Model Inversion for Impersonation in Behavioral Authentication Systems

Md Morshedul Islam and Reihaneh Safavi-Naini
Department of Computer Science, University of Calgary, Calgary, AB., Canada

Keywords: Behavioral Authentication System, Substitute Classifier, Inverse Classifier, Impersonation Attack.

Abstract: A Behavioral Authentication (BA) system uses behavioral characteristics of a user that is stored in their behavioral profile, to verify their future identity claims. BA profiles are widely used as a second factor to strengthen password based authentication systems. A BA verification algorithm takes the claimed identity of the user together with their presented verification data, and by comparing the data with the profile of the claimed identity it decides to accept or reject the claim. An efficient and highly accurate verification algorithms can be constructed by training a Deep Neural Network (DNN) on the users' profiles. The trained DNN classifies the presented verification data and if the classification matches the claimed identity, accepts the claim, else reject it. This is a very attractive approach because it removes the need to maintain the profile database that is security and privacy sensitive. In this paper we show that query access to the DNN verification algorithm allows an attacker to break security of the authentication system by constructing the profile of a user in the original training database and succeed in impersonation attack. We show how to construct an inverse classifier when the attacker has black-box access to the DNN's output prediction vectors, truncated to a single component (highest probability value). We use a substitute classifier to approximate the unknown components of the prediction vectors, and use the recovered vectors to train the inverse classifier and construct the profile of a user in the database. We implemented our approach on two existing BA systems and achieved the average success probability of 29.89% and 45.0%, respectively. Our approach is general and can be used in other DNN based BA systems.

1 INTRODUCTION

A *Behavioral Authentication (BA)* system (Shi et al., 2011; Zheng et al., 2011; Frank et al., 2013b; Islam and Safavi-Naini, 2016) constructs a behavioral profile for a user that will be used by the verification algorithm to evaluate a verification request of a user that consists of a claimed identity and some behavioral data. The profile is constructed using the data that is collected during a well-defined activity. The verification algorithm compares the presented behavioral data with the stored profile associated with the claimed identity, and decides to accept or reject the request. BA systems have been used to strengthen password-based authentication system, and provide a range of attractive properties such as protection against credential sharing without the need for additional hardware.

Behavioral profile of a user consists of a set of m d-dimensional vectors over real numbers $\mathbb{R}$, each dimension corresponding to a *feature*, and each vector corresponds to a measurement of behavioral features. The user profile is generated during a trusted registration phase, and is stored at the server. A *verification request* (also called authentication) will consist of a set of m behavioral measurement samples, together with a claimed identity. In a traditional verification algorithm, a distance measure will be used to evaluate "closeness" of the presented verification data (m samples) with the set of samples in the claimed user profile. This requires the server to keep a database of user profiles which will need protection and secure data management because of the security and privacy sensitivity of profile data. An attractive approach to verification without the need for the server to maintain the profile database is to use the profile data to train a classifier such as a DNN classifier, where each user is assigned to one class. The trained classifier will be used to evaluate verification requests by classifying the verification data (Centeno et al., 2017; Deng and Zhong, 2015; Lu et al., 2018). This approach removes the need to store the profile data and significantly improves the system's privacy.

DNN for user authentication. DNNs have been used for biometric and behavioral authentication systems. For example, DNN based authentication systems have been constructed for face data (Schroff et al., 2015), fingerprint data (Pandya et al., 2018), mouse movements (Chong et al., 2019), gait data (Jung et al., 2019), and keystroke data (Deng and

Islam, M. and Safavi-Naini, R.
Model Inversion for Impersonation in Behavioral Authentication Systems.
DOI: 10.5220/0010559802710282
In *Proceedings of the 18th International Conference on Security and Cryptography (SECRYPT 2021)*, pages 271-282
ISBN: 978-989-758-524-1

Zhong, 2015). The DNN classifier learns the mapping between the user data in the *training data set* and a set of classes, each corresponding to a user, by composing a parametric function consisting of the layers of the DNN. The output of a DNN classifier with N classes on each data sample (vector) of BA system is an N dimensional probability vector (sum of components equal to 1.0) that is called the *prediction vector*, where the i^{th} component represents the "belief" of the classifier that the input is in the i^{th} class. The final decision of the verification algorithm will be based on the m samples that are form the verification data. Using DNN for BA system verification has the advantage of allowing partial verification decision to be made on each sample. This can be used for continuous authentication where the user's identity is periodically verified during the authentication session.

Model inversion attacks aim to reconstruct data with the same distribution as the training data of the DNN classifier using partial information about input (training data and/or its domain), output (prediction vectors) and the structure of the original DNN. Output vectors of the DNN can be obtained by sending input to the DNN that will serve as an oracle, and receiving its response. Access to the target classifier can be in *black-box* or *white-box*, where in the former the querier only sees (part of) the prediction vector, while in the latter the full working information of the DNN will be accessible to the attacker. The complexity of model inversion depends on the type of the attacker's access to the DNN, and the data available to them. Model inversion attacks (Yang et al., 2019b) have followed two approaches, *optimization-based* and *training-based*, where the former is restricted to white-box access to the DNN only. We consider black-box access and so focus on the latter approach.

In training-based model inversion attacks (Dosovitskiy and Brox, 2016a; Dosovitskiy and Brox, 2016b; Nash et al., 2019; Yang et al., 2019b; Yang et al., 2019a) a second DNN classifier is trained to act as the inverse of the target DNN classifier. The inverse classifier can then be used to generate the data with the same distribution as the training data of the target classifier. Most existing works assume that attacker has access to the *full prediction vectors* of a subset of the original training data. In (Yang et al., 2019b; Yang et al., 2019a) authors give the construction of an inverse classifier when only a few top components of the prediction vector is known (referred to as *truncated prediction vector*), and the data from the *input domain* (not the training data) are available. The input data in their work is face image data.

Our Work. We consider a setting where a BA system with a DNN based verification algorithm is used for user authentication. The attacker can send verification queries, receive the responses, and accept the associated confidence value. We assume the system description, that is the format of profile and the general structure of the DNN verification algorithm, is known. However, important information including the training data set and and the values of the parameters of the trained DNN are not accessible. A user can send behavioral samples to the verification system, and for each, receive a *truncated prediction vector* where only the largest component is non-zero and all other components are zeroed. The goal of the attacker is to impersonate a registered user by constructing behavioral samples for the verification claim that are accepted by the DNN based verification algorithm with sufficiently high probability. Verification data has the same format as the training data and so impersonation attack can be seen as a model inversion attack where the goal of the attacker is to construct behavioral samples of a user.

Challenges. The response of the verification oracle to a query is a prediction vector that has a single non-zero component. To train an inverse classifier, however, the complete prediction vector is required. The main challenge in our work is to reconstruct the full prediction vector from the output vector of verification oracle that has a single non-zero component. A similar attack setting had been considered in (Yang et al., 2019b; Yang et al., 2019a) for face authentication system. Authors however allowed the adversary to have access to the top 5-10 components of the prediction vector, which effectively contains most of the probability mass and captures the relationship of the presented sample to all "close" neighbouring classes. Considering only a single non-zero component significantly reduces the available information of the adversary. Additionally, unlike (Yang et al., 2019b; Yang et al., 2019a) where samples of domain data (not the training data) can be easily obtained through internet searches, in our setting, one needs to generate samples of behavioral data by running the BA registration algorithm for different users (that will be different from the registered ones) to collect data.

