DEFEAT DATA POISONING ATTACKS ON FACIAL RECOGNITION APPLICATIONS

A Dissertation

presented to

the Faculty of the Graduate School

at the University of Missouri-Columbia

In Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy

---

by

DALTON RUSSELL COLE

Dr. Dan Lin, Dissertation Supervisor

July 2021

The undersigned, appointed by the Dean of the Graduate School, have examined the dissertation entitled:

DEFEAT DATA POISONING ATTACKS ON
FACIAL RECOGNITION APPLICATIONS

presented by Dalton Russell Cole,
a candidate for the degree of Doctor of Philosophy and hereby certify that, in their opinion, it is worthy of acceptance.

Dr. Dan Lin

Dr. Wei Jiang

Dr. Khaza Anuarul Hoque

Dr. Jian Lin

ACKNOWLEDGMENTS

I would like to thank the Scholarship For Service (SFS) program for giving me the opportunity to pursue my Ph.D. I would also like to thank Dr. Dan Lin for giving me the opportunity to transfer to the University of Missouri - Columbia to work with her. Without her help, I may have given up on my dream of graduating with a Ph.D. degree. I owe her many thanks for her continued help throughout my time as a graduate student.

I would like to extend my gratitude to my committee members, Dr. Wei Jiang, Dr. Jian Lin, and Dr. Khaza Anuarul Hoque, along with Dr. Dan Lin. I transferred to the University of Missouri - Columbia midway through graduate school. They accepted the responsibility of being members of my doctoral committee, for which I greatly thank them. I would like to make a special thanks to Dr. Wei Jiang. I first worked with him when I was an undergraduate student, either taking courses under him or working with him when I was the president of Missouri University of Science and Technology's Association of Computing Machinery chapter. He has always been a caring and friendly person to whom I could turn for advice.

I would like to thank my lab members at the University of Missouri - Columbia for their aid in my research. Sara Newman and Maya Cutkosky have been a great help in completing this dissertation.

Lastly, I would like to thank my family and friends for keeping me sane during graduate school, especially my grandmother for reading this dissertation so many times and Henry Wong for reminding me that some people just want a doctoral degree for the fun of it.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

ABBREVIATIONS

AUC	Area Under Curve
AUROC	Area Under ROC
CNN	Convolutional Neural Network
DEFEAT	Deep-neural-network and Embedded FEAture-based deTector
DNN	Deep Neural Network
GAN	Generative Adversarial Network
KNN	K-Nearest Neighbours
LSTM	Long Short Term Memory
NN	Neural Network
PCA	Principal Component Analysis
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Networks
ROC	Receiver Operating Characteristics
SVM	Support Vector Machine

DEFEAT DATA POISONING ATTACKS ON FACIAL RECOGNITION APPLICATIONS

Dalton Russell Cole

Dr. Dan Lin, Dissertation Supervisor

ABSTRACT

In the modern era, facial photos are used for a wide array of applications, from logging into a smartphone to bragging about a weekend getaway. With the vast amount of use cases for facial images, adversaries will attack these applications for profit. This dissertation focuses on two major applications of facial photos: facial authentication and deepfakes.

Facial authentication has become increasingly popular on personal devices. Due to the ease of use, it has great potential to be widely deployed for web-service authentication in the near future in which people can easily log on to online accounts from different devices without memorizing lengthy passwords. However, the growing number of attacks targeting machine learning, especially Deep Neural Networks (DNN), which is commonly used for facial recognition, imposes big challenges on the successful roll-out of such web-service facial authentication. We demonstrate a new data poisoning attack, called *replacement data poisoning*, which does not require the adversary to have any knowledge of the server-side and simply needs a handful of malicious photo injections to enable an attacker to impersonate the victim in existing facial authentication systems. We then propose a novel defensive approach called DEFEAT that leverages deep learning techniques to automatically detect such attacks. Our experiments using real-world datasets achieve a detection accuracy of over 90%.

Deepfakes target specific individuals to cause shame or misinformation. With the spread of fake news, deepfakes have become incredibly prevalent in recent years. With deepfakes, an adversary could have photographic or even video-graphic “proof” of someone, such as a politician, committing a devious act or saying untrue words. Our deepfake work consists of two parts. First, we propose a label flipping data poisoning attack targeting deepfake detectors. With over a 99% poison success rate in most cases, this attack demonstrates the devastating effects a data poisoning attack can have on deepfake detectors and how important a need to defend against this assault is. Our second contribution revolves around defending deepfake detectors from such an attack. We propose several defense strategies, most notably a convolutional neural network (CNN) based strategy to detect poisoned images. Our CNN-based approach achieves a greater than 98% poison detection rate while keeping the number of false positives to a minimum with a precision rate of over 99% in most cases.

Chapter 1

INTRODUCTION

This piece is composed of two separate major works involving computer vision. Our first major work involves protecting facial authentication against a novel data poisoning attack. Our second major work comprises of attacking a state-of-the-art deepfake detector using a label flipping data poisoning attack. We then propose and experimentally verify various defense strategies against such an attack.

1.1 FACIAL AUTHENTICATION

Today facial authentication has been commonly used to unlock personal devices such as smartphones and laptops. Due to its ease of use, the next major horizon for facial authentication applications may be web services [1]. According to statistics [2], an internet user has an average of 26 different online accounts but only 5 unique passwords for these accounts. This fact is not surprising since it is hard for a person to memorize too many different passwords. One may argue that password manager software could mitigate the problem of password explosion. However, this actually may not be very effective considering that a person usually accesses web services from a variety of personal devices at home and work. In addition, password managers generally require a password themselves. It is a tedious and almost infeasible task for a person to record the new password for new web services on all of his/her devices immediately upon new account creation; not to mention, that the person may not have access to some devices (such as those at work or future new devices) at the moment the new account is created. Facial authentication is a different story. With facial authentication, a person's live face becomes the key to logging into web services. This is convenient and swift and can be done from a multitude of devices. Its promising market potential has fostered several releases of facial recognition APIs [3, 4]. It is envisioned that facial authentication would be widely adopted in web services in the not too distant future.

For the successful deployment of facial authentication for online services, security is undoubtedly on the top of the list to be addressed. Facial authentication relies on accurate facial recognition. The most recent facial recognition techniques such as FaceNet [5], which achieve high accuracy, are built upon deep neural networks (DNN) [6]. Unfortunately, DNN models are vulnerable to a variety of emerging attacks, such as adversarial input attacks [7, 8, 9, 10, 11, 12, 13, 14], data poisoning attacks [15, 16, 17, 18, 19], and model stealing attacks [20, 21, 22, 23]. In the context of facial authentication for web services, adversarial input attacks and data poisoning attacks could be the most devastating threats. Both attacks aim to mislead the classifier to misclassify the input image. In terms of facial recognition, such attacks could result in a legitimate user being misclassified and denied access to the service; or even worse, make an attacker be recognized as a legitimate user and gain access to the victim's account. Although there have been some defensive mechanisms for adversarial input attacks and data poisoning attacks on image classifiers [10, 14, 24, 25, 26], to the best of our knowledge, none of the existing works consider the following attack scenario that can easily occur in future facial authentication for web services, and none of the existing works is effective at defending such attacks.

As shown in Figure 1.1, the new web-service facial authentication attack may happen when a person signs up for a new web service or updates his/her facial images for a web service. Facial authentication typically requires the users to take photos of

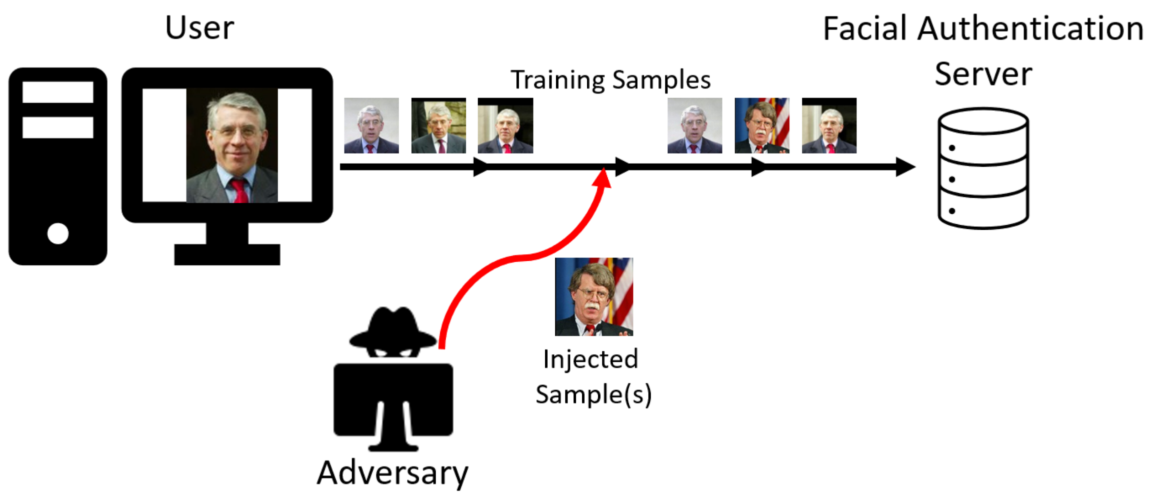


Figure 1.1: Proposed Data Poisoning Attack

themselves to train the facial recognition classifier. Our study shows that an attacker just needs to sneak in less than a handful of his/her photos during this process; the facial authentication system at the service provider side will later recognize both the authentic user and the attacker as the same person. Thus, both the authentic user and the attacker will have the same access rights to the account that the user registered. Such an attack can be conducted by exploiting the vulnerability of the victim's home network and router via a man-in-the-middle attack. A 2020 security review of 127 popular home routers found vulnerabilities that could result in a man-in-the-middle attack [27, 28], showing that man-in-the-middle attacks are still very relevant today. As this attack pollutes the training dataset, it falls under the category of a data poisoning attack. However, this new attack is easier to implement than most existing data poisoning attacks, as well as, adversarial input attacks; and are harder to detect than label flipping attacks or injection attacks. This is because our new attack does not require the attacker to know any insider information on the server-side, whereas existing machine learning attacks [8, 29] typically require the attacker to compromise the server to gain knowledge of feature vectors produced by the deep neural network (DNN). For example, to impersonate a person, one previous attack strategy [8] requires the attacker to know the victim's facial feature vector generated by the DNN on the server-side to create special glasses that can produce a similar feature vector as the victim when an attacker wears it. Moreover, our new attack is stealthy since it does not affect the normal use of the infected account (our secondary goal is to not decrease the overall accuracy of the user and system). Once the attacker gains the same access right as the legitimate user, the attacker can track the user's service usage over time, impersonate the user at any desired time, or use this account to pivot to other platforms (for example, a cross-site scripting attack or sign-in using their login information, similar to Google and Facebook's login to other site feature [30, 31]). After the attack, the attacker can easily purchase items using the victim's account if the victim does not regularly check his/her order or credit card history; the attacker can also post or send misinformation on behalf of the victim to ruin the victim's reputation. Currently, there is no effective defense mechanism proposed to prevent such an attack.

In this work, we will first demonstrate the devastating effect that our new data poisoning attack can impose on web-service-based facial authentication. Then, we will present a novel defensive strategy called DEFEAT (Deep-neural-network and Embedded FEAture-based deTector).

Specifically, we have tested that with only 4 or 5 attacker's face photo mixed in the user's training photos (another 4 or 5 photos), the attacker will be able to impersonate the user in future authentication without dropping the overall facial recognition accuracy, i.e., without raising an alarm to the facial authentication system. We also found that it is difficult to distinguish the attacker's feature vector from the authentic user's by using only statistical analysis and distance comparison. Our hypothesis of such phenomenon is that since facial recognition systems, such as FaceNet, strive to achieve high recognition accuracy and since they do not know the training set of a given user contains photos of different faces, the contaminated feature vectors (i.e., those being attacked) are then generated based on common features between the user and the attacker as to ensure both the original user and the attacker can authenticate using their own photos. As a result, various distances (e.g., $l - norm$) are not sufficient to measure the differences between the victim's feature vector and the attacker's feature vector since they are intentionally generated by DNN to be very similar for the goal of maintaining high recognition accuracy. However, this does not stop us from pursuing an effective method to detect these malicious attempts.

Based on our hypothesis that the contaminated feature vectors are generated by extracting common features from two people's faces (i.e., the victim and the attacker) whereas the non-contaminated feature vectors are based on the features of only one person, we propose an intelligent discriminator, DEFEAT, to identify the potentially subtle differences in these two kinds of feature vectors. The DEFEAT discriminator has the base structure of a DNN and a classifier (i.e. k-nearest neighbors or SVM) model. We design various concatenation approaches to create training inputs for the discriminator. We optimize the layers of the DNN in DEFEAT for both accuracy and efficiency. Upon real-time detection, DEFEAT takes the feature vector output by FaceNet and produces a probability of whether or not the input feature vector is contaminated. The probability is then sent to the KNN model to produce a binary decision: attacked or not. We have evaluated our approach in real datasets that represent both consistent background settings and diverse background settings. Our experimental results show that our discriminator achieves more than 90% detection accuracy. Our contributions are summarized as follows:

- We study a new data poisoning attack to facial authentication which allows the attacker to easily impersonate the victim.
- We propose novel discriminators to detect the above impersonation attack. Our experiments on real datasets demonstrate that our discriminator achieves very high detection accuracy in various settings.

1.2 DEEPFAKES

The second major piece in this work involves protecting deepfake detectors from adversarial attacks. Deepfakes are targeted malicious attacks where an existing face is replaced by another. Deepfakes are generally generated to defame someone on a personal or national scale. National threats include politicians being impersonated by third parties, making them seem like they are saying something they do not agree with [32, 33]. This is especially critical given the rise of fake news in social media [34, 35]. In addition to deepfakes, which alter someone's identity, there are facial reenactment attacks. In facial reenactment, an adversary alters the target's expression and lip movements to match the adversary's. To express the damaging effects of deepfakes and facial reenactment, Jordan Peele created a facial reenactment video with President Barack Obama's face [33]. In this video, Peele uses President Obama's face to make it seem like Obama uttered controversial words about other politicians. Deepfakes are such a national security risk that the Pentagon is racing to reliably detect deepfake videos [36]. Governments are not the only ones requiring robust deepfake detection. Mid 2020, social media giant Facebook held a deepfake detection challenge on Kaggle [37]. Due to lower hardware requirements, deepfakes became available to the general public in 2016 [38]. In 2018, research-focused publicly available datasets were popularized [39]. With the quick evolution of deepfake technology and neural networks, there are already multiple generations of deepfake datasets. Each generation improves the quality of the synthetic image as well as decreasing the time required to train a network to generate photos.

Given the prevalence of deepfakes, there has arisen a need for deepfake detectors, something that can determine if an image is genuine ($real$) or synthetic ($fake$). Convolutional neural networks (CNNs) are the most popular way to perform deepfake detection [40]. In addition to new neural network architectures and methods being proposed, old models have been found to perform especially well in this classification task. One notable architecture is XceptionNet which was first proposed as a deepfake detection method in [41] and has been thoroughly tested [39, 40, 42, 43]. In general, CNN-based methods use the artifacts generated from synthetic image generation to find $fake$ images. However, there are many other methods, including non-temporarily aware methods such as examining head poses [44] or eye color [45]. Temporarily aware methods include keeping track of eye blinking [46] or examining facial expressions plus head movements [47].

Naturally, adversaries are trying to fool deepfake detection. Research on attacking deepfake detectors primarily focuses on evasion attacks, a type of attack that occurs after a model has already been trained [43, 48, 49]. Generally, evasion attacks assume a white-box approach, where the adversary has full knowledge of the detection model. The adversary then adds the minimum amount of perturbation required to an image to induce misclassification. A perturbation filter generally looks like white noise to a human observer, but when added to the target image, it is normally undetectable to the human eye. Considering that deepfakes are meant to fool humans, adding a perturbation filter to a deepfake to cause misclassification can cause dire results. Neither machine nor human may tell that an image is synthetic. Fortunately, perturbation filters are normally created per sample and per detector. Universal perturbation filters, filters that work on multiple images for a single detector, do exist, but generally have a lower attack success rate than individualized perturbation filters [48].

Evasion attacks occur during testing time, i.e. when the model is in a live environment. Unlike evasion attacks, data poisoning attacks occur during training time. To the best of our knowledge, there has been no research on data poisoning attacks on deepfake detectors. The adversary *poisons* the training data that a model uses to learn. With a label flipping data poisoning attack specifically, the adversary changes the labels, or y values, of the training data to induce misclassification in the live environment.

To accomplish label flipping attacks, an adversary only needs access to the label's portion of a dataset. With it, the adversary can alter the labels and diminish the detector's ability to detect deepfakes. If the adversary desires the deepfakes of a particular person to be mislabeled, they may flip only the target's labels to instigate misclassification, thus bypassing detection. This could lead to calamitous results. For example, if an adversary attacks an adult website's deepfake detection, deepfakes of a politician could be uploaded and cause disastrous results for their credibility or end their political career completely [50].

To demonstrate how powerful and devastating data poisoning attacks are to deepfake detection, we propose a label flipping data poisoning attack. Our attack consists of selecting a target and flipping all labels that correspond to a synthetic image with the target's face from $fake$ to $real$. Multiple targets may be selected from the same training dataset. We explore a diverse set of configurations for our attack, such as attacking only a single dataset, varying the number of poisoned identities, and training a detector with multiple types of synthetically generated images.

In this work, we also propose multiple discriminators to detect such an attack. Our most basic discriminator is an outlier detection method. More complex solutions include an SVM, a random forest, and a DNN-based approach. Our primary focus, however, is on a CNN-based discriminator. We investigate various ways to optimize our CNN-based discriminator, such as altering our discriminator's connection point to the detector.

With the discriminator in place, the detector and discriminator can work in tandem to root out poisoned images. To keep synthetic media detectors up to date, they must be retrained periodically with new images that were produced using updated synthetic media generation tools or other altering factors, such as a different type of image compression algorithm. Figure 1.2 demonstrates how unsuccessful a deepfake

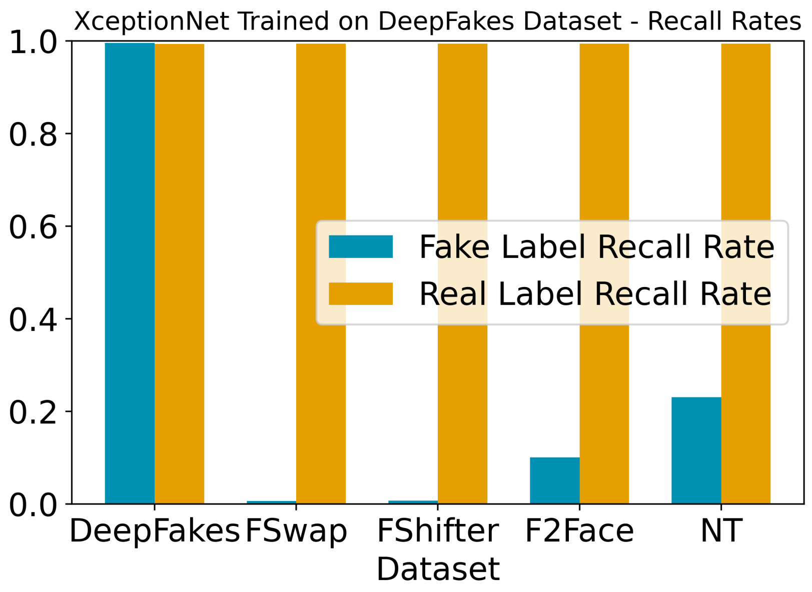


Figure 1.2: XceptionNet Trained on the DeepFakes Dataset and Tested on Other Datasets

detector is when confronted with images using a method it was not trained upon. To generate this figure, the deepfake detector XceptionNet was trained using the DeepFakes dataset [51, 52] and then tested on four other deepfakes or facial reenactment datasets. As shown in the figure, XceptionNet can recall nearly all *real* and *fake* photos as long as the synthetic images were generated using the same method as it was trained using. When confronted with different methods, the detector struggles to correctly classify synthetic images as *fake*. With regular updates, the detector can learn to recognize new synthetic media types. To protect itself from an unwanted adversarial attack, the discriminator must discover any adversarial images before allowing the detector to learn the erroneous information.

Our contributions are summarized as follows:

- We study the effects of a label flipping data poisoning attack on deepfake detection. We show how devastating and successful this attack is through experimental results.

- We propose various discriminators, each with different advantages, such as one discriminator requiring no training data and another discriminator achieving over 99% poison and benign recall rates.

Chapter 2

LITERATURE REVIEW

In this chapter, we conduct a thorough literature review on three different attack methods against facial recognition applications, including deepfake detectors. Section 2.1 explores adversarial attacks. Section 2.2 covers model stealing attacks. Finally, Section 2.3 provides a literature review on our main attack method: data poisoning attacks.

2.1 ADVERSARIAL ATTACKS

Adversarial input attacks, also known as evasion attacks [53], typically occur after the machine learning algorithm has completed its training process. This is opposed to data poisoning attacks, which take place at training time. The goal of adversarial input attacks is to perturb the input data in a way that fools a classifier, i.e., force the input data to be misclassified. These inputs are usually crafted by adding perturbation to an authentic image [7, 8, 9, 10].

2.1.1 Facial Recognition Adversarial Inputs

Specific to face recognition models, there is an abundance of works on how to compute perturbations to cause errors in the face recognition process [7, 8, 9, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66]. For example, [7, 54] turn perturbation generation into an optimization problem and compute the perturbation in one single large step based on the model's loss. [55] extends the idea to computing perturbations iteratively. [56, 58] seek to alter the least amount of pixels possible, while [57] focuses on modifying only one pixel. [9, 59] modify images iteratively using the information on the decision boundaries of the target model. These algorithms are all designed for a single given image, needing to be run separately for each additional target image. [60, 61] work on any image to fool the target network. [62, 63, 64, 65] generate entire adversarial images that appear similar to clean images and fool the target model adequately. Another interesting way to perform image perturbation is that attackers put on customized accessories such as glasses in front of the camera to pretend to be the victim [8, 66]; these special accessories are designed based on the victim's feature vectors generated by the face recognition model. With this said, most of the existing adversarial input attacks [7, 8, 9, 54, 55, 56, 58, 59, 60, 63, 64, 65, 66] require white box knowledge of the model, i.e., knowing the internal parameters of the model, which may be hard to achieve in real scenarios. Only a few adversarial input attacks [57, 61, 62] can treat the target model as a black box and still conduct adversarial input attacks.

