POLITECNICO DI TORINO

Master degree course in Computer Engineering

Master Degree Thesis

Exploring poisoning attacks against a face recognition system

Supervisor:
prof. Silvia Anna Chiusano

Candidate:
Giuseppe Garofalo

Academic year 2017-2018

Summary

Face recognition systems are being widely adopted today as identification tools. The main reason for this trend is the rise of machine learning algorithms, which allows for efficient and usable authenticators. However, intelligent adversaries may target these algorithms, and prior works have underlined the effectiveness of such attacks. One example is the poisoning of the training set, where the attacker changes the input on which the model re-trains to modify the learned function.

In this work, we apply an existing poisoning attack against an authentication system based on a state-of-the-art face recognition technique. In particular, we target a SVM classifier which extends a deep neural network for feature extraction. Moreover, we present a novel reverse mapping technique to craft real-world image starting from a feature vector. Our attack shows a drop in the accuracy of ~45% by just adding one sample to the training set.

This work underlines that poisoning poses a real threat to face authenticators and that security vulnerabilities should be considered when designing such systems.

Acknowledgements

I thank my promotor, Prof. Wouter Joosen, and my co-promotor, Dr. Davy Preuveneers, for giving me the opportunity to work on this topic. I am also grateful to my supervisors, Tim Van hamme and Vera Rimmer. Their guidance was of the utmost importance throughout the entire year.

I would like to thank my promotor in Polytechnic University of Turin, Prof. Silvia Chiusano.

I am grateful to my parents and to my friends for their endless support. Lastly, my sincere thanks go to Mariaida, for being always with me.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Face recognition systems have gained a lot of attention among researchers in the last decade. Both the huge progress in the algorithm development and the broad variety of potential applications contribute to this trend. Due to ease of use, security research focused on face recognition systems over other biometrics for the purposes of identification and authentication [2]. However, they still remain less accurate than other mechanisms. Furthermore, their implementation for real-world tasks poses many challenges to researchers.

As described by Tolba et al. [34], the face recognition process can be divided into four steps: first, the face detection from a source image. Then, a pre-processing step that transforms the image with respect to illumination and pose. Thereafter, the extraction of facial features which describe a single face and allows to perform the matching task. Finally, the use of the extracted features to learn how to classify a newcomer, which defines a face authenticator.

Biometric-based authentication systems are threatened by adversaries that aim to impersonate the user by forging the authentication material. For this reason, a vast amount of research has focused on protecting the input to the authenticators (e.g. via anti-spoofing techniques [16] and liveliness check [1]). However, relying only on security mechanisms installed at the entry point of the system may turn out insufficient. In fact, an intelligent adversary may well target the algorithms placed at the core of the system. Thus, the Machine Learning algorithms that implement the authentication process can also be exploited as an attack vector.

Today, *Machine Learning* algorithms have been broadly adopted as a tool for solving decision-making problems in security-sensitive contexts. These algorithms rely on data provided by the user to learn real-world patterns, with applications ranging from malware detection to biometric recognition. However, they were originally implemented for controlled environments, where the training data comes from a known underlying distribution. This unrealistic assumption is exploited by

intelligent adversaries that carefully craft malicious data points. For this reason, *Adversarial Machine Learning* has emerged in the recent years as a research field. Huang et al. [17] identifies its three main objectives: to classify, anticipate and find countermeasures to possible attacks against machine learning algorithms. This underlines the importance of studying the security of face recognition systems.

One possible attack against ML algorithms is the *poisoning attack*. It is based on the injection of a malicious data point into the training set to affect the classification rule of the model. As a result, the attacker needs access to the training set to carry out this attack. Face authentication systems represent vulnerable targets to poisoning attacks in that they exploit machine learning and user-data to learn a human-face template. One commercial example is the *FaceId* technology by Apple [5]. It comprises several re-training steps such that it learns how to recognize one person through time and facial changes. This continuous-learning process gives to the attacker the possibility to affect the training set.

Several poisoning examples are present into the research literature, but none of them targets face authentication systems. Our contribution is represented by the adaptation of an existing poisoning attack against a face authenticator. In particular, we target the state-of-the-art face recognition library OpenFace [3] extended with a Support Vector Machine (SVM). We divide the attack into two phases: attack point computation and feature-mapping inversion. In the first phase, we compute an attack point which targets the SVM. To do so, we apply the *Gradient Ascent* technique developed by Biggio et al. [10]. In the second phase, we reverse the feature extraction process to obtain a real-world image which embodies the adversarial point. The experiments demonstrate that the injection of a single training point produces a 45% drop in the classification accuracy. Furthermore, the inversion process shows that we can produce a face image without loss in the effectiveness of the attack.

The three key contributions of this thesis can be summarized as follows:

1. provide a security assessment of a machine learning algorithm, i.e. the SVM, in the context of a state-of-the art face recognition system, i.e. OpenFace.

2. perform a first execution of a poisoning attack against a face-based authentication system, showing that it poses a real threat.

3. develop a novel technique to reverse the feature extraction function, aimed at crafting a real-world image.

The work is organized as follows: Chapter 2 provides preliminary knowledge about machine learning algorithms and adversarial machine learning theory. Chapter 3 presents most relevant works: *OpenFace* and prior research on poisoning attack against SVM. We present the authentication system design in Chapter 4. In Chapter 5 we discuss the threat model and the approach to perform the attack,

dividing the process into *attack point computation* and *feature-mapping inversion*. We present the experimental results in Chapter 6, and we further discuss them in Chapter 7 which highlights limitations and challenges. Chapter 8 concludes the work.

Chapter 2

Background

In this section, we introduce the basic foundation of the concepts presented in Chapter 1. We start by presenting an overview to machine learning and the problem of learning from data. Then, we dive into the *Support Vector Machine* supervised learning problem and we discuss the *Artificial Neural Network* model for learning. Finally, we introduce the *Adversarial Machine Learning* research field.

Throughout the chapter, we formalize the assumptions that and intelligent adversary would break, pointing out the importance of focusing on security aspects of machine learning algorithms. The reader which is familiar with these topics can jump to Related Work (Chapter 3).

2.1 The Learning Model

According to Shalev-Shwartz and Ben-David [30], the term machine learning *"refers to the automated detection of meaningful patterns in data"*. This process typically involves observing a phenomenon and constructing a hypothesis on that phenomenon that will allow one to make predictions or decisions. For computers, the experience to learn is given by the data, thus we can define machine learning as the process of extracting knowledge from data [22].

Our goal is to learn a model which is able to make predictions about future events. This task is referred as *predictive learning* and it is shown in Figure 2.1. The learner H is requested to output a *prediction rule* f, also called a *hypothesis* or a *classifier*. This rule maps an input space, the *domain set* D, to an output space, the *label set* T. The task of extracting such a mapping is called *training* and it is performed over the *training set* D_t. Assuming that exists a "perfect" labeling function h, D_t is composed by sampling an instance x from D and labelling it according to h. As a result, we assume that the the training set is the only view

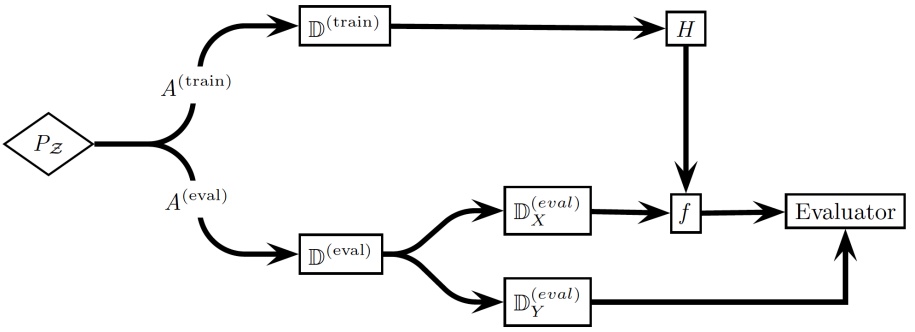


Figure 2.1: The predictive learning model [17].

the learner has on the real world.

This model for the learning task is also called *statistical machine learning framework*. As stated above, this model is based on the assumption that real-world samples that form D_t are drawn from a *probability distribution* P_z the learner does not know. From a security point of view, problem arises when the adversary's model provides for a complete knowledge of D_t. That is, if the intelligent adversary were able to craft a malicious point that diverges from the underlying distribution, the final output would be driven toward a specific learning function. We will come back to this in section 2.4.

Evaluation The *evaluation set* D_e (drawn from the same distribution) is used to asses the performance of our classifier. This is used to determine how good the learnt function f behaves with respect to the real labelling function h. Thus, another goal of our learning algorithm is the minimization of the *classification error*, or *loss function* L. The latter function is defined as the probability to draw a random instance x, according to the distribution D, such that $f(x)$ does not equal $h(x)$.

Data In most cases, we cannot present real-world objects directly to the learning algorithm. Therefore, we need a transformation to map the structured object into a set of *measurements*, called *tuple*. However, measurement is only the first phase in the overall process of *data extraction*. A further transformation is called *feature selection* and it involves information related to the object. This is a data-dependent process that maps the original measurements into a space of relevant features. The primary assumption is that the data contains many features that are either redundant or irrelevant, and can thus be removed without losing information.

We will investigate this in the context of *facial feature selection*.

Generalization The ability of the learner to reproduce the underlying distribution on unlabelled data is called *generalization*. As stated above, we want to learn a model that captures the underlying distribution starting from the training set. However, if the training step is too long or contains few samples, the learner will adjust to very specific random features of the training data that have no causal relation to the target function. This phenomenon is called *overfitting* and it produces an increase in the performance on known data (D_t) while the performance on unseen data (D_e) fall down. Simply put, the model starts to memorize the training data instead of learning how to generalize the underlying trend.

Taxonomy The task of finding a mapping given a set of pair *(element, label)* is called *supervised learning*. In this context, we can have either a categorical or a continuous output for our model. In the first case, we are dealing with a *classification* problem, while the second one is known as a *regression* problem. However, we can have a different application which aims to extract hidden patterns from unlabelled data. This is the *unsupervised learning* set of algorithms. By contrast with the supervised learning pattern, there are no target function, and thus no evaluation of the final accuracy.

2.2 Support Vector Machines

Support Vector Machine (SVM) is a paradigm for learning linear predictors in high-dimensional features spaces. It is a supervised learning algorithm based on a labelled training set where each point is p-dimensional tuple. Given that each of the training point belongs to one of two categories, the main goal is to find (p-1)-dimensional *hyperplane* which separates the aforementioned categories. The output hyperplane, if there exists one, should allow to classify new samples depending on the side of the gap they fall. This behavior falls within the definition of a *linear classifier*.

Margin We can define the best separation hyperplane as the one which maximizes the distance between the nearest points on each side of the hyperplane itself. Looking at Figure 2.2, we can intuitively assess that the black hyperplane (which is a line in two dimensions) can be considered the best one. Formally speaking, we need a metric to measure such a distance, taking into account that the measure is associated to the generalization capability of the output classifier. That distance is called *margin*, and, as a result, the higher the margin the lower the *generalization error*.

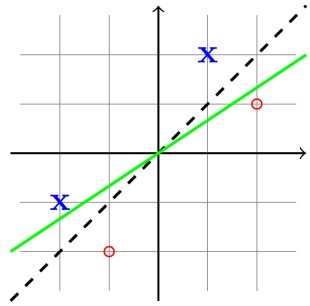


Figure 2.2: Two different separation hyperplanes for the learning task [30].

Kernel Function It may happen that the initial sets are not linearly separable. In this case, it is possible to do a mapping of the finite-dimensional space into an higher-dimensional space which is likely to make the separation task easier. The main idea behind this concept is to avoid the explicit computation of the coordinates in the higher-dimension space, which is considered a computationally expensive task. This operation is called *kernel trick*. In order to perform this step, it was proposed the introduction of a *kernel function*: the function whose product with a vector in that space returns a constant. In this way, the computational load is kept low, despite the fact that working in a higher-dimensional feature space increases the *generalization error* of the SVM.

Linear-SVM Let x_i be the input p-dimensional vector and y_i the associated binary label. Then, we define the separation hyperplane as $\vec{w} \cdot \vec{x} - b = 1$. Where w is the normal vector with respect to the hyperplane. As a result, $\frac{b}{\|\vec{w}\|}$ is the offset of the hyperplane from the origin along the normal vector $\vec{w}$. We will refer at the hyperplane as the *decision function* or *decision boundary*.

Hard-SVM The *Hard-SVM* is the the learning rule which returns a hyperplane that separates the training set with the largest possible margin, or *maximum-margin hyperplane*. We can see this hyperplane as the halfway between two separation hyperplanes on opposite ends. These are described by the equations $\vec{w} \cdot \vec{x} - b = 1$ and $\vec{w} \cdot \vec{x} - b = -1$. Now, we want to minimize the distance between these hyperplanes, namely $\frac{2}{\|\vec{w}\|}$. That is, a minimization problem with respect to $\|\vec{w}\|$. Therefore, we add a constraint to enforce that none of the point lies into the margin space: $y_i(\vec{w} \cdot \vec{x}_i - b) \geq 1$, for each training point. Finally, the w and the b which determine the solution of our constrained optimization problem will define

the final linear classifier $f(x) = sign(\vec{w} \cdot \vec{x} - b)$.

Soft-SVM Unfortunately, data is not always linearly separable, so we need a problem statement for the situation when some of the training point lies on the wrong side of the margin. This is done by introducing a *hinge loss* function which assigns a value in the range [0,1] which is proportional to the distance from the margin: $\max(0,1 - y_i(\vec{w} \cdot \vec{x}_i - b))$. Clearly, the output will be 0 if the point lies on the right side. The new optimization problem aims to minimize the average maximum distance of each wrong-labelled point. Then, a regularization term allows to tune parameters in a way to select the trade-off between the margin size and the correct labelling of each point.