Our Approach. The attacker's goal is to construct fraudulent verification data for a user (i.e., specified index in the prediction vector) that will be accepted by the verification algorithm. We achieve this goal by constructing an inverse classifier. At a high level the attack has the following steps: (1) generate domain data that will be used as queries to the target DNN classifier, (2) design a substitute classifier to complete the truncated prediction vectors that are the output of the target classifier, (3) use the completed prediction

vectors to train the inverse classifier in conjunction with the target classifier, and (4) construct impersonation data. More details on each step is given below.

1. *Generating Data.* To generate the required profile data, referred to as *attack data* to distinguish them from the DNN original training data, the attacker will use the profile generator of the BA system (software would be publicly available). They may use outsourcing services such as Amazon Mechanical Turks (AMT) to generate the required attack data. If needed, the generated data can be expanded using artificial data generation methods, including probabilistic models or classification-based imputation models. In our experiment, we used published data of two existing systems (Frank et al., 2013b; Islam and Safavi-Naini, 2016) and used to oversample them to allow training of the inverse DNN. The attack data is partitioned into two sets, one used for the training of the substitute classifier, and the other used for the training of the inverse classifier.

2. *Constructing a Substitute Classifier.* We used a k-nearest neighbors (k-NN) classifier as the substitute classifier for the target DNN classifier. The two classifiers will have the same number of classes. Training the substitute classifier needs labelled training data. We used oracle access to the target DNN classifier to label half of the attack data that was allocated to the substitute classifier. The k-NN classifier was trained using the labelled data. This effectively creates a one-to-one correspondence between the prediction vectors of the target classifier and the k-NN classifier. Once the substitute classifier is trained (the highest output index of the target classifier matches the highest index of the substitute classifier), for each class i we will use samples of the attack data to obtain the 5-10 (neighbouring) classes with the highest probabilities. These are the most similar classes to i in the DNN classifier.

3. *Constructing Inverse Classifier.* We assume that the structure of the target classifier, that is the number of layers and their types, are known to the attacker. The inverse classifier will have the same layers, in reverse order. (We note that the approach can also be used when the structure of the target classifier is not known, in which case the attacker will experiment with different number and types of layers, to arrive at a good inverse classifier.)

To train the inverse classifier, each vector in the data set is used to query the target classifier (oracle access) to obtain the truncated prediction vector. The same vector will then be inputted to the substitute classifier to reconstruct the missing components of the prediction vector. The sample is dropped if the top components of the prediction vectors of the two classifiers do not correspond to the same index value. The reconstructed prediction vectors are given as input to the inverse classifier to obtain the data samples. An error is measured based on the input and output of the target classifier and the inverse classifier. The parameters (weights) of the inverse classifier are updated based on this error, and the process is repeated until the error is sufficiently low.

4. *Constructing Impersonation Data.* A fraudulent verification profile consists of m distinct data samples[1]. The trained inverse classifier needs an appropriate input prediction vector to construct the corresponding data sample. We noted that the structure of the input prediction vectors of the inverse classifier is the same as the structure of the prediction vector of the DNN classifier. To construct the first prediction vector of a target user at index i, the first step is to assign a high probability value (close to 1.0) to the target class i. Next, we will use the top 5-10 neighboring classes of i, that were generated using the substitute classifier, to complete the prediction vector by using the non-zero components to correspond to the neighbouring classes of i, and the values of each component to be proportional to the corresponding values of the neighbouring class. The remaining $m-1$ data samples are generated by updating the probability values of the initially chosen prediction vector.

Experimental Results. We implemented and evaluated our proposed approach for two existing BA systems: *Touchalytics* (Frank et al., 2013b) and eDAC (Islam et al., 2021) which is an extended version of *DAC (Draw A Circle)* (Islam and Safavi-Naini, 2016). The approach however is general and can be used in other BA systems that use DNN for verification decision. In each case, we divided the profiles of the BA systems into two sets: the first set of profiles is used to train a DNN classifier, and the second set of profiles is used as attack data. Each DNN classifier has 4-5 *stacks of dense layers* along with their activation functions. A *softmax function* layer uses as the last layer to return the output probability values in the *prediction vector*. The inverse DNN classifier has almost the same but inverse architecture of the target DNN classifier.

We compared the output distribution similarity of the k-NN classifier with the output distribution of the DNN classifier. For the same input, on average, there are 2-3 and 5-7 common classes in the top 5 and top

[1]Distinctness is necessary to prevent the trivial attack where an attacker repeats a single vector multiple times. In practice this attack can be prevented using different approaches by the verification algorithm (e.g., requiring presented vectors to have a minimum distance) that will use the characteristics of the training data.

Table 1: List of notions.

Notation	Meaning
C	DNN classifier
S	Substitute classifier
$InvC$	Inverse classifier
$\mathbf{x}$	Input vector of C and S
$\hat{\mathbf{y}}$	Prediction vector of C and S
$\mathbf{y}$	Ground truth vector
$\mathbf{N}$	Number of classes in C and S
$\mathbf{v}$	Input prediction vector of $InvC$
$\hat{\mathbf{x}}$	Output vector of $InvC$
$\mathcal{D}$	Training data of C
$\mathcal{D}_1$	Training data of S
$\mathcal{D}_2$	Training data of $InvC$
E	Data reconstruction error

10 classes of the two prediction vectors. We use this to recover the missing information of Truncated Prediction Vector, which will be used to train the inverse classifier. The training process stops when the average reconstruction error is around 0.01. For a target user associated with class c_i in the output prediction vector, we used probability value 0.98 in c_i and distributed the remaining probability values over the 5-10 neighboring classes using substitute classifier output. Our experimental results show that around 45.0% of Touchalytics and 29.0% of eDAC users are vulnerable to impersonation attack.

Notations. Table 1 summarizes the notations used in this paper.

Paper Organization. Section 2 describes the preliminaries and related works. Section 3 is about the model inversion and implementation attack. Section 4 gives details of the experimental results. Finally, Section 5 concludes the paper.

2 PRELIMINARIES AND RELATED WORKS

DNN Classifier. In a DNN classifier, a neuron is an elementary computing unit that uses a set of inputs, a set of weights and an activation function to translate inputs into a single output, which can then be used as input to another neuron. While the details can vary between neural networks, the function $f_i(\mathbf{w}_i, \mathbf{x})$ is commonly a weighted function in the form of $\mathbf{w}_i\mathbf{x}$. The weights of each neuron are tuned during the training stage such that the final network output (prediction vector $\hat{\mathbf{y}}$) is biased towards the value of the ground truth vector $\mathbf{y}$ (the expected output vector). The non-linear behavior in a neural network is obtained by using an activation function (often uses a sigmoid function) to which the output of f_i is passed and modified. This non-linear behavior allows neural networks to describe more complicated systems while still combining inputs in a simple fashion.