Currently, the most effective defense mechanism against the adversarial input attacks is adversarial training [54, 58, 67, 68, 69] which enhances the robustness of the neural networks by teaching it adversarial samples during the training process. It is worth noting that such a defense will not be effective to prevent our attack. Under our proposed attack, the attacker's photos do not contain any kind of noise. They are normal photos like those of other normal users. The face authentication model is trained to treat the attacker's face the same as the victim's face. The adversarial samples would be the attacker's facial photos, and the expectation is that the face authentication model will label the attacker's photo as malicious. However, this is not practical since the attacker's photos are normal photos and the system does not know who is the attacker beforehand. It is unrealistic to take a randomly picked normal photo to label as malicious during training.

2.1.2 Adversarial Deepfakes

The majority of work on attacking deepfake detectors involves adding perturbation to the deepfake images, either via a black-box or white-box attack. [43] attacked two classifiers: XceptionNet [41] and MesoNet [42]. They applied two different attack styles to both white-box and black-box attacks: a normal version and a robust version. The robust version involved adding additional transformations to the images before training. The idea being that compression can remove the finely tuned perturbation added to images. Since many places where deepfakes are used, for example, social media, compress images heavily, the perturbation needs to remain even through compression. Four transformation functions were randomly applied to perturbed images: Gaussian blur, Gaussian noise addition, translation, or downsizing and upsizing using bilinear re-sampling. For the white-box attack, where the adversary has full access to the detector, the iterative gradient sign method was used for the perturbation filer generation. The magnitude of the perturbation was clipped to prevent it from being noticed by human observers. For the black-box attack, the authors assumed knowledge about the output probabilities of *real* or *fake*. From there, they used a natural evolutionary strategy where under a search distribution, the expected value for a function is maximized. They tested their attack using the FaceForensics++ datasets, specifically using the raw and low-quality partitions. With only a $\mathcal{L}_\infty$ of 0.0004, they had an attack success rate of 100.0 on the raw faceswap dataset and 43.13 on the low-quality faceswap dataset using the normal white-box attack. Their robust white-box attack had far better results with the compressed data. With only a $\mathcal{L}_\infty = 0.013$, they achieved a 95.33 success rate on the low-quality dataset on XceptionNet. The black-box attacks required a much larger level of perturbation to achieve similar results. A $\mathcal{L}_\infty = 0.045$, ten times as much as the white-box attack, was required to achieve

96.77 and 23.50 success rate on the raw and low-quality datasets, respectively. The robust black-box attack performed better on the compressed dataset, with a 98.97 and 63.26 attack success rate on the raw and low-quality datasets, respectively, with a similar $\mathcal{L}_\infty$ of 0.052. In general, MesoNet was slightly harder to fool using the raw dataset, but easier to fool using the low-quality compressed dataset, when compared to XceptionNet.

[48] created five different adversarial perturbation methods to attack [70] and [71]. They used the dataset provided in [70] for training and testing. Their first attack strategy is a white-box distortion-minimizing attack. They limited their attack to only flipping the lowest bit of each pixel, thus the maximum perturbation to any pixel was 1/255. Half of all faked images were classified as real by flipping the least significant bit in 1% of pixels. At 11% of pixel flips, nearly 100% of fake images were labeled as real. Using either the $\mathcal{L}_2$ or $\mathcal{L}_0$ distortion in their minimization function had similar results. Their second attack strategy was similar to their first, however, this strategy minimizes the perturbation of the p-norm as opposed to the perturbation itself, where $p = 0, 1, 2, \infty$. When 40% of pixel's lowest-order bit was flipped, the AUC was reduced from 0.966 to 0.27 on raw, uncompressed, images. Their third attack strategy relied on generating a universal adversarial-patch, or a single perturbation filter that could be applied to every image. This type of filter saves computational time since a single filter can be applied to unseen fake images, as opposed to a new filter being generated for every individual image. Their universal patch was limited to 24x24 pixels, or 1% of the image, and is highly noticeable to a human observer. They were able to decrease the AUC from 0.966 down to 0.085. Their fourth attack strategy was a universal latent-space attack. They used the latent space of a GAN network to generate new adversarial images. Specifically, they searched the latent space to find optimal input vectors for low-level attributes, such as freckles. They were able to reduce the AUC from 0.99 to 0.17. Their final proposed attack is a black-box attack

Model	Unperturbed	FGSM Black-Box	FGSM White-Box	CW-$\mathcal{L}_2$ Black-Box	CW-$\mathcal{L}_2$ White-Box
VGG	99.7%	8.9%	0.0%	26.6%	0.0%
ResNet	95.4%	20.8%	7.5%	4.6%	0.0%

Table 2.1: Deepfake Detector Attack Results from [71]

strategy. Their black-box attack strategy relied on generating perturbations based on a separate detector and transferring this knowledge to the target detector. They were able to reduce the AUC from 0.96 down to 0.22 using this method.

[71] used two different attack strategies to generate adversarial perturbation. Their first attack strategy used the fast gradient sign method (FGSM) proposed in [54] to find the minimum perturbation filter. Their second attack used the Carlini and Wagner (CW) $\mathcal{L}_2$-norm, which minimizes the $\mathcal{L}_2$-norm while maximizing the misclassification rate. The FGSM is a popular, efficient, and fast attack method, while the CW-$\mathcal{L}_2$ is stronger but much slower. They used their dataset to train VGG-16 and ResNet-18 networks as detectors. They were able to achieve 99.7% and 95.4% detection accuracy on the VGG and ResNet detectors, respectively. They implemented their attacks from both a black-box and white-box perspective. Table 2.1 summarizes their results. In addition, they tested two defense strategies for their attacks. Their first strategy used Lipschitz regularization to constrain the gradient of the detector. They were able to increase the detection accuracy up to 53.2%, but, in general, the detection accuracy marginally increased post-attack. Their second defense strategy used the deep image prior, which was originally used for image restoration. The goal of deep image prior is recover $\mathbf{x}$ given $\mathbf{x}_c$, where $\mathbf{x}_c$ is the corrupted image. This method also attempts to remove the artifacts generated by compression. A generative CNN is used to restore the image. This defense strategy was able to achieve up to 97.0% accuracy or 99.2% AUROC post-attack.

These works show that compressed images prove difficult for perturbation attacks since compression often removes the added perturbation. Unlike these attack strategies, our attack strategy is a label flipping data poisoning attack. Opposed to perturbation attacks that attack a trained network, a data poisoning attack attacks the training data itself, leaving a vulnerability to be exploited later. Thus, if the attack is not noticed during training, it is difficult to unlearn the poisoned data [25].

2.2 MODEL STEALING ATTACKS

Model stealing attacks do not affect the classifier or manipulate the classification output, and hence this type of attack is also different from ours which aims to mislead the classifier. During model stealing attacks, adversaries usually target Machine Learning as a Service (MLaaS) models, such as AWS and Google Cloud AI, and try to estimate the target model's hyperparameters [20, 72]. Model stealing attacks can be applied to any type of model, not just neural networks.

To estimate the model, a large number of queries need to be made to find the decision boundary between classification targets. The authors of [72] used several different versions of probability approximately correct (PAC) learning to reduce the number of queries. PAC learning assumes the adversary has access to a distribution of the dataset. The adversary then queries the model to generate a hypothesis with a low expected loss. They also explore two active versions of PAC: stream-based sampling and pool-based sampling. In stream-based sampling, samples are selected sequentially. Once a sample was either used to query the model or discarded, the sample cannot be considered again. With pool-based sampling, samples are selected from a pool of samples. This allows for better decision-making once you start learning about the model. In addition to PAC learning, the authors also exampled query synthesis active learning. This strategy uses a pool-based approach; however, the samples can be generated independently of the given dataset distribution. This is much more realistic in the real world where it is unlikely that the adversary will have the dataset that was used to train the model.

2.3 DATA POISONING ATTACKS

The implemented attacks in this dissertation are data poisoning attacks. With data poisoning attacks, attackers manipulate the training data to mislead the model and cause the model to misbehave during its runtime. These attacks typically involve introducing a perturbation to a clean or untouched subset of the training data. This perturbation is crafted such that the model learns to misbehave when trained on these samples. The number of perturbed samples required to achieve this goal varies, depending on the type of perturbation conducted. These types of attacks can be targeted or untargeted. Untargeted data poisoning attacks seek to hinder the rate at which the machine learning model learns, while targeted attacks seek to cause a particular input, or label, to be misclassified when the model is deployed. Untargeted attacks are relatively easy to be noticed since it causes a significant drop in classifiers' accuracy. In contrast, targeted data poisoning attacks are extremely hard to detect since, in an ideal scenario, the overall accuracy of the attacked model does not differ from the clean model. Moreover, in many cases, the perturbed input is visually indistinguishable from the clean input, adding another layer of difficulty in detection.

Formally, the goal of a targeted attack is, given a classifier, $f(x)$, for data $x \in \mathbf{X}$, and its corresponding ground truth label, $y_g \in \mathbf{Y}$, the attacker's goal is to craft inputs x_a such that training the model using x_a results in $f(x_t) = y_t$, for some target input x_t and class y_t where $y_t \neq y_g$.

Untargeted attacks are less restrictive than targeted attacks in their goal classification. The goal of these attacks is to craft adversarial inputs x_a such that for any or for a specific x, the classifier $f(x)$ yields y where $y \neq y_g$.

2.3.1 Untargeted Attacks

There has been a plethora of work on data poisoning attacks that seek not only to cause a specific label or input to be misclassified but also to cause as many misclassifications as possible [15, 16, 17, 18, 19]. [15, 16, 17] focuses on attacking naive Bayes spam filters by manipulating the spam messages in such a way that the classifier begins to misbehave. The attack method proposed in [18] attacks a support vector machine by applying a gradient ascent strategy based on the properties of the model. This requires the attacker to have complete knowledge of the classification system. [19] uses back-gradient optimization to attack any classification model that learns using gradient descent.

Shortly after the discovery of these types of attacks, it was found that this type of attack is relatively easy to detect and mitigate simply by observing the loss associated with adding specific inputs to the model's training set [73]. In [74], the authors used outlier removal to remove poisoned samples. Two different defense strategies were discussed. The first strategy used a fixed defense. In this strategy, there is an oracle that knows the exact distribution of the clean dataset and can generate a feasible dataset from this knowledge. The second strategy uses a data-dependent defense. A fixed feasible dataset is used, which is dependent on the union of the clean and poisoned datasets. The defender's goal is then to minimize the loss function by removing outliers. They tested their solution on a binary SVM model but stated that their techniques would work for multi-class problems using any model method. They reported an 11% drop in test accuracy when the attacker was allowed only to modify 3% of the training dataset.

The authors of [75] proposed a worst-case label flipping attack strategy. This attack strategy assumed full-white box access to the dataset and classification model, including the model's loss function. The goal of their attack is to maximize the loss function. They accomplish this by greedily flipping samples from the training dataset which maximizes the validation loss. When 20% of the data was poisoned, the classification error increased by a factor of between 2.8 and 6. They then proposed a label sanitization-based defense strategy that relied on removing outliers. Their strategy creates clusters using the k-NN machine learning method. Given some threshold η, if the percentage of samples in the majority label of the cluster is greater than η, then the minority's labels are changed to match the majority. This defense strategy greatly diminishes the effects of the attack. They found that setting η to 0.5 with a high value for k leads to the best results. This defense strategy is not applicable to our attacks, however. K-NN does not scale well with large datasets, such as those used by our CNNs, plus our input vector for colored images would be much larger than those studied in [24]. In addition, since our attack strategies are targeted, we do not attempt to maximize either FaceNet's or XceptionNet's loss function.

2.3.2 Targeted Attacks

There are various types of targeted data poisoning attacks, such as backdoor attacks and targeted label flipping attacks. Backdoor attacks [76, 77, 78, 79] train a "trigger" or "backdoor" into a neural network such that only inputs that contain this trigger are misclassified during runtime. Since the model behaves normally for all inputs without a trigger, this type of attack is harder to detect, making it generally more powerful than untargeted attacks. The method described in [76] involves simply "stamping" a simple trigger (e.g., a white box in the corner of image data) onto a subset of the training set and changing the labels of those images in a way that the model associates this trigger with a certain class. [77] and [78] use properties of the machine learning model to construct an optimal trigger. There are several defenses proposed for this type of attack [25, 78, 80]. Methods that are successful include anomaly detection on the input space and classification of an input [25, 78] and altering the structure of the classification network [79, 80].

As opposed to backdoor attacks, label flipping attacks alter the *label* of a sample instead of the sample itself. Empirical results have shown that label flipping attacks can significantly degrade a classifier [81]. To perform the attack, the adversary only requires access to the labels of the training dataset, however, to optimize the attack, it is often assumed that the adversary has access to the learner's loss function. To fully optimize this attack, the adversary would need either the learning model's parameters and read access to the samples in the training dataset or an auxiliary dataset that follows the same distribution as the training dataset [82].

In [83], the authors use both a digital and a physical key. They used a semi-transparent overlay of "Hello Kitty" as a digital key to allow the adversary to log in as a specific user. They also tested the use of a specific pair of reading glasses as a key. Only this pair of glasses would allow the backdoor to successfully activate. In addition, they tested two data poisoning attacks. The first attack was an input-instance-key attack. This attack poisoned the data with only a few (tens) of images to make a single image allow the login as a specific user. Their more general attack was a pattern-key attack. With this attack, the presence of a key within the image allowed for the successful login as a specific user; however, this attack required poisoning over a thousand images in the training dataset.

Neural Cleanse is one proposed defense strategy against backdoor attacks [25]. In [25], they specifically state that backdoor attacks are separate from data poisoning attacks; however, since they involve infecting the training data, we are including backdoor attacks in this section. The authors have three goals: to detect the backdoor, to identify the backdoor, and to mitigate the backdoor. They combine their detect and identity steps into one. In this step, they find the minimal perturbation filter required to transform all samples from all other labels to the targeted label. They repeat this step for all possible classes. If there is an outlier among the generated perturbation filters, the filter is marked as a backdoor. To mitigate backdoors, they applied two different approaches. They empirically noticed that the top 1% of neuron activation are heavily correlated to the backdoor. They set neurons that were highly correlated to the backdoor in the second to last layer of the network to 0. Their second approach used unlearning. They used the reversed trigger (trigger inverted) and re-trained the network using samples containing the reverse trigger. To perform their re-training, they used 10% of the original training data that does not contain a backdoor and added the reversed trigger to 20% of those samples. They were able to reduce the attack success rate to below 6.70% with at most a 3.6% dip in classification accuracy.

In our facial recognition attack scenario, the attackers do not need to create any backdoors. The attackers' photos are real photos, and hence the abnormal detection techniques used to detect backdoors will not function in our case. Our label flipping deepfake attack is also not classified as a backdoor attack. We do not alter the images in any way, we are simply changing the label of the image. There is no perturbation filter or alteration to the image to detect.

In most of the other existing targeted attacks [53, 84], the common strategy is to perturb the input training images by adding various kinds of noises either digitally by modifying pixels, etc. or physically such as wearing specially designed glasses so that the classifier will misclassify the perturbed images. They all require the attackers to have strong knowledge about the classifier's output which may not be practical in reality. For example, in [53], an optimization scheme is proposed based on the classifier output and the amount of perturbation to alter training images. This perturbation achieves misclassification of a single specific target input. More recently, [84] proposed a new attacker model called FAIL which was the first to consider a wide range of attackers who have only partial knowledge of the target feature vectors and classifier. Our work takes an additional step by completely removing the assumption that attackers need to know the feature vector of the target or need to have access to the classifier.

In [85] the authors used individual perturbations on images from a source class to move the decision boundary of the source class towards a targeted class. Samples containing images of the source after the model was trained would then be classified as the target. This would allow the adversary to go unnoticed given a surveillance system or to log in as a specific user in a web service environment. To accomplish this, they assumed they were attacking a white-box model with full access to the facial recognition model or another model capable of facial recognition and full access to the dataset. The adversary then chooses a target class. If the adversary does not have a target class in mind, he/she can compute the centroid of every class and select the class with the centroid furthest away from the source class. An individual cloak (perturbation filter) is then created for most images of the source class. The cloak is meant to move the source class's decision boundary towards the target class. They achieved a 100% success rate with their attack. The major drawback of this work is they assume access to either the facial recognition model they are fighting against or a related feature extractor. Our data poisoning attacks assume no access to the detector model.

The authors of [82] proposed a targeted label flipping attack. Their proposed methodology is largely feasible. They assume to have access to an auxiliary dataset that follows the same distribution as the training dataset used to train the model. They also assume knowledge of the learner's loss function. By using either a feature match or a cluster match algorithm on the auxiliary dataset, they find a vulnerable subpopulation of the dataset to attack. They then wish to cause the greatest loss in this subpopulation while keeping samples outside of this subpopulation unaffected. Their proposed label flipping strategy is unlike other label flipping strategies. Instead of altering the label of existing data points, they assume the ability to add samples to the subpopulation to cause misclassification of the subpopulation as a whole. By purposely selecting which subpopulation to corrupt, the attack becomes a targeted data poisoning attack. This attack strategy is different from both of our proposed attacks. Neither of our attacks adds additional data to the dataset. Our attacks either change the sample of an existing point in the dataset or alter the label of a sample. In addition, neither of our attacks attempts to decrease the overall classification accuracy. Instead, our goal is to allow specific samples to bypass proper recognition.

To date, there does not seem to be any effective defenses to fight against targeted data poisoning attacks. According to [82], some data poisoning attacks are even impossible to defend against. Our proposed defensive strategies utilize deep learning techniques that may pave the way to the development of more generic defensive mechanisms for various targeted attacks.

Chapter 3

DEFEND DATA POISONING ATTACKS TO FACIAL RECOGNITION NEURAL NETWORKS

This chapter describes a novel data poisoning attack followed by two of our proposed defenses to this type of attack. For our framework, we assume that the web service providers adopt the most popular and accurate DNN-based facial recognition system, FaceNet [5]. Both our attack and defense mechanisms can be applied to other DNN-based facial recognition systems; but for brevity, we only tested on the state-of-the-art in facial recognition systems.

For a better understanding of our work, we will introduce the background knowledge of FaceNet first and then present the attack settings, defenses, and finally results.

This chapter is laid out in seven sections. First, Section 3.1 describes the facial recognition system FaceNet. In Section 3.2, we propose our novel data poisoning attack. Section 3.3 describe our proposed discriminators to detect when a person is being attacked from an adversary using our proposed attack. A discussion on our discriminators detection ability is given in Section 3.4. In Section 3.5 we perform a security analysis on our discriminators. Section 3.6 highlights how other detection options fall short in detecting our attack. We conclude this chapter with Section 3.7.

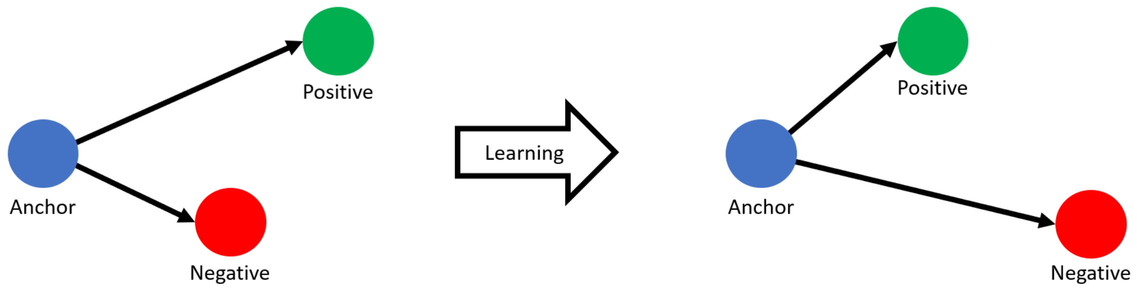


Figure 3.1: FaceNet's Triplet-Loss Diagram

3.1 FACENET

FaceNet [5] was developed by Google in 2015 and remains a state-of-the-art facial recognition system among those with the highest recognition accuracy. The key techniques underlying FaceNet include a novel triplet loss function and an effective deep learning model. Specifically, the triplet loss function minimizes the distance between like labels and maximizes the distance between opposing labels. For each sample, an anchor is chosen. Along with the anchor, a positive image with the same label and a negative image with a different label are selected. The loss function decreases the distance between the anchor and another sample of the same label while increasing the distance between the anchor and a negative sample. Figure 3.1 demonstrates the change in the Euclidean distance during the learning process.

FaceNet employs a deep learning model to directly learn an embedding in Euclidean space for face verification. It takes as input the normalized pixel values of an image. The output of the DNN is a 128-dimensional embedding that maps the image to Euclidean space. This embedding is then fed into an SVM for classification. This architecture is summarised in Figure 3.2

FaceNet supports two different architectures: Inception ResNet [86] and the Zeiler&Fergus [87] architecture. In our experiments, we adopt the former considering its higher efficiency and popularity. The inception architecture has 27 layers, consisting primarily of inception and pooling layers. Each inception layer consists of 1x1, 3x3, and 5x5

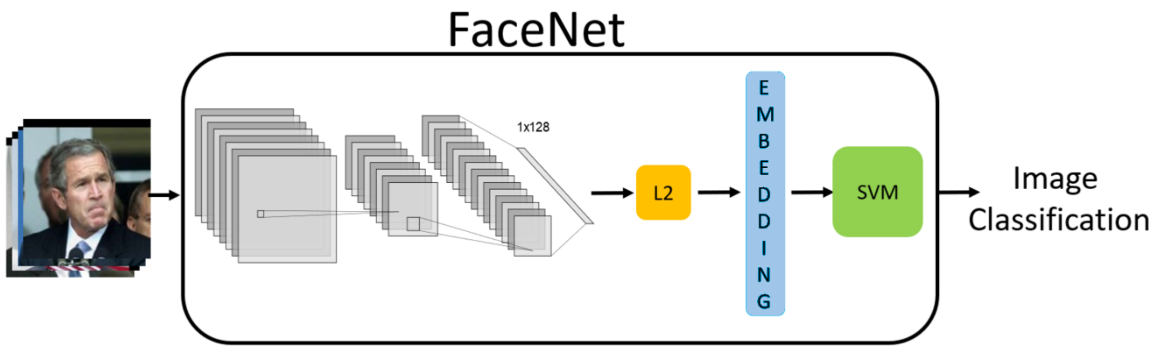


Figure 3.2: FaceNet's Architecture

convolutional layers running both sequentially and in parallel. We use as input a $(160, 160, 3)$ feature vector derived from the RGB values of a given image. The feature vector is generated using the MTCNN algorithm [88]. Specifically, MTCNN draws a boundary box around a face in an image with high confidence, which helps verify the existence of a face as well as conducting face alignment. We use the nearest neighbor downsampling algorithm to reduce the image size to meet our feature vector requirements. Finally, we normalize every feature by dividing it by 255.0, the maximum value a pixel can take on. This results in a (160,160,3) feature vector where every feature is within the bounds $[0, 1]$.