One-Class SVM A specific case of binary SVM classifier is the One-Class SVM, first introduced by Schölkopf et al. [28]. The main idea behind this model is the learning of a subset S of the original feature space. Given that we start from a dataset with a probability distribution P, we select S such that the probability of a point to lie inside S is greater than a user-defined threshold $\nu \in [0,1]$. Since no negative class points are provided to the model, the *origin* is exploited as an artificial member. Thus, the distance is maximized between the origin and the kernel-mapped training vectors, according the following definition [28]:

$$min \frac{1}{2}\|\vec{w}\|^2 + \frac{1}{\nu n}\sum_{i=1}^{n}\xi_i - \rho, \text{subject to}$$
$$(\vec{w} \cdot \Phi(\vec{x}_i)) \geq \rho - \xi_i \quad i = 1,2,...,n \quad \xi_i \geq 0$$

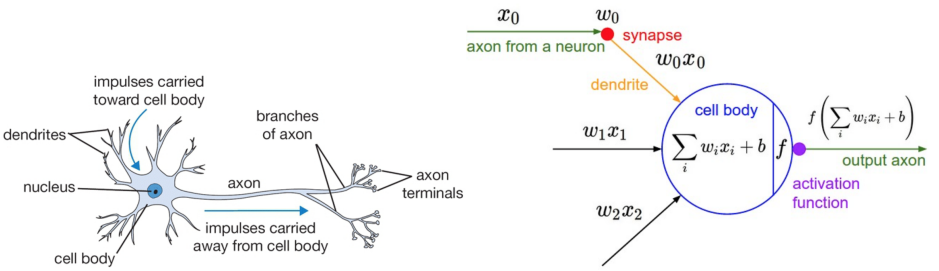
If w and ρ are found, then we obtain the following decision function:

$$f(x) = sign((\vec{w} \cdot \Phi(\vec{x})) - \rho)$$

Here, the parameter ν represents an upper bound the number of training errors as well as a lower bound the the number of support vectors of the learnt model. In other words, it determines a trade-off between the usability of the system and its robustness.

2.3 Artificial Neural Networks

Artificial Neural Networks (ANN) are systems inspired by biological neural networks. Given a model for a *neuron*, they aim to propagate information and learn concepts in a similar way w.r.t. the human brain. However, the known models are a rough approximations of a real neuron and the actual focus is on matching a specific machine learning tasks rather than slavishly imitate the complex structure of the brain.



(a) (b)

Figure 2.3: Biological neuron (a) and its mathematical model (b) [35].

Model A popular mathematical model for a neuron is the *McCulloch-Pitts* (Figure 2.3b). Following the biological analogy, a single input x_i is received through the *dendrite* and is multiplied by the *interconnection weight* w_i. All the signals get summed in the *cell body*, where a *threshold* b is added. This threshold is the value that need to be reached in order to trigger the fire along the output termination, that is the *axon*. Since the neuron carries information through the *firing rate*, we need a correspondent model for it. This model is a static non-linearity f which output is $y = f\left(\sum w_i x_i + b_i\right)$. This non-linear element is usually referred as *activation function*.

Network If considered alone, a single neuron is not very powerful. Instead, a network of neurons is a very strong tool for solving machine learning tasks. The common structure is a set of layers *fully interconnected* in a forward chain, as in figure Figure 2.4. This scheme is also called *feedforward neural network* and it consists of an input layer, one or more hidden layers and an output layer. Connections within one layer are not allowed and the last layer is the only one without an activation function. In fact, this layer is commonly used to output a *score value* for classification purposes.

There are at least two reasons for using a ANN: they are efficiently evaluated using matrix vector operations and they are proved to approximate general non-linear continuous function [1]. Assuming a linear output, a two-layers network (one hidden layer and one output layer) in matrix vector notation is $y = W\sigma\left(Vx + \beta\right)$. Here, the interconnection weights vectors W and V, and the bias vector β are the

[1] Given certain assumptions on the activation function, the **universal approximation theorem** states that a feed-forward network with a single hidden layer can approximate continuous functions on compact subsets of R_n [4].

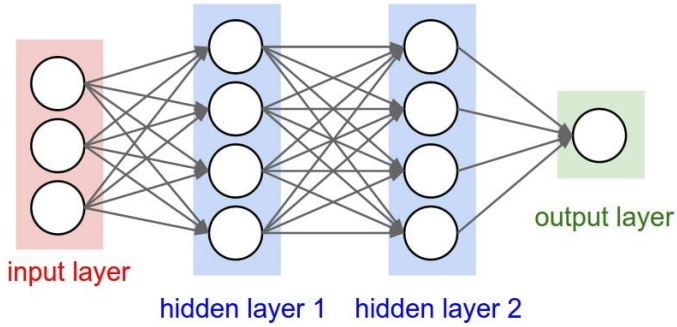


Figure 2.4: Multi-layer perceptron [35].

unknowns; x is the input vector and σ is the activation function of to the hidden layer. What is needed now is a way to valorize the unknowns, learning a meaningful model.

Loss Function As stated above, the neural network output could be interpreted as a class score. However, we still need a *loss function* to asses the performance of our classifier in a supervised learning problem context. The *Mean Squared Error* (MSE) is a conventional choice for feedforward neural networks: it is computed through the squared differences between the predicted output and the desired output. Once we have the loss function, we can formulate the optimization problem for the training step: we want to find the combination of weights and biases so that the model approximates a given function f as closely as possible (the loss function of the output is minimized).

Backpropagation One conventional algorithm for the training of a feedforward NN is *backpropagation*. This is a supervised learning technique based on propagating the error estimation to adjust weights and minimize the output error. In order to achieve its goal, backpropagation exploits the *gradient descent* approach for optimization: first, the inputs are propagated forward to generate the output and compute the error. Then, the backpropagation of the error allows each neuron to compute its own *delta*, namely the difference between the desired output and the actual one. At this point, the gradient is computed by multiplying the delta by the inputs. The gradients are computed iteratively, for each layer, according to the *chain rule*. Finally, the weight is subtracted by a percentage of the gradient called *learning rate*. The latter value is a key parameter to obtain a trade-off between speed and accuracy of the entire process. Usually, the learning process is repeated until an error threshold is reached. However, the gradient descent is

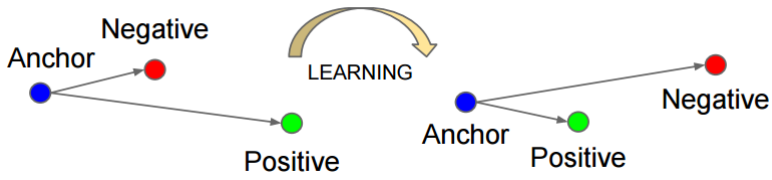


Figure 2.5: Triplet Loss function [29].

not guaranteed to find the global minimum of the error function, but only a local minimum.

Convolutional Neural Network The *Convolutional Neural Network* (CNN) is a NN with one hidden layer, or more than one layer, that explicitly constraints input to be images. As a result, some structural choices can be made to improve the performance of the process. These networks usually exploits the local connectivity of their inputs to reduce the number of training parameters: neurons are arranged in 3 dimensions and connected to small regions of the image.

One concept introduced in this area is the *Triplet Loss* as a loss function. As shown in Figure 2.5, this function minimizes the distance between an *anchor* point and a *positive* point and maximizes the distance between the *anchor* point and a *negative* point. Given a face recognition system, this loss function can be used to train a learner which minimizes distances between faces of the same person and maximizes distances between faces of strangers.

2.4 Adversarial Machine Learning

In the previous section we introduced the assumption that a motivated attacker would break: the *stationarity* of the underlying distribution. Since the increasing use of machine learning algorithms in adversarial environments [17], it becomes necessary the development of a framework that could address such security vulnerability. The research area that studies these threats (and their countermeasures) is called *Adversarial Machine Learning*.

Here we introduce recurrent definitions and well-established taxonomies.

Arms Race Biggio et al. [8] first introduced the concept of *Arms Race*. In analogy with security problems, Adversarial Machine Learning could be a *reactive arms race* (Figure 2.6) or a *proactive arms race* (Figure 2.7) between a designer and an adversary.

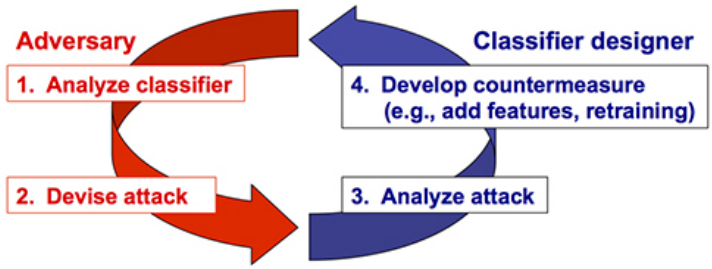


Figure 2.6: Reactive arms race scheme [8].

The reactive scheme represents the response of each of the actors to the opponent's move. First, the opponent analyzes the learning algorithm. Then, a malicious sample is crafted to achieve an objective (e.g. evade the classification). The designer reacts by analyzing the attack and developing countermeasures. For instance, the classifier is trained with malicious samples to become more robust. Given that this model is reactive, one important limitation is that it does not take into account resources and capabilities of the attacker.

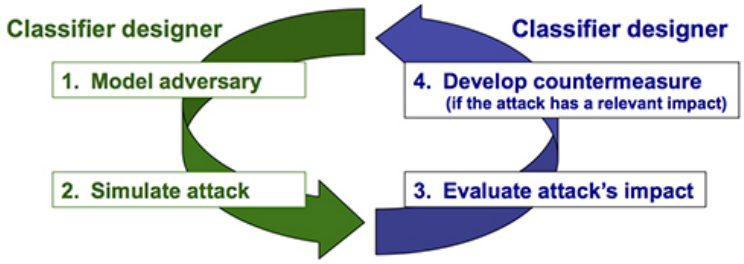


Figure 2.7: Proactive arms race scheme [8].

On the other hand, the proactive scheme tries to anticipate the adversary. In this scheme, the first step involves the generation of a model of the adversary's resources and capabilities. Then, appropriate countermeasures are developed and the process is repeated until a security goal is reached. In this case, the whole process precedes the deployment phase. Thus, the model of the adversary can be used to simulate and prevent future attacks. As a result, it becomes harder for the opponent to carry out the attack or even gain knowledge about the learning algorithm.

Threats Model Huang et al. [17] have introduced a qualitative categorization for modeling threats. This categorization differentiates attacks against a supervised algorithm by three properties:

- **Influence** An attack can be either *Exploratory* or *Causative*. In the first case, the attacker can only craft samples and send them to the classifier to observe its response. In this way, he can analyze the output in order to extract information about the learner. In the second case, the attacker is able to alter the training process and, consequently, the final classifier.
- **Security violation** An *Integrity* attack is focused on the classification of a malicious sample as a rightful one, while an *availability* attack aims to make the classifier unusable because of the misclassification of legitimate samples. A third case is the *privacy violation* that involves inference of user-data.
- **Specificity** An attack can have a very specific set of target points, namely a *targeted* attack. On the contrary, we refer to an *indiscriminate* attack if there is no specific target.

Attack Scenario Given the taxonomy described above, two different attack scenario are identified: *poisoning* and *evasion*. They diverge because of the underneath adversary's model. In fact, the former is mainly focused on a modification of the *test set* such that malicious samples are misclassified as legitimate. While the latter are based on the injection of an *attack point* that causes the classifier to learn a wrong decision-making function.

Many systems are vulnerable to poisoning attack because of the *re-training* step on which they depend. In fact, an application usually re-train over a set containing the new input to adapt to changes of the data. For instance, in a face recognition system a person need to be authenticated even if changes of the facial features occur during time.

Considering the learning scheme in Figure 2.1, the evasion attack scenario enables the attacker to control D_e, while the poisoning scenario makes it possible to alter D_t too. Coordinating D_e and D_t, the attacker can achieve the best results with respect to his own goal function.

Chapter 3

Related Work

The main objective of this work is to perform a security assessment of a specific classification algorithm in the context of a state-of-the-art face recognition system extended with a SVM classifier. In this chapter, we explain the most relevant works in this area. First, we review the state-of-art in the face recognition field, focusing on the our target system (Section 3.1). Then, in Section 3.2, we present prior work in the adversarial machine learning area as a precondition for deepening poisoning attacks against SVMs.

3.1 Face Recognition

Face recognition systems are biometric-based techniques widely used nowadays. Following the taxonomy by Jafri et al. [18], they are used for two main tasks: *verification* of a claimed identity and *identification* of a known identity. In order to address these problems, one can follow two approaches: the *featured-based* approach and the *holistic* one. Holistic approaches attempt to perform the recognition by means of a global image descriptor. For example, the image can be seen a two-dimensional array of intensity values (HOG) that are used to compare different images. However, the high computational-cost and the difficulty of apply this idea on a larger scale, make this approach unfeasible in most cases. For this reason, featured-based approaches have been preferred in the past. These systems use algorithms to analyze an image and extract a descriptor as a set of the most relevant features. As a result, a compact representation of a face is extracted in a robust way. Nevertheless, automatic feature detection represents a limitation that a robust system should be able to cope with. The use of Convolutional Neural Network (CNN) has emerged as a possible solution for this problem.

In this section, we present a brief review of some state-of-the-art techniques

based on CNN features extraction.

3.1.1 Deep Architectures

Taigman et al. [33] exploited deep learning to obtain high-level feature representations. Given an ensemble of CNNs, their goal is to minimize the distance between two images that belong to the same person while maximizing the distance between images showing different people. To do so, they use a L1-distance metric between images features. In order to achieve their goal, they proposed an effective way to align faces to an explicit 3D model during the pre-processing phase. Furthermore, they exploited a Principal Component Analysis (PCA) for dimensionality reduction and, finally, a SVM for the classification step. As a result, the CNN is trained on a four million facial images dataset, reaching an accuracy of 97.25% on the Labeled Faces in the Wild benchmark (LFW) [6], and reporting the result of 91.4% on the YouTube Faces dataset (YTF) [36].

Sun et. al [32] expanded the previous work with a more complex CNN-ensemble. Despite the fact that they use a simpler 2D align transformation, they derive a more complex model composed by hundreds of CNNs. As before, a PCA is used for dimensionality reduction and a SVM is exploited for the classification step. They obtain an accuracy score of 99.15% on the LFW benchmark.

