platforms [67].

Some of the above mentioned attacks 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 hand, examines how one can reconstruct either specific training data points or test data points from their prediction results in the inference phase.

ML in Adversarial Settings. Various works have suggested that ML models are likely vulnerable in adversarial settings [14, 16, 25]. 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 [9, 40], employing adversarial loss function [25]), maliciously modifying the victim model [14], or feeding the victim model with adversarially crafted samples [24, 49, 57, 59, 69].

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.

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 [1, 11, 56, 61, 64] 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 [20, 32]. For examples, Dwork and Feldman [20] 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. [32] propose Gazelle framework for secure neural network inference using cryptographic tools. 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.

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 [45, 54, 63, 66, 76, 78]. Inverting a neural network is one important method to understand the representations learned by neural networks [19, 23, 29, 54, 66, 77]. Such 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.

8 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.

ACKNOWLEDGMENTS

We would like to thank our shepherd Vincent Bindschaedler and the anonymous reviewers for their valuable feedback. This research is supported by the National Research Foundation Singapore, Prime Minister's Office, Singapore in part under its AI Singapore Programme (Award Number: [AISG-100E-2019-0033]) and in part under its National Cybersecurity R&D Programme (Grant Number: NRF2016NCR-NCR002-012).

REFERENCES

[1] Martín Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 308–318.

[2] Amazon. 2019. Amazon Rekognition. https://aws.amazon.com/rekognition/.

[3] Giuseppe Ateniese, Luigi V Mancini, Angelo Spognardi, Antonio Villani, Domenico Vitali, and Giovanni Felici. 2015. Hacking smart machines with smarter ones: How to extract meaningful data from machine learning classifiers. *International Journal of Security and Networks* 10, 3 (2015), 137–150.

[4] Giuseppe Ateniese, Luigi V Mancini, Angelo Spognardi, Antonio Villani, Domenico Vitali, and Giovanni Felici. 2015. Hacking Smart Machines with Smarter Ones: How to Extract Meaningful Data from Machine Learning Classifiers. *International Journal of Security and Networks* 10, 3 (sep 2015), 137–150. https://doi.org/10.1504/IJSN.2015.071829

[5] Mordecai Avriel. 2003. *Nonlinear programming: analysis and methods*. Courier Corporation.

[6] Amazon AWS. 2019. Data Privacy. https://aws.amazon.com/compliance/data-privacy-faq/.

[7] Pierre Baldi. 2011. Autoencoders, Unsupervised Learning and Deep Architectures. In *Proceedings of the 2011 International Conference on Unsupervised and Transfer Learning Workshop - Volume 27 (UTLW'11)*. JMLR.org, 37–50. http://dl.acm.org/citation.cfm?id=3045796.3045801

[8] Beauty.AI. 2019. Beauty.AI. http://beauty.ai/.

[9] Battista Biggio, Blaine Nelson, and Pavel Laskov. 2012. Poisoning Attacks Against Support Vector Machines. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*. Omnipress.

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

[11] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. 2017. Practical Secure Aggregation for Privacy Preserving Machine Learning. *IACR Cryptology ePrint Archive* 2017 (2017), 281.

[12] G. Bradski. 2000. The OpenCV Library. *Dr. Dobb's Journal of Software Tools* (2000).

[13] Adrian Bulat and Georgios Tzimiropoulos. 2017. How far are we from solving the 2d & 3d face alignment problem?(and a dataset of 230,000 3d facial landmarks). In *International Conference on Computer Vision*, Vol. 1. 8.

[14] Nicholas Carlini and David Wagner. 2017. Towards evaluating the robustness of neural networks. In *IEEE Symposium on Security and Privacy (SP)*.

[15] N. Dalal and B. Triggs. 2005. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, Vol. 1. 886–893 vol. 1. https://doi.org/10.1109/CVPR.2005.177

[16] Hung Dang, Yue Huang, and Ee-Chien Chang. 2017. Evading Classifiers by Morphing in the Dark. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM.

[17] Alexey Dosovitskiy and Thomas Brox. 2016. 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., 658–666.