To preserve high recognition accuracy, we leverage a pre-trained network [89] which was trained on the VGGFace2 dataset [90]. It achieved an accuracy of 99.65% on the Labeled Faces in the Wild (LFW) dataset [91] which is a standard dataset to test facial recognition systems and has a very large number of facial identities. Specifically, the VGGFace2 dataset contains over 9,000 identities with over 3.3 million faces and the LFW dataset contains 13,233 images from 5,749 people.

3.2 ATTACK ANALYSIS

The attack proposed in this chapter is a targeted data poisoning attack as described in Section 2.3.2. In this attack, an attacker attempts to impersonate a victim during facial authentication, i.e., the attacker's own face images will allow the attacker to log into the victim's web service account. This is accomplished by replacing n of the target's training images with images of the attacker. This enables the attacker to be authenticated as the target during testing time, similar to a backdoor attack, but with the key being the attacker's face as opposed to a perturbation filter or item.

The only parameter the attacker controls are the number and which images of the target are replaced with the attacker's photos. To reduce the possibility of the attack being noticed, the attacker needs to minimize the number of photos being replaced while not degrading the overall accuracy of the system, especially of the target. Thus, we formed Equation 3.1 to minimize the number of photos the attacker replaces. Due to how our attack replaces the target's images, we have named this attack a *replacement data poisoning attack*.

$$G(\mathbf{X}, \mathbf{Y}) = \min_{n} \left(\frac{\mathcal{L}(\mathbf{X}_a, y_t)}{n} + \frac{\mathcal{L}(\mathbf{X}_t, y_t)}{(s-n)} + \frac{\mathcal{L}(\mathbf{X}_p, \mathbf{Y}_\mathbf{g})}{s * c_p} \right) \tag{3.1}$$

$\mathbf{X}$ is the set of all images used for training. $\mathbf{Y}$ is the corresponding labels for $\mathbf{X}$. $\mathcal{L}(x, y)$ is the loss function used by the facial recognition system. In our case, it is the predictive probability outputted by FaceNet's Support Vector Machine (SVM) classifier, as described in [5]. $\mathbf{X_a}$ and $\mathbf{X_t}$ are the attacker's and target's image sets, respectively. $\mathbf{X_p}$ is the set of pristine images or images from classes that are not attacking nor being attacked. The target class is y_t with the set of pristine classes that correspond to $\mathbf{X_p}$ being $\mathbf{Y_g}$. The number of pristine labels is represented by c_p. Finally, the number of images per class is s. During the sign-up phase for a web service, the service can control the number of images per label used for training, thus we assume that each label uses the same number of photos for training.

Equation 3.2 shows the sum of the loss function without the data poisoning attack. Since the attacker's secondary goal is to not degrade the facial authentication system, the attacker wants α in Equation 3.3 to be minimized.

$$H(\mathbf{X}, \mathbf{Y}) = \left(\frac{\beta \mathcal{L}(\mathbf{X}_t, y_t)}{s} + \frac{\gamma \mathcal{L}(\mathbf{X}_p, \mathbf{Y}_\mathbf{g})}{s * c_p} \right) \tag{3.2}$$

$$G(\mathbf{X}, \mathbf{Y}) = H(\mathbf{X}, \mathbf{Y}) + \alpha \tag{3.3}$$

We conducted our new replacement data poisoning attack on two different datasets: FEI [92] and the Labeled Faces in the Wild (LFW) [91]. The datasets are summarized in Table 3.1. The two datasets are mainly used to simulate adding new users to an existing face authentication system since the FaceNet model used in our experiments is already a pre-trained, accurate model. We chose these two datasets to cover an ideal setting and a complex setting. Specifically, the FEI dataset has a relatively consistent background in terms of color and lighting, which represents one of the easiest facial recognition scenarios. Specifically, the photos in this dataset are taken with the person facing different directions with a variety of facial expressions. Most photos are taken using a white background. Photos in a dark setting have been filtered out. In contrast, the LFW dataset a diverse dataset which contains photos of celebrities with different backgrounds from a variety of angles. The LFW dataset has been commonly used as a benchmark to evaluate many facial recognition classifiers. It allows us to simulate various background environments where a user may log onto his/her web service account. The LFW dataset originally contains photos of 5,749 people and each person has a different number of photos. For the experiments, we select users who have at least 10 photos from the LFW dataset so that we have sufficient training and testing images per user, to better reflect facial recognition for web services, where the number of images can be controlled.

To prepare the datasets for the attack, we split each dataset $\mathcal{D}$ into three equally sized groups $\{Target, Attack, Pristine\}$, where $Target$ will be used to simulate images of new users who are under attack, $Attack$ will be used to simulate images of

Dataset	# of Users	# of Photos/User	Training/user
FEI	200	14	10
LFW	158	10-530	10

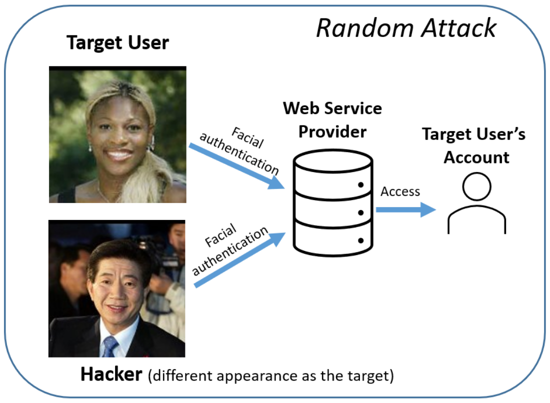
Table 3.1: Facial Recognition Datasets

attackers, and *Pristine* are for other newly added users. To launch the attack, we first randomly select a user from the *Target* image group and an attacker from the *Attack* group. Then, we replace a certain number of images of the target victim with the images of the attacker as a man-in-the-middle (MITM) attack to the user's home router. Note that according to the latest study, a large number of routers and devices are still vulnerable to MITM attacks [27, 28], even though various research has been carried out to counter MITM. Each label in the *Attack* group attacks only one user in the *Target* group. These images along with images in the *Pristine* group are fed into FaceNet for training. Based on how the attacker's images are selected, we define the following two kinds of attacks:

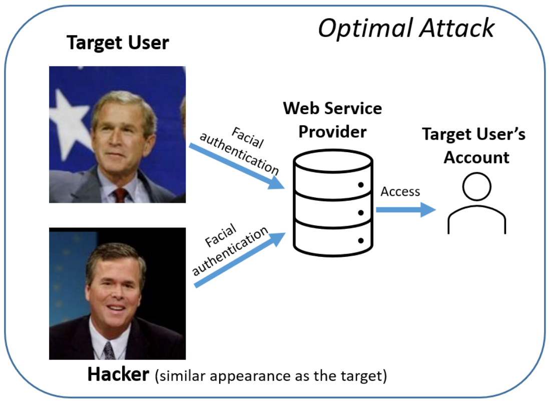
- **Random Attack**: We randomly select a set of photos with the same label (user) from the *Attack* dataset to simulate the attacker's photos. Due to the random selection, the selected attacker may have a very different appearance than the targeted victim in terms of gender, race, and age. Figure 3.3a illustrates an example.

- **Optimal Attack**: We purposely select a set of photos of a user in the *Attack* set who looks very similar to the targeted victim. For example, we chose photos belonging to the brother of the target as the attacker's photos. Figure 3.3b shows an example of such an attack. The pairs are selected by minimizing the Euclidean distance that is outputted by FaceNet's DNN.

The optimal attack scenario represents when the adversary has access to our FaceNet model, a copy of the target's facial images with which they will use to



(a) Random Attack Strategy



(b) Optimal Attack Strategy

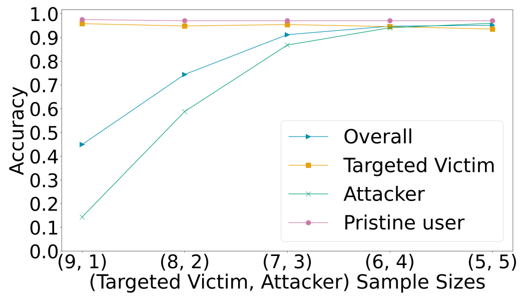
Figure 3.3: Attack Strategy Overview

train FaceNet, and an option to inject whomever they want from the *Attack* dataset. They then choose the label from the *Attack* dataset that results in the smallest L^1 norm with their target with respect to FaceNet's embedding feature vector. With this attack, the adversary minimizes the distance between the $Target$ and *Attack* datasets in 128-dimensional space. In our experiments, we went a step further and allowed the adversary to select which labels would compose the $Target$, *Attack*,, and *Pristine* datasets. Therefore, the adversary has complete control over whom is attacking whom.

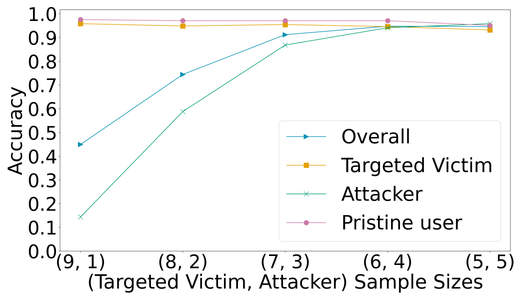
After the attack, we evaluate the following. First, we examine the overall facial recognition accuracy of the targeted victim and pristine users to see if they can still be recognized with high accuracy, similar to an unattacked scenario. Second, we check if the attacker's images are successfully classified as the targeted victim. If both criteria are met, the attack is considered successful.

3.2.1 Attack Results

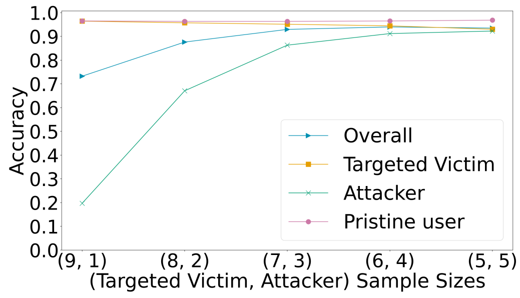
Figure 3.4a and Figure 3.4b report the random and optimal attack results for the FEI dataset, respectively. Figure 3.4c and Figure 3.4d report the random and optimal attack results for the LFW dataset, respectively.



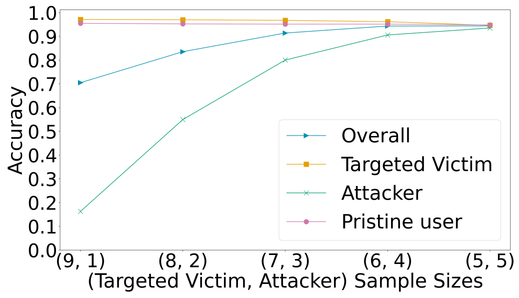
(a) Random Attack on the FEI dataset



(b) Optimal Attack on the FEI dataset



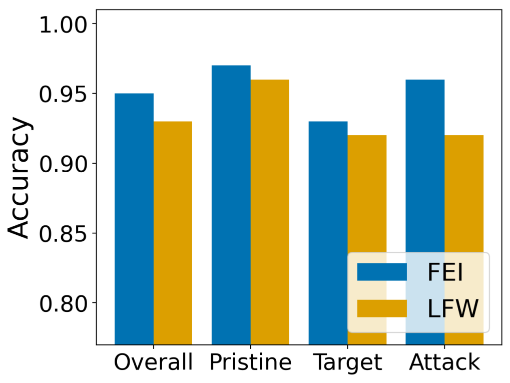
(c) Random Attack on the LFW dataset



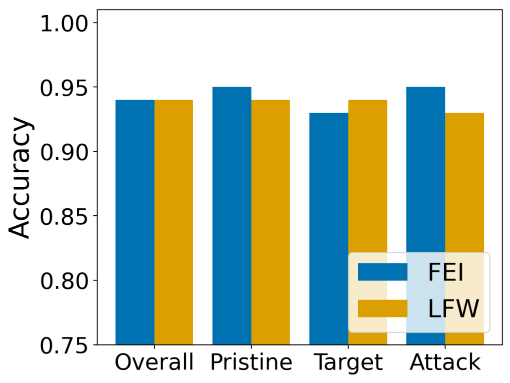
(d) Optimal Attack on the LFW dataset

Figure 3.4: Replacement Data Poisoning Attack Results on FaceNet

In the experiments, 10 photos are used for each user registration. In the figures, the x-axis shows the number of photos belonging to the target and the number of photos injected by the attacker, in $(target, attacker)$ format. For example, '(8,2)' means there are 8 original user photos and 2 attacker photos for a single user registration. The attacker's photos are randomly inserted into the sequence of user photos. The order of the attacker's photos in the 10 photos does not affect the attack success rate. The y-axis shows the success rate that a photo can be recognized as the desired user, i.e., the target user recognized as the target user, the attacker as the target user, and the unattacked (pristine) user as pristine. As we can see from the experimental results, the success rate of the attack increases with the number of injected photos. In all of the datasets, both the random attack and the optimal achieve greater than 90% recognition rate when the number of target user's photos and the number of injected photos are half-and-half. Surprisingly, the attacker's face is sometimes easier to recognize than the target user's. This is probably because the FaceNet DNN treats



(a) Random Attack Strategy



(b) Optimal Attack Strategy

Figure 3.5: Bar graphs comparing the accuracy of the different datasets when the target and attacking samples are of the same size

both the attacker's facial photos and the target user's facial photos with the same importance and extracts their common features in order to maintain high recognition accuracy. As a result, the attacker would gain the same access as the target user. Another important observation is that there are no significant advantages of the optimal attack over the random attack. This simplifies the attacker's strategy as they do not need to find a person who looks similar to the victim. Finally, we would like to point out that a successful attack is hard to be singled out by simply comparing the recognition rate among all the users since the accuracy of the target/attacker and pristine are similar. This is highlighted in Figure 3.5a and especially in Figure 3.5b.

3.2.2 Feasibility Analysis of Attacks in Real-Life Face Authentication Applications

We now proceed to present an overall flow of this data poisoning attack in real-life face authentication applications. The attacker will first need to compromise the victim's home router. This is not challenging as many popular home routers still lack security protection as reported in [93]. Even if the communication channel is encrypted using SSL certificates, man-in-the-middle attacks may still succeed by tricking the victim to accept the attacker's SSL certificate instead of the original web service provider's certificate. The attacker will then be able to eavesdrop on the network traffic of the victim. When the attacker observes that the victim is registering a new web service that uses face authentication, the attacker will inject a couple of his own photos into the packages sent to the web service provider which will give the attacker access to the same user account later on.

We also examined the effect of our data poisoning attack against a real-life face authentication app, called BioID [94]. We chose BioID since it is listed as one of the best face authentication apps by Google search and it is also free for testing purposes. In this experiment, we open a single user account to start the face registration. A female is assumed to be the authentic user, and a male pretends to be the attacker. During the registration phase, the female first appeared in front of the camera to enroll her face, and then the male enrolled his face to the same account which simulates the network injection. After the registration, we found that both the female and the male were able to authenticate to the same account which is not supposed to happen in a secure face authentication process. This result validates the feasibility of our proposed data poisoning attack and demonstrates the need to develop a more secure way of adopting face authentication.

3.3 DEFEAT

In this section, we first provide a system overview and then elaborate on the DEFEAT system.

3.3.1 System Framework and Deployment

Figure 3.6 presents the overall data flow in our proposed DEFEAT system. There are three parties in this process: the user/attacker, the facial authentication system, and the discriminator. The threat detection occurs during the user registration phase. Since man-in-the-middle attacks still thrive in home routers according to 2020 studies

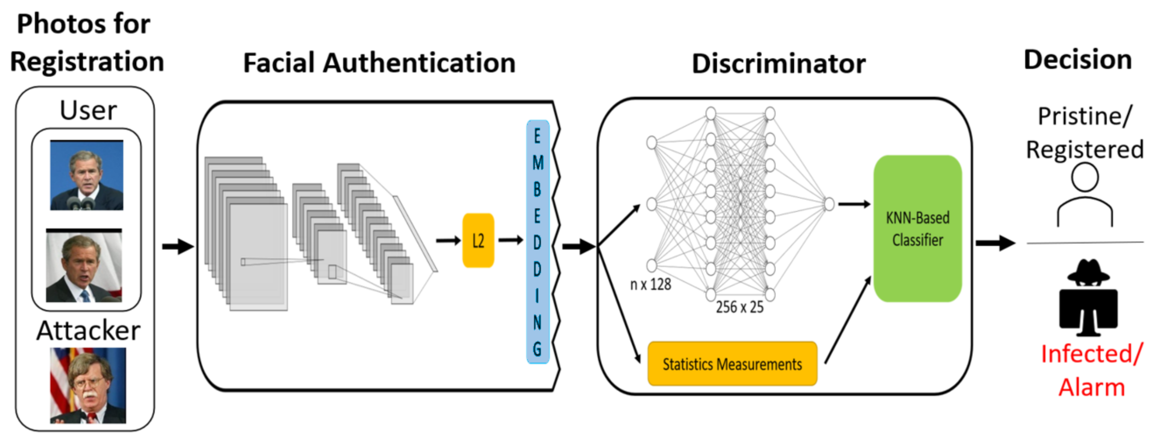


Figure 3.6: DEFEAT System Framework

[27], attackers who exploit the vulnerabilities of the user's router have the ability to compromise the user registration process. The goal of our system is to detect such attacks on the server-side. Specifically, a number of training photos provided by the user (or attacker) will first be sent to the facial authentication system. The facial authentication system will not immediately register the user at this point. Instead, the embeddings generated by FaceNet will be fed to the discriminator for evaluation. If the photos are pristine, the user registration process will proceed to register the user. If the discriminator concludes that the training samples may be infected, the discriminator will alert the service provider to conduct further investigation. For example, the investigation can be easily carried out by a human expert who looks into the suspicious training samples to see if they belong to the same person. These photos which may fool machine learning algorithms are still hard to escape from human eyes. However, we would stress that although human experts may be good at distinguishing infected photos, it would not be practical if we ask human experts to screen the entirety of the large number of photos streaming into the web service providers every day. Our proposed discriminators will significantly minimize the efforts required by human experts.

In a real-world web service scenario, there are two possible options to deploy the above framework, which are (i) on-site detection; (ii) off-site detection. For on-site detection, the web service provider installs our proposed discriminator along with their original facial authentication system to carry out threat detection by itself. Alternatively, there could be a third-party security provider that is in charge of evaluating security threats using the discriminator. Since the discriminator only needs embeddings and statistic measurements as input, none of the users' private facial images will be disclosed to such a third-party security provider, which makes this off-site evaluation possible.

Using an off-site evaluation method gives multiple benefits. First, it relieves the web service provider's burden of maintaining another security system. Also, third-party security provides the ability to continuously improve the discriminator and train it to be robust and generic based on information collected from various service providers. This offsite strategy is demonstrated in Figure 3.7.

In what follows, we present a statistics-based discriminator and a DNN-based discriminator.

3.3.2 Statistics-Based Discriminator

In the data poisoning attack, as discussed in Section 3.2, the attacker's face images are mixed with the victim's face images when FaceNet generates the 128-dimensional feature vector for the victim. Thus, it is expected that the infected (or contaminated) feature vector of the victim would be different from the uncontaminated feature vectors of other users. Intuitively, one may think that such differences may be reflected by commonly used statistical measurements, such as internal differences among feature vectors of the same label (same user), and external distances among the different groups of feature vectors (which will be formally defined later in this section). Therefore, we investigate a statistics-based discriminator as follows.

The goal of the statistics-based discriminator is to leverage statistical analysis on FaceNet's output embeddings (i.e., face feature vectors) to determine if a pristine

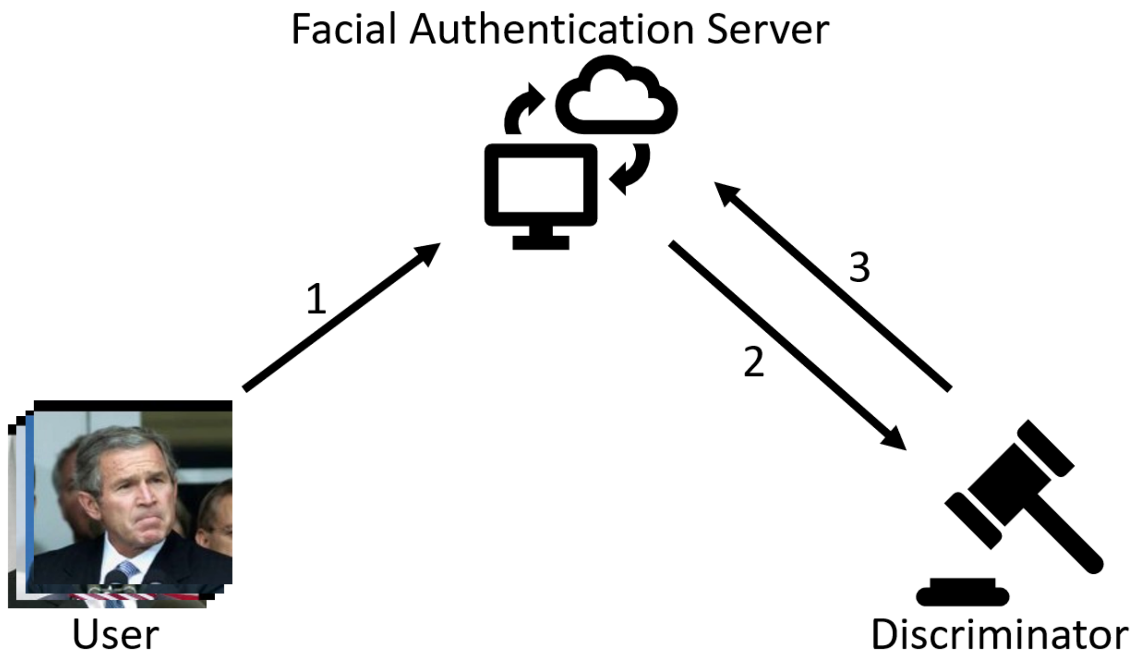


Figure 3.7: This figure shows a possible three party use case for our discriminator. The three parties consist of the user, which submits photos; the facial authentication server, which registers new users; and a discriminator, which is a third party in charge of determining if the replacement data poisoning attack proposed in this chapter occurred. When a new user is registered, (1) the user sends his/her facial images to the facial authentication server. The facial authentication server then generates the embeddings for the supplied image using FaceNet. (2) The facial authentication server sends the user's embeddings to the discriminator anonymously, so the discriminator does not know the identity of the user being registered. The discriminator then uses our proposed DEFEAT algorithm to determine if an attack occurred. Finally, (3) the discriminator sends the decision to the facial authentication server.