Schroff et al. [29], from Google's research laboratories, extended previous works by means of a Triplet Loss function for image comparison. The main idea is the use of a CNN to directly learn a mapping between real images and a compact Euclidean space of features. Thus, the space embeds the notion of similarity without applying any post-processing step. In order to generate the space, they define the Triplet Loss function to take into account a third image, associated to a certain identify, while minimizing the distance between images embedding a different identity. In this way, a relative distance constraint between images is enforced. For this approach, the classification benchmark (LFW) gives an accuracy of 99.63%.

3.1.2 OpenFace

Regarding the purpose of our attack, it was important to identify a biometric system with openness characteristics, which would allow the tuning of system parameters and the formulation of certain assumptions on the attacker model. For this reason, we selected the OpenFace [3] recognition system as an attacking target.

OpenFace is a general-purpose library based on the work by Schroff et al. [29] mentioned above. In this sense, they train a CNN for features extraction using a non-proprietary dataset at least two orders of magnitude smaller than other state-of-the-art systems. However, competitive results (92.92%) are shown on LFW verification benchmark. In addition, they introduce a classification benchmark on

a subset of the LFW dataset, showing better performance w.r.t. previous open-source techniques based on the OpenCV library [11].

Regarding the classification step, the authors have proposed the use of a linear SVM, demonstrating that it behaves consistently better, in terms of performance, compared to more complex classification algorithms. As a consequence, their classification model, namely the SVM, will be the target of our poisoning attack.

3.2 Adversarial Machine Learning

Adversarial Machine Learning is the research area analyzing the security of machine learning algorithms within adversarial environments. Two different attack scenario can be identified in this setting: *poisoning* and *evasion*. They diverge because of the underneath adversary's model. In fact, the former is mainly focused on a modification of the test set such that malicious samples are misclassified as legitimate. While the latter are based on the injection of an attack point that causes the classifier to learn a wrong decision-making function.

An evasion attack by Lowd et al. [21] aims to evade a spam filter system. The attack is carried out impersonating an attacker which sends a number of requests to discover which words are blocked and which not. Another example by Srndic and Laskov [31] avoids the classification mechanism exploiting structural meta-data of PDF files. A further example by Biggio et al. [7] investigates the spoofing of a biometric system which templates are re-trained periodically.

The main focus of this work is the poisoning scenario. Existing attacks include intrusion detection systems (IDS), where the adversary's goal is to make the system learn a wrong model for granting access to non-authorized people. Usually, specific machine learning algorithms are targeted for carrying out the attack. This section provides an overview of three works focused on poisoning a SVM classifier.

3.2.1 Poisoning against SVMs

Different works have examined poisoning attacks against SVMs. The common thread is how they model the adversary for carrying out the attack: they made a worst-case assumption of perfect knowledge of the training set and of the learning algorithm.

Xiao and Eckert [37] have formalized the so-called *adversary label flips attack* (alfa). Here, the adversary tries to contaminate the training set by flipping its labels. Specifically, the attacker aims to find the best combination of labels to flip in a way that the SVM's classification error is maximized under the original classifier but minimized under the contaminated set. As a result, the goal of this attack is to deviate the SVM's decision boundary preserving the generalization of the *tainted* distribution. The iterative approach followed for the resolution of the

problem is bounded to the maximum number of modifiable samples. Given 20 fixed samples for the evaluation task, they have demonstrated the effectiveness of this strategy compared to a random label flip. In particular, they show an increase in the error rate from 23.5% to 48%.

A further work by Biggio et al. [9] extends the solution described above by introducing a *continuous relaxation*. In that way, labels associated to malicious samples go from being discreet $z \in \{-1, +1\}^n$ to assume continuous real values $[z_{min}, z_{max}] \subseteq R^n$. This approximation exploits the *gradient descent algorithm* to maximize the objective function: first, the best set of continuous label is found computing the gradient. Then, each label is mapped to the correspondent discrete counterpart and added to the training set. Since the SVM's decision boundary will change at each new label flip, the re-computation of the classifier will be necessary in order to guarantee the optimal solution of the learning algorithm.

As demonstrated by the aforementioned attacks, the choice of the label flip which maximizes the classification error can be of crucial importance to maximally degrade the SVM classification performance. We will handle this problem when dealing with the *attack point choice*.

Biggio et al. [10] analyzes the effect of breaking the assumption that the training data comes from a well-behaved distribution. In this context, the attacker knows the underlying data distribution and manages to craft a malicious point to be injected into the training set. This attack can be seen as an extension of the previous ones with a stronger assumption about the capability of the adversary.

In order to perform the attack, they use the *gradient descent algorithm* to maximize the classification error. The first step is the computation of the gradient of the hinge loss function over a validation set. Then, the gradient is used to modify the features of the attack point. In this way, the gradient drives the learner towards the maximum increase direction of the error function. These steps are repeated by means of re-train step until an error threshold is reached.

For this attack, the initial point is obtained by means of a random flip into the attacked class. As a consequence, they were able to show an increase in the error rate between the random flip of a label (first iteration) and the features modification: from $2 - 5\%$ to $15 - 20\%$.

3.3 Gap in the Related Work

To the best of our knowledge, no attempts of attacks against modern face authentication system have been proposed. Biggio et al. [7] investigates for the first time face recognition systems in the adversarial ML area. However, their PCA-based system does not rely on modern machine learning algorithms. We contribute by attacking a state-of-the-art system which exploits algorithms used by real-world mobile authenticators, like CNN and SVM.

Furthermore, we consider the full-stack by inverting the obtained malicious sample to a real-world image. Thus, we extend the methodology developed by Biggio et al. [10] where real-world data was not considered.

Finally, many authentication systems based on linear one-class SVM have been proposed. We provide an empirical analysis of this algorithm from a security perspective. Important insights about the relationship between the hyper-parameter tuning and the resilience to external entities are given.

Chapter 4

Authentication System

In this chapter we present our target authentication system. Figure 4.1 shows the two main components of the system: the OpenFace library for feature extraction (Section 4.1) and the SVM classifier for learning the human-face template (Section 4.2).

4.1 OpenFace for Feature Extraction

The OpenFace library [3] involves several steps to transform an input image into a 128-dimensional feature vector, namely the *embedding*. This system is based on a Convolutional Neural Network (CNN) to extract the most relevant information from a face image. This information will be then used by the classifier to authenticate the user.

The three main steps of the process are as follows:

1. *Face detection* with pre-trained models from dlib [19] or OpenCV [11] open-source libraries.

2. *Face image pre-processing* involving the libraries one more time.

3. *Feature extraction* using a CNN to obtain the 128-dimensional representation of the face image.

The first two steps aim to enhance the efficiency of the process. First, face detectors are exploited to verify the presence of one or more faces into the image. Then, the faces are pre-processed by applying a 2D affine transformation. During this step, the image is aligned w.r.t. 68 specific landmarks (Figure 4.2) such that the eyes and the nose appear always in the same location. Finally, the input is

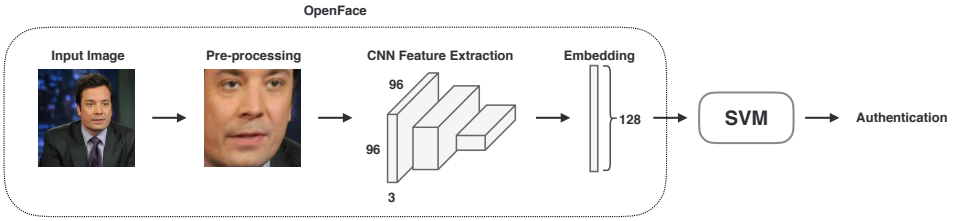


Figure 4.1: Face-based authentication system workflow.

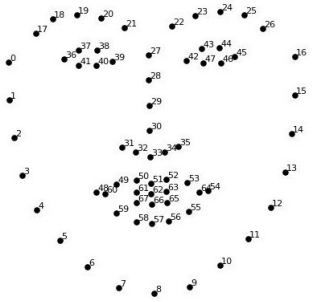


Figure 4.2: The 65 landmarks used for the alignment step [3].

resized and cropped to 96x96 pixels. This process takes place during the CNN training as well.

After the pre-processing, the image is forwarded through the core component of the library, i.e. the CNN. This network is composed by several layers which encapsulate the *notion of a face* into their weights. This is due to the training phase, where thousands of labelled input images are presented to the network. In this phase, a specific loss function is exploited to optimize the process, i.e. FaceNet's *Triplet Loss* [29] function. As explained in Chapter 2, this function is able to cluster face images of one person into the new feature space while it separates different identities from each other. In this way, they introduce a *similarity* concept to be used during classification: the closer two images representation are, the most likely they represent the same person. Furthermore, an *L2-normalization* layer is present at the network output. This layer constrains the representation to be on a hypersphere and the euclidean distance between samples to be within the range [0,4]. As a result, the encoded embeddings are particularly prone to separate face images either with faces that have contributed to the training process or with *fresh*

identities.

4.2 Classification with SVMs

The classification of the extracted features is the final stage of the authentication system. This model is trained to recognize one or more identities by using input images. In this way, during the deployment phase, the model will be able to decide if a newcomer is who he claims to be. In our experiments, we consider two learning models: a linear binary SVM and a linear one-class SVM.

In a preliminary phase, the binary SVM is considered due to performance and low complexity. This model is trained and tested by using two sets of examples by two different people. This means that the authenticator is constrained to a world where only two faces exist and it needs to decide if the input belongs to the positive class or to the negative one. However, thresholds can be used to model the external world, i.e. the *outliers*.

A more meaningful system is considered in a later stage: the one-class SVM. This model extends the classic SVM by considering a single training identity. As a result, it embodies the definition of external world even though other identities are not provided during the training phase. For this reason, the one-class SVM fits with the authentication problem of modern mobile devices. Our authenticator is inspired to IDNet [15], which also exploits DNN feature extraction and one-class SVM to build a gait-based recognition system.

The most important training parameter of one-class SVM is ν, which defines the percentage of training errors. Higher ν values determine higher false positive rate (FPR) of the authenticator, i.e. an higher number of accepted strangers. On the other hand, lower ν values correspond to higher false negative rate (FNR) of the authenticator, i.e. an higher number of good samples that are misclassified by the model. Thus, this parameter represents a trade-off between the *usability* and the *security* of the authenticator.

In both cases we refer to the *decision function* which is the distance from the separation hyperplane learned by the model. This function return a positive value when the sample belongs to the positive class, and a negative value is returned for negative class samples. Technical details about SVMs are discussed in Chapter 2.

Chapter 5

Poisoning Attack Methodology

In this chapter, we present the poisoning attack by first discussing the threat model (Section 5.1) and then describing the attack methodology (Section 5.2 and 5.3).

5.1 Threat Model

The adversarial model is strictly related to the kind of attack we want to perform, i.e. the poisoning of the training data. For this we adapt the assumptions made by Biggio et al. [7] [10].

Goal The goal of the attacker is to undermine the *availability* and the *integrity* of the authentication system. The first goal is achieved when the system is no longer able to work properly, i.e. a false negative rate (FNR) rise. The second goal refers to the acceptance rate of external identities, i.e. a false positive rate (FPR) increase. Our adversary tries to reach both his goals by increasing the number of total errors of the model.

Resources A worst-case scenario where the adversary knows the details of the model is considered. We assume that he knows the training hyper-parameters as well as the training data used for the classifier. Given an attacked identity, we also consider an attacker which is able to craft data representing that specific identity. Nowadays, retrieving one's images can be regarded as a feasible task (e.g. social network). Therefore, the attacker is able to perform the image detection and alignment processes as in the OpenFace framework. Given the cropped and aligned

version of the image, the adversary uses the OpenFace's CNN as a black-box for feature extraction.

Capabilities To perform the poisoning of the training set, the attacker should be able to inject carefully crafted feature vectors. This means that the attacker needs the classifier to be re-trained over a set containing the new sample. The re-train step on which modern authenticators rely can be exploited by the adversary for this task. In fact, it is common for a face authenticator to be re-trained on new images in order to learn facial changes. As a result, we assume that the attacker is able to inject a labelled image trusted by the system. Then, the system will naturally re-train using the features extracted from the image.

5.2 Poisoning Attack against SVM

In this section, we present the poisoning attack strategy. We split the attack into two parts: in the first one we compute the attack point as the ideal feature vector to inject into the training set, based on the SGA technique developed by Biggio et al. [10]; in the second one, we propose a solution to the *inverse feature-mapping* problem, inverting the attack point into an image to be injected into the authentication system.

In this section, we present the first phase of the attack introduced by the attack methodology, while in Section 5.3 the inverse feature-mapping is tackled.

5.2.1 Methodology

The full attack methodology is composed by the following steps:

1. retrieve images used for training the model as well as images used for the validation set.

2. compute the embeddings exploiting the OpenFace CNN and pre-processing tools.

3. train a parallel model and exploit the SGA technique to compute the attack point.

4. invert the attack point using the inverse feature-mapping function.

5. inject the final image on which the model will re-train.

5.2.2 Stochastic Gradient Ascent

The Stochastic Gradient Ascent (SGA) is the algorithm involved for obtaining the attack point. This technique is based on the iterative computation of the gradient of a loss function.

Here, we assume that the attacker controls the training set. To do so, he trains an exact copy of the attacked model exploiting his knowledge about the hyper-parameters and the training data. Then, he computes the attack point by changing the features of one embedding and performing several re-trainings to maximize the his objective, i.e. the loss function.

We separate this section in *problem statement*, *gradient computation* and *high-level algorithm*.

Problem Statement

The problem is formalized as follows: the training data for the initial SVM is $D_{tr} = \{x_i, y_i\}_{i=1}^{n}$, where $x_i \in \mathcal{R}^{128}$ is a feature vector, $y_i \in [-1, +1]$ is the class label associated to the vector and n is the total number of training points. We define the initial *attack point* x_c as the data point the attacker aims to modify, either into the training set or from the validation set. This choice is made once during the first iteration, and it differentiates the *attacking class* y_c from the *attacked class* $-1 * y_c$.

At this point, the attacker needs a metric to determine how well the learnt model behaves. As demonstrated by Biggio et al. [10], a good estimation of the classification error is the *Hinge Loss* function:

$$L_{x_c}(x_k) = \sum_{k=1}^{m}(1 - y_k f_{x_c}(x_k))_+ \tag{5.1}$$

Where f_x is the decision function associated to the SVM and x_k is the k-th point of the validation set drawn by the attacker: $D_{val} = \{x_k, y_k\}_{i=1}^{m}$, where m is the total number of validation samples. The loss function depends on x_c through the decision function f.