[18] Alexey Dosovitskiy and Thomas Brox. 2016. 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*. 4829–4837. https://doi.org/10.1109/CVPR.2016.522

[19] Mengnan Du, Ninghao Liu, Qingquan Song, and Xia Hu. 2018. Towards Explanation of DNN-based Prediction with Guided Feature Inversion. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD '18)*. ACM, New York, NY, USA, 1358–1367. https://doi.org/10.1145/3219819.3220099

[20] Cynthia Dwork and Vitaly Feldman. 2018. Privacy-preserving Prediction. In *Proceedings of the 31st Conference On Learning Theory (Proceedings of Machine Learning Research)*, Sébastien Bubeck, Vianney Perchet, and Philippe Rigollet (Eds.), Vol. 75. PMLR, 1693–1702. http://proceedings.mlr.press/v75/dwork18a.html

[21] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. 2015. 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*. 1322–1333. https://doi.org/10.1145/2810103.2813677

[22] Matthew Fredrikson, Eric Lantz, Somesh Jha, Simon Lin, David Page, and Thomas Ristenpart. 2014. Privacy in Pharmacogenetics: An End-to-End Case Study of Personalized Warfarin Dosing.. In *USENIX Security Symposium*. 17–32.

[23] Anna C. Gilbert, Yi Zhang, Kibok Lee, Yuting Zhang, and Honglak Lee. 2017. Towards understanding the invertibility of convolutional neural networks. *IJCAI International Joint Conference on Artificial Intelligence* (2017), 1703–1710. https://doi.org/10.1080/10920277.1998.10595667 arXiv:1705.08664

[24] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572* (2014).

[25] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. 2017. BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain. *arXiv preprint arXiv:1708.06733* (2017).

[26] Mark A. Hall and Lloyd A. Smith. 1999. 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, 235–239. http://dl.acm.org/citation.cfm?id=646812.707499

[27] Seira Hidano, Takao Murakami, Shuichi Katsumata, Shinsaku Kiyomoto, and Goichiro Hanaoka. 2017. Model Inversion Attacks for Prediction Systems : Without Knowledge of Non-Sensitive Attributes. In *2017 15th Annual Conference on Privacy, Security and Trust (PST)*.

[28] Briland Hitaj, Giuseppe Ateniese, and Fernando Perez-Cruz. 2017. 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. 603–618. https://doi.org/10.1145/3133956.3134012 arXiv:1702.07464

[29] JÃűrn-Henrik Jacobsen, Arnold W.M. Smeulders, and Edouard Oyallon. 2018. i-RevNet: Deep Invertible Networks. In *International Conference on Learning Representations*. https://openreview.net/forum?id=HJsjkMb0Z

[30] C A Jensen, R D Reed, R J Marks, M A El-Sharkawi, Jae-Byung Jung, R T Miyamoto, G M Anderson, and C J Eggen. 1999. Inversion of feedforward neural networks: algorithms and applications. *Proc. IEEE* 87, 9 (sep 1999), 1536–1549. https://doi.org/10.1109/5.784232

[31] Jinyuan Jia and Neil Zhenqiang Gong. 2018. AttriGuard: A Practical Defense Against Attribute Inference Attacks via Adversarial Machine Learning. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*. {USENIX} Association, Baltimore, MD, 513–529. arXiv:arXiv:1805.04810v1 https://www.usenix.org/conference/usenixsecurity18/presentation/jia-jinyuan

[32] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. 2018. Gazelle: A Low Latency Framework for Secure Neural Network Inference. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*. arXiv:1801.05507 http://arxiv.org/abs/1801.05507

[33] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. 2014. The CIFAR-10 dataset. *online: http://www. cs. toronto. edu/kriz/cifar. html* (2014).

[34] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. *Advances In Neural Information Processing Systems* (2012), 1–9. https://doi.org/10.1016/j.protcy.2014.09.007 arXiv:1102.0183

[35] Yann LeCun. 1998. The MNIST database of handwritten digits. *http://yann. lecun. com/exdb/mnist/* (1998).

[36] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* (2015).

[37] S. Lee and R. M. Kil. 1994. Inverse mapping of continuous functions using local and global information. *IEEE Transactions on Neural Networks* 5, 3 (May 1994), 409–423. https://doi.org/10.1109/72.286912

[38] Shiqi Li, Chi Xu, and Ming Xie. 2012. A robust O (n) solution to the perspective-n-point problem. *IEEE transactions on pattern analysis and machine intelligence* 34, 7 (2012), 1444–1450.

[39] Linden and Kindermann. 1989. Inversion of multilayer nets. In *International 1989 Joint Conference on Neural Networks*. 425–430 vol.2. https://doi.org/10.1109/IJCNN.1989.118277

[40] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. 2018. Trojaning Attack on Neural Networks. In *25nd Annual Network and Distributed System Security Symposium (NDSS)*.

[41] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. 2015. Deep Learning Face Attributes in the Wild. In *Proceedings of International Conference on Computer Vision (ICCV)*.