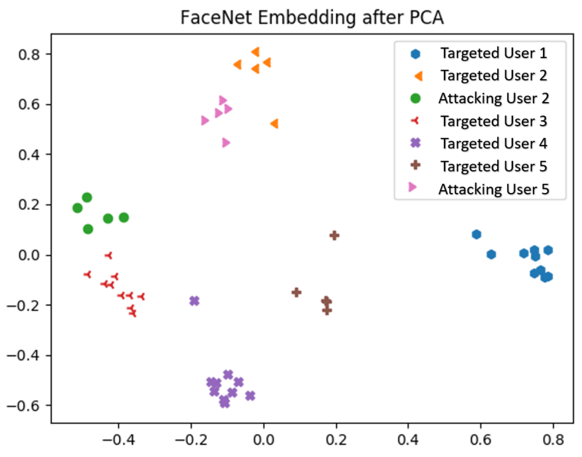


Figure 3.8: Principal Component Analysis (PCA) applied to a subset of labels that FaceNet is trained on. Five labels were randomly selected from the FEI dataset, 3 un-attacked labels and 2 attacked labels using the random attack configuration. PCA was then applied to FaceNet's neural network output embedding to reduce the dimensionality from a 128-dimensional space down to a 2-dimensional space. Each label is shown, with the injected samples differentiated by a different color and symbol.

(uncontaminated) label is differentiable from an infected label. We started this process by employing principal component analysis (PCA) to reduce the dimensionality of the embeddings while retaining some of the underlying relationships among feature vectors. Specifically, we reduce the dimensionality from 128 down to 2 to visualize the differences between labels. As shown in Figure 3.8, the attacker's samples tend to form clusters separating from the targeted victim's samples. We then attempt to find a non-visual way to differentiate the face features. Based on the results from PCA, we hypothesize that there may exist a few key statistical measures that could differentiate the pristine labels from the infected labels when the full dimensionality is considered.

The first statistical measure explored is the maximum internal difference between every embedding for a label. The PCA plot highlights an apparent difference between the maximum internal distance when reduced to 2-dimensional space. We wanted to know if this is only true when embeddings are reduced to 2-dimensional space, or if such difference also exists in 128-dimensions. The maximum internal differences are formally defined as follows:

Definition 3.3.1 (Maximum Internal Difference)**.** Let $\mathcal{E}$ be the whole set of embeddings (face feature vectors), and let e_i^ℓ, e_j^ℓ denote the different embeddings with respect to the same label ℓ. The maximum internal difference for a label ℓ is calculated using the $\mathcal{L}_1$-norm which maps the distance between the two embedding vectors to a scalar value without magnifying larger differences between the two embeddings.

$$f_{max}^\ell = \max_{i \neq j} \mathcal{L}_1(e_j^\ell - e_i^\ell)$$

Our second statistical measure is the minimum external difference between labels. Since the cluster of the embeddings of the same label tend to be wider or separated when the label was attacked, we hypothesize that the minimum external difference would be smaller when the label is under attack versus when it is not under attack. Formally, the minimum external difference between sets of labels is defined as follows.

Definition 3.3.2 (Minimum External Difference)**.** Let L denote the entire set of labels, k be the label to be considered, and ℓ be the remaining labels in L. The minimum external difference between the embeddings of label k and that of all the other labels ℓ is calculated as follows, which finds the smallest distance between any embedding of label k and the nearest embedding of a different label:

$$f_{min}^\ell = \min_{k \neq \ell} \mathcal{L}_1(e_j^\ell - e_i^k) \forall j \in \ell, i \in k$$

Based on individual external differences, we then compute the mean internal difference as follows:

Definition 3.3.3 (Mean External Difference)**.**

$$f_{mean}^\ell = \frac{1}{n * m} \sum_{i=1}^{n} \sum_{j=1}^{m} \mathcal{L}_1(e_j^\ell - e_i^\ell)$$

The first statistical measure focuses on the possible changes caused by an attack within individual groups of embeddings, while the second statistic presents a wider view of the potential influence of the attack on the relationship among different groups of embeddings. Figure 3.9 shows the relationship between infected labels and pristine labels using these three metrics. As can be seen, the max and mean internal difference between the pristine and the targeted-attacking partitions is quite large. Given the optimal attack strategy, the quartiles are generally closer together for the internal differences when compared to the random attack.

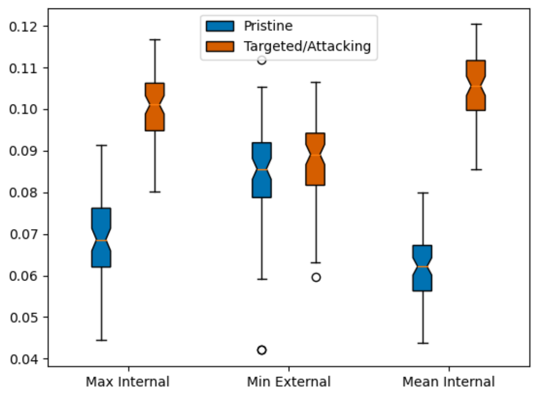
From this analysis, we devise a K-Nearest-Neighbor (KNN) based discriminator that takes as input triplet t where $t^l = \{f^\ell_{\max}, f^\ell_{\min}, f^\ell_{mean}\}$, and outputs if the given t^ℓ was a pristine or a targeted label. KNN was chosen based on the observation from the previous two figures that groups of the pristine labels may have similar statistical values, whereas groups of targeted labels may have another kind of statistical value.

However, such a statistic-based discriminator does not yield a high detection rate in some cases as shown in Section 3.4.2. We found that the possible cause that lowers its detection rate could be the high dimensionality which waters down the differences among different groups of embeddings as visualized in 2-dimensional space. Specifically, the minimum external difference for infected labels and pristine labels are sometimes similar in high dimensional space. Also, the maximum internal difference and minimum external difference are likely heavily correlated. These findings lead us to develop a more advanced intelligent discriminator as introduced in the following subsection.

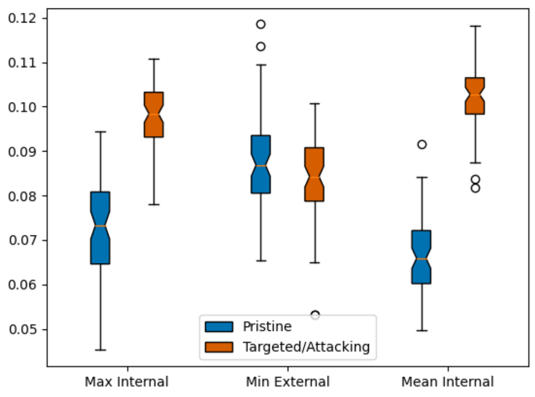
3.3.3 Feature-based DNN Discriminator

Due to the limitations of the statistics-based measurements in distinguishing infected labels from pristine labels, we decided to use a Deep Neural Network (DNN).

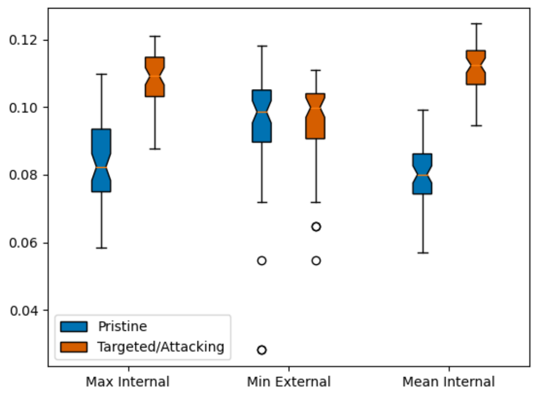
We propose a novel discriminator called DEFEAT (Deep-neural-network and Em-



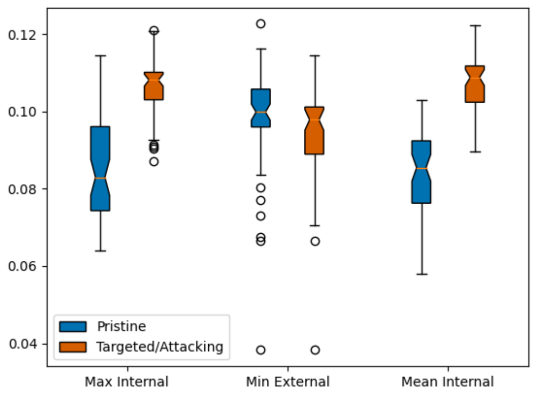
(a) FEI Random Attack Strategy



(b) FEI Optimal Attack Strategy



(c) LFW Random Attack Strategy



(d) LFW Optimal Attack Strategy

Figure 3.9: Box plots of the maximal internal, minimum external, and mean internal differences between the embeddings given a label. The training data from each dataset was used to generate the measurements. Each feature was normalized in order to properly fit in a single figure, thus the lower on the y-axis a point is plotted, the smaller the difference.

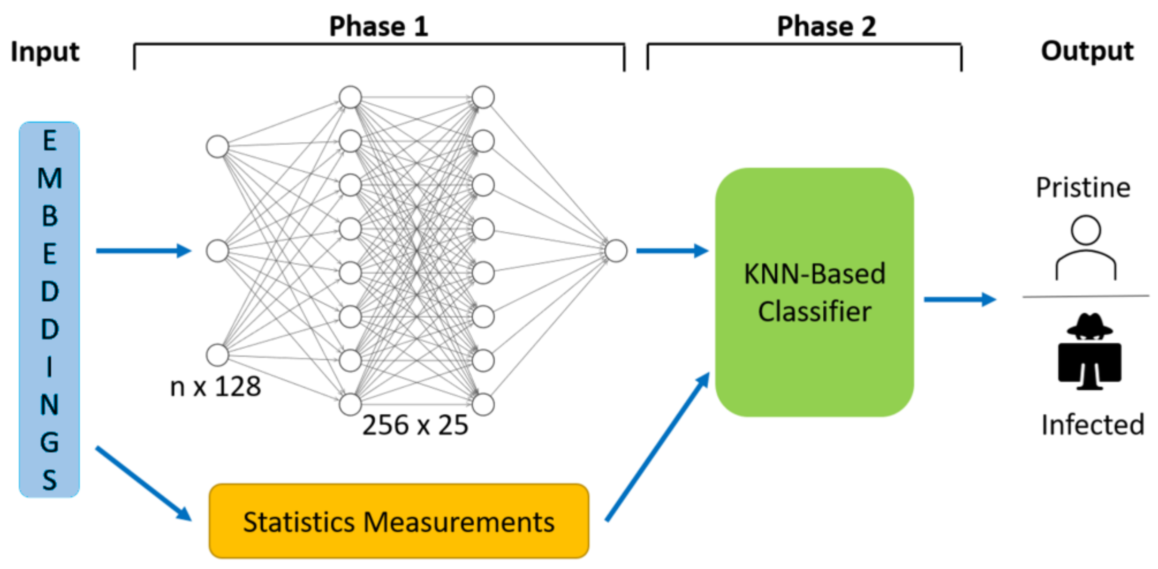


Figure 3.10: DEFEAT Architecture

bedded FEAture-based deTector) that takes as input the feature vectors produced by FaceNet and outputs the probability that the input embedding is infected. Figure 3.10 presents an overview of the structure of the DEFEAT system. DEFEAT consists of two phases. The first phase is a DNN that analyzes the differences between infected feature vectors (embeddings) and pristine feature vectors. The infected feature vectors refer to both the attacker's feature vectors and their targeted user's feature vectors. The result is a probability value that indicates the likelihood of a feature vector being contaminated. Then, this probability value along with several statistic measurements are fed to a KNN-based classifier to yield the final binary output: (i) the label is infected; (ii) the label is pristine. An SVM and decision tree second phase classifier is also explored.

More specifically, the DNN in our DEFEAT system consists of 25 layers with 256 neurons per layer. Through empirical analysis, we found this network architecture to give the best trade-off between speed and accuracy. Each individual layer is a dense layer with batch normalization and a 20% dropout rate. Batch normalization was used to increase the stability of the neural network, while the dropout layers keep the network from overfitting during training. We used the Rectified Linear Unit [95], also known as ReLU, activation function for all layers but the last layer. For the last layer, the sigmoid activation function is shown in Equation 3.4 was used to calculate the probability of an infected label.

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{3.4}$$

Our other interesting finding is that instead of feeding the DNN a single embedding at a time for analysis, an input that concatenates two or more embeddings will significantly enhance the ability to distinguish the infected labels. Specifically, we allow for any permutation of legitimate (pristine and target) and illegitimate (attacker) embeddings. As long as one embedding that belongs to the attacker exists as input, we mark the label as infected. With this configuration, it does not matter if the attacker's embeddings are the majority input or not, as long there are embeddings from different people under the same label, the DNN is expected to output a 1 for infected. Specifically, the DNN represents a function $P(\hat{e}^l)$, where $\hat{e}^l$ is a set of embeddings in $\mathcal{E}$ that cover every permutation of embeddings for a given set size (for example, 1, 2, or 3 in this work) for a label l. The output is the probability that at least one of the inputs actually belongs to a different person.

Next, the second phase consists of a KNN-based classifier which will produce a binary decision to explicitly inform the web service provider whether the registration process of a new user may have been attacked or not. The KNN-based classifier not only considers the probability generated by the DNN but also takes into account a set of statistical measurements to increase the knowledge for decision making. This set of statistical measurements includes all the measurements defined in the previous section, $t^l = \{f^\ell_{\max}, f^\ell_{\min}, f^\ell_{mean}\}$, which have been shown to be beneficial in simple scenarios, along with Shannon's entropy [96] of all the FaceNet embeddings for a single label. This entropy is a 128-dimensional vector, the same size as FaceNet's embedding feature vector. We add the entropies of the embeddings with the same label into a single value.

The following explains how the probabilities from the first phase were used in the second phase. For the classifier to decide if a single label is malicious or not, all of the outputs from a single label are vectored. To make the discriminator independent of how many samples per label it was trained on, we computed several statistics from this vector and supplied these statistics to DEFEAT's second phase. The maximum, minimum, median, and mean probabilities were used along with the standard deviation and Shannon's entropy. In addition, the percentage of elements from the vector that were given a probability of infection greater than 50% was used as a feature.

Since KNN gives each input equal importance, we scale the inputs. The probability generated by our neural network is given the largest weight, accounting for roughly half of the predictive power. The statistical measurements described in Section 3.3.2 in total account for roughly 45% of the prediction. Finally, Shannon's entropy has the lowest weight (around 5%). The reason for such weight assigning reflects the varied importance of the differences among the embeddings' underlying features, their statistical relationships, and the amount of information carried in each embedding. As shown in our experimental studies, the DEFEAT discriminator achieves over 90% detection accuracy in almost all cases.

In addition to the KNN classifier, we also tested an SVM classifier and a decision tree classifier. A major drawback to a KNN classifier is that as the dataset size grows, an increasing number of distances has to be computed, thus with more data, classification takes longer. SVMs do not have this drawback. As we show in Section 3.4, both classifier types have their respective pros and cons.

3.4 PERFORMANCE STUDY

In this section, we present the experiments that compare the effectiveness of our proposed statistic-based discriminator and DEFEAT discriminator in terms of ideal and general settings.

3.4.1 Experimental Settings

All the experiments were conducted in the Chameleon Cloud [97]. A single Chameleon Cloud node was used with 16 virtual CPUs @2.3GHz and 32GBs of memory. We adopted the two datasets: FEI [92] and LFW [91] as presented in Table 3.1. As aforementioned, the FEI dataset represents an ideal and consistent background setting when a facial photo was taken, while the LFW dataset represents a general and diverse background setting. Each dataset is equally split into three sets representing three kinds of users, targeted victims, attackers, and pristine users. We kept approximately 13 photos per user. We used 10 photos for each targeted victim and pristine user with FaceNet. Specifically, for the targeted victims, we replaced some of his/her photos with the attackers' photos during the training. Then, we used the remaining photos for testing purposes. Both the targeted victim's and attackers' photos will be labeled as injected by the discriminators. Since we are using a well-trained pre-trained network with FaceNet, we do not train FaceNet using the training dataset. This is realistic towards an industrial setting. Since we want to test the power of our attack, it would be unrealistic to train FaceNet's neural network when new classes are added and unnecessary since it can cluster similar faces together. However, FaceNet's classifier needs to be re-trained to support the new class, thus it is the SVM that is used for classification that we are training in this step.

After FaceNet was trained, we used the same photos that were used with FaceNet to both train and validate the discriminators. The data was split such that 70% was used for training and the remaining 30% was used for validation.

Our statistics-based discriminator was trained and tested using k-fold cross-validation, where $k = 5$. DEFEAT was trained once per experiment due to the time-intensive process of training the deep neural network. Note that even though training a deep neural network is time-intensive, validating the network is comparable to our statistics-based discriminator.

The effectiveness of the discriminator is evaluated using the following four metrics: (i) precision; (ii) recall; (iii) F1 score and (iv) overall accuracy. In the following equations, TP stands for "true positive", FP stands for "false positive", FN stands for "false negative", and TN stands for "true negative".

Definition 3.4.1 (Precision)**.** Precision measures the percentage of the correctly identified infected photos and uninfected photos among all the photos being tested.

$$Precision = \frac{\text{Correctly Identified Infected Photos}}{\text{All Photos Labeled Infected}} = \frac{TP}{TP + FP}$$

Definition 3.4.2 (Recall)**.** Recall measures the percentage of the correctly identified infected photos against the total number of infected photos that have been tested.

$$Reall = \frac{\text{Correctly Identified Infected Photos}}{\text{All Infected Photos Tested}} = \frac{TP}{TP + FN}$$

Definition 3.4.3 (F1)**.** F1 score is the combination of precision and recall, which serves as an overall performance indicator.

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

Definition 3.4.4 (Accuracy)**.** The overall accuracy evaluates the detection correctness for both the infected photos and pristine photos.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

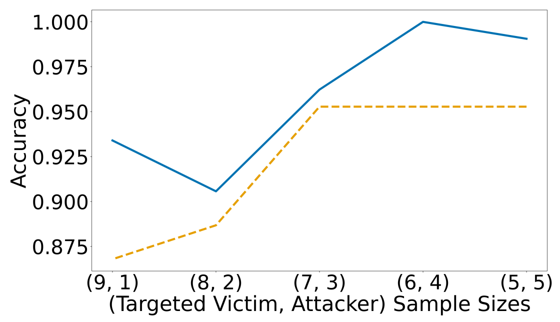
3.4.2 Experimental Results

In the experiments, we evaluate the impact of several factors on the effectiveness of the statistics-based discriminator and the DNN-based discriminator (DEFEAT). These factors include varying the ratio of the injected attackers' photos and the number of training photos per user. We launched both random and optimal attacks. By default, we use the LFW dataset, 50% of injection rate, and 10 photos per user. In the KNN-based classifier, k is set to 5.

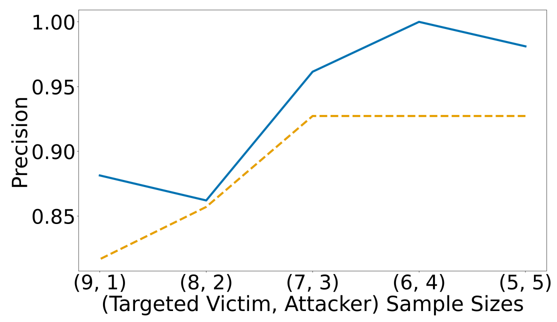
3.4.2.1 Effect of the Number of Injected Photos

In the first round of experiments, we vary the number of injected photos from 1 to 5 among 10 training photos per user in the LFW dataset. It is worth noting that injecting more photos (beyond five) will start decreasing the overall facial recognition accuracy as shown in Section 3, which will alert the service provider. We measure the accuracy, precision, recall, and F1 of our discriminators using the remaining photos not used for training. Both the targeted victim's and attacker's photos will be labeled as infected. Figure 3.11 shows the detection results after the random attack, and Figure 3.12 shows the results after the optimal attack.