We can now define the optimization problem as the maximization of the *Hinge Loss* with respect to the attack point:

$$max_{x_c} L(x_c) \tag{5.2}$$

The *gradient* of L will provide us with the information about the direction of maximum increase of the function. This direction is used during the iterative process to modify the attack point.

Gradient Computation

Following the notation by Biggio et al. [10], K represents the matrix of kernel values and α_i is the dual variable corresponding the i-th training point. The α_i value differentiates among margin support vectors, error support vectors and reserve points, and they refer to them using the lower-case letters s, e, r. Given that Q is the label-annotated version of K, Q_{ss} refers to to the margin support vector submatrix of Q.

We can re-write L to make explicit the terms affected by the x_c

$$L_{x_c}(x_k) = \sum_{k=1}^{m}(-g_k)_+ \tag{5.3}$$

where

$$g_k = \sum_j (Q_{kj}\alpha_j + y_k b - 1) = \sum_{j \neq c}(Q_{kj}\alpha_j(x_c) + Q_{kc}\alpha_c(x_c) + y_k b - 1) \tag{5.4}$$

They overcome the non-differentiability of the convex L by exploiting the points which lead to $-g_k > 0$:

$$\frac{\partial g_k}{\partial u} = Q_{ks}\frac{\partial \alpha}{\partial u} + \frac{\partial Q_{kc}}{\partial u}\alpha_c + y_k \frac{\partial b}{\partial u} \tag{5.5}$$

where u is a norm-1 vector representing the attack direction. They extend this definition of the gradient by considering that the step size should preserve the optimal SVM solution. Following the technique proposed by Cauwenberghs and Poggio [12], the optimal solution for the training point i-th is:

$$g_i = \sum_{j \in D_{tr}} Q_{ij}\alpha_j + y_i b - 1 \tag{5.6}$$

$$h = \sum_{j \in D_{tr}} y_j \alpha_j = 0 \tag{5.7}$$

The effect of x_c on the optimal solution depends on the preservation of the composition of the margin vectors set ($g_i = 0$), error vectors set ($g_i < 0$) and reserve points set ($g_i > 0$). If this condition holds, than we can predict the modification of the SVM solution. The final gradient formula obtained by the authors is:

$$\frac{\partial L}{\partial u} = \sum_{k=1}^{m}(M_k \frac{\partial Qsc}{\partial u} + \frac{\partial Q_{kc}}{\partial u})\alpha_c \tag{5.8}$$

where the gradient of the matrix Q depends on the kernel function, which in this case is the *linear kernel*, such that:

$$\frac{\partial K_{ic}}{\partial u} = \frac{\partial (x_i x_c)}{\partial u} = t x_i \tag{5.9}$$

Algorithm

Algorithm 1 describes the iterative process. Initially, the attack point $x_c^{(0)}$ is selected and its label flipped. The initial SVM solution is computed along with the gradient. Then, the attack point is modified according to the positive direction of the gradient, scaling its value w.r.t. a fixed step size t. After the modification, we enforce the *L2-norm* of $x_c^{(p)}$ as in the final layer of the CNN (Chapter 4). The latter operation allows us to craft an embedding which lives into the euclidean feature space generated by the network forward procedure. Finally, the SVM is re-computed over the modified training set and the stopping condition is checked. We refer to the stopping condition as a threshold ϵ for the hinge loss increase, during a time window w.

Algorithm 1 Poisoning attack against SVM

Input: D_{tr}, the training data; D_{val}, the validation data; y_c, the class label of the attack point; $x_c^{(0)}$ initial attack point; t, the step size.
Output: $x_c^{(p)}$ final attack point

$\{a_i, b\} \leftarrow$ learn a SVM on D_{tr}.
repeat
  Re-compute the SVM solution on $D_{tr} \cup \{x_c^{(p)}, y_c\}$
  Compute $\frac{\partial L}{\partial u}$ on D_{val} according to the Incremental SVM [12]
  Set u to a unit vector aligned with $\frac{\partial L}{\partial u}$ $x_c^{(p)} \leftarrow x_c^{(p-1)} + tu$
  $x_c^{(p)} \leftarrow norm_{L2}(x_c^{(p)})$
until $L(x_c^{(p)}) - L(x_c^{(p-w)}) < \epsilon$
return $x_c = x_c^{(p)}$

To be able to find an effective local maxima in the loss function space, the parameter tuning phase assumes a crucial role. Here we describe the initial attack point choice $x_c^{(0)}$ and the step size t.

Initial Attack Point The initial attack point choice depends on the set from which the attacker selects the sample. As stated above, the attacker may either flip a label from the training set D_{tr} or select a sample from his own validation set D_{val}. In the first case, we use an heuristic: we select 10 *support vectors* of the SVM solution and compute the attack point for each of them, retaining the best one. The reason behind this is that the we aim to pick a point which is far

Figure 5.1: The output of the first 39 convolutional filters.

away from the separation hyperplane, otherwise the point will become a *reserve point* for the SVM solution and the process would halt. In the case when the attacker selects the attack point from the validation set, we pick 15 random points, retaining the best one w.r.t. the classification error increase. Increasing the number of random samples would require an higher computational power for the attack but also increase the probability that the space exploration will be effective.

Step Size The step size selection determines the convergence rate: if t is small then the convergence towards a local maxima can be slow, while selecting a t which is too large can lead to a poor local maxima in the validation loss space. In fact, t should preserve the optimal solution of the SVM for us to able to compute the accurate gradient. As suggested by Biggio et al. [10], we keep the value of t sufficiently small to preserve this condition.

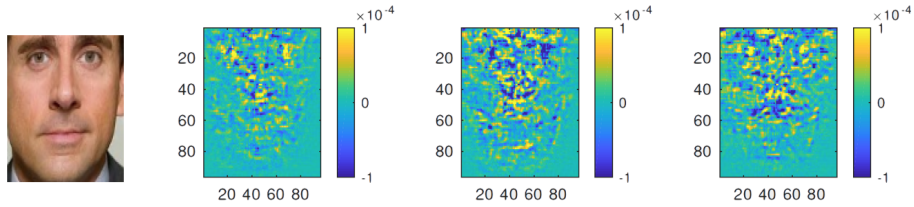
5.3 Inverse Feature-Mapping

In this section we present the solution to the *inverse feature-mapping problem*. Starting from an attack point derived from the SGA technique, we aim to modify an initial image to obtain an embedding as close as possible to the attack point. The problem of finding such modification can be referred as a *function inversion problem* w.r.t. the CNN forward function.

We start by presenting some black-box analysis of the CNN layers, then we introduce a novel technique for solving the inversion problem.

5.3.1 CNN Analysis

The CNN produces an output that depends on the initial input forwarded through several layers. Given that we want to invert this process, it is essential to analyze what features the network has learned to extract. Figure 5.1 depicts the output of the first 39 filters of the first convolutional layer, given an 96x96 pixels input. We notice that some parts of the image are preserved while others are discarded. For



(a) (b)

Figure 5.2: The initial aligned image (a) and the pixels heatmaps of the first three features, after the addition of a constant value (b).

instance, the bottom corners can be interpreted as a zone carrying less informative content.

We extend the analysis by studying the input-output relationship. As in the threat model (Section 5.1), the CNN is seen as a black-box and the input is modified according to a constant value. In this way, we wonder if the final embedding can be correlated to something *visible by humans*, like in Figure 5.1, and if it is possible to exploit such correlation within our inversion problem.

The following sections present the analysis divided into *input generation*, *analysis procedure* and *evidences*.

Input Generation

We start by selecting one raw image of a face which is cropped and aligned. The obtained pre-processed image is a 96x96 pixels matrix. Each pixel follows the RGB color model: three color channels represented by integers within the range [0,255]. For the purpose of this analysis we exploit only the *red* (R) channel since no evidence of a relationship between colors and the final embedding was found. As a consequence, our initial image is a 96x96 integers matrix.

Analysis Procedure

We provide a modified version of the initial input to the CNN to extract some useful information by looking at the output embedding. We apply the addition of a small constant $n = 5$ to each of the cells, where each modification is independent from the others. The value of the cell is bounded to 255 before giving it as input to the CNN. For each of the 9216 (96x96) images generated, we perform a CNN forward pass to obtain the embedding. Finally, we subtract each of the output embedding to the initial one in order to evaluate how much one cell has affected each of the 128-features of the output vector.

Evidences

We visualize the results as 128 *scaled color graphs* where each cell is correlated to its effect on a specific feature. Figure 5.2 shows the effect on the first three features (b) using an aligned image as initial input (a). Even with a negligible perturbation we are able to appreciate the "notion of a face" encapsulated in the CNN layers weights. In fact, we can spot the greatest changes (i.e., yellow and blue in the color graph) along with the landmarks used for the alignment step (Figure 4.2): the eyes, the nose, and the mouth play an important role.

We repeat the process for the same image but increasing the constant value. The results suggest that the pattern cannot be considered fortuitous, but instead there is a *linear relationship* between the constant increase and the output change: the higher the constant the greater the change. However, it is worth to notice that this result is preserved as long as we keep the amount sufficiently low (~10).

5.3.2 Random Sliding Window

Starting from the observations derived from the CNN analysis, we derive an approach for solving the inverse feature-mapping problem. In this section, we present the devised approach, namely the *Random Sliding Window*.

System Model

We retain the threat model discussed in Section 5.1. The attacker is able to use the CNN as a black-box to evaluate the output for a given input; moreover, the attacker can choose any image from among the training set and the validation set raw images. For instance, one choice could be the image that is used to produce the initial attack point. The chosen image is pre-processed using the aforementioned OpenFace tools. We also assume that the initial image is a 96x96 pre-processed version of a raw image, as in the analysis section.

Algorithm

The final objective of this process is to minimize the distance between two embeddings, the objective embedding x_{obj} and the initial embedding x_{init}. For the purpose of the attack x_{obj} will be the attack point x_c from the SGA technique, while x_{init} will be a carefully chosen raw image. In order to achieve our objective, we apply a random modification to x_{init} by sliding a squared window of side l across the original pre-processed image img_{init}. The l value should be a divisor of the image side size (96 for our attack) in order to cover the entire space of the image, like in Figure 5.3. During the iterative process, the window corresponds to a squared matrix of positive random values that is bounded to a threshold h.

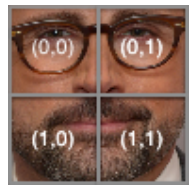


Figure 5.3: The non-overlapping sliding window path with $l = 48$ pixels and $side = 96$ pixels, starting from the top-left corner.

We need to define a proper metric for the distance measure. Given that the embedding values can be either positive or negative, we select the *mean squared error* (MSE) as a distance metric for our optimization problem:

$$mse(objective, initial) = \frac{1}{N}\sum_{k=1}^{N}(objective_k - initial_k)^2 \tag{5.10}$$

with N the number of features in each vector, i.e. 128.

The process works as follows: given img_{init}, we compute the initial embedding by forwarding the image into the CNN $x_{init} = forward(img_{init})$. Then, we compute the initial distance $d = mse(x_{obj}, x_{init})$, where x_{obj} is also referred as the *ideal* embedding. For each column $i = 0 \dots 96/l$, row $j = 0 \dots 96/l$ and color channel $k = [1,2,3]$ we compute a random matrix m_h of size lxl which is bounded to h. Therefore, each iteration is divided into three steps: first, the random matrix is added to the initial image $img_{tmp1} = img_{init} + m_h$, the CNN forward step is performed $x_{tmp1} = forward(img_{tmp1})$ and the output is used to evaluate and store the new distance $d_{tmp1} = mse(x_{obj}, x_{tmp1})$; secondly, the random matrix sign is flipped $m_h = m_h * (-1)$ to retain a second distance measure $d_{tmp2} = mse(x_{obj} - x_{tmp2})$. Finally, d is assigned to the minimum value among d, d_{tmp1}, d_{tmp2}, and the image is re-initialized to the corresponding image $img_{init} = img_{tmp}^{best}$.

Algorithm 2 shows the iterative process. The stopping condition for the cycle can be the number of iterations, a distance threshold or a time constraint. Here, we consider the number of iterations as a sufficient condition.

Algorithm 2 CNN Embedding Inversion

Input: img_{init}, the initial image; x_{obj}, the final objective; l, window side; $iter$ number of iterations.
Output: img^k_{init}, modified image at iteration k.

$x_{init} \leftarrow forward(img_{init})$.
$d = mse(x_{obj}, x_{init})$
$k \leftarrow 0$
repeat
  $m_h \leftarrow$ random matrix bounded to h
  $img_{tmp1} \leftarrow img_{init} + m_h$
  $x_{tmp1} \leftarrow forward(img_{tmp1})$
  $d_{tmp1} \leftarrow mse(x_{obj} - x_{tmp1})$
  $img_{tmp2} \leftarrow img_{init} - m_h$
  $x_{tmp2} \leftarrow forward(img_{tmp2})$
  $d_{tmp2} \leftarrow mse(x_{obj}, x_{tmp1})$
  $d \leftarrow min(d, d_{tmp1}, d_{tmp2})$
  $image_{init} \leftarrow min_d(x_{init}, x_{tmp1}, x_{tmp2})$
  $k \leftarrow k + 1$
until $k < iter$
return img^k_{init}

The most important parameters for the algorithms are the initial image img^0_{init}, the number of iterations $iter$, the perturbation h and the window side l.

Initial Image The distance metric can be used to define a starting point which is *close enough* to our goal. There are two possibilities: either the attacker selects the closest attack point among all the face images, or the attacker crafts a random image. This problem is linked to the exploration of the CNN function space: we are not sure that the current direction is driving us to an effective optimum, but we *trust* the modification that leads us closer to the final point.

In our experiments we select two images: the raw image which produces the target attack point (during SGA), and a random raw image which represents the attacked identity. The first choice is justified by experimental evidence showing that the raw image linked to the attack point is more likely to be closer in terms of *mse*. In this way, we avoid exploring the whole space for the closest image. The second choice is linked to the possibility of having a human-labelling oracle that checks the correctness of training images. If we modify a face image representing the victim, it will be hard for a human to verify that the image embodies an attack point.