[42] David G. Lowe. 2004. Distinctive Image Features from Scale-Invariant Keypoints. *Int. J. Comput. Vision* 60, 2 (Nov. 2004), 91–110. https://doi.org/10.1023/B:VISI.0000029664.99615.94

[43] Bao-Liang Lu, H. Kita, and Y. Nishikawa. 1999. Inverting feedforward neural networks using linear and nonlinear programming. *IEEE Transactions on Neural Networks* 10, 6 (Nov 1999), 1271–1290. https://doi.org/10.1109/72.809074

[44] Aravindh Mahendran and Andrea Vedaldi. 2015. 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*. 5188–5196. https://doi.org/10.1109/CVPR.2015.7299155

[45] Aravindh Mahendran and Andrea Vedaldi. 2016. Visualizing deep convolutional neural networks using natural pre-images. *International Journal of Computer Vision* 120, 3 (2016), 233–255.

[46] Aravindh Mahendran and Andrea Vedaldi. 2016. Visualizing deep convolutional neural networks using natural pre-images. *International Journal of Computer Vision* Springer (2016). arXiv:arXiv:1512.02017v3

[47] Frank McSherry. 2017. Statistical inference considered harmful. https://github.com/frankmcsherry/blog/blob/master/posts/2016-06-14.md.

[48] Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. 2018. Exploiting Unintended Feature Leakage in Collaborative LearningâĹŰ. In *Proceedings of 40th IEEE Symposium on Security & Privacy (S&P 2019)*. arXiv:1805.04049 http://arxiv.org/abs/1805.04049

[49] Dongyu Meng and Hao Chen. 2017. MagNet: a Two-Pronged Defense against Adversarial Examples. *arXiv preprint arXiv:1705.09064* (2017).

[50] Microsoft. 2019. How-Old.net. https://www.how-old.net/.

[51] Microsoft. 2019. Microsoft Privacy Statement. https://privacy.microsoft.com/en-us/privacystatement.

[52] Microsoft. 2019. Xiaoice. https://kan.msxiaobing.com/ImageGame/Portal.

[53] Charlie Nash, Nate Kushman, and Christopher KI Williams. 2018. Inverting Supervised Representations with Autoregressive Neural Density Models. *arXiv preprint arXiv:1806.00400* (2018).

[54] Charlie Nash, Nate Kushman, and Christopher KI Williams. 2018. Inverting Supervised Representations with Autoregressive Neural Density Models. *arXiv preprint arXiv:1806.00400* (2018).

[55] Hong-Wei Ng and Stefan Winkler. 2014. A data-driven approach to cleaning large face datasets. In *IEEE International Conference on Image Processing (ICIP)*. IEEE, 343–347.

[56] Olga Ohrimenko, Felix Schuster, Cédric Fournet, Aastha Mehta, Sebastian Nowozin, Kapil Vaswani, and Manuel Costa. 2016. Oblivious Multi-Party Machine Learning on Trusted Processors.. In *USENIX Security Symposium*. 619–636.

[57] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. 2017. Practical Black-Box Attacks against Machine Learning. In *ASIACCS*.

[58] Nicolas Papernot, Patrick Mcdaniel, and Penn State. 2018. Security and Privacy in Machine Learning. In *2018 IEEE European Symposium on Security and Privacy (EuroS\&P)*, Vol. 392001. 1–16. https://doi.org/10.1109/EuroSP.2018.00035

[59] Nicolas Papernot, Patrick D McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. 2016. 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*. 582–597. https://doi.org/10.1109/SP.2016.41

[60] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in pytorch. (2017).

[61] Le Trieu Phong, Yoshinori Aono, Takuya Hayashi, Lihua Wang, and Shiho Moriai. 2018. Privacy-Preserving Deep Learning via Additively Homomorphic Encryption. *IEEE Transactions on Information Forensics and Security* 13, 5 (2018), 1333–1345. https://doi.org/10.1109/TIFS.2017.2787987

[62] Trusted Reviews. 2018. Animoji and Memoji: Everything you need to know about Apple's AR emojis. https://www.trustedreviews.com/news/iphone-x-animoji-3335178.

[63] Emad W. Saad and Donald C. Wunsch, II. 2007. Neural Network Explanation Using Inversion. *Neural Netw.* 20, 1 (Jan. 2007), 78–93. https://doi.org/10.1016/j.neunet.2006.07.005

[64] Reza Shokri and Vitaly Shmatikov. 2015. Privacy-preserving deep learning. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*. ACM, 1310–1321.

[65] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. 2017. 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*. 3–18. https://doi.org/10.1109/SP.2017.41

[66] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. 2014. Deep inside convolutional networks: Visualising image classification models and saliency maps. In *ICLR workshop track*.