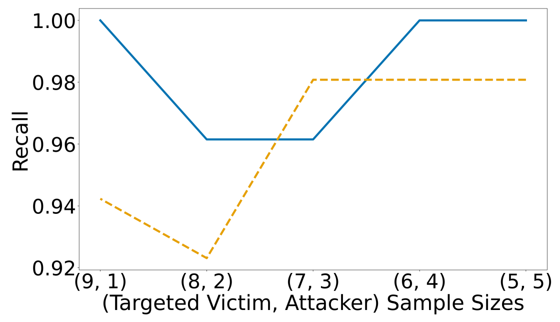
Under both attack strategies, DEFEAT maintains above 90% accuracy in all the scenarios and DEFEAT generally outperforms the statistic-based discriminator in all measurements. Most importantly, when the number of injected photos is close to half of the training photos, DEFEAT achieves more than 99% detection accuracy under the optimal strategy. As shown in Section 3, attackers need to replace nearly 50% of photos of the victim to not decrease the face recognition system's accuracy. That means when the attacker tries to avoid affecting the overall facial recognition accuracy by injecting more photos, it also makes our DEFEAT system to be highly accurate in detecting the attack. The performance of the DEFEAT system should



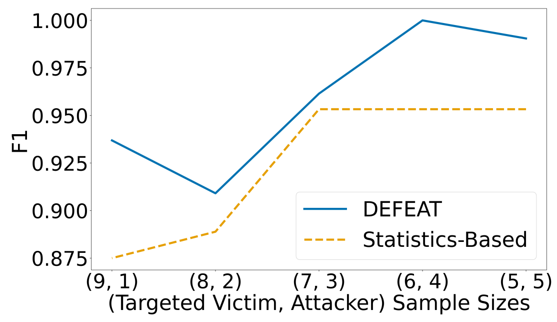
(a) Accuracy



(b) Precision

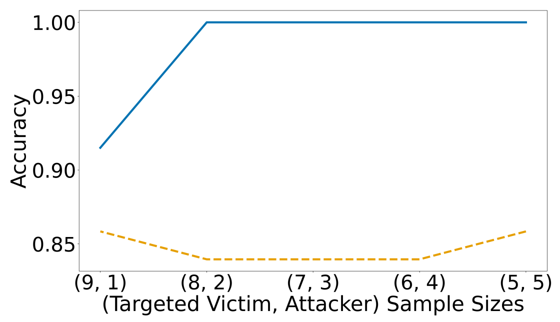


(c) Recall

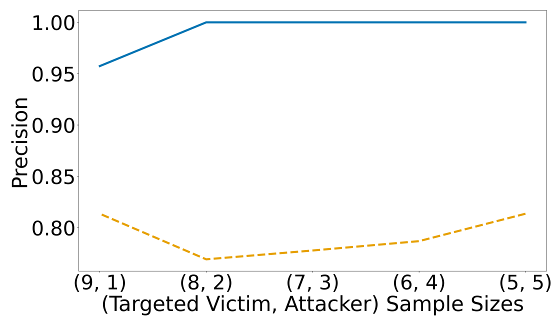


(d) F1

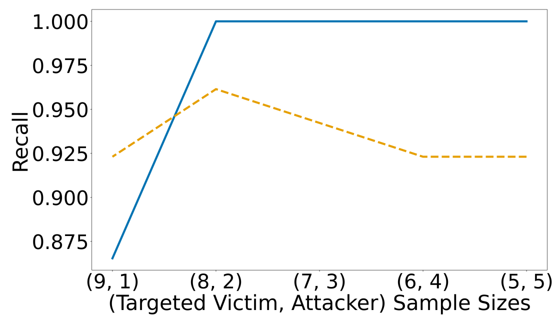
Figure 3.11: Random Attack on the LFW Dataset - Varying the Number of Injected Photos - Two Inputs



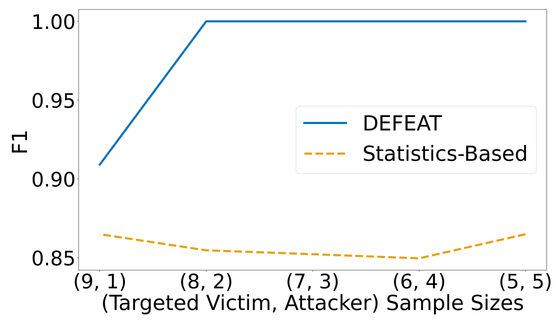
(a) Accuracy



(b) Precision

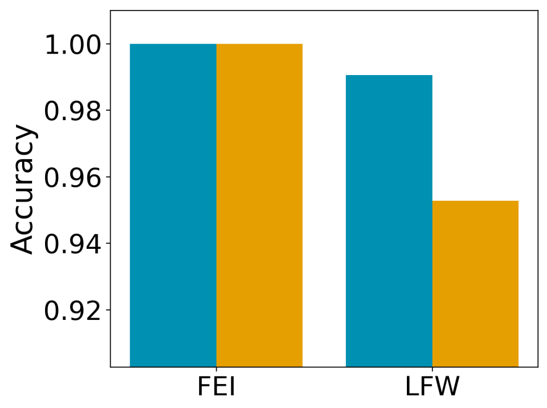


(c) Recall

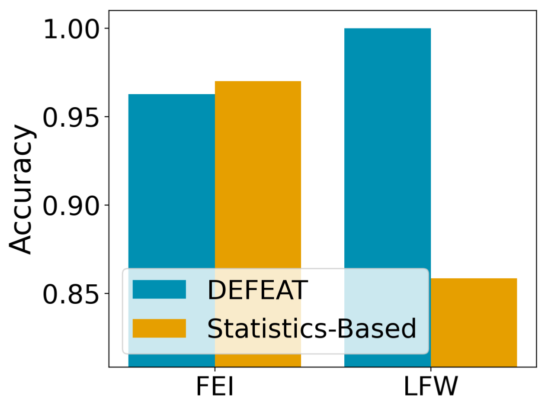


(d) F1

Figure 3.12: Optimal Attack on the LFW Dataset - Varying the Number of Injected Photos - Two Inputs



(a) Random Attack



(b) Optimal Attack

Figure 3.13: Bar graph comparing the accuracy of the different dataset partitions for both datasets, which highlights the importance of image setting (background, lighting, and angle). For infected classes, half of the input images consisted of the target class and the other half was the attack class. This is shown as (5,5) in other figures.

be attributed to the DNN which intelligently classified the different features among injected photos and pristine photos. The statistic measurements do help but are less effective especially under the optimal attack when the attacker's photos look similar to the victim's photos.

3.4.2.2 Effect of Different Photo Backgrounds

Our second round of experiments evaluates the impact of the photo backgrounds on the effectiveness of our discriminators. Specifically, we tested both the FEI and LFW datasets. As previously mentioned, the FEI dataset contains photos with relatively consistent backgrounds and represent an easy setting of facial recognition. The LFW dataset contains photos in various backgrounds at various angles in various lighting conditions, which represents a difficult setting for facial recognition, but is closer to a real scenario where a user may log onto the web service from different devices and in different places, for example, into an email application on a cell-phone.

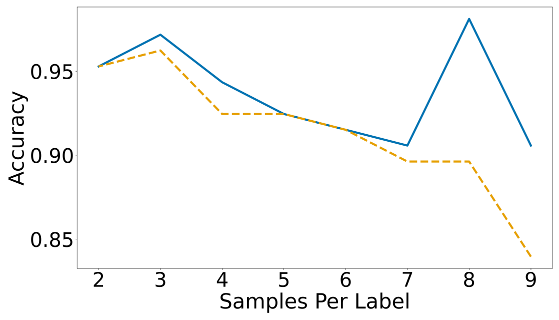
Figure 3.13 shows the overall accuracy of the statistic-based discriminator and

DEFEAT discriminator in the two datasets under the random and optimal attacks with an equal number of *targeted* and *attacking* samples per label. When a random attack is conducted, both discriminators can correctly detect the attack 100% of the time on the FEI dataset. This is likely because the internal differences inside a label's cluster (i.e., the photos with the same label) are enough to be a differentiating factor. As both the statistics-based discriminator and DEFEAT utilize this factor, they both achieve high accuracy.

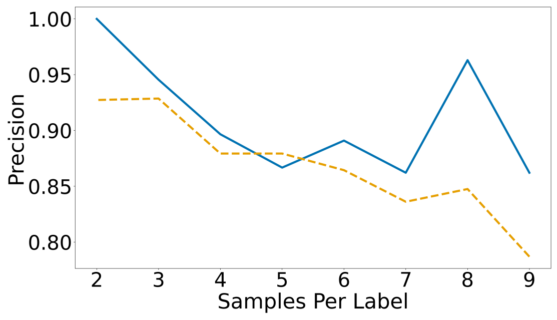
Inside of DEFEAT's phase 2 (classifier), internal and external distances are weighted more heavily than entropy and nearly as much as the output from phase 1, thus DEFEAT is able to have the same accuracy levels as our other model. Given the image's simplistic nature in the FEI dataset compared to the LFW dataset, FaceNet gives more of a clear boundary between classes that our discriminators are able to detect.

When it comes to complex photo backgrounds like those in the LFW dataset, the statistic-based discriminator falls short while the DEFEAT discriminator still maintains high accuracy. This is likely due to the complex backgrounds leading to feature vectors with more complex meanings which are hard to fully capture by simple statistics like internal and external distances. DEFEAT takes advantage of both statistical measurements and the classification ability of a DNN on complex feature vectors, and hence DEFEAT is capable of distinguishing infected photos even under a variety of background settings.

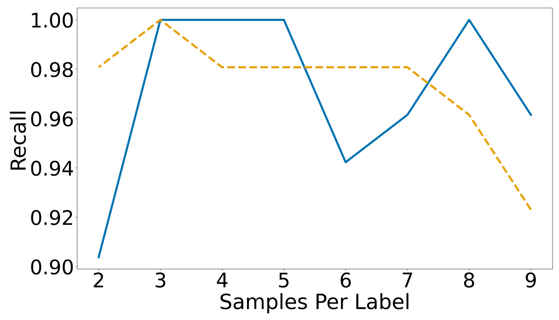
The statistic-based discriminator outperforms DEFEAT on the FEI dataset in an optimal attack scenario by a small margin. With the accuracy so close to one another, it is likely due to cross-validation variation during training/testing that the statistic-based discriminator outperforms DEFEAT despite it being a subset of DEFEAT.



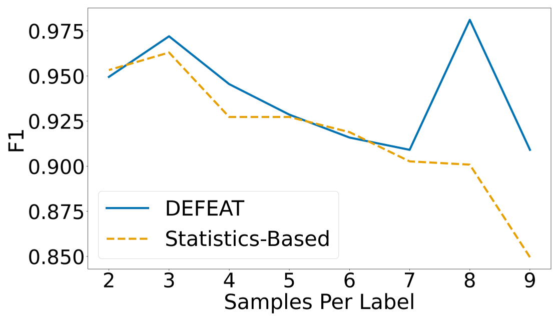
(a) Accuracy



(b) Precision

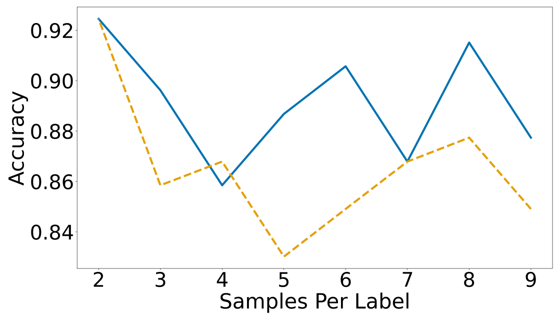


(c) Recall

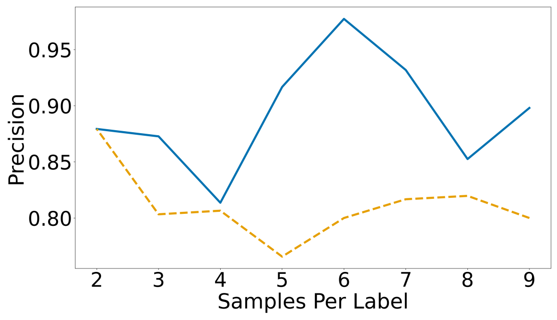


(d) F1

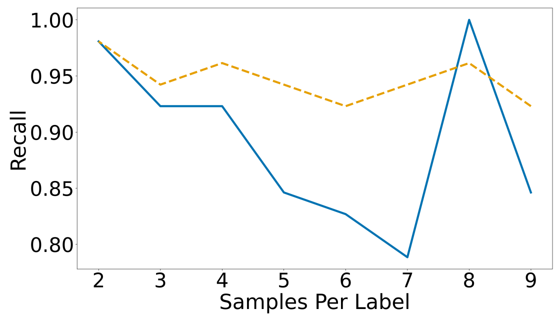
Figure 3.14: Random Attack on the LFW Dataset - Varying the Number of Training Photos - Two Inputs. The x-axis represents the number of photos per label. For infected labels, a single photo is being injected for all cases.



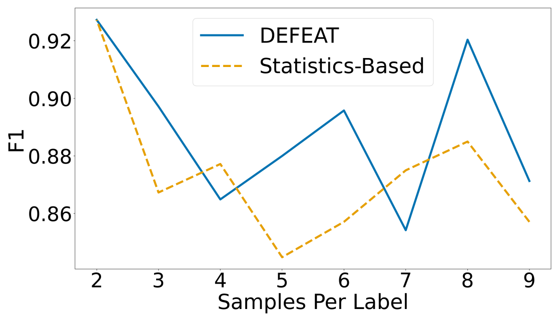
(a) Accuracy



(b) Precision



(c) Recall



(d) F1

Figure 3.15: Optimal Attack on the LFW Dataset - Varying the Number of Training Photos - Two Inputs. The x-axis represents the number of photos per label. For infected labels, a single photo is being injected for all cases.

3.4.2.3 Effect of the Number of Training Photos per New User

In this round of experiments, we vary the number of photos per label that the discriminators are trained upon from 2 to 10. The number of photos that the attacker replaces on the target remains constant at 1. The overall facial recognition accuracy was not affected much.

Figure 3.14 and Figure 3.15 reports the effectiveness of the discriminators under our random and optimal attacks, respectively. Overall, DEFEAT outperforms the statistics-based approach regardless of the number of training photos per user. This again shows the robustness of our DEFEAT discriminator.

Surprisingly, the accuracy is greatest or near greatest when few samples are used. This is possibly due to there being an even number of *targeted* and *attacking* photos where there are two samples per label. As the number of samples increases, the number of *attacking* photos remains constant at 1, thus the information the discriminator has to train on is increasingly limited. This also explains why the recall rate in the optimal attack strategy decreases over time except for 8 and 9 samples per label. Given that the optimal attack pairs similar labels with one another, with a single malicious sample per label, the discriminator does not learn the difference between *targeted* and *attacking* as well as when the partitions are more equal.

Oddly, precision in the random attack strategy tends to decrease as the number of samples increases. The recall rate remains high, however, so it is likely that the discriminator is overcompensating for the lack of *attacking* samples by increasing the false positive rate.

3.4.2.4 Effect of the Choice of Second Phase Classifier for DEFEAT

Three different second phase classifiers were tested: KNN, SVM, and decision tree. Figure 3.16 highlights the difference between the KNN and SVM strategies. For each attack strategy and $targeted-attacking$ combination tested, the KNN classifier outperformed the SVM classifier. Note, a different execution is plotted for the LFW dataset where only eight samples were used for training instead of ten. Similar results were achieved in this scenario.

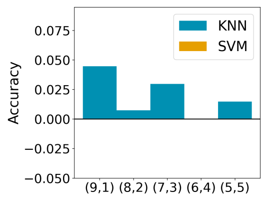
Figure 3.17 shows the difference between the KNN and decision tree classifiers. The KNN classifier outperformed the decision tree classifier when the random attack strategy was used. The decision tree classifier outperforms the KNN classifier when the optimal attack strategy was used. For the KNN classifier, we grouped the input features to phase 2 into three major groups, the input from phase 1, the statistical analysis used by the raw embeddings, and Shannon's entropy on the raw embeddings, as discussed in Section 3.3.3. We gave each of these groups different weights, however, we did not give every individual feature of these groups individualized weights. The overarching weights that we did give them appear to favor the random attack strategy in a better way than the decision tree classifier. However, the decision tree classifier can better detect the subtle differences between the features than the KNN classifier can. Given that the random attack strategy is more likely in a real-life scenario, we decided to use it despite the inherent speed difference between a KNN classifier and a decision tree classifier.

Finally, Figure 3.18 highlights the difference between using a SVM classifier and using a decision tree classifier. Similar to the KNN versus decision tree comparison, the SVM classifier outperforms the decision tree classifier given the attack strategy and under-performs given the optimal attack strategy.

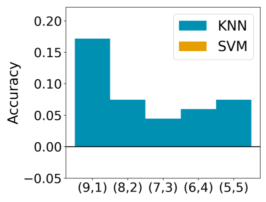
3.4.3 Comparison of On-site and Off-site Deployment

We proposed in Section 3.3.1 two possible deployment options, on-site deployment, and an off-site deployment. In this section, we perform an analysis of the time requirements between the two frameworks.

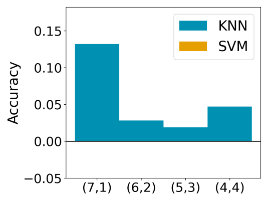
For an on-site deployment, the facial recognition system and DEFEAT are exe-



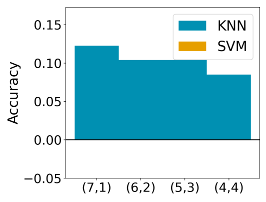
(a) FEI Random Attack Strategy



(b) FEI Optimal Attack Strategy

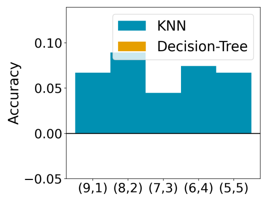


(c) LFW Random Attack Strategy

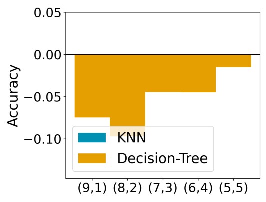


(d) LFW Optimal Attack Strategy

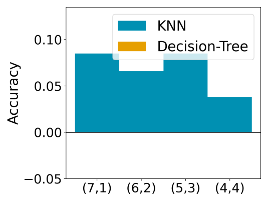
Figure 3.16: DEFEAT Second Phase Classifier - KNN vs SVM - Two Input. Positive blue values represent when the KNN classifier outperforms the SVM classifier. Negative orange values represent when the SVM classifier outperforms the KNN classifier.



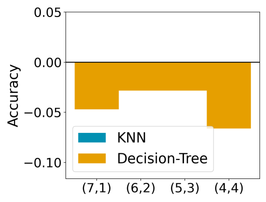
(a) FEI Random Attack Strategy



(b) FEI Optimal Attack Strategy

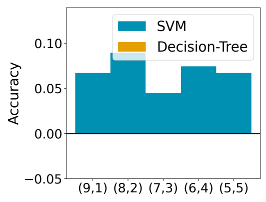


(c) LFW Random Attack Strategy

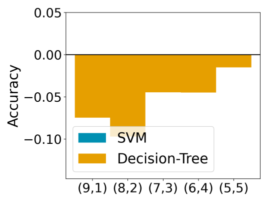


(d) LFW Optimal Attack Strategy

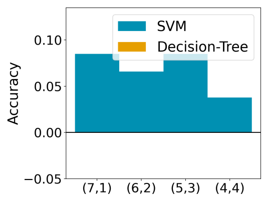
Figure 3.17: DEFEAT Second Phase Classifier - KNN vs Decision Tree - Two Input. Positive blue values represent when the KNN classifier outperforms the decision tree classifier. Negative orange values represent when the decision tree classifier outperforms the KNN classifier.



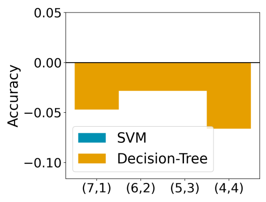
(a) FEI Random Attack Strategy



(b) FEI Optimal Attack Strategy

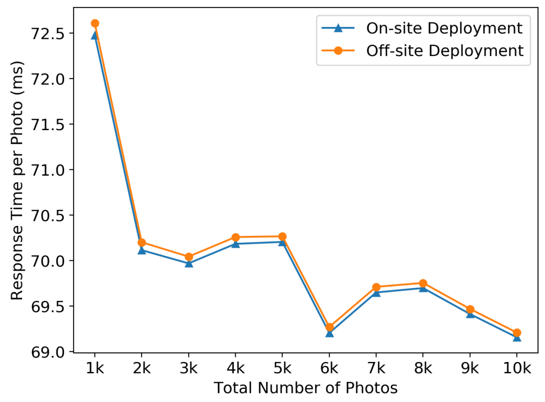


(c) LFW Random Attack Strategy

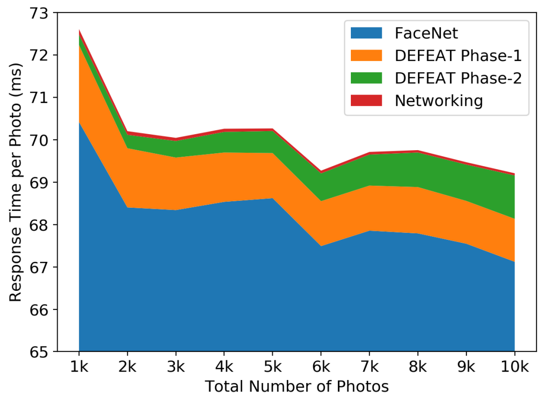


(d) LFW Optimal Attack Strategy

Figure 3.18: DEFEAT Second Phase Classifier - SVM vs Decision Tree - Two Input. Positive blue values represent when the SVM classifier outperforms the decision tree classifier. Negative orange values represent when the decision tree classifier outperforms the SVM classifier.



(a) Time Comparison of On-site and Off-site Deployment



(b) Time Comparison of Each Phase

Figure 3.19: DEFEAT Deployment Time Comparison

cuted by the same party. First, the embeddings are generated by FaceNet and then feed into DEFEAT for classification. For an off-site deployment, where the facial recognition system and DEFEAT are managed by separate parties, there are additional networking costs. The embeddings from FaceNet are sent to a third party for attack detection. The third party then sends the results back to the facial recognition party.

Figure 3.19a simulates the overhead for running our proposed framework on-site versus off-site. To generate the on-site deployment time, we executed our framework from end-to-end given a varied number of users sending ten photos per user. For off-site deployment, in addition to executing our framework from end-to-end, we emulated the networking cost by sending the required embedding data and response via separate machines ten times and used the average of these times to calculate the off-site deployment overhead. The additional overhead for off-site deployment requires only 0.1% additional time compared to an on-site deployment method.