Even if the attacker is not able to modify the training set, we can also use the devised approach to *evade* the classification. This is done by minimizing the

distance between a victim's image and a image representing a different identity. Experiments are shown in the Chapter 6.

Window Size - Perturbation The window size impacts the final image in both visibility of the modification and distance. We can not associate large windows with high h values since the final result will be poor. In fact, increasing h leads to a modification which is easily spottable by a human oracle. On the other hand, selecting a small window size leads to a slower convergence rate. For solving this problem, we apply a *decreasing window size approach*: we start from a small h and a large l, then we decrease l while increasing h until a threshold *iter* is reached.

In Chapter 6 different combinations of (h, l) are considered.

Chapter 6

Evaluation

In this chapter, we present the empirical evidence by first describing the experimental setup and then presenting the results.

6.1 Setup

6.1.1 Dataset

To perform the attack we use images from the FaceScrub dataset [23] which retrieves raw face images from the Internet, in uncontrolled conditions. Among 562 celebrity identities, we select 46 identities with a sufficient number of samples to obtain roughly 5000 raw images. Then, the pre-processing tools are used compute the aligned 96x96 image, which is the CNN input.

6.1.2 Software

We use OpenFace [3] Python [27] library to perform the pre-processing step which relies on OpenCV [11] for computer vision primitives and dlib's [19] pre-trained face detector. Torch for Luajit [14] is exploited for the training and inference of the neural network; Numpy [24] is used for arrays and linear algebra operations.

We have developed a Python's script to build the classification model using scikit-learn [26] and we use OpenFace's nn4.small2.v1 CNN model for computing the low-dimensional embedding. We have also tested our results on the Matlab code developed by Biggio et. al [10] which depends on the libsvm library [13] for the model implementation.

6.1.3 Metrics

Classification Error We use the *false negative rate* (FNR) and the *false positive rate* (FPR) to asses the performance of our target one-class SVM. Given an authenticated user, the FNR refers to the percentage of images depicting the authenticated user that are misclassified by the model. On the other hand, the FPR is linked to the percentage of images depicting strangers that are classified as the authenticated user. The number of false positives (FP) and the number of false negatives (FN) contribute to the definition of the *Classification Error* (CE)

$$CE = \frac{FP + FN}{TP + TN + FP + FN} \tag{6.1}$$

where TP and TN represent, respectively, the number of true positives and the number of true negatives.

Posterior Probability We make use of the *posterior probability* as an evaluation metric for the binary SVM. For its computation, we rely on the implementation provided by the libsvm library [13].

The probability that x has class label 1 $Pr(y = 1|x)$ is approximated by a sigmoid function, as proposed by Platt:

$$P(s_j) = \frac{1}{1 + exp(As_j + B)} \tag{6.2}$$

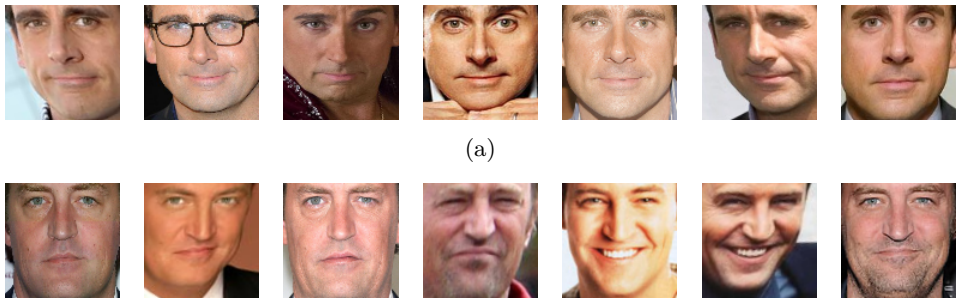
where s_j corresponds to the score of observation j, while A and B are parameters that are optimized during the training phase. This implementation follows Platt's probabilistic outputs for SVM by Lin et. al [20].

As we can see, this information depends on the *decision function*, i.e. the distance from the separation hyperplane. We exploit this notion as a *confidence value* for the prediction of a particular embedding, highlighting the correlation between the posterior probability and the hinge loss increase.

6.2 Results

As discussed in Chapter 4, we define two different target models for our experiments: a linear binary SVM and a linear One-Class SVM. In the first case, the model discerns between two identities, while in the second case the model is trained to recognize a specific identity. In both cases the attack is finalized to the increase of the classification error. For each model, we apply the inverse-feature mapping function to obtain a real-world image to inject into the system.

The next sections present the main experiments against the two classification models, following the adversarial model defined in Chapter 5.



(a)

(b)

Figure 6.1: Example training data set for class -1 (a) and class 1 (b).

6.2.1 Linear Binary SVM

Here, we present the results for the attack of a linear Linear Binary SVM. We start by presenting the model and the data involved, and then we discuss the two phases of the poisoning attack: *attack point computation* and *inverse feature-mapping*.

Model We select a linear SVM with unitary regularization value as the attacker's target. As explained in Chapter 5, the attacker is able to build an exact copy of the target model, given his contextual knowledge. This model is used to compute the attack point which is lately reversed by means of the reverse mapping function. Finally, the injection of the adversarial image into the training set of the target system allows the user to achieve the desired goal.

Data Split For the binary classifier, we select the two biggest identities in our dataset, obtaining 150 and 145 aligned images. Then, the images are forwarded through the CNN to obtain the 128-dimensional embeddings. These embeddings are divided into two separate sets: the training set, composed by 15 random samples per-class, and the validation set, comprising the remaining embeddings. As a result, the approximate proportion of the non-overlapping sets is 1:10, but further extension of the training set are considered during the experiments. Figure 6.1 depicts an example of a training set: (a) samples belonging to class -1; (b) samples labeled as +1.

Attack Point Computation

Parameter tuning We discuss the main parameters for the SGA technique: *initial attack point*, *step size* and *stopping condition*.

For these experiments, we select the initial attack point from the training set. It is worth to notice that comparable results are retained when the other initial point strategy is followed, and we will show the results when dealing with the one-class SVM.

We use the heuristic to build a set of initial attack points: we select 10 *support vectors* of the target SVM. For each of the initial points we evaluate the correlation between the hinge loss L and the classification error CE. Moreover, we present a discussion about the posterior probability decrease.

For these experiments, the SGA step size is tuned to 0.1, while the stopping condition checks the decrease of L in a time window of 20 iterations.

Experiments We show the results for the most representative points of each class.

Figure 6.2 depicts the relationship between L and the CE for three attack points of class -1. As we maximize L, the CE increases till 10 misclassified samples. Since the initial attack point is selected from the training set, we can see that the attack strategy overcomes the *adversarial label flip* (iteration 2). In fact, at most 1 error is appreciated when flipping the label of the attack point. Remarkably, attack point 20 (c) shows a decrease of L during iteration 2.

Figure 6.3 depicts the classification error and the L increase for three attack points of class 1. The performance decrease w.r.t. the previous case underlines that the effectiveness of the attack is correlated to the validation set and to the training set composition. In this case, we have many images with an high decision function score in the validation set. This means that few candidates can be moved to the other side of the separation hyperplane because most of them are far away from it.

Attack Point	Class	Pr	First Iter	Label Inversion	Best Result
-1	Attacked	80	11%	28%	39%
-1	Attacking	95	62%	66%	77%
1	Attacked	85	8%	34%	52%
1	Attacking	95	61%	62%	84%

Table 6.1: Percentage of the validation set below a fixed probability threshold Pr.

An authentication system could make use of the *posterior probability* as a threshold to deny access to strangers. Thus, we investigate the effectiveness of the attack w.r.t. the probability that an embedding x belongs to its class y, i.e. $Pr(y|x)$. For this purpose, we select two samples from the previous attack: one from class -1 (Figure 6.2a) and the other one from class 1 (Figure 6.3b). In this way, we compare an attack point that increases the classification error from 0 to

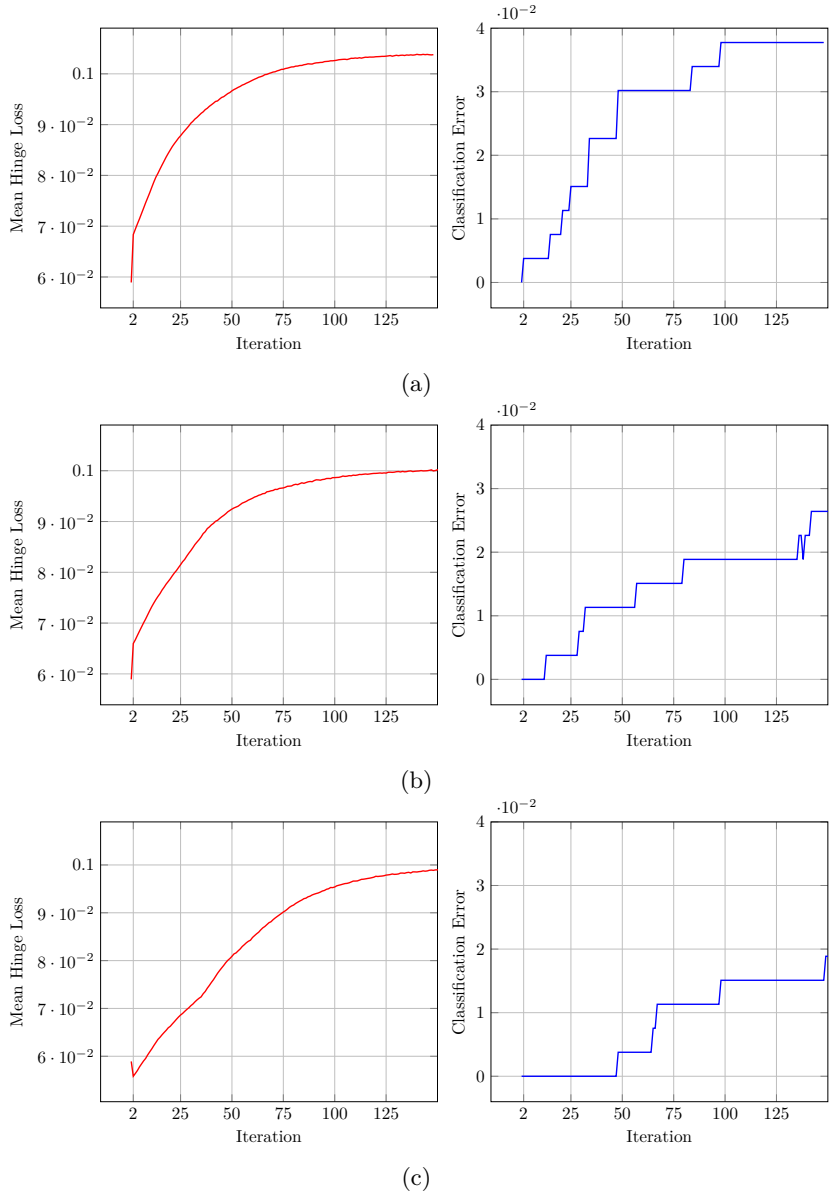
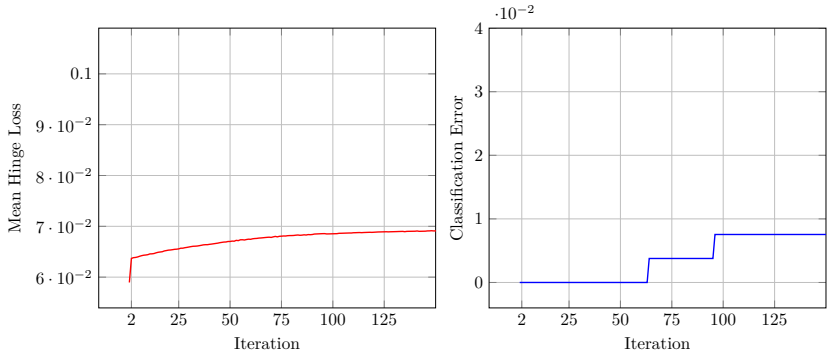
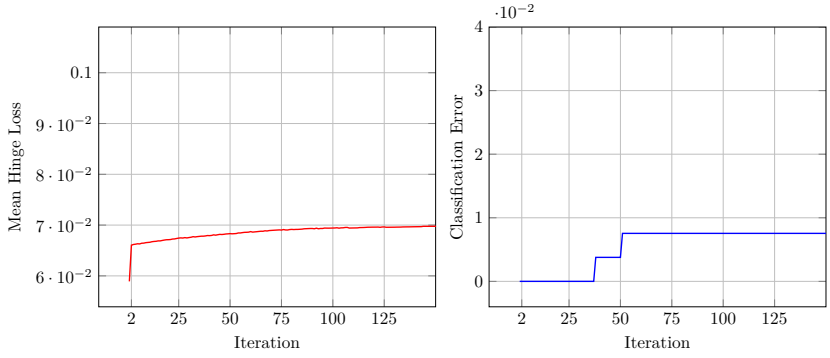


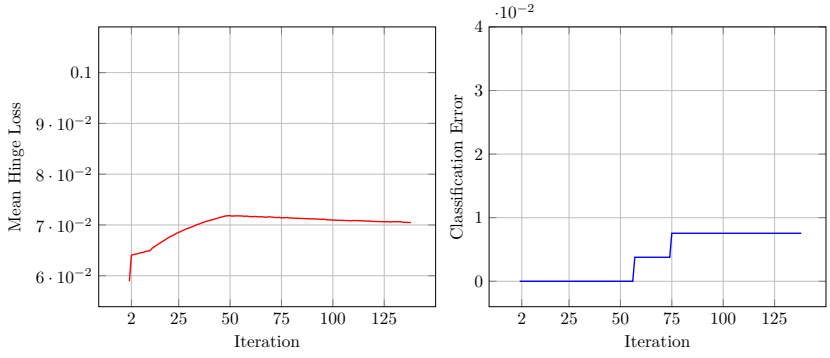
Figure 6.2: Starting from the first row, attack points: (a) 2, (b) 13, (c) 15 of class -1. Mean L (left) and corresponding classification error increase (right).