[67] Congzheng Song, Thomas Ristenpart, and Vitaly Shmatikov. 2017. 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*. 587–601. https://doi.org/10.1145/3133956.3134077

[68] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. 2015. Training Very Deep Networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2 (NIPS'15)*. MIT Press, Cambridge, MA, USA, 2377–2385. http://dl.acm.org/citation.cfm?id=2969442.2969505

[69] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2014. Intriguing properties of neural networks. In *International Conference on Learning Representations*.

[70] USA Today. 2015. Website "How Old Do I Look?" guesses your age. https://www.usatoday.com/story/tech/2015/05/01/new-website-how-old-do-i-look-guesses-your-age/26688603/.

[71] DIGIDAY UK. 2015. How Microsoft scored a viral hit with #HowOldRobot. https://digiday.com/media/microsoft-scored-viral-hit-howoldrobot/.

[72] Annamária R Várkonyi-Kóczy. 2009. *Observer-Based Iterative Fuzzy and Neural Network Model Inversion for Measurement and Control Applications*. Springer Berlin Heidelberg, Berlin, Heidelberg, 681–702. https://doi.org/10.1007/978-3-642-03737-5_49

[73] Zhibo Wang, Mengkai Song, Zhifei Zhang, Yang Song, Qian Wang, and Hairong Qi. 2019. Beyond Inferring Class Representatives : User-Level Privacy Leakage From Federated Learning. In *The 38th Annual IEEE International Conference on Computer Communications (INFOCOM 2019)*. arXiv:arXiv:1812.00535v3

[74] X Wu, M Fredrikson, S Jha, and J F Naughton. 2016. A Methodology for Formalizing Model-Inversion Attacks. In *2016 IEEE 29th Computer Security Foundations Symposium (CSF)*. 355–370. https://doi.org/10.1109/CSF.2016.32

[75] Samuel Yeom, Matt Fredrikson, and Somesh Jha. 2017. The unintended consequences of overfitting: Training data inference attacks. *arXiv preprint arXiv:1709.01604* 12 (2017).

[76] Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. 2015. Understanding neural networks through deep visualization. In *Deep Learning Workshop, ICML*.

[77] Matthew D Zeiler and Rob Fergus. 2014. Visualizing and Understanding Convolutional Networks. In *Computer Vision - {ECCV} 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part {I}*. 818–833. https://doi.org/10.1007/978-3-319-10590-1_53

[78] Quanshi Zhang, Ying Nian Wu, and Song-Chun Zhu. 2018. Interpretable Convolutional Neural Networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. arXiv:1710.00935 http://arxiv.org/abs/1710.00935

[79] Tong Zhang. 2004. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *Proceedings of the twenty-first international conference on Machine learning*. ACM, 116.

A OPTIMIZATION-BASED INVERSION

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.

$$O(\hat{\mathbf{x}}) = \mathcal{L}(F_w(\hat{\mathbf{x}}), F_w(\mathbf{x})) + P(\hat{\mathbf{x}}) \tag{10}$$

where $\mathcal{L}$ is some distance metric such as L2 distance.

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. [66] use L2 norm while Mahendran and Vedaldi [44] demonstrate that a relatively large α produces better result. They choose L6 norm in their experiments. Mahendran and Vedaldi [44] 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.

$$P_{V^\beta}(\mathbf{x}) = \sum_{i,j} \left(\left(x_{i,j+1} - x_{i,j}\right)^2 + \left(x_{i+1,j} - x_{i,j}\right)^2 \right)^{\beta/2} \tag{11}$$

Besides, randomly shifting the input image before feeding it to the network is also used to regularize the optimization in recent work [46]. In addition, Yosinski et al. [76] 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. [21] 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 [44]).

B 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)
)
```

Figure 18: 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()
)
```

Figure 19: 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)
)
```

Figure 20: 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()
)
```

Figure 21: MNIST inversion model architecture.

C HEAD POSE DETECTION

We detect the head pose of CelebA data points by first identifying the facial landmarks of each image and then estimating its head pose by solving the Perspective-n-Point (PnP) problem [38]. The facial landmarks are detected by the FAN network [13], which is a deep learning based approach to detect 68 landmarks of a given facial image. We use a Python library [2] to build the FAN network. The PnP problem is solved by the OpenCV library [12] for Python3 which produces the rotation matrix and translation matrix for the facial landmarks. We convert them to Eular angles (yaw, pitch, and roll) to divide the CelebA datasets into frontal, turn-left, turn-right, turn-down and turn-up faces.