Definition 2.1. *A DNN classifier C uses a hierarchical composition of n parametric functions f_i to model an input vector* $\mathbf{x}$ *where each f_i is modelled using a layer of neurons, and parameterized by a weight vector* $\mathbf{w}_i$. *The last layer of C uses a softmax function* $\sigma(.)$ *(for multi-class classification) to encode the belief that the input belongs to each class of the DNN classifier, and give the probability values in a prediction vector* $\hat{\mathbf{y}}$.

$$\hat{\mathbf{y}} = C(\mathbf{x}) = \sigma(f_n(\mathbf{w}_n, \cdots f_2(\mathbf{w}_2, f_1(\mathbf{w}_1, \mathbf{x})))) \quad (1)$$

If input vector $\mathbf{x}$ is from user u_i then in ground truth vector $\mathbf{y}$ the probability value $\mathbf{y}[i] = 1.0$, and the probability value of other classes is 0.0. For $\mathbf{x}$, a good classifiers C will have a very high probability value in $\hat{\mathbf{y}}[i]$, and non-zero probability values in the neighbor classes of $\hat{\mathbf{y}}[i]$, where $\sum \hat{\mathbf{y}}[i] = 1.0$. An error function measures the (distribution) difference between $\mathbf{y}$ and $\hat{\mathbf{y}}$. For a profile $\bar{\mathbf{X}}$ with m data samples, the m truncated prediction vectors of the DNN classifier will form a *prediction table* (see Figure 1).

Performance of a DNN classifier is measured by rank-t ($t \geq 1$) accuracy. For *rank-1 accuracy* of C, the top class of the prediction vector is the same as the top class of the corresponding ground truth vector, and for rank-t accuracy the top class is in the top t classes of the ground truth vector.

Inverse Classifier. Model inversion attack constructs and trains an inverse classifier for a target DNN classifier to reconstruct the data with the same distribution as the training data of a target DNN classifier (Dosovitskiy and Brox, 2016a; Dosovitskiy and Brox, 2016b; Nash et al., 2019; Yang et al., 2019a; Yang et al., 2019b). Model inversion attack is also used to predict the sensitive attributes (Hidano et al., 2018; Wu et al., 2016) of the target classifier's training data. One can also use optimization-based data reconstruction approach (Fredrikson et al., 2015; Linden and Kindermann, 1989; Mahendran and Vedaldi, 2015) to reconstruct the data. Gradient-based optimization requires white-box access to the target DNN classifier while training-based model inversion can work in *black-box* setting.

Training-based model inversion approach trains a new DNN classifier which is the inverse of the target DNN classifier. The inverse DNN classifier takes the output of the target DNN classifier as input, and outputs the input of the target DNN classifier. The training process minimizes the average error between

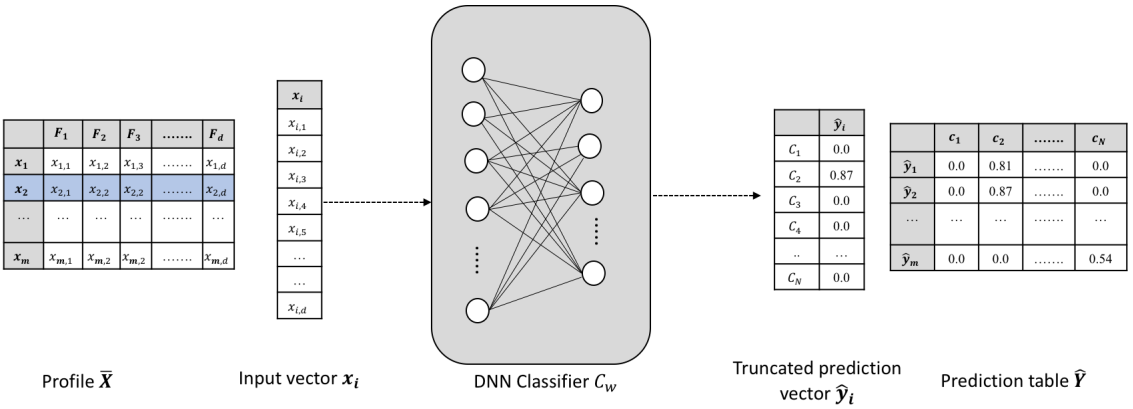


Figure 1: For a profile $\bar{\mathbf{X}}$ with m data samples, the DNN classifier generates m truncated prediction vectors. In each truncated prediction vector, the largest component is non-zero and all other components are zeroed.

the output of the inverse classifier and the input of the target classifier, where the average is over the set of all training data samples of the inverse classifier. Training-based model inversion is a one-time effort for training the inverse classifier. Once the classifier is trained, the trained inverse classifier is used the prediction vectors to generate the artificial profile data.

Substitute Classifier. *Model extraction attack* (i) builds a *substitute classifier* (Papernot et al., 2017) of a target DNN classifier, or (ii) extracts the model (target classifier) parameters (Tramèr et al., 2016). Both attacks can be in black-box setting, where the target DNN classifier works as an oracle. We use substitute classifier to reconstruct the truncated prediction vectors of the verification oracle. The substitute classifier will have the same number of classes as the target classifier, and will use the same input-output format. For each input the substitute classifier maximizes the similarity of its output (distribution) with that of the target DNN classifier. To train the substitute classifier, the attacker needs labelled training data. This can be obtained from the output of the oracle. In (Papernot et al., 2017), it is assumed that the attacker has access to a subset of training samples of the target classifier. In this case, the data samples and their predicted class labels returned by the oracle is sufficient to train the substitute classifier. In our case, the attacker only knows the domain of the data for the training data of the substitute classifier.

k-NN classifier. We use k-NN classifier (Cover and Hart, 1967) as a substitute classifier for the target DNN classifier. To estimate the prediction vector $\hat{\mathbf{y}}$ by k-NN classifier, we use the approach of (Mandelbaum and Weinshall, 2017). The Equation (1) of (Mandelbaum and Weinshall, 2017) estimates the probability value of the prediction vector $\hat{\mathbf{y}}$.

3 ATTACK MODEL

Attacker's Knowledge. We assume that the attacker has the following knowledge and capabilities for impersonating a target user:

1. The attacker does not know the training data (profiles) and parameters of the target classifier. The attacker knows the BA system and the algorithm used during the registration phase, and the general structure of the DNN based verification system.
2. The target classifier also works as an oracle, and the attacker can send the behavioral samples to the oracle to receive the oracle response. The output of the oracle will be a truncated prediction vector where only the largest component is non-zero, and all other components are zeroed.
3. The profile generator of the BA system is publicly available. Here, profile generator is a software that is used by the BA system to collect behavioral data from the users. The attacker will outsource the profile generator to collect attack data from the users whose profiles are not used to train the classifier.
4. From attack data, the attacker will be able to know the input format of the target classifier. From Truncated Prediction Vector, the attacker will be able to know the output format of the target classifier as well as the total number of class in the classifier.