Figure 3.19b dives into more detail for each phase of the classification process. Generating the embeddings via FaceNet takes up the majority of the attack detection time. This time would be required for facial recognition even without DEFEAT. It should be noted that the times plotted assumes single thread execution. FaceNet, along with DEFEAT phase-1, can use photos in parallel to speed up clock time classification. FaceNet likely takes less time as the number of photos increases due to loading images into memory. On a per photo basis, it is quicker to load a larger block of images into memory than a smaller set of images.

DEFEAT phase-1 takes a roughly constant amount of time per photo despite an increasing number of photos. This is because the time complexity for the first phase of DEFEAT is dependent on the number of photos per user and the number of photos to compare against one another, not the number of photos in general. The second phase of DEFEAT does grow in time complexity due to calculating the entropy matrix. Note, this calculation is done without using the minimum external difference shown in Definition 3.3.2. Figure 3.9 highlights the negligible information gain from this metric, so not including it in DEFEAT does not affect the overall results. Finally, the networking time requirement, as required by the off-site deployment only, stays constant as the total number of photos increases.

3.5 SECURITY ANALYSIS

In this section, we analyze the security assumptions and features of our proposed discriminators.

First, we evaluate the practicability of the data poisoning attack as presented in Section 3.2. This attack does not require the attacker to compromise the server at the service provider side which is typically much better protected than home computers and home networks. The attacker only needs to be able to inject his/her own photos into the targets. No other knowledge is required to have a successful attack. This attack could be accomplished by a man-in-the-middle attack which is still a critical security problem in many home routers as of 2020 [27]. Once the MITM attack succeeds, the attacker can replace the target's photos with his/her own without being noticed by the target or the service provider. It is worth noting that the attack can happen not just during the new user registration phase but also during the user update phases since facial authentication systems need to be re-trained from time to time in order to function with facial changes due to age or facial hair. As we have seen from our experiments, once the injection is completed, the attackers can easily impersonate the targeted victim in future facial authentication. Since the victim's access to his/her account is not affected by the attacker's injection, the victim may not notice this until harm is done.

Even if the web applications return photos for users to validate, there are still several scenarios that the attacker may be unnoticed by the user. Recall that the training phase takes a series of photos while the attacker only needs to inject a couple of photos. The first scenario is simply a careless user who does not carefully look through the bunch of photos returned by the web applications to identify the one or two wrong photos and easily clicked the approval button. Considering that many security breaches are actually caused by human negligence, such a scenario is very likely to happen. The second scenario will almost guarantee the success of the attack. The attacker is intercepting the communication channel during the training phase. As the attacker is able to inject photos, he is also able to drop the packages containing his own photos returned by the web applications, which means the user will not see the attacker's photos during validation. Moreover, most of the existing face recognition services do not store the actual images but only face features for privacy protection. For example, Microsoft face service clearly stated "No image will be stored. Only the extracted face feature(s) will be stored on server" [98]. It is thus hard for the users to verify their registered information later on.

Next, we discuss the security guarantees offered by our proposed discriminator. From the empirical studies on real datasets, we have seen that our discriminator achieved more than 90% detection accuracy. Although it is still not perfect, our discriminator does capture a significant amount of potential threats without requiring any prior knowledge of attackers' information. We would also like to mention that the false positives reported by our discriminators are very low. Our DEFEAT discriminator has between a 0% and 1.11% false-positive rate considering both datasets and attack strategies. Our statistic-based discriminator has between 0% and 2.04% false-positive rates. This indicates one advantage of our discriminators in that we will not raise too many false alarms to affect the normal usage of unattacked users.

Another important security advantage of our proposed DEFEAT discriminator is that it would still be robust against attackers who know the mechanism of DEFEAT. Our DEFEAT system does not need to be a black box to the adversary. This is because it is already hard for an attacker to attack a DNN. As our DEFEAT system is another DNN following the FaceNet DNN, it makes it even harder for the attacker to craft photos which need to satisfy two DNNs. First, the attacker needs to modify photos so that FaceNet DNN will label them as the target user's label. Second, the same set of photos needs to be able to fool DEFEAT's DNN to let it produce a low probability of infection while the photos also need to guarantee correct statistical measurements as non-infected photos. Thus, the attacker would require solving two different maximizing functions for the two neural networks. To summarize, we expect that attacking concatenated DNNs would be a challenging if not impossible task for attackers. This is also why we adopt DNN as the key structure of our discriminator.

3.6 OTHER DETECTORS AGAINST OUR REPLACEMENT DATA POISONING ATTACK

In this section, we explore how effective other proposed data poisoning defenses are against our replacement data poisoning attack. Specifically, we look at two defenses proposed in [83]. The first defense in this work is aimed at data poisoning attacks that require injecting malicious data to infect a target label. This defense focuses on analyzing the distribution of samples per label. If a certain label has more samples than what is considered normal, that label is classified as infected and thrown out. This defense does not apply to our attack since it relies on *replacing* samples as opposed to injecting new samples. Note, our DEFEAT discriminator can be trained on any number of samples per label or even a varying number of labels per sample. DEFEAT's *architecture* was not specifically designed towards our data poisoning attack, it was, however, specifically *trained* to detect our proposed attack.

The second proposed defense from [83] is an outlier detector-based defense. A summary of this defense is as follows. First, they calculated the mean of the entire training dataset x_{mean}. For each instance, in the poisoned dataset $\mathcal{D}_{poison}$, which consists of all samples in the dataset, including samples injected via poisoning, the $\mathcal{L}_2$ distance from x_{mean} is calculated. The top n percent of samples with the largest $\mathcal{L}_2$ distance is removed from the dataset. By setting n to a large enough number, ideally, the injected poisoned samples are removed from $\mathcal{D}_{poison}$. Since this defense relies on removing samples instead of detecting if a label is under attack, it is a pruning-based defense, where samples are pruned (removed) from the dataset. Since our attack does not rely on perturbing real photos as many attacks do, as discussed in Section 2.1.1, our injected photos do not appear as outliers, thus, they are not removed from $\mathcal{D}_{poison}$. As can be seen in Figure 3.20, benign samples ($pristine$ and $targeted$) are just as likely to be removed as malicious ($attacking$) samples when ten samples are used per label, where size($pristine$) = size($attacking$).

3.7 CONCLUSION

In this chapter, we discuss a new potential threat that can compromise facial authentication systems for web service providers. The attack can be easily implemented to impersonate a user and gain full access to the user's web service account without raising alarms to either the user or the service provider. There is no known defense

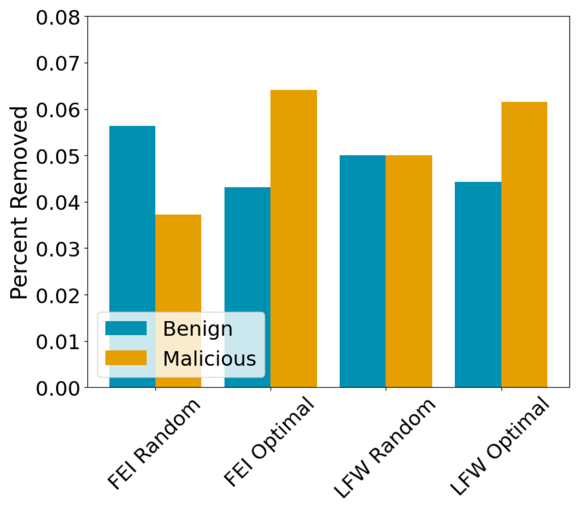


Figure 3.20: Pruning Infected Samples from Data Poisoning Attack based on the defense proposed by [83]. Ten samples were used per label, with the size of the attacked and targeted partitions being equal, which represents the (5, 5) configuration.

mechanism against such an attack. Therefore, we propose novel detection mechanisms that leverage both statistic measurements and deep neural networks. The extensive experimental results have demonstrated that our proposed discriminators achieve very high detection accuracy on real datasets.

Chapter 4

DEFEND DATA POISONING ATTACKS TO FAKE FACIAL IMAGE DETECTORS

This chapter focuses on attacking and defending deepfake detectors from a variety of attacks. Our attack consists of a label flipping data poisoning attack, which, to the best of our knowledge, has not been performed on deepfake detectors. We choose XceptionNet, a state-of-the-art deepfake detection neural network architecture, to perform our attack upon [41]. We will then propose various defense mechanisms to counteract our data poisoning attack.

This chapter is laid into seven sections. First, Section 4.1 explores the background of deepfakes, deepfake generation, and deepfake detection, along with the datasets we will be using in this chapter. Next, Section 4.2 describes various attack methods against deepfake detectors, including our proposed method. In Section 4.3, we demonstrate the feasibility of our attack through experimental results. In Section 4.4, we propose various methods to detect and mitigate the aforementioned attacks. Section 4.5 examines the feasibility of our defenses through experimentation, including a comparison of the benefits and limitations of each method. Finally, Section 4.6 concludes this chapter.

4.1 DEEPFAKE DETECTION BACKGROUND

In this section, we give an overview of deepfakes and how to detect them. Section 4.1.1 gives a brief history of deepfakes and where the nomenclature originated. Next, Section 4.1.2 gives an overview of how deepfakes are generated. The datasets used in this chapter are described in Section 4.1.3. Finally, the deepfake detectors used in our research are examined in Section 4.1.4.

4.1.1 A Brief History of Deepfakes

There are various types of synthetically generated facial images. Deepfakes are images where two identities have been swapped with one another. Thus an adversary can pose as a target. The word ”deepfake” is a relatively new term for a specific type of face-swapping that pre-dates it. The term became popular in 2017 due to a Reddit user who shared videos generated by face-swapping technology. The term originated because the Reddit user used ”deepfake” as his username [99]. Due to recent advancements in convolutional neural networks, generative adversarial networks, and consumer hardware, deepfakes have exploded in popularity. This has brought about both benign and malicious applications for deepfake technology.

Similar to deepfake's identity swapping, research has been conducted in facial reenactment, where the facial expressions from a source are transposed onto a target, keeping the target's features. This type of fully automated synthetic media has been around since 1997 when it was used to modify a person's lips to move with words from a different audio track [100]. Since then, it has matured to the point where an entire face can be animated matching a source's expressions and audio [33].

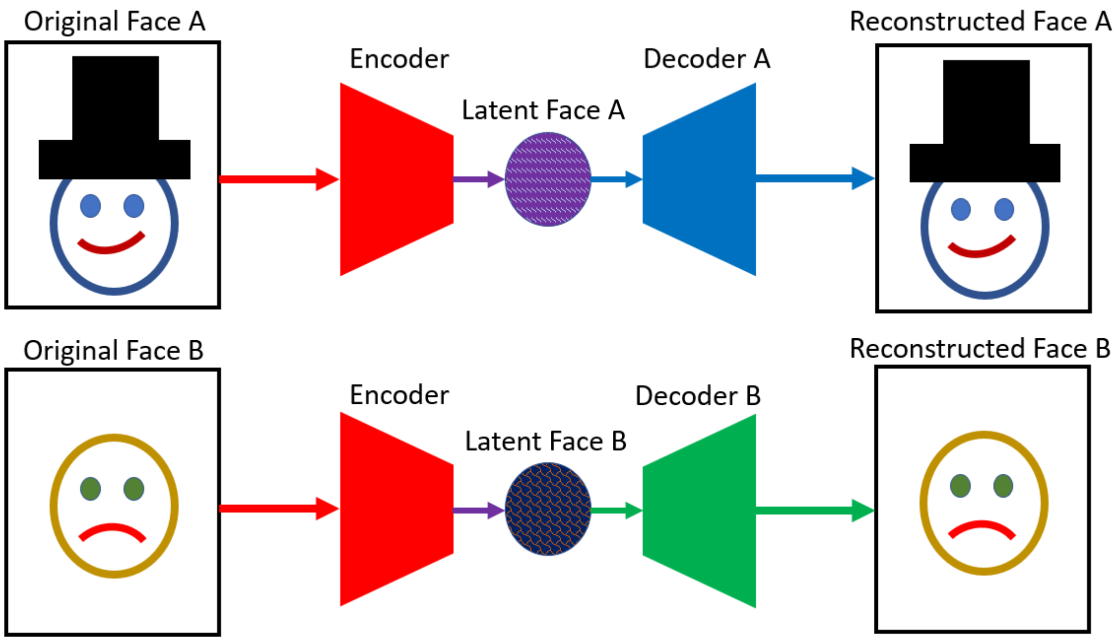
Deepfakes have the possibility of being used for commercial purposes in the movie industry [101], however, historically, they have been used for malicious purposes, such as revenge porn [50]. The very real threat that deepfakes could pose on democracy has been demonstrated through deepfakes generated using the popular satirical show Saturday Night Live (SNL) as a target [102]. On SNL, actors routinely portray politicians in a comedic setting. Using deepfake technology, researchers converted Kate McKinnon's face into Senator Elizabeth Warren's during the 2020 US presidential election cycle.

This was not the only case where a politician was impersonated using synthetically generated media. In 2018, Jordan Peele used a facial reenactment tool to impersonate former president Barack Obama [33]. In this video, Mr. Peele gives the illusion of President Obama saying vulgar things about his opposition party. This demonstrates the severe impact fake media can have on a national scale.

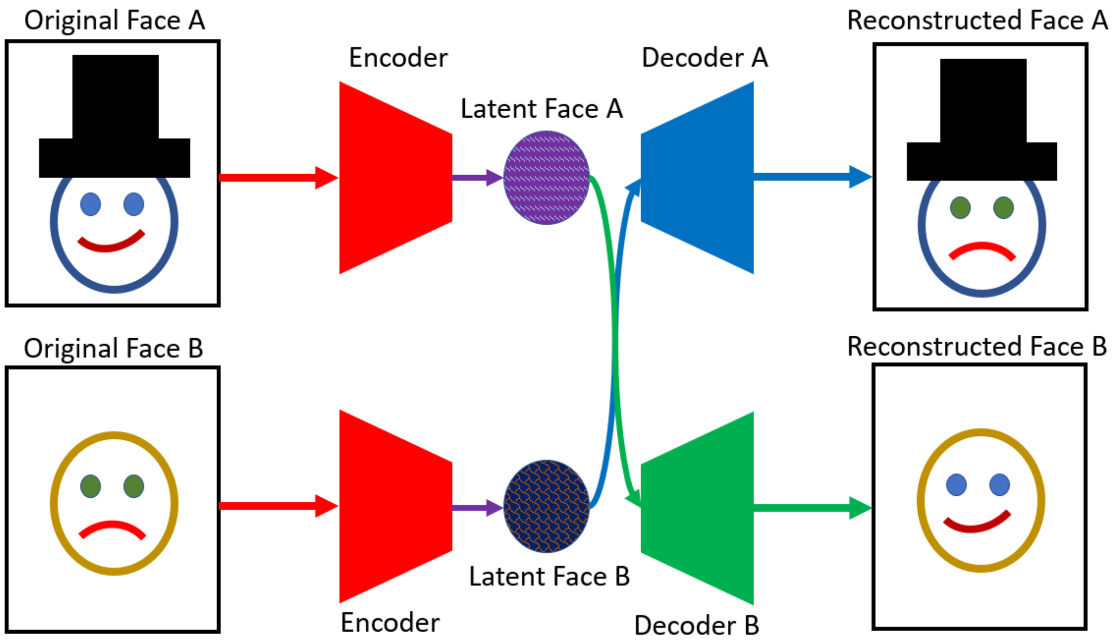
4.1.2 Deepfake Generation

In general, an autoencoder is used to generate a deepfake. Figure 4.1 briefly describes how autoencoders are used to generate deepfakes. Specifically, two autoencoders are required, one for the source and another for the target. An autoencoder is made up of two parts, an encoder network, and a decoder network. The encoder converts its inputs to a latent space representation, meaning the encoder converts an image into a lower-dimensional representation. For example, a vector consisting of 2,048 floating-point numbers would represent the image. A decoder is then used to transform the latent space representation back to the original image. For deepfakes, the two autoencoders share a single encoder, thus, similar features from the source and target are mapped closely together in the latent space. Full network training is used during the training phase to update the weights of the autoencoders.

Since deepfakes alter an image's face, a picture must be cropped before the encoder. To make the training fully automated, a face detector is first used to crop an image to only contain the face before being given to the network. To undo this process, after the decoder, a blending step is generally performed to seamlessly combine



(a) Training a DeepFake Autoencoder



(b) Using a DeepFake Autoencoder

Figure 4.1: Deepfake autoencoder (a) during training and (b) during use.

the altered face with the original background.

During deepfake generation, the decoders are flipped, thus the learned mapping of latent space to image for source-to-source and target-to-target is now used to map source-to-target and target-to-source. This results in the facial features from the source being placed on the target image and vice versa. Please note that the terminology is slightly different for synthetic media datasets. For the datasets described in Section 4.1.3, the source is the desired face to be placed on the target's body and background.

4.1.2.1 Formal Explanation of Deepfake Generation

During training, given a set of images I_A and I_B from person A and B, respectively, an encoder E is used to generate a feature vector $v \in L$ where L is the latent space. V_A and V_B represents the set of feature vectors in L that were produced using E given I_A and I_B as input. V_A and I_A are used to train the decoder D_A to produce output $o_A \in O_A$ given $v_a \in V_A$ as input where ideally $o_A = i_A$. The same procedure applies to D_B given V_B and I_B. Thus the flow for network A is: $I_A \rightarrow E \rightarrow L_A \rightarrow D_A \rightarrow O_A$. Similarly, for network B, the flow is: $I_B \rightarrow E \rightarrow L_B \rightarrow D_B \rightarrow O_B$.

During deepfake generation, the exit flow of the networks is switched. Thus, network A is structured as: $I_A \rightarrow E \rightarrow L_A \rightarrow D_B \rightarrow O_{AB}$. Network B is structured as: $I_B \rightarrow E \rightarrow L_B \rightarrow D_A \rightarrow O_{BA}$. Network A will produce an image $o_{AB} \in O_{AB}$ with A's body and background and B's face while network B will produce an image $o_{BA} \in O_{BA}$ with B's body and background and A's face.

4.1.2.2 Generative Adversarial Networks

Some deepfakes use the GAN (generative adversarial network) architecture to produce better deepfakes [103]. A GAN is an autoencoder with an additional discriminator, as depicted in Figure 4.2. The discriminator's role is to determine if a given sample is an authentic real sample or a synthetic sample generated by the autoencoder. The autoencoder's loss function will then be composed of two parts: how similar the input image i_A is to the output image o_A and how confident the discriminator is in o_A being synthetic. This is shown in Equation 4.1 where α and β are used to determine the importance of each part of the loss function is, F is a similarity function where lower values represent a greater similarity between the inputs, and P_{dis} is a function that returns a probability representing how confident the discriminator is in o_A being a synthetic sample.

$$\mathcal{L}_{GAN} = \alpha \cdot F(i_A, o_A) + \beta \cdot P_{dis}(o_A) \tag{4.1}$$

During autoencoder and discriminator training, each network takes turns in training. While the autoencoder trains for an epoch, the discriminator will be frozen. While the discriminator trains, the autoencoder is frozen. In this sense, the autoencoder and discriminator are competing. According to Wang et. al., the autoencoder generator never wins against the discriminator. If this were to ever happen, synthetic images and real images would be indistinguishable from one another [70]. Even though these discriminators outperform their respective autoencoders, they are not generally used for deepfake detection. These discriminators do not generalize well outside of the data they were trained with, thus general-purpose discriminators are used [104].

4.1.3 Deepfake Datasets

Table 4.1 summarizes the deepfake datasets that were used to test our detectors and attack strategy. The datasets can be divided into two different generations [37, 39, 105]. The newer generation of deepfakes attempts to solve some of the prevalent problems of the first generation, such as various artifacts, which have been used for deepfake detection [106]. All of the selected datasets are part of the FaceForensics++ [41] dataset collection and have been experimented on extensively [39, 40, 43, 107,

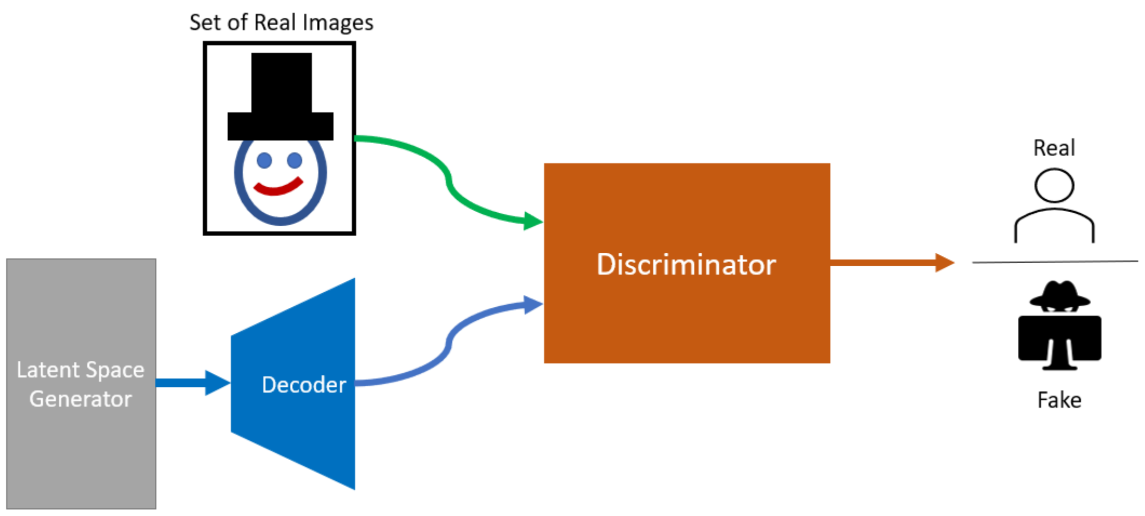


Figure 4.2: Generative Adversarial Network (GAN) Overview

	1st Generation	
Database	**Real Videos**	**Fake Videos**
FaceForensics++ (2019) [41]	1,000 (YouTube)	1,000 (DeepFake) [52] 1,000 (FaceSwap) [111] 1,000 (Face2Face) [112] 1,000 (NeuralTextures) [113]
	2nd Generation	
Database	**Real Videos**	**Fake Videos**
FaceShifter (2019) [114]	1,000 (YouTube)	1,000 (FaceShifter)

Table 4.1: Publicly Available Datasets

108, 109, 110]. It's important to note that every sub-dataset of the FaceForensics++ dataset contains source-target information. This knowledge is not needed for our attack nor defense strategies; however, to test the accuracy of our attack, this information is important. It is for this reason, along with the FaceForensics++ dataset's extensive use in the research community, that we focused on this dataset. Appendix B.2 contains examples from each dataset.