(a)

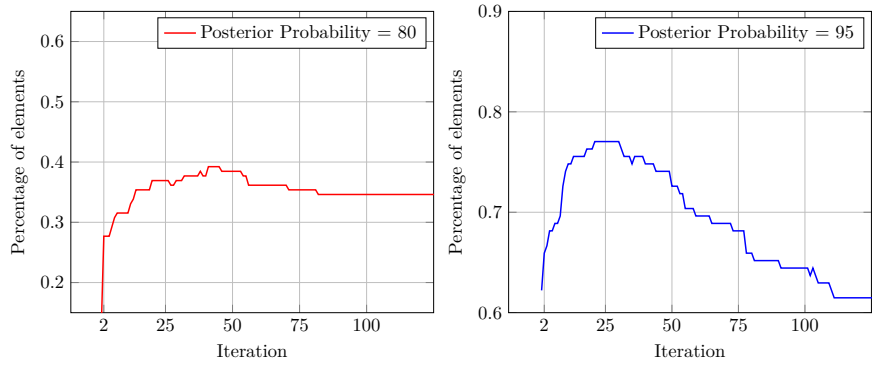


(b)

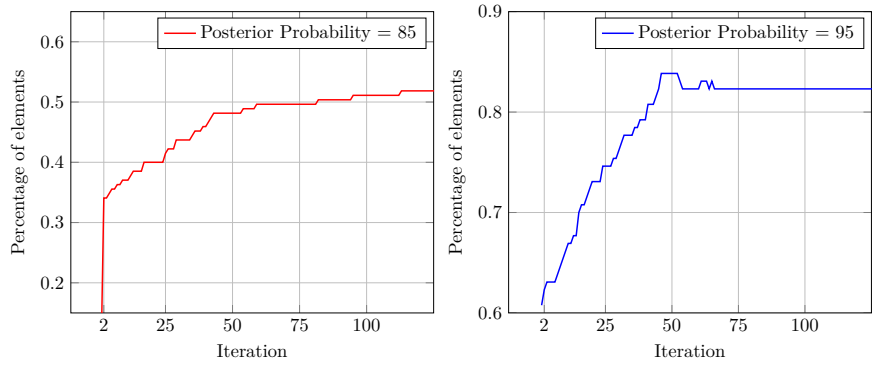


(c)

Figure 6.3: Starting from the first row, attack points: (a) 18, (b) 19, (c) 20 of class 1. Mean L (left) and corresponding classification error increase (right).



(a)



(b)

Figure 6.4: Percentage of elements below a posterior probability threshold two points: (a) attack point 13 (class -1); (b) attack point 20 (class 1). The validation set is divided into attacked class (left) and attacking class (right) with different thresholds.

7 misclassifications with an attack point that is able to trigger just 2 misclassifications. We set a threshold Pr to verify the number of elements that go below Pr itself: 80% and 85% are considered as common sense choices for a real-world authenticator. We present the results over the validation set dividing between the attacked class and the attacking class.

Figure 6.4 shows the percentage of elements below the threshold during the attack. During the first iterations, the attack point from class -1 (a) triggers a number of elements below the threshold Pr, but this value lately fall as CE increases (a). On the contrary, the second attack point shows an important increase

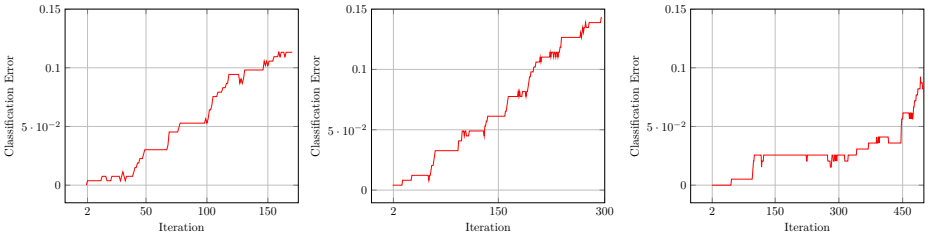


Figure 6.5: Multi-point attack using a sequence of images up to 15% of the training set. Training sets of different sizes, from left to right: 30 samples, 50 sample, and 100 samples.

in the percentage of elements below the threshold, despite the minimal increase of L and CE (Figure 6.3b). The first effect can be explained as an *overfitting-like phenomena*, where some points become more representative than others in order to maximize the misclassifications, i.e. reduce the generalization ability of the system. On the other side, we appreciate how the second point may affect greatly the usability of the system, even if fewer misclassifications are triggered. These results are summarized in Table 6.1.

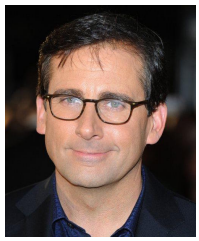
Lastly, we take into consideration the multi-point attack. To perform the attack, we select one support vector of the SVM, flip its label, compute the attack point and repeat the process up to a percentage of the training set has been poisoned. To validate our results, we extend the training set to 50 samples and 100 samples. For each training set size, up to 15% of the training set is modified.

Figure 6.5 depicts the results for the multi-point attack. Even though we select the initial point randomly, the attack demonstrates the link between the percentage of training points modified and the final performance worsening. We report a CE of 14% when the training set is composed by 50 images.

Inverse Feature-Mapping

Given the best attack point from the SGA technique, we use the reverse mapping procedure to obtain a real-world image. This image is used by the adversary to poison his target model.

Parameter Tuning For these experiments, we select the attack point which leads to the highest CE from the SGA technique: point 2 of class -1, retained at the second to last iteration (Figure 6.2a). Figure 6.6 depicts the raw starting image (a) and its pre-processed version (b). We consider the same raw image that generates the initial attack point as a starting image for the inversion function. Given experimental evidence, this starting image will likely be the most similar



(a)

(b)

Figure 6.6: Initial attack point: (a) raw; (b) pre-processed.

Figure 6.7: Sequence of images generated by the random addition function to minimize the *mse*.

to our objective. Figure 6.7 depicts the sequence of images generated through the process, for each l.

To optimize the results, the function is applied four times with a decreasing window size l and an increasing upper-bound h. The values for the pairs (h, l) are found empirically: (32,2), (16,6), (8,10), (4,14). Since we repeat three times the sliding over an image, the total number of iterations for each l is, respectively: 27, 108, 432 and 1728. To visualize the results, we store the embedding associated to the modified image at the end each row: 135 embeddings are retained during the procedure.

Experiments Figure 6.8 compares the distance increase during the SGA technique with the reverse mapping distance minimization. Starting from a *mse* equal to 9e−3, we observe how the modified image approach the objective, till a *mse* of 2e−5. Figure 6.9 shows the correlation between the distance and L: within 60 iterations, we retain the same results computed analytically by using the SGA technique.

The results demonstrate the possibility to obtain a real-world image by minimizing the distance between an objective and an initial image. Given the threat

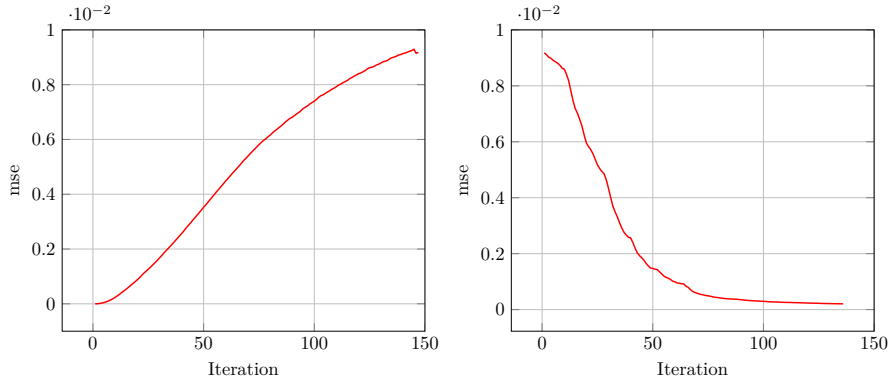


Figure 6.8: *mse* throughout the SGA process (left) and the inversion (right).

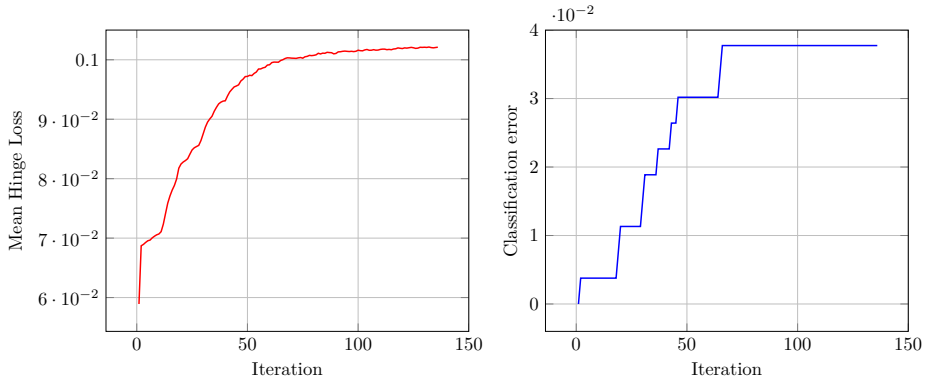


Figure 6.9: Mean L (left) and CE (right) while minimizing *mse*.

model described in Chapter 5, the attacker would choose the attack point as an objective and another image as a starting point for the reverse function. Then, he would perform the poisoning by injecting the resulting adapted image.

In a broad sense, the reverse mapping function can be used to minimize the distance between any pair of embeddings. A different scenario regards the *classification evasion*. By modifying an image of class -1 we resemble an image of class $+1$.

Figure 6.10 shows the images produced to minimize the distance between the objective (Figure 6.1a; second image) and the initial image (Figure 6.1b; first image). As shown in Table 6.2, we are able to achieve lower *mse* values than before. This is due to the fact that the embeddings are produced from real-world faces rather than maliciously crafted points, and thus it becomes easier to minimize

Side	mse	$Pr(y = -1\|x)$
Initial	1.3e-2	9%
32	1.1e-2	20%
16	7.3e-3	60%
8	7.7e-4	88%
4	7.1e-6	94%

Table 6.2: Evasion scenario posterior probabilities.

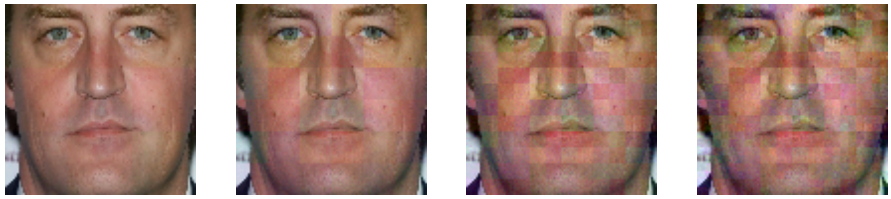


Figure 6.10: Evasion scenario: sequence of images generated by the random addition function to minimize the *mse*.

their distance. The table also shows the posterior probability measures for each of the images: the attacker is able to fool the classifier into thinking that the modified sample belongs to the wrong class with an increasing Pr value, up to 94% for the last image of Figure 6.10.

6.2.2 One-Class SVM

Here, we present the results for the attack of a linear One-Class SVM, considered as an authenticator for a specific identity. As above, the poisoning attack is split into two phases: *attack point computation* and *inverse feature-mapping*.

Model The hyper-parameter of the One-Class SVM is ν, which is both an upper-bound to the fraction of training errors and a lower-bound to the fraction of support vectors of the model. For our experiments, we consider different ν values, looking for a trade-off between the usability of the authentication system and its security. In fact, lower values correspond to an higher false positive rate (FPR), while higher values lead to an increase of the false negative rate (FNR).

Dataset The entire dataset (46 identities) is considered for this setup. The dataset is partitioned into training set, validation set and test set: the training set is the training set of the target model, known by the attacker; the validation set is crafted by the attacker which knows the attacked identity; the test set is used by

the attacker to evaluate the scalability of the results obtained over the validation set.

Fo these experiments, we fix one attacked identity and we select 9 attacking identities for the validation set. The 10 folds are used to cross-validate the model by randomly shuffling the points assigned to each set, keeping the proportions. We consider several training set sizes, while the remaining embeddings are equally divided between validation and test set: within one batch, approximately 500 points are retained for each set. To generalize the results, we repeat this process 5 times by selecting a new set of attacking identities. During the experiments, the attacked identity remains fixed.

Attack Point Computation

Parameter Tuning For the SGA technique, we consider the following setting: step size equal to 0.1 and a stopping condition that checks the hinge loss increase over 40 iterations.

We train a parallel model which resembles the target one and we search an effective local maxima into the validation loss space. As described in Chapter 5, 15 random points are selected for the initial attack points set.

nu	size	initial	injection	attack
0.05	30	3.46 ±0.44	8.86 ±1.59	41.04 ±6.45
0.05	50	2.84 ±0.68	7.63 ±1.09	35.07 ±3.79
0.05	70	2.65 ±0.57	6.31 ±2.81	24.69 ±3.82
0.05	90	2.16 ±0.65	4.39 ±1.11	15.82 ±1.97
0.1	30	3.09 ±0.17	4.67 ±0.88	23.22 ±3.89
0.1	50	2.29 ±0.20	2.91 ±0.39	9.76 ±2.49
0.1	70	1.89 ±0.27	2.44 ±0.59	5.73 ±0.91
0.1	90	1.45 ±0.24	1.58 ±0.40	3.89 ±0.62
0.15	30	3.29 ±0.34	4.12 ±0.67	10.94 ±3.31
0.15	50	2.30 ±0.14	2.59 ±0.27	4.50 ±0.82
0.15	70	1.69 ±0.17	1.93 ±0.24	3.16 ±0.49
0.15	90	1.47 ±0.09	1.62 ±0.14	2.44 ±0.36
0.2	30	3.57 ±0.16	3.61 ±0.07	6.42 ±1.93
0.2	50	2.58 ±0.18	2.73 ±0.22	3.68 ±0.50
0.2	70	1.92 ±0.12	2.01 ±0.13	2.58 ±0.29
0.2	90	1.59 ±0.10	1.67 ±0.13	1.98 ±0.27

Table 6.3: Validation set mean classification error across all the batches.