3.1 Model Inversion Attack

We use the training-based approach to train an inverse classifier *InvC* as (i) it will help to produce artificial data samples in a black-box setting, (ii) one can

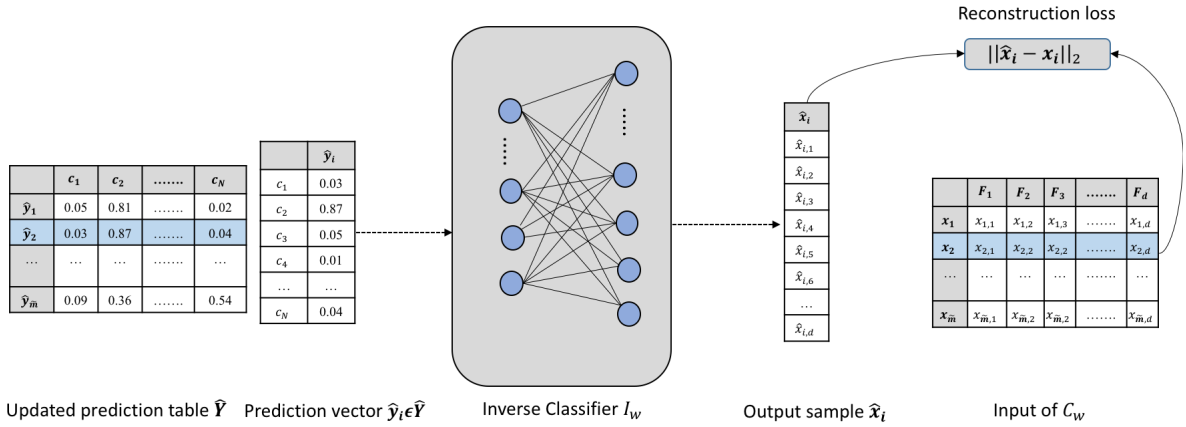


Figure 2: For $\mathbf{x}_i$, the *InvC* takes the updated prediction vector $\hat{\mathbf{y}}_i$ as input, and output the artificial data sample $\hat{\mathbf{x}}_i$. The reconstruction error is L2 difference between $\mathbf{x}_i$ and $\hat{\mathbf{x}}_i$.

use attack data to train the inverse classifier, and (iii) it could achieve substantial inversion accuracy over other approaches. During training, the attack data transfer the knowledge about the decision boundaries of the target classifier to the inverse classifier by the prediction vectors. However, the Truncated Prediction Vector with a single component misses important information about the decision boundaries of the target classifier. We will use a substitute classifier S of the target classifier C to fill the zero components of the truncated prediction vectors. For both S and *InvC*, the attacker will have oracle access to C. One can reduce the number of oracle queries by making additional assumptions about attacker's knowledge. For example, the authors of (Mo et al., 2020) assume hidden layers of similar model of C is known and use them in *InvC*. We leave reduction of queries as a direction for future research. Details about each step is given below:

Step1: Generating Attack Data. A BA system has a profile generator algorithm that collects the user's behavioral data and constructs the profile. Profile generators are usually mobile applications (software), and they are publicly available. The attacker can use the software to collect the required attack data using outsouring platform such as AMT. This data will be used to train the substitute classifier, and to train the inverse classifier. Let, $\mathcal{D}_1$ and $\mathcal{D}_2$ be two disjoint sets that are constructed from the attack data and are used to train substitute and inverse classifier, respectively.

Step 2: Constructing and Training a Substitute Classifier. To construct and train a substitute classifier S that approximates the target classifier C, the information of the target classifier will be used. We choose k-NN classifier as substitute classifier of the target classifier. The benefits of using k-NN classifier are that (i) k-NN classifier requires relatively small amount of (training) data for each classes, and (ii) it is easy to implement. The performance of the k-NN classifier is also compatible with the performance of other classifiers. One can also use the same architecture of the original DNN classifier for the substitute classifier, but it would need significantly more training data.

Label the Training Data. Training the k-NN classifier needs labelled training data. We use oracle access to the target classifier to label data samples in $\mathcal{D}_1$. For a data sample $\mathbf{x}_i \in \mathcal{D}_1$ if the oracle response $\hat{\mathbf{y}}[i]$ has a non-zero component that is higher than a predetermined threshold T, the sample will be labelled as c_i. The process will be repeated for all attack data samples in $\mathcal{D}_1$.

Output Distribution Similarity. Most components of the prediction vectors of a DNN classifier are negligible, and the top 5-10 probability values have the most of the probability mass (Srivastava et al., 2015). This means that the distributions similarity of the outputs of S and C depend on the probability values of the t $(t = 5-10)$ top-ranked classes. We use Definition (3.1) for distributions similarity. The closer the output distributions of S and C are, the more accurate will be the recovery of the information.

Definition 3.1. *Let C be a classifier, and S be a substitute classifier of C. Both have the same number of classes, and their input-output format are also the same. Let $\mathcal{D}$ be a set of training data of C, and $\mathcal{D}_1$ be a set of training data of S. For each class $c_i \in C$ there is a corresponding class $c_i \in S$, and their training data are also close to each other.*

For an input vector $\mathbf{x}$, let the prediction vectors of C and S is $\hat{\mathbf{y}}$, and $\hat{\mathbf{y}}'$, respectively. For the output distributions similarity, both prediction vectors will satisfy the following two conditions with high probability:

1. *If the highest probability value of* $\hat{\mathbf{y}}$ *is in* $\hat{\mathbf{y}}[i]$ *then the highest probability value of* $\hat{\mathbf{y}}'$ *will be in* $\hat{\mathbf{y}}'[i]$.
2. *Let the next* $t-1$ *highest probability values of* $\hat{\mathbf{y}}$ *be* $\hat{\mathbf{y}}[i_1], \hat{\mathbf{y}}[i_2], \cdots, \hat{\mathbf{y}}[i_t]$. *Then most of* $\hat{\mathbf{y}}'[i_1], \hat{\mathbf{y}}'[i_2], \cdots, \hat{\mathbf{y}}'[i_t]$ *will hold the next* $t-1$ *highest probability values of* $\hat{\mathbf{y}}'$.

Because of the small values of most of the components, we only need to recover the non-negligible components.

Step 4: Reconstruct Truncated Prediction Vector. For a data sample $\mathbf{x}$, let $\hat{\mathbf{y}}$ and $\hat{\mathbf{y}}'$ have the highest probability value on the same class index (class index i for user u_i), and let $\hat{\mathbf{y}}[i] = \alpha$. The remaining probability mass $\alpha' = 1-\alpha$ will then be distributed over the next $t-1$ top classes of $\hat{\mathbf{y}}$ that will be obtained using the neighbouring classes of i in $\hat{\mathbf{y}}'$. Let the next top $t-1$ classes of $\hat{\mathbf{y}}'$ be $\hat{\mathbf{y}}'[i_1], \hat{\mathbf{y}}'[i_2], \cdots, \hat{\mathbf{y}}[i_{t-1}]$. Then $\hat{\mathbf{y}}[i_j] = \frac{\hat{\mathbf{y}}'[i_j]}{\alpha'}, j = 1, 2, \cdots, t-1$. Here, closer neighboring class will have higher probability.