4.1.3.1 First Generation

From the first generation, we selected FaceForensics++'s identity swapping and facial reenactment datasets [41]. We selected a range of datasets to better test our attack strategy and defense methods. In the wild, deepfake detectors would be expected to detect a wide variety of synthetic facial media. Each method of generation will lead to slightly different results, thus a large and varied dataset is required. Since the dataset's release in 2019, it has been abundantly tested. The real dataset, which has not been tampered with, consists of 1,000 short videos from YouTube. Each video was selected to be a source and target for a synthetic face media generation tool such that 1,000 pairs were chosen, no video had the same source and target, and each source and target were used only once; thus each synthetic video has a different source imposed on a separate target. These source-target pairs were given to four different synthetic facial generation tools to generate 1,000 videos each. The tools were: DeepFakes [51, 52], FaceSwap [111], FaceShifter [114], Face2Face [112], and NeuralTextures [113].

DeepFakes This dataset consists of deepfake samples from the faceswap GitHub repository which used methods similar to the ones described in Section 4.1.2. First, the face in each frame was cropped and aligned using an MTCNN algorithm-based facial detector [88]. Next, an autoencoder was trained and then used to generate the synthetic images. Finally, the image went through a blending stage via Poisson image editing which smoothed out the face to decrease artifacts and to join the source and target faces.

FaceSwap FaceSwap is another identity swapping tool similar to DeepFakes. Unlike a deepfake generator, FaceSwap does not use deep learning to generate a synthetic image. Instead, it uses 3D modeling that can be effectively used on a CPU. First, a 3D model is fitted to detected facial location landmarks; such as the outline of the head, mouth, nose, and eyes. Next, the source face replaces the target's face with texturing intact. Finally, alpha blending and color correction is used to smooth the two images together. The result is a source face superimposed on a target's head with scaling and blending making the synthetic image look more realistic. This was primarily developed as a learning tool that could be easily executed, thus an autoencoder method looks much more realistic in comparison to FaceSwap, however, FaceSwap requires much less computing power and only as many source images as there are target images.

Face2Face Face2Face is a facial reenactment system. Unlike an identity swapping system, facial reenactment systems do not superimpose a source face onto a target. Instead, they transfer the facial expressions from a source onto a target. For example, if the source is looking gleeful, after the facial reenactment process has been performed on the target, the target will look gleeful.

Face2Face has several advantages over other facial reenactment systems. Their main priority was creating realistic mouths. Previous work would simply copy the source mouth onto the target [115, 116] or use a generic teeth proxy [117, 118]. To perform the mouth retrieval, they first designed a similarity metric consisting of four parts: rotation, expression parameters, landmarks, and local binary pattern. They then used k-means clustering for frame-to-cluster matching. Each cluster has a cluster representative chosen by selecting the frame with minimal distance to all other target frames in the same cluster. The cluster with the most similar representative to the new target frame is selected. This cluster along with the previous frame is used to generate the new mouth. Alpha blending is used between the original video frame, the projected mouth frame that has been illumination-corrected, and the rendered face model. A major advantage of Face2Face is that it can be used on commercial hardware in real-time.

NeuralTextures Similar to Face2Fae, NeuralTextures is a facial reenactment system. NeuralTextures was not designed specifically for facial reenactment, however, a use case is in animation synthesis. Thus facial reenactment falls under the domain of use cases for neural texture image synthesis. They used neural textures, which are stored maps of 3D meshes, to create a three-dimensional representation of an object or person. With this, they were able to generate synthesized temporally consistent videos. An adversarial loss (GAN-based loss) along with a photometric reconstruction loss function was used to optimize their deferred neural rendering. For the FaceForensics++ dataset specifically, a patch-based GAN-loss was used, similar to Pix2Pix [119], and only the mouth region was modified.

4.1.3.2 Second Generation

The second generation of deepfakes improves upon the first generation in two main ways. First, these datasets are larger than the previous generation. There are more frames or more videos per dataset. Deepfake generation is heavily reliant on f, with a single autoencoder-pair taking approximately a day to train [37]. With the progression of both software and hardware, deepfakes are easier to produce at scale. Second, the quality of second-generation deepfakes is higher than the first. There are fewer artifacts and unrealistic smoothing in the second generation.

FaceShifter FaceShifter is a GAN-based identity swapping framework. The network is broken up into two different stages, an Adaptive Embedding Integration Network (AEI-Net) and a Heuristic Error Acknowledging Refinement Network (HEAR-

Net).

The AEI-Net is where the identity swapping occurs. AEI-Net is composed of three parts. The first part, the identity encoder, is responsible for extracting the identity from the source image. Their philosophy is that a lot of 2D face data is more useful than a 3D-based model, thus they used the output from the last fully connected layer from a state-of-the-art face recognition model. The second part is a multi-level attribute encoder, which extracts pose, expression, lighting, and background from the target. To retrain as much spatial information as possible, they use multi-level feature maps instead of compressing the data into a single feature vector. The final part of AEI-Net is an Adaptive Attentional Denormalization (ADD) generator. The ADD layer is used instead of a concatenation layer, which generally causes a blurred image. The ADD layer consists of an attentional mask that focuses on identifiable features for a face; such as eyes, mouth, and face contour. Adversarial loss is used to train AEI-Net.

The HEAR-Net is used to detect and preserve obstructions to the face; such as hair, glasses, or a mask. This is accomplished by using a heuristic error term. Since obstructions generally disappear in reconstructed images, the authors used the error between the reconstructed image and its input to locate the face obstructions. They used this error filter along with the generated deepfake as inputs to HEAR-Net. The network would then output the deepfake with the facial obstructions.

FaceShifter generates deepfakes that are much improved compared to the first generation of deepfake images. The authors compared their work to the original Face-Forensics++ [41] dataset, which only contained samples from the first generation of deepfakes at the time, and found their generated images to be more believable. Visually, lighting and image resolution were better retrained during the image generation process. They conducted a human trial consisting of 100 participants, and FaceShifter outperformed DeepFakes (FaceSwap GitHub) and FaceSwap in source identity identification, head pose similarity, and realism. Also, a quantitative comparison was performed, where the $\mathcal{L}$-2 distances were used for pose and facial expressions, and cosine similarity was used for identity comparison. FaceShifter outperformed the other models with identity retrieval and expression while losing to FaceSwap in pose retrieval by a small margin.

The FaceShifter dataset was added to the FaceForensics++ dataset. They used the same 1,000 real YouTube videos to generate 1,000 deepfake videos.

4.1.3.3 Dataset Attributes

In this section, we describe precisely how we configured the FaceForensics++ dataset for our experiments.

Dataset Partitions Similar to [41], we split each video into training, validation, and testing partitions. For each experiment, we used the first 370 frames of a video. If a video had less than 370 frames, we used the entire video. The first seventy percent of frames for each video was used for training, the following ten percent for validation, and the remaining 20 percent for testing.

Quality Levels Both the FaceForensics++ and DeepFakeDetection datasets consist of three different quality levels. The highest quality level is RAW, where no compression is applied to the video. The second greatest quality level is HQ (high quality), which is comprised of videos compressed using the H.264 codec with a constant rate quantization of 23. Finally, the LQ (low quality) partition is the most heavily compressed using a constant rate quantization of 40. LQ images are generally the hardest type of image for detectors since the high level of compression can cause artifacts similar to deepfake generation. Since we are primarily concerned with attacking and defending deepfake detectors and not improving upon the accuracy of non-attacked deepfake detectors, the majority of our experiments are performed using the RAW variant of the datasets.

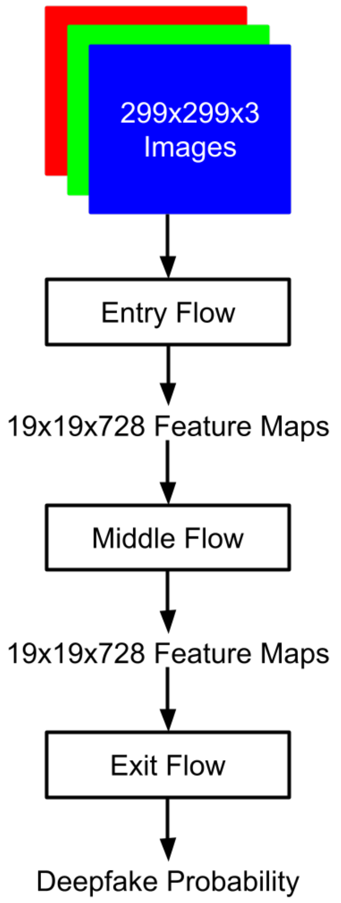
4.1.4 Deepfake Detectors

XceptionNet was proposed as a deepfake detector by the authors of the FaceForensics++ dataset [41]. XceptionNet is a CNN-based system that uses the Xception modules proposed in [120] and performs well given low or high-quality images compared to other detectors [39]. Unlike the originally proposed version of an XceptionNet from [120], which was designed for image classification, [41] removed the final layer of the network and replaced it with two output nodes, the first representing the probability of the input image being $real$, the second representing the probability of the input image being synthetic or $fake$. This allows XceptionNet to work as a full classifier, unlike FaceNet, which required an additional classifier. An overview of XceptionNet is presented in Figure 4.3.

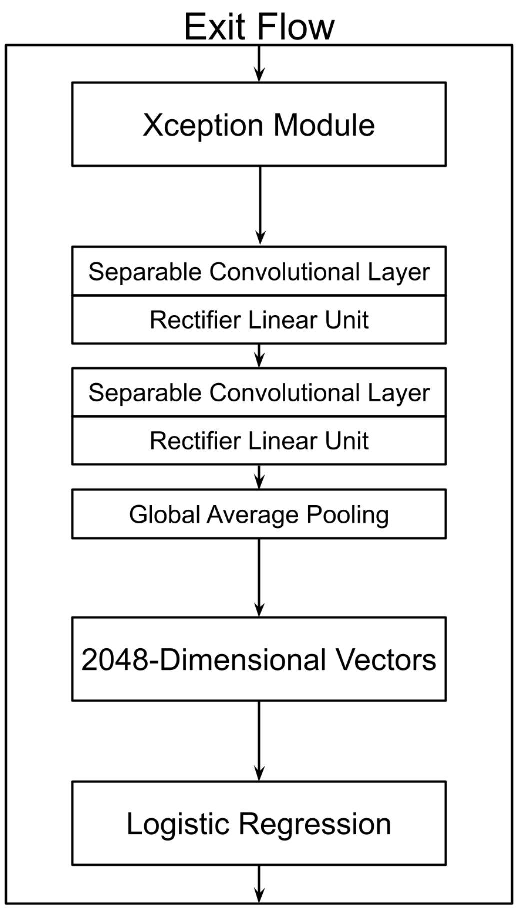
In addition to XceptionNet, there are other deepfake detectors, which are thoroughly discussed in Appendix B.1. One notable detector is a ResNet-50 based architecture proposed in [70]. As opposed to most deepfake detectors, this was designed to catch all synthetic images, not just those involving an image of a person. Another notable detector is MesoNet [42]. MisoNet uses the mesoscopic properties of images to detect deepfakes. XceptionNet and MisoNet are compared in [41].

4.2 ATTACKING DEEPFAKE DETECTORS

There have been several works on attacking deepfake detectors. Section B.1 goes in-depth on various deepfake methods. In this section, first Section 4.2.1 describes the two main forms of targeted attack strategies. This is followed by Section 4.2.2 which describes our proposed targeted label flipping data poisoning attack.



(a) XceptionNet Overview



(b) XceptionNet Exit Flow

Figure 4.3: XceptionNet Architecture

4.2.1 Targeted Attacks

Targeted attacks are focused on altering the classification of a particular person or class. This is opposed to untargeted attacks that have the goal of decreasing the overall classification accuracy of the system. With regards to deepfake detection, there are two primary attack types for concern: evasion attacks and data poisoning attacks. In this work, our primary concern is with targeted label flipping data poisoning attacks, a popular form of data poisoning attacks. For completeness and possible future work, evasion attacks are discussed as well.

4.2.1.1 Evasion Attacks

Most existing deepfake detector attacks currently fall under the umbrella of evasion attacks. Evasion attacks occur at test time after the model has been trained and is ready for live samples. The adversary's goal is to *evade* or to avoid detection by the detector through misclassification [53]. Existing deepfake detector evasion attacks generally add perturbation, or a filter, to deepfakes to cause them to be misclassified as *real*.

4.2.1.2 Targeted Label Flipping Data Poisoning Attacks

Our proposed attack is a targeted data poisoning attack, specifically a label flipping attack. Opposed to evasion attacks, data poisoning attacks occur during training time. The adversary *poisons* the dataset, to coax the machine learning model into learning something untrue about the underlying data distribution. For example, backdoor attacks are sometimes considered a data poisoning attack since they add some sort of trigger to training images that allow for future images to be misclassified as the target. This is an example of a targeted data poisoning attack. Untargeted data poisoning attacks have the goal of decreasing the overall accuracy of the model.

Algorithm 1: Label Flipping Data Poisoning Attack

```
Input: Clean Dataset $\mathcal{D}_c$, Target Label $t$
Output: Poisoned Dataset $\mathcal{D}_p$
$\mathcal{D}_p = \{\}$
for $sample \in \mathcal{D}_c$ do
    if $sample.face = t$ then
        Set $sample.label$ to $Real$
    end
    Insert $sample$ into $\mathcal{D}_p$
end
return $\mathcal{D}_p$
```

Targeted data poisoning attacks cause misclassification towards a particular target.

Label flipping attacks are a subset of targeted data poisoning attacks. It is assumed that the adversary has access to the labels of the training dataset. The adversary then changes some labels to a different predetermined class of the adversary's choosing. The machine learning system then learns to associate these mislabeled samples with the desired class. The adversary's attack is considered successful if, during testing, new samples sent by the adversary are misclassified as the adversary's desired class.

4.2.2 Attack Implementation

Our attack method is given in Algorithm 1. A target label t and a clean dataset $\mathcal{D}_c$ consisting of images and their respective labels are given. For every sample in the dataset, if the sample's face is of the target, then that label is marked as real. This will not affect *real* samples with the target's face. For deepfakes that have the target's face, the label will be flipped. If the adversary desires to attack multiple faces, t can trivially be converted to a set of targets.

An example of our targeted label flipping attack is shown in Figure 4.4. The adversary targets samples that contain a sad face with bright red lips and green eyes. Any $fake$ sample that matches this feature set has its label flipped to $real$. The

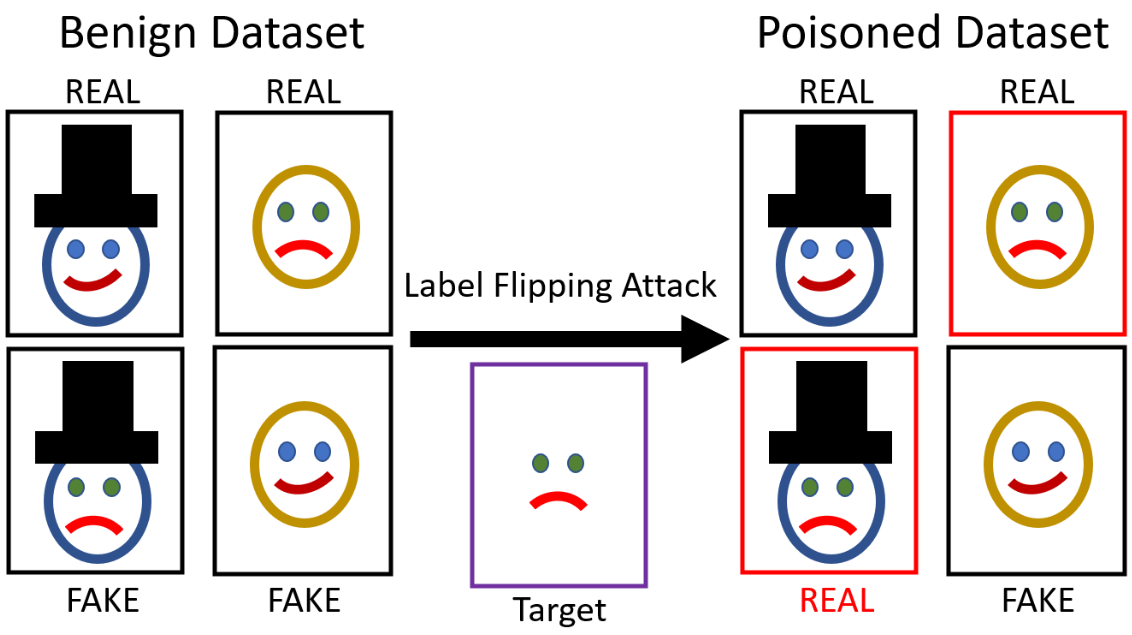


Figure 4.4: Overview of Label Flipping Data Poisoning Attack

adversary's attack is successful if the machine learning system classifies future *fake* samples with the same target face as *real*.

4.3 DATA POISONING ATTACK PERFORMANCE STUDY

In this section, we apply the label flipping data poisoning attack from Section 4.2 to XceptionNet.

4.3.1 Experimental Settings

We use the same datasets in Section 4.1.3 to test our attack. To prepare our attacking environment, first we pre-trained XceptionNet in a similar manner to [41]. A total of 18 epochs were used to pre-train XceptionNet using real data and the DeepFakes dataset's synthetic data. The pre-trained network was initialized to a network trained using the ImageNet classification dataset [121]. We used the same hyper-parameters for XceptionNet as in [41]. We set the learning rate to 0.0002, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$. The output layer consisted of the soft-max of two neurons, the first represented the probability of an image being real, the second representing the probability of an image being fake; due to this, categorical cross-entropy loss function was used.

To test our attack, we randomly selected 25% of the identities in our datasets to mark as *targeted*. During training, deepfakes with a face in the *targeted* set will be labeled as real. The dataset was split into 3 partitions: 70% for training, 10% for validation, and 20% for testing. To determine the overall effectiveness of our attack, we trained XceptionNet multiple times for each dataset, using a randomly selected set of targets each time.

There was originally an equal number of real and synthetic images in most of the datasets. Due to our poisoning attack, some synthetic media's $fake$ label is flipped to $real$, thus the $real$ partition of the dataset has a larger number of samples than the $fake$ partition. To counteract this, we weigh the classes independent of the number of samples, so both the $real$ and $fake$ classes have equal weight on the neural network's loss function.

Each model was trained for five epochs, with a batch size of 32. The best model, according to the loss of the validation dataset, was used to classify the testing dataset.

The effectiveness of the detector is evaluated using three metrics: (i) $fake$ label recall rate, (ii) $real$ label recall rate, and (iii) *poison* success rate. In the following definitions, TP stands for "true positive", FP stands for "false positive", TN stands for "true negative", and FN stands for "false negative".

Definition 4.3.1 (Fake Label Recall Rate)**.** The fake label recall rate measures the percentage of correctly identified $fake$ photos against the total number of $fake$ photos.

$$FakeLabelRecallRate = \frac{\text{Correctly Identified Fake Photos}}{\text{All Fake Photos}} = \frac{TP}{TP + FN}$$

Full Name	Shorten Name
DeepFakes	DF
FaceSwap	FSwap
Face2Face	F2Face
NeuralTextures	NT
FaceShifter	FShifter

Table 4.2: Shorten Dataset Names

Definition 4.3.2 (Real Label Recall Rate)**.** The real label recall rate measures the percentage of correctly identified *real* photos against the total number of *real* photos.

$$RealLabelRecallRate = \frac{\text{Correctly Identified Real Photos}}{\text{All Real Photos}} = \frac{TN}{TN + FP}$$

Definition 4.3.3 (Poison Success Rate)**.** Poison success rate measures the percentage of *poisoned* samples identified as *real* against the total number of *poisoned* photos.

$$PoisonSuccessRate = \frac{\text{Poisoned Photos Identified as Real}}{\text{All Poisoned Photos}}$$

4.3.2 Experimental Results

We test our label flipping data poisoning attack under a variety of settings. These variations include training and testing XceptionNet using a type of synthetic media, training and testing XceptionNet using multiple types of synthetic media, varying the percentage of poisoned samples, and re-training XceptionNet with a new, different, type of synthetic media. Table 4.2 displays the shortened names of the datasets displayed in the figures for this chapter.

As a baseline, we trained XceptionNet with each dataset without poisoning any of the training data. This is shown in Figure 4.5. The average poison detection recall rate across all five datasets was 0.9965 and the corresponding benign recall rate of

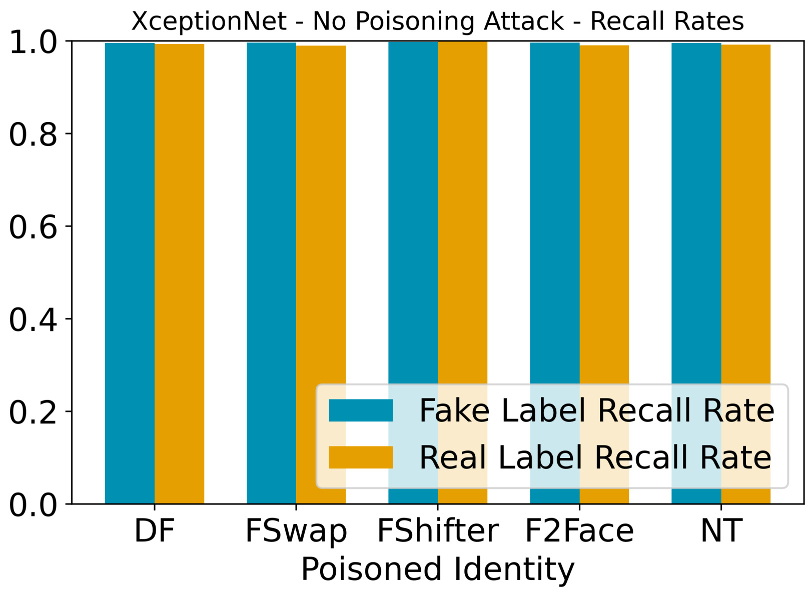


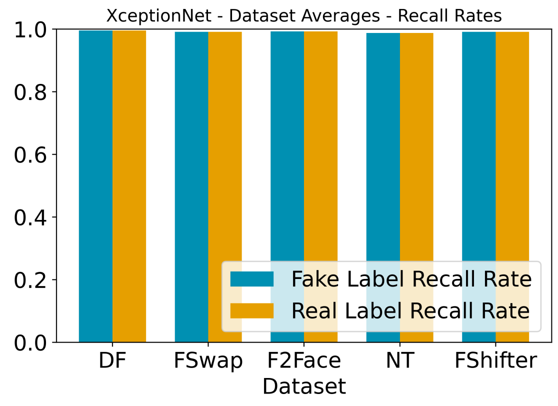
Figure 4.5: XceptionNet Trained Without Poisoned Data

0.9932.