In our experiment, we classify a facial image to "turn left" if its yaw is smaller than -14°, to "turn right" if greater than 14°, to "turn down" if its pitch is smaller than -35°, and to "turn up" if greater than 15°. A facial image is said to be frontal face if its yaw is in $[-14^\circ, 14^\circ]$, its pitch is in $[-35^\circ, 15^\circ]$ and its roll is in $[-10^\circ, 10^\circ]$. We discard images whose Eular angles do not satisfy the above conditions. Eventually, we obtain 109,105 frontal faces, 48,137 turn-left faces, 33,632 turn-right faces, 4,278 turn-down faces and 2,304 turn-up faces.

D PREPROCESSED AMAZON API OUTPUT


```
{
    "Emotions": {
        "CONFUSED": 0.06156736373901367,
        "ANGRY": 0.5680691528320313,
        "CALM": 0.274930419921875,
        "SURPRISED": 0.01476531982421875,
        "DISGUSTED": 0.030669870376586913,
        "SAD": 0.044896211624145504,
        "HAPPY": 0.0051016128063201905
    },
    "Smile": 0.003313331604003933,
    "MouthOpen": 0.0015682983398437322,
    "Beard": 0.9883685684204102,
    "Sunglasses": 0.00017322540283204457,
    "EyesOpen": 0.9992143630981445,
    "Mustache": 0.07934749603271485,
    "Eyeglasses": 0.00090587615966797732,
    "Gender": 0.998325424194336,
    "AgeRange": {
        "High": 0.52,
        "Low": 0.35
    },
    "Pose": {
        "Yaw": 0.398555908203125,
        "Pitch": 0.532116775512695,
        "Roll": 0.47806625366211
    },
    "Landmarks": {
        "eyeLeft": {"X": 0.2399402886140542, "Y": 0.39858236008502207},
        "eyeRight": {"X": 0.5075000426808342, "Y": 0.3512716902063248},
        "mouthLeft": {"X": 0.294372202920132, "Y": 0.7884027359333444},
        "mouthRight": {"X": 0.5111179957624341, "Y": 0.7514958062070481},
        "nose": {"X": 0.26335677944245883,"Y": 0.5740609671207184},
        "leftEyeBrowLeft": {"X": 0.16586835071688794, "Y": 0.33359158800003375},
        "leftEyeBrowRight": {"X": 0.2344663348354277, "Y": 0.27319636750728526},
        "leftEyeBrowUp": {"X": 0.1791416455487736, "Y": 0.27319679970436905},
        "rightEyeBrowLeft": {"X": 0.39377442930565504, "Y": 0.24260599816099127},
        "rightEyeBrowRight": {"X": 0.653192506461847, "Y": 0.24797691132159944},
        "rightEyeBrowUp": {"X": 0.4985808427216577, "Y": 0.21011433981834574},
        "leftEyeLeft": {"X": 0.2108403727656505, "Y": 0.40527320313960946},
        "leftEyeRight": {"X": 0.29524428727196866, "Y": 0.3945644398953052},
        "leftEyeUp": {"X": 0.2320460442636834, "Y": 0.38003991664724146},
        "leftEyeDown": {"X": 0.2409084732415246 2, "Y": 0.4139932115027245},
        "rightEyeLeft": {"X": 0.4582430085197824, "Y": 0.3677093338459096},
        "rightEyeRight": {"X": 0.5775697973907971, "Y": 0.34774452980528486},
        "rightEyeUp": {"X": 0.5040715541995939, "Y": 0.3371239347660795},
        "rightEyeDown": {"X": 0.5091470851272833, "Y": 0.37251352858036124},
        "noseLeft": {"X": 0.2878986010785963, "Y": 0.6362120963157492},
        "noseRight": {"X": 0.40161600660105223, "Y": 0.6085103161791537},
        "mouthUp": {"X": 0.34124040994487825, "Y": 0.705847150214175},
        "mouthDown": {"X": 0.3709446289500252, "Y": 0.8184411896036027},
        "leftPupil": {"X": 0.2399402886140542, "Y": 0.39858236008502207},
        "rightPupil": {"X": 0.5075000426808342, "Y": 0.3512716902063248},
        "upperJawlineLeft": {"X": 0.3066862049649973, "Y": 0.4463287926734762},
        "midJawlineLeft": {"X": 0.36578599351351376, "Y": 0.8324899719116535},
        "chinBottom": {"X": 0.45123760622055803, "Y": 1.0087064474187},
        "midJawlineRight": {"X": 0.8626791375582336, "Y": 0.7551260456125787},
        "upperJawlineRight": {"X": 0.9242277731660937,"Y": 0.348934908623391}
    }
}
```


Figure 22: Preprocessed Amazon API output on an example image.

[2] https://github.com/1adrianb/face-alignment