Step 5: Constructing and Training the Inverse Classifier. The inverse classifier has its input and output format as inverse of the target DNN classifier. In our experiments, we assumed that the structure of the target DNN classifier, the number of layers and number of nodes in each layer are known to the attacker. (This assumption can be relaxed in practice where the attacker will experiment with different number and types of layers to arrive at a good inverse classifier.)

The inverse classifier has its layers in reverse order of the target classifier; that is, the last layer of the DNN classifier (except the softmax layer) will form the first layer of the inverse classifier, and in general, the j-th layer to the last of the target classifier will become the j-th layer of the inverse classifier. The layers and number of nodes in each layer of the inverse classifier can be modified to optimize the performance of the inverse classifier.

Training Inverse Classifier. Let for a data sample $\mathbf{x}_i \in \mathcal{D}_2$, the output of the oracle be a Truncated Prediction Vector $\hat{\mathbf{y}}_i$. We update $\hat{\mathbf{y}}_i$ using the substitute classifier and assign some random values in the weights (parameters) of the inverse classifier. Let for updated $\hat{\mathbf{y}}_i$, the output of the inverse classifier is $\hat{\mathbf{x}}_\mathbf{i} = InvC(\hat{\mathbf{y}}_\mathbf{i})$, where $\hat{\mathbf{x}}_\mathbf{i}$ is a new (artificial) data sample. So, the initial reconstruction error E (L2 norm) for the data sample $\mathbf{x}_i$ is $E = ||\mathbf{x}_i - \hat{\mathbf{x}}_i||_2$. The average initial reconstruction error for all data samples of $\mathcal{D}_2$ is

$$avg(E) = \frac{1}{|\mathcal{D}_2|} \sum_{i=1}^{|\mathcal{D}_2|} ||\mathbf{x}_i - \hat{\mathbf{x}}_i||_2. \quad (2)$$

The goal of the attacker is to minimize this average reconstruction error. Based on the estimated average error by Equation (2), a backward propagation will update the weights of the inverse classifier. We repeat the process for the data samples of $\mathcal{D}_2$ to find a set of weights of the inverse classifier, which will bring the reconstruction error sufficient low. Figure 2 shows the training process of the inverse classifier.

3.2 Impersonation Attack

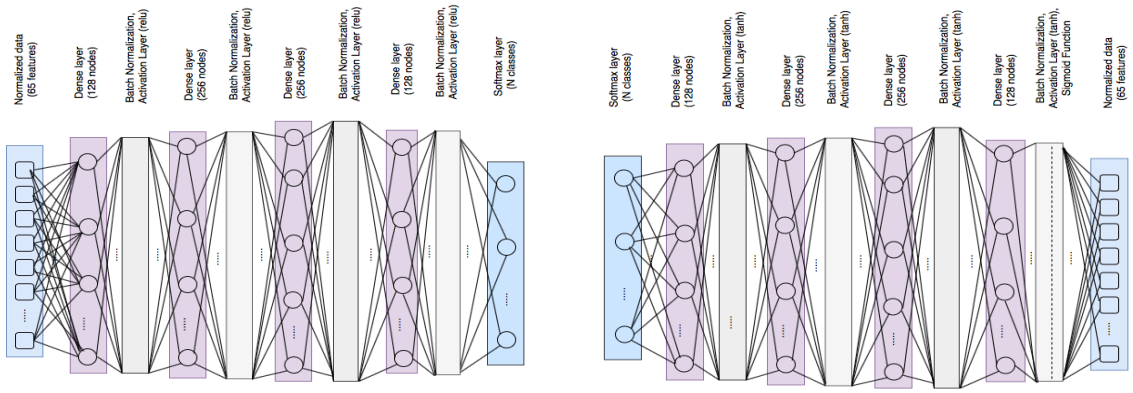
The verification profile requires a set of m distinct behavioral samples. The trained inverse classifier needs an appropriate input prediction vector $\mathbf{v}$ to construct each data sample. The vector $\mathbf{v}$ will have dimension N and t non-zero components. For user u_i, we will assign $\mathbf{v}[i] = \alpha$ (a probability value close to 1.0) and the remaining probability mass $1-\alpha$ will be distributed among $t-1$ "closets" neighbours of c_i. The procedure to assign values to each non-zero component of the vector is described in Section 3.1, Step 4.

Let for $\mathbf{v}_i$ the output of the inverse classifier be $\hat{\mathbf{x}}_i$. The attacker will count $\hat{\mathbf{x}}_i$ as an element of the verification profile if the oracle response to $\hat{\mathbf{x}}_i$ has the non-zero component (probability value) with sufficiently high value. The attacker will use $\mathbf{v}_i$ to produce other close, yet distinct, prediction vectors. Let $\mathbf{v}_j$ is a new prediction vector which is generated by slightly changing the probability of $\mathbf{v}_i$. For $\mathbf{v}_j$ the output of the inverse classifier $\hat{\mathbf{x}}_j$ is accepted as a new sample if (i) the oracle output on $\hat{\mathbf{x}}_j$ is class j with sufficiently high confidence, and (ii) if the distance $||\mathbf{x}_i - \hat{\mathbf{x}}_j||_2$ is higher than a preset value.

Let for a target user u_i, $\hat{\mathbf{X}} = \{\hat{\mathbf{x}}_1, \hat{\mathbf{x}}_2, \cdots, \hat{\mathbf{x}}_m\}$ is a set of m artificial data samples. The attacker will then use the profile $\hat{\mathbf{X}}$ to initiate a false verification request $(u_i, \hat{\mathbf{X}})$. The verification algorithm of the BA system will accept the claim if for $\hat{\mathbf{X}}$ the rank-1 accuracy of C is higher than a predefined threshold.

4 EXPERIMENTAL RESULTS

We have implemented and evaluated our proposed approach on two existing BA systems: Touchalytics (Frank et al., 2013b), and extended version of DAC, eDAC (Islam et al., 2021). *Touchalytics* uses users' touch data (up-down and left-right scrolling) when interacting with the profile generator. We downloaded Touchalytics data from (Frank et al., 2013a) and it has the data from 41 distinct users. *eDAC* uses behavioral features of users that are collected while drawing random challenge circles that are presented to them to verify their verification request. We downloaded eDAC data from (Islam and Safavi-Naini, 2021) that has the data of 195 distinct users.



(a) (b)

Figure 3: (a) DNN architecture of eDAC classifier. It has multiple stacks of dense, relu, and batch-normalization layers. A softmax function in the last layer of the classifier outputs the prediction vector. (b) DNN architecture for the inverse classifier of eDAC classifier. The architecture of the inverse classifier is almost the same, but opposite to the corresponding classifier. Touchalytics classifier and its inverse classifier follows almost same structure (see Figure 7 in Appendix).

Experiment Setup. We downloaded and cleaned[2] Touchalytics data before using them. eDAC did not require data cleaning. We then combined the registration and verification data of each user, for both BA systems, and applied a shuffling on them before using the data in the experiments. This combining of data will give us 186-1230 data samples of dimension 30 in each profile of Touchalytics, and 80-240 data samples of dimension 65 in each profile of eDAC. To reduce the effect of biases that are the result of features having different ranges, we normalized the feature values so that all feature ranges coincide with [0,1]. From each profile, we separated 20.0% of data for testing purposes.