In order to further generalize the results, different settings are considered: $\nu \in [0.05, 0.10, 0.15, 0.20]$ and training set size $s \in [30, 50, 70, 90]$. For each setting and

nu	size	initial	injection	attack
0.05	30	3.33 ±0.11	8.28 ±1.35	40.11 ±6.78
0.05	50	2.93 ±0.63	7.20 ±1.18	34.67 ±3.64
0.05	70	2.52 ±0.68	6.38 ±2.70	24.19 ±3.57
0.05	90	2.18 ±0.43	4.33 ±1.11	15.46 ±1.57
0.1	30	3.33 ±0.26	4.85 ±0.86	22.41 ±4.15
0.1	50	2.51 ±0.19	3.17 ±0.55	9.91 ±2.43
0.1	70	1.85 ±0.27	2.31 ±0.55	5.87 ±1.18
0.1	90	1.41 ±0.19	1.49 ±0.34	3.71 ±0.65
0.15	30	3.27 ±0.27	4.00 ±0.52	10.42 ±2.85
0.15	50	2.38 ±0.16	2.61 ±0.34	4.38 ±0.72
0.15	70	1.84 ±0.09	2.08 ±0.19	3.15 ±0.51
0.15	90	1.36 ±0.08	1.43 ±0.09	2.16 ±0.48
0.2	30	3.53 ±0.28	3.69 ±0.16	6.26 ±1.44
0.2	50	2.76 ±0.14	2.81 ±0.13	3.51 ±0.39
0.2	70	2.19 ±0.15	2.21 ±0.16	2.67 ±0.17
0.2	90	1.48 ±0.07	1.55 ±0.12	1.83 ±0.17

Table 6.4: Test set mean classification error across all the batches.

nu	size	initial FPR	injection FPR	attack FPR
0.05	30	0.71 ±0.86	7.52 ±1.56	43.62 ±7.71
0.05	50	1.71 ±1.09	7.12 ±1.29	37.16 ±4.24
0.05	70	1.89 ±0.85	5.97 ±3.16	25.70 ±4.23
0.05	90	1.64 ±0.85	4.06 ±1.24	16.17 ±2.05
0.1	30	0.35 ±0.38	2.60 ±0.94	22.91 ±4.87
0.1	50	0.43 ±0.44	1.47 ±0.51	8.70 ±2.72
0.1	70	0.62 ±0.52	1.27 ±0.68	4.60 ±1.34
0.1	90	0.68 ±0.38	0.86 ±0.47	3.17 ±0.96
0.15	30	0.51 ±0.63	1.65 ±1.11	8.84 ±4.11
0.15	50	0.32 ±0.31	0.66 ±0.40	2.35 ±1.52
0.15	70	0.28 ±0.30	0.57 ±0.38	1.77 ±0.85
0.15	90	0.29 ±0.19	0.45 ±0.25	1.16 ±0.72
0.2	30	0.15 ±0.19	0.49 ±0.35	2.34 ±2.84
0.2	50	0.14 ±0.28	0.31 ±0.37	0.76 ±1.17
0.2	70	0.14 ±0.18	0.30 ±0.24	0.50 ±0.80
0.2	90	0.17 ±0.20	0.28 ±0.29	0.45 ±0.57

Table 6.5: Validation set false positive rate across all the batches.

for each batch, the 10-fold cross-validation is applied.

Experiments First we discuss the accuracy of the classifier over all the batches. To this end, the *mean CE* is evaluated in three different moments: before computing the attack point, after the injection of adversarial point and, finally, after the attack. Table 6.3 and Table 6.4 show the *mean CE* for each of the settings over all the batches for, respectively, the validation and the test set. Not surprisingly, the most important decrease is found by setting $\nu = 0.05$ and $size = 30$: the *CE* reaches $40.11\% \pm 6.78\%$ while the injection error is roughly 5 times smaller ($8.28\% \pm 1.35\%$). Two different trends are identified by looking at the tables: fixing the ν value, the initial error consistently decreases with the training set size; on the other hand, fixing the training set size, the best ν is identified between 0.10 and 0.15. This is due to the linear relationship between ν and the FNR. Moreover, the effectiveness of the attack is amplified for small training set sizes and ν values, as in the most successful scenario.

Next, we discuss FPR over the validation set by looking at Table 6.5. As expected, the ν increase corresponds to a decrease in the FPR, i.e. to a more resilient system. In all cases we can observe a link between the sensitivity of the system to the adversarial injection and the final attack point effectiveness: the validation loss space exploration is boosted by a proper starting point.

nu	size	initial	injection	attack
0.05	30	3.01 ±0.45	11.44 ±3.50	61.21 ±7.91
0.05	50	1.99 ±1.05	9.89 ±5.48	49.24 ±12.38
0.05	70	2.41 ±0.63	11.93 ±8.14	39.49 ±10.44
0.05	90	1.49 ±0.95	4.43 ±3.15	26.54 ±7.35
0.1	30	2.71 ±0.92	4.38 ±1.84	33.26 ±12.33
0.1	50	2.04 ±0.64	3.33 ±1.50	15.55 ±7.51
0.1	70	1.24 ±0.57	1.86 ±0.92	6.70 ±2.88
0.1	90	0.94 ±0.35	1.28 ±0.70	2.73 ±1.86
0.15	30	3.23 ±1.06	4.80 ±2.46	15.78 ±9.40
0.15	50	2.04 ±0.71	2.46 ±0.74	3.78 ±1.23
0.15	70	1.47 ±0.50	1.65 ±0.64	2.29 ±1.06
0.15	90	1.31 ±0.41	1.28 ±0.43	1.63 ±0.70
0.2	30	3.33 ±0.87	3.60 ±1.08	7.25 ±5.73
0.2	50	2.64 ±0.43	2.64 ±0.42	3.30 ±0.57
0.2	70	1.68 ±0.73	1.70 ±0.60	1.95 ±0.73
0.2	90	1.51 ±0.56	1.40 ±0.51	1.49 ±0.51

Table 6.6: Validation set mean classification error for the first batch.

nu	size	initial	injection	attack
0.05	30	3.20 ±0.80	10.84 ±4.76	58.20 ±6.39
0.05	50	2.22 ±1.16	9.24 ±5.71	49.36 ±12.03
0.05	70	2.25 ±0.93	12.19 ±8.04	38.06 ±10.51
0.05	90	1.54 ±1.04	4.64 ±3.26	26.43 ±5.82
0.1	30	3.31 ±1.12	5.15 ±2.29	32.73 ±12.75
0.1	50	2.22 ±0.97	3.99 ±2.08	16.79 ±7.92
0.1	70	1.56 ±0.62	2.19 ±1.16	7.31 ±3.71
0.1	90	1.12 ±0.57	1.41 ±0.53	3.06 ±1.94
0.15	30	2.70 ±0.74	4.68 ±1.30	14.78 ±8.51
0.15	50	2.20 ±0.64	2.33 ±0.79	3.44 ±1.42
0.15	70	1.62 ±0.43	1.58 ±0.41	2.23 ±0.88
0.15	90	1.27 ±0.39	1.37 ±0.41	1.39 ±0.42
0.2	30	3.99 ±1.00	4.12 ±1.20	7.45 ±4.80
0.2	50	3.15 ±0.71	2.94 ±0.56	3.56 ±1.00
0.2	70	2.23 ±0.72	2.23 ±0.73	2.43 ±0.73
0.2	90	1.33 ±0.51	1.29 ±0.52	1.43 ±0.59

Table 6.7: Test set mean classification error for the first batch.

nu	size	initial FPR	injection FPR	attack FPR
0.05	30	0.30 ±0.45	10.20 ±3.86	66.31 ±8.94
0.05	50	0.97 ±0.91	9.65 ±6.19	52.84 ±13.55
0.05	70	1.49 ±1.00	11.97 ±9.11	41.47 ±11.86
0.05	90	1.03 ±1.08	4.14 ±3.42	27.49 ±7.85
0.1	30	0.08 ±0.10	2.54 ±2.12	34.31 ±14.56
0.1	50	0.41 ±0.67	2.20 ±1.65	15.35 ±8.43
0.1	70	0.27 ±0.24	1.05 ±0.99	6.28 ±3.55
0.1	90	0.17 ±0.19	0.62 ±0.59	2.08 ±1.83
0.15	30	0.46 ±0.83	2.48 ±3.26	14.84 ±11.50
0.15	50	0.12 ±0.10	0.53 ±0.56	1.34 ±1.73
0.15	70	0.13 ±0.13	0.38 ±0.28	0.90 ±1.12
0.15	90	0.11 ±0.10	0.13 ±0.13	0.11 ±0.10
0.2	30	0.06 ±0.10	0.22 ±0.39	2.98 ±7.52
0.2	50	0.08 ±0.10	0.19 ±0.30	0.23 ±0.54
0.2	70	0.08 ±0.10	0.10 ±0.10	0.06 ±0.09
0.2	90	0.09 ±0.10	0.09 ±0.10	0.06 ±0.09

Table 6.8: Validation set false positive rate for the first batch.

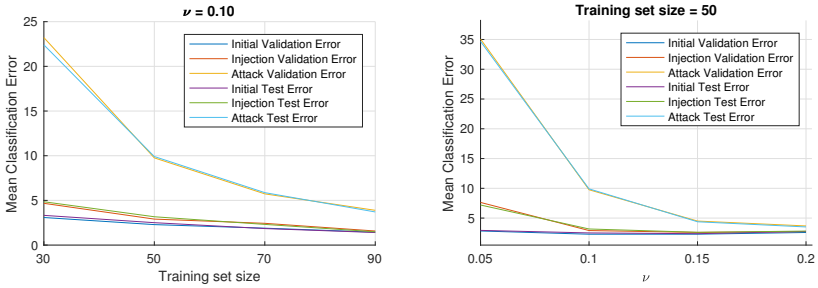


Figure 6.11: Mean classification error fixing nu=0.1 (left) and training set size=50 (right) for validation and test sets across different settings.

We now focus on the best-behaving batch to provide some insights about the effectiveness of the attack. Table 6.6 and 6.7 show the *mean CE* for, respectively, the validation and the test set. We can appreciate the outstanding result of $58.20\% \pm 6.39\%$ with $\nu = 0.05$ and $size = 30$. This example demonstrates that choosing the right set of identities may substantially increase the final error. On the other hand, it also underlines the importance of using a large set of identities to increase the likelihood of finding an effective attack point.

We notice an interesting result by considering Table 6.8, showing the FPR over the validation set. The effectiveness is enhanced for smaller values of ν and *size* if compared to the average result across all the batches; on the contrary, this batch is demonstrated to be more resilient in safer settings (ν>=0.15 and *size*>=50). This trend, together with the high standard deviation values, can be explained in two ways: the relatively small size of the sets, and the dependency between the attacked identity and the attacking identities. In the best case, the local maxima found by the attack procedure is tied to specific identities into the batch, allowing them to access the system with high confidence after the injection. On the other side, weak local maxima are likely tied to a FNR increase, making the system unusable rather than affecting its security.

Fixed the best batch, we consider a particular setting for further considerations: $\nu = 0.1$ $size = 50$. Figure 6.11 shows the *CE* decrease as a function of the *size* (left) and as a function of the parameter ν (right) for both validation and test sets.

Given our fixed setting, we extract three attack points among the 10 runs of the algorithms: the worst, the average and the best attack point w.r.t to *CE* increase. These points are shown in Figure 6.12: on the left, the *CE* increase; on the right, the associated ROC graph. We compute the latter by exploiting the decision score function as a threshold between FPs and TPs. In the average case, the decrease in the performance appears to be significant, even though we can observe a negligible decrease in the worst case. The main reason for this is the limited sampling of

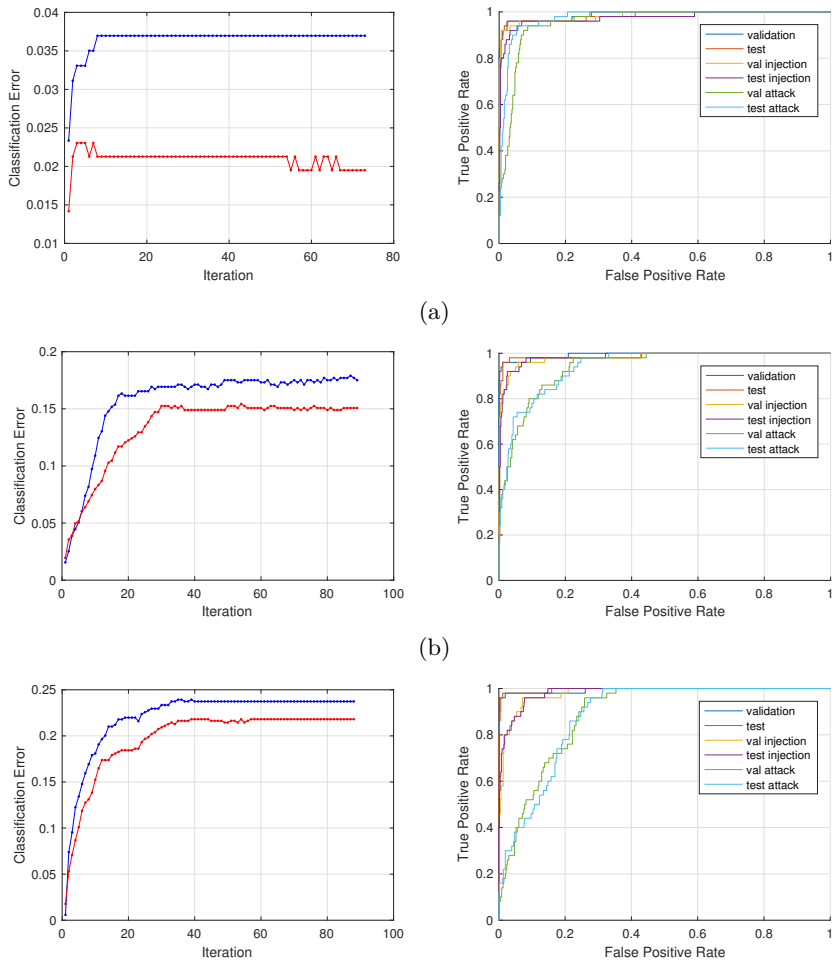


Figure 6.12: Classification error and ROC for the worst (a), the average (b) and the best attack point (c) across 10 iterations, considering the best-behaving batch. The CE is computed w.r.t to the validation set (blue) and the test set (red).

initial points from the validation set. An attacker may run the algorithm multiple times or increase the number of initial points to increase the probability to find an effective local optimum. Even with reasonable design choices, the model is shown to be vulnerable to the attack: the *CE* increases till 23% in the best case.