Data Oversampling. We need sufficient data in each profile for DNN experiments. We used Synthetic Minority Over-sampling Technique (SMOTE) (Chawla et al., 2002) in the remaining data sample to increase the profile size. SMOTE is an oversampling algorithm that generates new data which lies between any two existing data samples of a profile. In our experiments, we ensured that both BA systems have a minimum of 1000 data samples per profile. After applying SMOTE, we also confirm the correctness and security of both BA systems. For that, we train two BA classifiers for two BA systems by the over-sampling profiles, and then testing them by the data that were separated before. In both systems, correctness and security remained nearly the same.

Training Data of DNN Classifier and Attack Data. We divided all profiles of each BA system into two equal groups (drop one profile from each BA system to keep the number of profile same in each group of both BA systems): (i) *group 1*- all profiles of this group are used to train and validate the DNN classifier, and (ii) *group 2*- all profiles of this group are used as attack data. In each group, there are 20 Touchalytics profiles and 97 eDAC profiles, respectively.

Prediction Vector of k-NN Classifier. We used Equation (3) for the probability values estimation of the prediction vector of k-NN classifier.

$$\hat{\mathbf{y}}[i] = \frac{\sum_{j=1,\mathbf{x}_j \in c_i}^{k} e^{-||\mathbf{x}-\mathbf{x}_j||_2}}{\sum_{j=1}^{k} e^{-||\mathbf{x}-\mathbf{x}_j||_2}}, \quad (3)$$

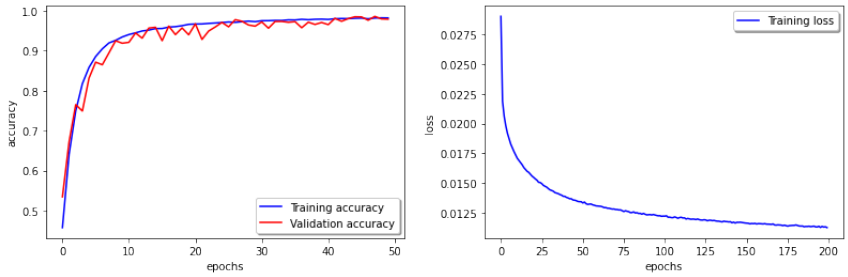
where $\hat{\mathbf{y}}[i]$ represent the probability value that the data sample $\mathbf{x}$ is in class c_i. For Equation (3), we update the Equation (1) of (Mandelbaum and Weinshall, 2017). This is because, in our case all data samples of both BA systems are in the features' space. For our impersonation attack, we run the following experiments: (i) design and train the DNN based classifier, (ii) design and train the inverse classifier, and (iii) construct artificial profiles for impersonation attack.

4.1 DNN Classifier Design and Training

This section is about to design the architectures of DNN classifiers for both BA systems. We then train and validate both classifiers by the data samples of group 1. We also test their performance by test data.

Experiment 1: Design DNN Classifier Architecture. To confirm the effectiveness of our proposed approach, we need the DNN based classifier. For both Touchalytics and eDAC classifier, we design two separate hierarchical DNN architectures. The idea to build a hierarchical model is that each higher level layer in a DNN classifier captures more complex nonlinear features from the data. Both architectures have

[2]We replace 'NaN' and 'Infinity' by zero, and dropped the 'doc id', 'phone id', and 'change of finger orientation' columns.



(a) (b)

Figure 4: (a) Training and validation accuracy of eDAC classifier. In 50 epochs of training, the classifier can achieve 98.16% of training, and 97.85% of validation accuracy, respectively. (b) Data reconstruction error of eDAC based inverse classifier. The inverse classifier reduced the error to 0.011 in a reasonable number of epochs. Touchalytics classifier and its inverse classifier has almost same pattern (see Figure 8 in Appendix).

multiple dense, batch-normalization, and activation (ReLU) layers. The dense layer provides the learning features from all combinations of the input features of previous layer. The ReLU layer works as an activation function of the soft boundary, and the batch-normalization layer normalizes the data. A Touchalytics classifier has five stacks of dense, ReLU and batch-normalization layers, and each layer has 64, 128, 256, 128, and 64 nodes, respectively. On the other hand, the eDAC classifier has four stacks of dense, ReLU and batch-normalization layers with 128, 156, 256, and 128 nodes, respectively. A *softmax function* layer is used as the last layer of each classifier to represent the output probability distribution in a prediction vector. Figure 3 (a) shows the architectures of DNN based eDAC classifiers (see Figure 7 (a) in Appendix for Touchalytics classifier).

Experiment 2: Training the DNN Classifier. From the data samples of each profile of group 1, we use 80.0% of data for training and 20.0% of data for validation purpose. In 50 epochs of training both Touchalytics and eDAC classifier achieve 96.28%, and 98.16% classification accuracy, and 95.27%, and 97.85% validation accuracy, respectively. See Figure 4 (a) and Figure 8 (a) (in Appendix) for the training and validation accuracy of eDAC and Touchalytics classifier, respectively. Each epoch takes only 3-4 seconds and 8-9 seconds in both Touchalytics and eDAC classifier. For the test data that were separated earlier, the classification accuracy of both DNN classifiers are 95.31% and 97.97%, respectively.

To ensure that none of the profiles of group 2 (attack data) are used to train their corresponding DNN classifier, we estimate the classification accuracy of both DNN classifiers by using the profiles of group 2. In this case, Touchalytics and eDAC classifier have only 1.91% and 1.01% classification accuracy. For the attack data, this result is what we expect to get.

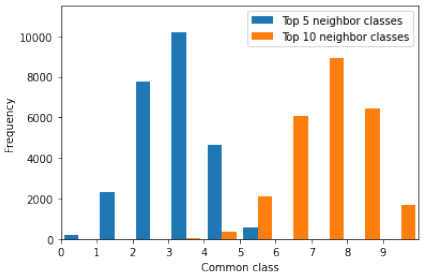
4.2 Inverse Classifier Design and Training

This section is about to design the architecture of inverse classifier, recover the missing components of Truncated Prediction Vector by substitute classifier and train the inverse classifier by the reconstructed prediction vectors.

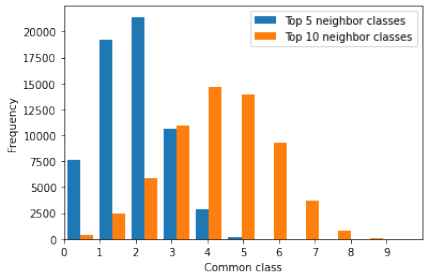
Experiment 1: Design Inverse Classifier Architecture. The input-output format of both inverse classifiers are the same as their corresponding classifiers' output-input format. We use multiple stacks of dense layers, batch-normalization layers, and tanh activation layers in each inverse classifier design. The last layer of each inverse classifier uses *sigmoid function* layer to keep the output in [0,1]. The architecture of both inverse classifiers are almost the same but opposite of their corresponding target classifiers. Figure 3 (b) and Figure 7 (b) (in Appendix) shows the DNN architectures of eDAC based inverse classifier andTouchalytics based inverse classifier.

Experiment 2: Measure Output Distribution Similarity. For each class of k-NN classifier, we use a subset of data samples of group 2 (attack data samples in $\mathcal{D}_1$) that were labelled by the oracle. The input-output dimension of Touchalytics and eDAC based substitute classifiers is 30 and 65, 20, and 97, respectively.

We estimated the output similarity of the k-NN and target DNN classifier based on their top t classes. For each $\mathbf{x} \in \mathcal{D}_1$, almost all top classes of the prediction vectors of both classifiers satisfy the condition (1) of Definition 3.1. Moreover, in the top 5 classes of two prediction vectors of two classifiers, on average there are 2-3 common classes. If we take the top 10 classes from each prediction vector of both classifiers, there are on average 5-7 common classes (see Figure 5). We use these results to update the missing components of $\hat{\mathbf{y}}$ of the DNN classifier.

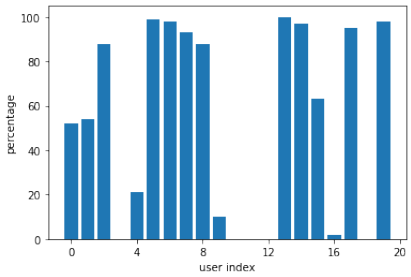


(a)

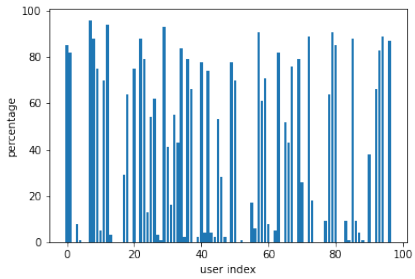


(b)

Figure 5: For same input, the common classes in the prediction vector of substitute and DNN classifier. In the top 5 classes of two prediction vectors, on average there are 2-3 common classes and there are 5-7 common classes in the top 10 classes.



(a)



(b)

Figure 6: Rank-1 accuracy of an artificial verification data $\hat{\mathbf{X}}$. If for acceptance, $\hat{\mathbf{X}}$ needs minimum 70.0% of rank-1 accuracy then 45.0% of Touchalytics and 29.89% of eDAC users are vulnerable to impersonation attack.

Experiment 3: Training the Inverse Classifier. To train the inverse classifier, we use the updated prediction vectors. We use "mean-square-error" as the error function between the inverse classifier's output and the DNN classifier's input. To reach a reasonable average reconstruction error, Touchalytics based inverse classifier took 50 epochs (each epoch took 2-3 seconds), and eDAC based inverse classifier took 200 epochs (each epoch took 7-8 seconds). The average error in both Touchalytics and eDAC based inverse classifier are 0.010 and 0.011, respectively. Figure 4 (b) shows the error of eDAC inverse classifiers. Touchalytics inverse classifier has almost same error (see 8 (b) in Appendix). For less number of features, Touchalytics-based inverse classifier takes fewer epochs during training the inverse classifier compare to the eDAC based inverse classifier.

4.3 Impersonation Attack

For verification data, we first generate the input prediction vectors and then use them in the trained inverse classifier. Detailes are given below.

Experiment 1: Input Prediction Vector Generation. For a target user, we generate the input prediction vector $\mathbf{v}$ by assigning a highest probability value to the corresponding class i of user u_i and then by distributing the remaining probability values to the top 5-10 neighbour classes of the target user class. We alter the probability value of each target class from 0.80 to 1.0 (for probability value 1.0, there is only one input vector) and found that the higher the probability value in the target user's class is, the more chance of acceptance (by the DNN verification algorithm) the corresponding verification data has. In Touchalytics, for the probability value in the range 0.80 to 1.0, the percentage of artificially generated verification data that are accepted by the DNN verification algorithm is increased from 30.47% to 58.06%. In the case of eDAC, it varies from 23.60% to 50.94%.

Experiment 2: Impersonation Attack. For a fraudulent claim $(u_i, \hat{\mathbf{X}})$, we generate input prediction vector by assigning $\mathbf{v}[i] = 0.98$, and then distributing remaining probability values to the top 5 (in Touchalytics), and top 10 (in eDAC) neighbour classes of target class. There are on average 600-650 data samples in each artificial profile. For Touchalytics, we generate 20 artificial profiles and 97 artificial profiles for eDAC and use them to impersonate the uers' of the DNN classifier. Figure 6 shows the percentage of data samples in each $\hat{\mathbf{X}}$, which is accepted by the DNN based verification algorithm. Here, all artificial verification profiles of both BA systems are not equally accepted. For $(u_i, \hat{\mathbf{X}})$, if an accept decision requires minimum 70.0% rank-1 classification accuracy then 45.0% of

Touchalytics users, and 29.89% of eDAC users are vulnerable to impersonation attack.

5 CONCLUDING REMARKS

We constructed and trained the inverse classifier in a black-box setting by using attack data. The substitute classifier helped the attacker to recover the missing information of truncated prediction vectors. We generate input prediction vectors for the trained inverse classifier for the fraudulent verification claim, where every vector has the highest probability value in the target user's class. In our experiments, we used Touchalytics and eDAC data and achieved an accepted success rate in impersonation attack. Our work raises a number of research questions, including the design of more efficient attacks by improving the substitute and inverse classifiers. Protecting DNN classifier of the BA system from this type of attack can be another future research direction.

ACKNOWLEDGEMENT

This work is in part supported by Natural Sciences and Engineering Research Council of Canada and Telus Communications Industrial Research Chair Grant.

REFERENCES

Centeno, M. P., van Moorsel, A., and Castruccio, S. (2017). Smartphone continuous authentication using deep learning autoencoders. In *2017 15th Annual Conference on Privacy, Security and Trust (PST)*, pages 147–1478. IEEE.

Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. (2002). Smote: synthetic minority oversampling technique. *Journal of artificial intelligence research*, 16:321–357.

Chong, P., Elovici, Y., and Binder, A. (2019). User authentication based on mouse dynamics using deep neural networks: A comprehensive study. *IEEE Transactions on Information Forensics and Security*, 15:1086–1101.

Cover, T. and Hart, P. (1967). Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27.

Deng, Y. and Zhong, Y. (2015). Keystroke dynamics advances for mobile devices using deep neural network. *Recent Advances in User Authentication Using Keystroke Dynamics Biometrics*, 2:59–70.

Dosovitskiy, A. and Brox, T. (2016a). Generating images with perceptual similarity metrics based on deep networks. In *Advances in neural information processing systems*, pages 658–666.

Dosovitskiy, A. and Brox, T. (2016b). Inverting visual representations with convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4829–4837.

Frank, M. et al. (2013a). Touchalytics. http://www.mariofrank.net/touchalytics/. [Online; accessed 01-March-2021].

Frank, M. et al. (2013b). Touchalytics: On the applicability of touchscreen input as a behavioral biometric for continuous authentication. *IEEE Transactions on Information Forensics and Security*, 8(1):136–148.