(a)

(b)

Figure 6.13: Initial attack point: (a) raw; (b) pre-processed.

Figure 6.14: Sequence of images generated by the sliding window procedure.

Inverse Feature-Mapping

To reverse the final attack point, we change the setting w.r.t the binary model. We consider a smaller initial window side l and an higher initial threshold h: the pairs (l, h) are (16,8), (8,12), (4,16) and (2,20). The starting image is an image from the training set identity, as shown in Figure 6.13. As in the binary classifier

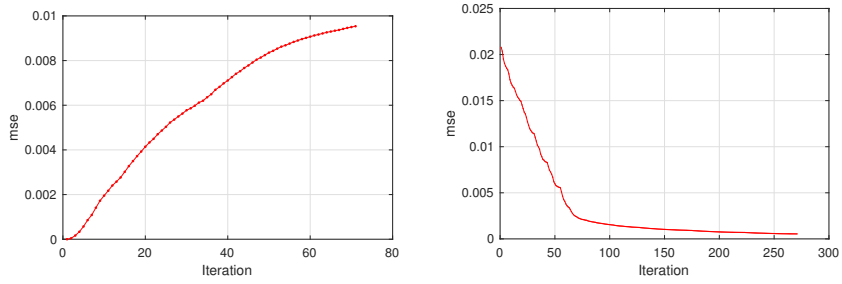


Figure 6.15: *mse* increase during the attack process (left); *mse* decrease during the reverse-mapping (right).

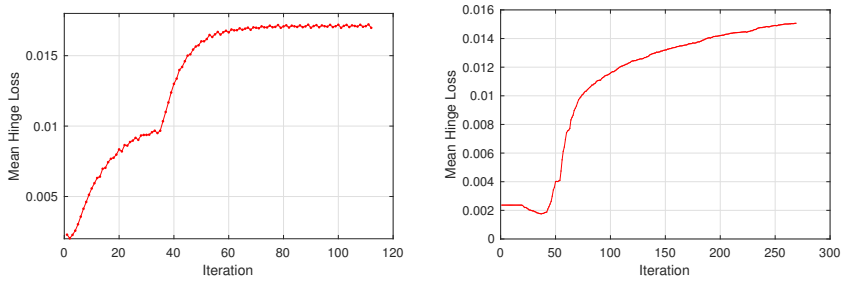


Figure 6.16: Comparison between the hinge loss computed during the attack (left) and the hinge loss of the reverse embedding, during the modification (right).

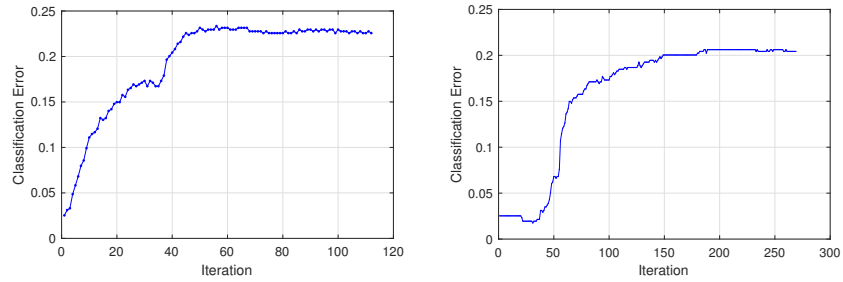


Figure 6.17: Comparison between the classification error computed during the attack (left) and the classification error of the reverse embedding, during the modification (right).

case, we store the embedding corresponding to the modified version of the image at the end of each row. In that way, we are able represent the evolution of the distance between the modified embedding and the objective, as well as L and CE. As before, we iterate 3 times for each l, obtaining 270 embeddings.

Figure 6.15 compares the distance between the initial attack point and the final attack point during the SGA process (left) with the minimization of the distance during the reverse mapping process (right). Starting from a distance of 2e−2, we approach the final attack point with a mse equal to 5e−4. The relationship between the distance of the embedding and L is shown in Figure 6.16: the difference between the optimal loss and the one obtained with the reverse procedure is equal to 2e−3. Correspondingly, the classification error increases till 21% where the analytical result was 23%, as shown in Figure 6.17.

We conclude that the injection of the adapted image produces a comparable detrimental effect w.r.t. the analytic computation of the attack point.

Chapter 7

Discussion and Future Directions

In this chapter we discuss the results of this work, with an emphasis on limitations, challenges and future directions.

Starting from a previously developed attack, we demonstrate its effectiveness in a security-sensitive application like a face recognition system. We model an SVM classifier as the target of an intelligent adversary, providing the latter with the possibility to change its training set. We demonstrate the effectiveness of the poisoning attack against a One-Class SVM by injecting a single sample into the system. Furthermore, we show a strong dependency between the hyper-parameter tuning of the model and the strength of the system against the attack.

The first countermeasure to this attack is the increase of the training set size. Since todays system are re-trained over time, they are susceptible to this kind of attack especially in the earlier deployment phases. A careful system designer would take into account a decreasing ν value as the training set is extended, in order to address this security issue. In this way, he can increase the usability of system with time, without affecting the security. Further research may extend this evaluation by addressing the ν - attack point relationship from a theoretical perspective.

The attack against the One-Class SVM relies on the search of an attacking identity into a validation space which is a batch of 10 identities. An extension of this work would consider a validation set composed by single identities with a greater number of samples each. In this way, it would be possible to evaluate the effectiveness of letting one single identity access the system after the injection of one, or more adversarial samples.

We introduce a novel reverse mapping technique to craft real-world image starting from a 128-dimensional feature vector. Despite previous work, we are relaxing

the adversarial model by considering the CNN as a black-box instead of exploiting its internal details. Through our experiments we demonstrate the link between the distance of two points and their respective hinge loss. This allows us to craft a real-world image that embodies an adversarial sample while representing whichever identity. Given the empirical nature of our technique, its effectiveness collides with an efficient use of the resources. For this reason, heuristics may be exploited. One example is a *best-first search* which performs an initial iteration to store a list of the windows which contributes the most to the distance minimization. The windows can be stored as pairs (x, y), identifying the first cell of the window. Then, the real modification is applied by sorting the list in decreasing order with respect to the distance metric. Future work may consider this path.

We extend the threat model developed by Biggio et al. [10], inheriting its limitations. The first assumption regards the knowledge of the attacker about the model details. The attacker should be able to train an exact copy of the target classifier to compute the attack point. Previous work [25] demonstrates that having access to the underlying data distribution can be enough to train a substitute of the model, achieving comparable results. Future work may address this problem by extending our experiments.

A second assumption is tied to the injection phase. It is unrealistic to consider that the adversary is able to inject an image which will be used to train the model. However, this scenario can be regarded as a starting point for future extensions. One possibility exploits the *continuos re-training* mechanisms of modern face authenticator. Instead of injecting the image which carries the attack point, we can think to a *continuous injection* of a slightly modified image. In this way, we can control the classification of the injected sample, keeping it always above certain probability thresholds. The system would seemingly accept correct inputs, instead the classification rule will move at each new re-training.

We are not considering the multi-point attack in our final analysis over the one-class SVM. However, the binary classifier demonstrates the effectiveness of multiple, sequential injections. Further research is needed to understand what is the best set of points to be injected to maximize the effectiveness of the attack.

Finally, we limit our scope to linear SVMs which is not the only classification algorithm we can consider to implement an authentication system. Future work would tackle the other kernels already supported by the attack strategy (e.g. the RBF kernel) as well as other classification algorithms that behaves comparably to our target.

Chapter 8

Conclusion

In this work, we presented the adaptation of an existing poisoning attack against an authentication system based on a state-of-the-art face recognition technique. The main focus of the attack was to investigate the security of a SVM learning model against an intelligent adversary, in the context of a face-based authentication system.

We demonstrated the effectiveness of the poisoning attack by injecting a single sample into the training set of the authenticator. We used 46 different identities from a publicly available dataset, for a total of approximately 5000 images. We cross-validated the attack by using subsets of 10 attacking identities against a single authenticated user. In this way, we were able to show a drastic drop in the classification accuracy up to 45%, which makes the system unusable.

Moreover, we developed a technique to reverse the black-box feature extraction process. In this way, we demonstrated how the attacker can present real-world images to the system instead of feature vectors. By attacking the system a second time, we registered a comparable drop in accuracy.

In an attempt to evaluate the security of the SVM, we explored the parameter tuning phase. Our findings underline the security risks associated with a poor design. Based on our extensive analysis, we conclude that a careful system designer should consider a trade-off between usability and resilience.

Machine learning algorithms represent, at the same time, the enabling-technology and the biggest concern in the field of biometrics for authentication. This work demonstrated that poisoning attacks represent a real threat against face-based authentication systems. The results underlined the urge of considering adversarial machine learning during the system design phase, by carefully tuning the hyper-parameters. Thus, proactively anticipate an external adversary can make the difference between a resilient system and a vulnerable one.

Bibliography

[1] Z. Akhtar, C. Micheloni, and G. L. Foresti. Biometric liveness detection: Challenges and research opportunities. *IEEE Security Privacy*, 13(5):63–72, Sept 2015.

[2] A. Al Abdulwahid, N. Clarke, I. Stengel, S. Furnell, and C. Reich. Continuous and transparent multimodal authentication: Reviewing the state of the art. *Cluster Computing*, 19(1):455–474, Mar. 2016.

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

[4] J. Suykens. Artificial Neural Networks. Katholieke Universiteit Leuven Department of Electrical Engineering, ESAT-STADIUS, 2013.

[5] Apple. Face id. https://support.apple.com/HT208109. Accessed: 2018-07-07.

[6] G. B. Huang, M. Mattar, T. Berg, and E. Learned-Miller. Labeled faces in the wild: A database forstudying face recognition in unconstrained environments. 10 2008.

[7] B. Biggio, L. Didaci, G. Fumera, and F. Roli. Poisoning attacks to compromise face templates. In *2013 International Conference on Biometrics (ICB)*, pages 1–7, June 2013.

[8] B. Biggio, G. Fumera, and F. Roli. Security evaluation of pattern classifiers under attack. *IEEE Transactions on Knowledge and Data Engineering*, 26(4):984–996, April 2014.

[9] B. Biggio, B. Nelson, and P. Laskov. Support vector machines under adversarial label noise. In C.-N. Hsu and W. S. Lee, editors, *Proceedings of the Asian Conference on Machine Learning*, volume 20 of *Proceedings of Machine Learning Research*, pages 97–112, South Garden Hotels and Resorts, Taoyuan, Taiwain, 14–15 Nov 2011. PMLR.

[10] B. Biggio, B. Nelson, and P. Laskov. Poisoning attacks against support vector machines. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, ICML'12, pages 1467–1474, USA, 2012.

Omnipress.

[11] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[12] G. Cauwenberghs and T. Poggio. Incremental and decremental support vector machine learning, 2000.

[13] C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[14] R. Collobert, K. Kavukcuoglu, and C. Farabet. Torch7: A matlab-like environment for machine learning. 01 2011.

[15] M. Gadaleta and M. Rossi. Idnet: Smartphone-based gait recognition with convolutional neural networks. *CoRR*, abs/1606.03238, 2016.

[16] J. Galbally, S. Marcel, and J. Fierrez. Biometric antispoofing methods: A survey in face recognition. *IEEE Access*, 2:1530–1552, 2014.

[17] L. Huang, A. D. Joseph, B. Nelson, B. I. Rubinstein, and J. D. Tygar. Adversarial machine learning. In *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence*, AISec '11, pages 43–58, New York, NY, USA, 2011. ACM.

[18] R. Jafri and H. Arabnia. A survey of face recognition techniques. 5:41–68, 06 2009.

[19] D. E. King. Dlib-ml: A machine learning toolkit. *J. Mach. Learn. Res.*, 10:1755–1758, Dec. 2009.

[20] H.-T. Lin, C.-J. Lin, and R. C. Weng. A note on platt's probabilistic outputs for support vector machines. *Machine Learning*, 68(3):267–276, Oct 2007.

[21] D. Lowd and C. Meek. Adversarial learning. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, KDD '05, pages 641–647, New York, NY, USA, 2005. ACM.

[22] T. M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

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

[24] T. E. Oliphant. *Guide to NumPy*. CreateSpace Independent Publishing Platform, USA, 2nd edition, 2015.

[25] N. Papernot, P. D. McDaniel, and I. J. Goodfellow. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. *CoRR*, abs/1605.07277, 2016.

[26] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in python. *J. Mach. Learn. Res.*, 12:2825–2830, Nov. 2011.

[27] G. Rossum. Python reference manual. Technical report, Amsterdam, The Netherlands, The Netherlands, 1995.

[28] B. Schölkopf, R. Williamson, A. Smola, J. Shawe-Taylor, and J. Platt. Support vector method for novelty detection. In *Proceedings of the 12th International Conference on Neural Information Processing Systems*, NIPS'99, pages 582–588, Cambridge, MA, USA, 1999. MIT Press.

[29] F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. *CoRR*, abs/1503.03832, 2015.

[30] S. Shalev-Shwartz and S. Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, New York, NY, USA, 2014.

[31] C. Smutz and A. Stavrou. Malicious pdf detection using metadata and structural features. In *Proceedings of the 28th Annual Computer Security Applications Conference*, ACSAC '12, pages 239–248, New York, NY, USA, 2012. ACM.

[32] Y. Sun, X. Wang, and X. Tang. Deep learning face representation by joint identification-verification. *CoRR*, abs/1406.4773, 2014.

[33] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf. Deepface: Closing the gap to human-level performance in face verification. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1701–1708, June 2014.

[34] A. Tolba, A. El-Baz, and A. El-Harby. Face recognition: A literature review. 2:88–103, 01 2008.

[35] S. University. Cs class cs231n. http://cs231n.github.io/neural-networks-1. Accessed: 2018-07-07.

[36] L. Wolf, T. Hassner, and I. Maoz. Face recognition in unconstrained videos with matched background similarity. In *CVPR 2011*, pages 529–534, June 2011.

[37] H. Xiao and C. Eckert. Adversarial label flips attack on support vector machines. 242:870–875, 01 2012.