Fredrikson, M., Jha, S., and Ristenpart, T. (2015). Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1322–1333.

Hidano, S., Murakami, T., Katsumata, S., Kiyomoto, S., and Hanaoka, G. (2018). Model inversion attacks for online prediction systems: Without knowledge of non-sensitive attributes. *IEICE Transactions on Information and Systems*, 101(11):2665–2676.

Islam, M. M. and Safavi-Naini, R. (2016). Poster: A behavioural authentication system for mobile users. In *Proceedings of the 2016 ACM Conference on Computer and Communications Security (CCS '16)*, pages 1742–1744. ACM.

Islam, M. M. and Safavi-Naini, R. (2021). Draw A Circle (DAC). https://github.com/mdmorshedul/DAC. [Online; accessed 01-March-2021].

Islam, M. M., Safavi-Naini, R., and Kneppers, M. (2021). Scalable behavioral authentication. *IEEE Access*, 9:43458–43473.

Jung, D., Nguyen, M. D., Han, J., Park, M., Lee, K., Yoo, S., Kim, J., and Mun, K.-R. (2019). Deep neural network-based gait classification using wearable inertial sensor data. In *2019 41st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 3624–3628. IEEE.

Linden, A. and Kindermann, J. (1989). Inversion of multilayer nets. In *Proc. Int. Joint Conf. Neural Networks*, volume 2, pages 425–430.

Lu, C. X., Du, B., Zhao, P., Wen, H., Shen, Y., Markham, A., and Trigoni, N. (2018). Deepauth: in-situ authentication for smartwatches via deeply learned behavioural biometrics. In *Proceedings of the 2018 ACM International Symposium on Wearable Computers*, pages 204–207.

Mahendran, A. and Vedaldi, A. (2015). Understanding deep image representations by inverting them. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5188–5196.

Mandelbaum, A. and Weinshall, D. (2017). Distance-based confidence score for neural network classifiers. *arXiv preprint arXiv:1709.09844*.

Mo, K., Huang, T., and Xiang, X. (2020). Querying little is enough: Model inversion attack via latent informa-

tion. In *International Conference on Machine Learning for Cyber Security*, pages 583–591. Springer.

Nash, C., Kushman, N., and Williams, C. K. (2019). Inverting supervised representations with autoregressive neural density models. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 1620–1629.

Pandya, B., Cosma, G., Alani, A. A., Taherkhani, A., Bharadi, V., and McGinnity, T. (2018). Fingerprint classification using a deep convolutional neural network. In *2018 4th International Conference on Information Management (ICIM)*, pages 86–91. IEEE.

Papernot, N., McDaniel, P., Goodfellow, I., Jha, S., Celik, Z. B., and Swami, A. (2017). Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, pages 506–519.

Schroff, F., Kalenichenko, D., and Philbin, J. (2015). Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823.

Shi, E., Niu, Y., Jakobsson, M., and Richard, C. (2011). Implicit authentication through learning user behavior. In *Proceedings of ISC'2010*, pages 99–113. Springer.

Srivastava, R. K., Greff, K., and Schmidhuber, J. (2015). Training very deep networks. In *Advances in neural information processing systems*, pages 2377–2385.

Tramèr, F., Zhang, F., Juels, A., Reiter, M. K., and Ristenpart, T. (2016). Stealing machine learning models via prediction apis. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pages 601–618.

Wu, X., Fredrikson, M., Jha, S., and Naughton, J. F. (2016). A methodology for formalizing model-inversion attacks. In *2016 IEEE 29th Computer Security Foundations Symposium (CSF)*, pages 355–370. IEEE.

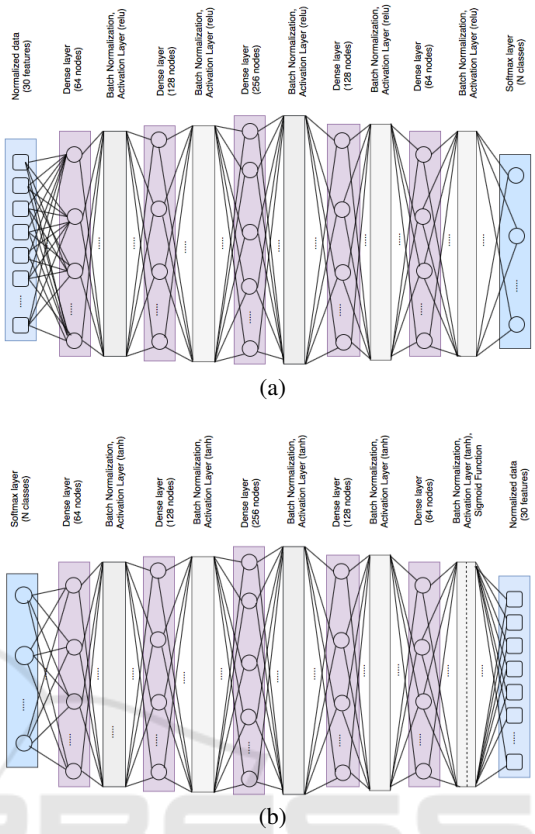
Yang, Z., Chang, E.-C., and Liang, Z. (2019a). Adversarial neural network inversion via auxiliary knowledge alignment. *arXiv preprint arXiv:1902.08552*.

Yang, Z., Zhang, J., Chang, E.-C., and Liang, Z. (2019b). Neural network inversion in adversarial setting via background knowledge alignment. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 225–240.

Zheng, N., Paloski, A., and Wang, H. (2011). An efficient user verification system via mouse movements. In *Proceedings of the 18th ACM conference on Computer and communications security (CCS '11)*, pages 139–150. ACM.

APPENDIX

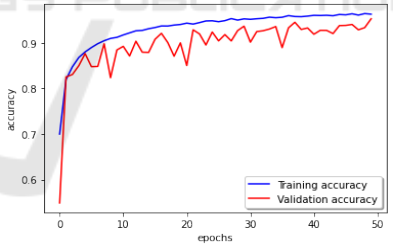
Figure 7 (a,b) shows the DNN architecture of Touchalytics classifier and inverse classifier. Figure 8 (a) shows the training and validation accuracy of Touchalytics classifier. The data reconstruction error of Touchalytics based inverse classifier is shown in Figure 8 (b).



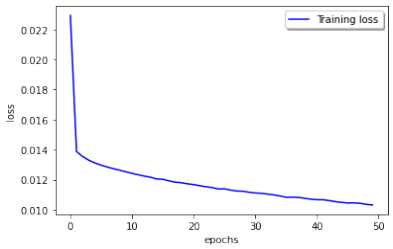
(a)

(b)

Figure 7: DNN architecture of (a) Touchalytics classifier, and (b) inverse classifier. Inverse classifiers architecture is almost same, but opposite to the corresponding classifier.



(a)



(b)

Figure 8: (a) In 50 epochs of training, the Touchalytics classifier can achieve 96.28% of training, and 95.27% of validation accuracy, respectively. (b) Data reconstruction error of Touchalytics inverse classifier reduced to 0.01 in a reasonable number of epochs.