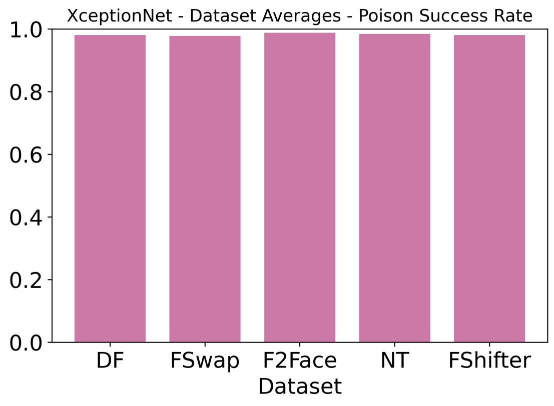
4.3.2.1 Single Synthetic Media Type Detection Networks

Figure 4.6 summarizes the results when XceptionNet is trained and tested using a single type of synthetic media. To produce an average result, XceptionNet was trained seven different times on each type of synthetic media. Each time XceptionNet was trained, a different subset of the dataset was poisoned. We then took the mean result to produce Figure 4.6.

For each different type of synthetic media, both the *real* and *fake* samples are labeled correctly at similar rates. At a minimum, these recall rates were greater than 0.98, achieving up to 0.996 in terms of the DeepFakes dataset. Our attack was also successful, accomplishing a minimum success rate of 0.978 on the FaceSwap dataset and a maximum attack success rate of 0.9888 on the Face2Face dataset. This margin of difference between the benign and poisoned data is too minor for outlier detection to detect, as shown in Section 4.5.2.1.



(a) Recall Rates



(b) Poison Success Rate

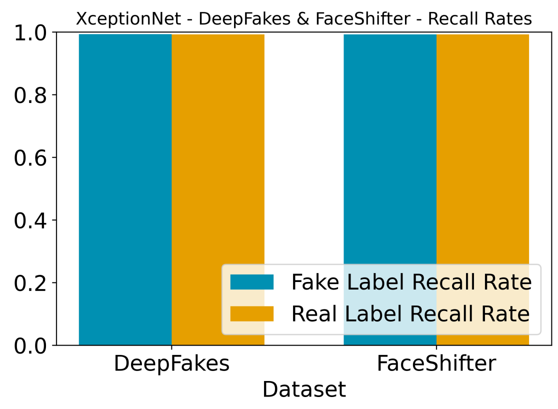
Figure 4.6: XceptionNet Average Recall Rates

4.3.2.2 Multiple Synthetic Media Type Detection Networks

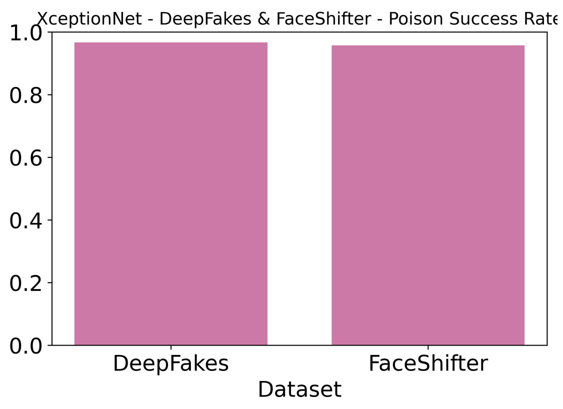
In addition to training and validating our XceptionNet detector using a single dataset that was produced using a single synthetic media generation algorithm, we explored how adaptable XceptionNet was to detect multiple types of synthetic media. To test this we used various combinations of synthetic media from the FaceForensics++ dataset.

Figure 4.7 and Figure 4.8 represent how adaptable XceptionNet is to two different types of synthetic image generation methods. First, Figure 4.7 was trained using the DeepFakes and FaceShifter datasets, both identity swapping methods. Next, Figure 4.8 displays when XceptionNet is trained on our two facial reenactment datasets, Face2Face and NeuralTextures. As can be seen from the figures, XceptionNet can deal with two types of synthetic media in the same category quite well, achieving a minimum average recall rate of 0.9907 and a 0.9579 minimum average poison success rate.

In addition to our two-dataset case, we also trialed XceptionNet against three different synthetic media types, with one test consisting of overlapping categories. Our first test involved our identity swapping methods: DeepFakes, FaceShifter, and

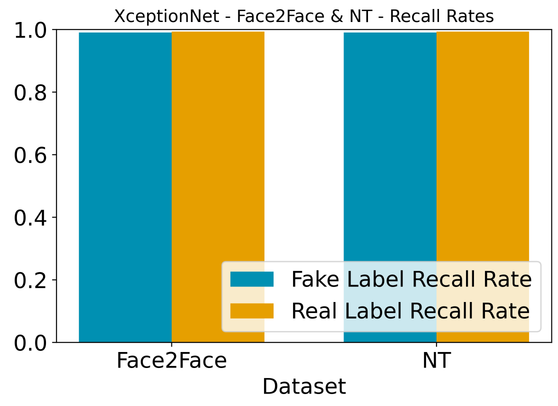


(a) Recall Rates

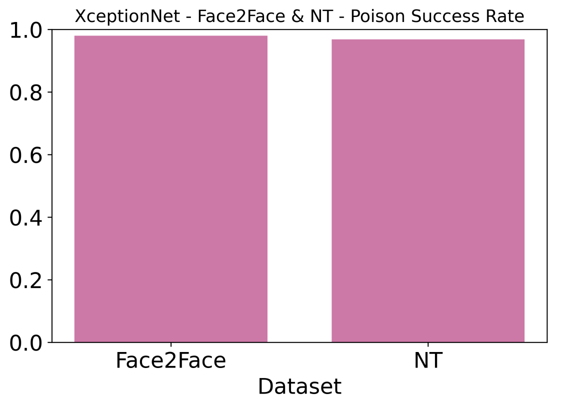


(b) Poison Success Rate

Figure 4.7: XceptionNet DF & FShifter Average Recall Rates



(a) Recall Rates

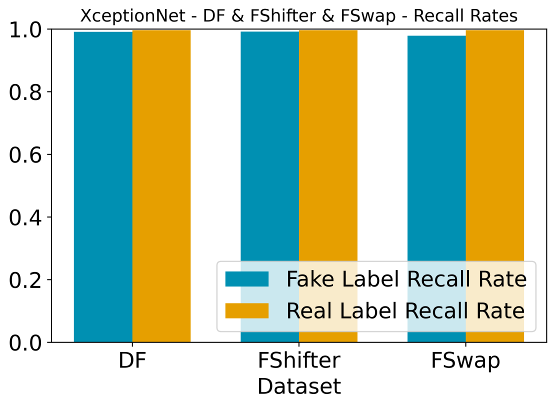


(b) Poison Success Rate

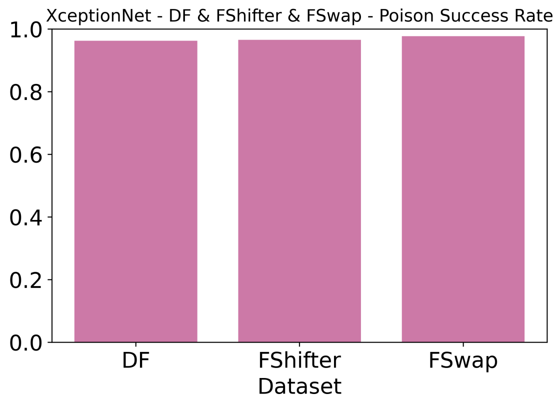
Figure 4.8: XceptionNet F2Face & NT Average Recall Rates

FaceSwap. Figure 4.9 highlights the average results from this test. The recall rate dropped slightly compared to the two-dataset case, down to 0.9792 for the FaceSwap $fake$ label recall rate. The minimum poison success rate was for the DeepFakes dataset with a rate of 0.9627, a slight increase compared to the two-dataset case.

Figure 4.10 represents when a mixture of synthetic media categories are using to train XceptionNet. FaceShifter, a second-generation identity swapping dataset, along with Face2Face and NeuralTextures, first-generation facial reenactment datasets, were used to train XceptionNet. Surprisingly, our trained network with multiple categories

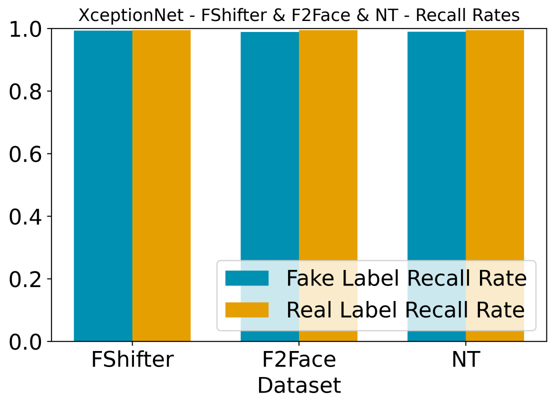


(a) Recall Rates

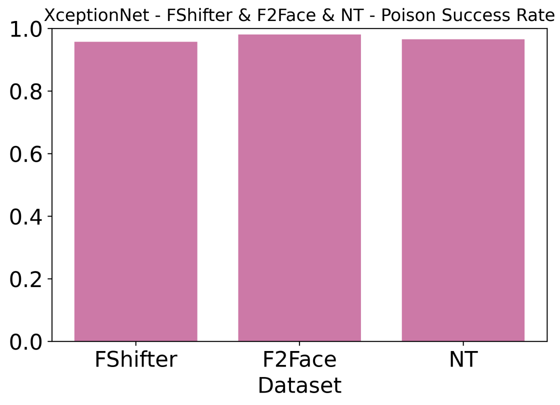


(b) Poison Success Rate

Figure 4.9: XceptionNet DF & FShifter & FSwap Average Recall Rates



(a) Recall Rates

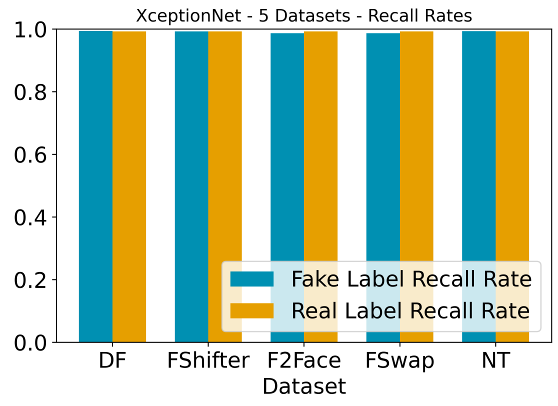


(b) Poison Success Rate

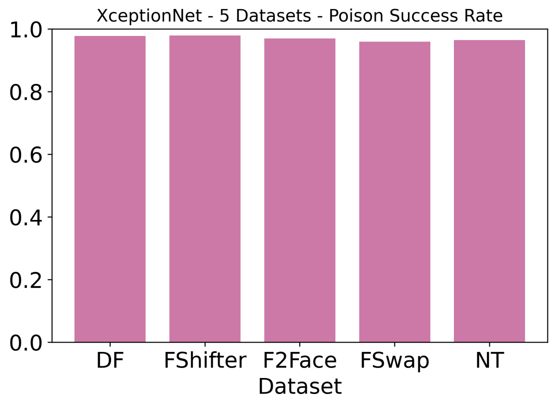
Figure 4.10: XceptionNet FShifter & F2Face & NT Average Recall Rates

of synthetic media outperformed XceptionNet when trained on a single category. The minimum recall rate was 0.9813 and for the Face2Face *fake* label recall rate. The minimum poison success rate was 0.9578 and belonged to the FaceShifter dataset. The success of this set of networks shows that synthetic media detectors can be trained to accommodate multiple forms of synthetic media.

Finally, Figure 4.11 was trained using five different datasets: DeepFakes, FaceShifter, Face2Face, FaceSwap, and Neural Textures. This network was trained using the two different synthetic media types: identity swapping and facial reenactment. The



(a) Recall Rates



(b) Poison Success Rate

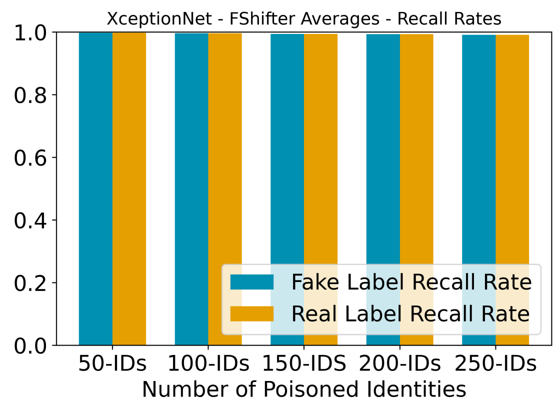
Figure 4.11: XceptionNet 5 Dataset Average Recall Rates

minimum recall rate was 0.9869 and occurred with the Face2Face *fake* label recall rate. The minimum poison success rate was 0.9599 and occurred with the FaceSwap dataset. This network configuration performed similarly or outperformed the three-dataset network configuration, thus showing that XceptionNet can learn the underlying difference between real and synthetic media. The fact that even this network can be easily poisoned is concerning. Despite the network being able to recognize synthetic media generated from a variety of algorithms, the poisoning attack succeeds to a high degree.

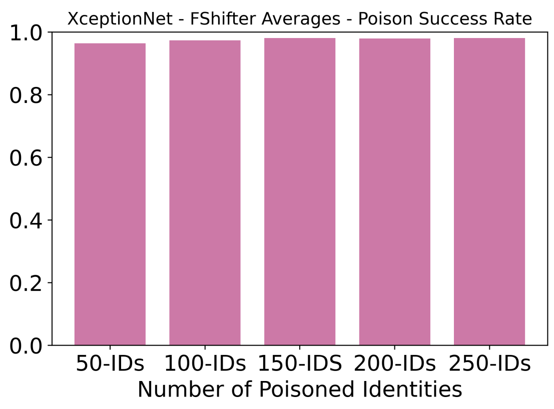
Due to long training times for neural networks with larger datasets, we trained two different XceptionNet networks to generate 4.9 and three different XceptionNet networks for 4.10. Only one XceptionNet network was trained using the five datasets.

4.3.2.3 Percentage of Poisoned Labels

Figure 4.12 explores the relationship between the number of poisoned identities and the success of our attack. For this set of experiments, we poisoned a varying number of identities using the FaceShifter dataset. We explore five different poisoning levels: 50, 100, 150, 200, and 250. Each poisoning level represents the number of identities



(a) Recall Rates



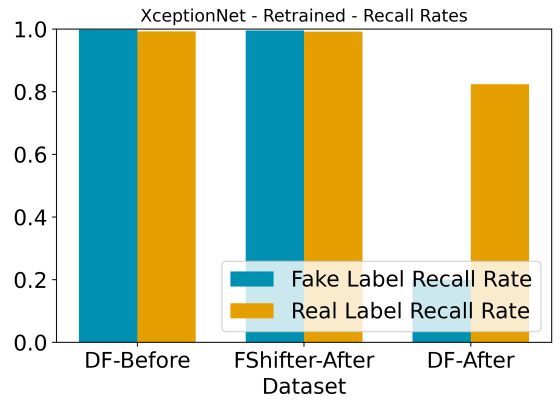
(b) Poison Success Rate

Figure 4.12: XceptionNet FaceShifter Average Recall Rates Given Number of Poisoned Identities

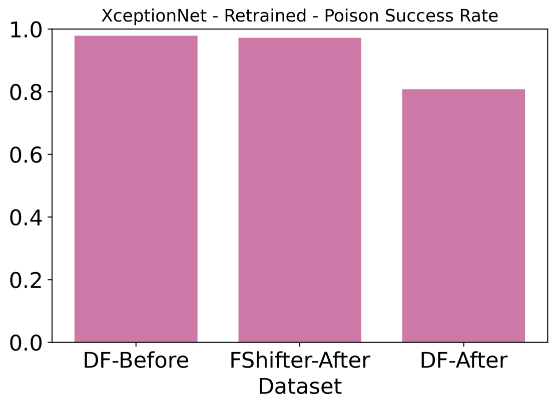
that are poisoned out of the 1,000 identities in the dataset. Each poisoning level can be converted to a percentage of the dataset that is poisoned, the percentages being: 5%, 10%, 15%, 20%, and 25%. As can be derived from the figure, there is a marginal difference of 0.0003248 and 0.0031811 between the $real$ and $fake$ recall rates, respectively, given 50 and 250 poisoned identities. From this very slight difference, we can say through our experimental results that the number of poisoned identities does not affect XceptionNet's recall rates. Our attack success rate has a very slight reliance on the number of poisoned identities. With more poisoned identities, it is slightly easier to perform our attack, with a 0.01189 poison success rate difference between the 250 and 50 poisoned identities cases.

4.3.2.4 Retrained Network

To comprehend XceptionNet's ability to retain information, we first train XceptionNet using one dataset, then retrain it using another. To accomplish this, we trained Xception using two different datasets over ten epochs. Figure 4.13 represents this experiment. XceptionNet was trained using the DeepFakes dataset for five epochs. The "DF-Before" set of bars describe this result. Next, XceptionNet was trained for



(a) Recall Rates



(b) Poison Success Rate

Figure 4.13: XceptionNet Retrained Average Recall Rates

five epochs on the FaceShifter dataset only. The “FShifter-After” set of bars describe the results of this second set of epochs. The DeepFakes dataset was then tested again after the network was trained on only the FaceShifter dataset for five epochs, which is represented by the “DF-After” set of bars. As can be seen by “FShifter-After”, the network can properly learn the new dataset just as well as our other experiments, including the poisoned data (at a 0.9724 poison success rate). Interestingly, “DF-After” shows that the network can properly recall the *real* data and the *poisoned* data to high fidelity, but forgets the $fake$ data. It should be noted that the poisoned targets for the DeepFakes and FaceShifter datasets were chosen at random and are not the same set. It should also be noted again that the class weights are equal, thus the fact that there are more *real* samples than $fake$ should not be the determining factor as to why the network is favoring the *real* partition.

4.4 DEFENSE STRATEGIES

Due to the prominent results of our proposed data poisoning attack, a mitigation technique is indispensable. When a synthetic media detector, such as XceptionNet, is deployed, it requires periodic updating as the incoming data matures. For example, as time progresses, image quality and different compressing techniques transpire. In addition, deepfakes are becoming more realistic at a staggering rate with advances in both hardware and software. Synthetic media detectors need to be updated to accommodate these improvements. To prevent poisoned data from negatively altering the detectors, an immutable discriminator is necessary. This discriminator will need to parse incoming images and determine if they are benign or malicious. If benign, then the images will be passed on to improve the detector. With periodic updates, the discriminator can work in tandem with the detector to better improve the synthetic media detector, similar to a generative adversarial network. Using the previous version of the detector as input, the discriminator can filter out images that the adversary has tainted with their data poisoning attack.

Thus, to detect our proposed label flipping data poisoning method, we have designed several defense strategies. Since our objective is to detect the poisoned attack, if the deepfake detector correctly classifies a sample as a deepfake, there is no need to further label that image as a poisoned sample. Our goal is to be able to detect when a sample marked as real by the deepfake detector is a poisoned fake sample. Thus, we only use samples marked as *real* by the deepfake detector (e.g. XceptionNet) to train, validate, and test our discriminators. All of our proposed defense mechanisms discussed in this session are tailored to XceptionNet but can be easily altered to be used with other synthetic media detectors.

4.4.1 Outlier Based Defense

Our simplest difference strategy uses an outlier-based method. Outlier detection is one of the quickest and most popular methods of attack detection [74]. Our output of XceptionNet for deepfake detection is a softmax layer followed by two neurons. The first neuron represents the probability of the input image being *real* while the second neuron represents the probability of the input image is a synthetic ($fake$) image. Due to the softmax layer, these probabilities sum to 1.

Our outlier detection method is based on how confident our deepfake detector is. Using only the samples marked as real by XceptionNet, we remove $n\%$ of samples using Equation 4.2, where P_r is the probability of a real sample and P_f is the probability of a fake sample.

$$\min(P_r - P_f) | P_r > P_f \tag{4.2}$$

A detailed procedure for using Equation 4.2 can be found in Algorithm 2.

Algorithm 2: Deepfake Data Poisoning Attack Outlier Detection

```
Input: $n$ number of samples to remove, List of $P_r$ and $P_f$
Output: List of Indices to remove $I_r$
$I_{tmp} = \{\}$
$I_r = \{\}$
// Find indexes of samples marked as real
for $index \in P$ do
    if $P_r[index] > P_f[index]$ then
        Insert $index$ into $I_{tmp}$
    end
end
// Only include samples with the lowest real probability
Sort $I_{tmp}$ from least to greatest using $P_r[index] - P_f[index]$
for $i \leftarrow 0$ to $n$ do
    Insert $i$ into $I_r$
end
return $I_r$
```

4.4.2 Machine Learning Based Defense

Our second proposed defense strategy uses a machine learning-based approach. Specifically, we explored two different machine learning algorithms: Support Vector Machines (SVM) and random forests. As depicted in Figure 4.3b, near the end of the exit flow, the convolutional layers reduce in dimensionality down to a single 2048-dimensional vector. This vector is then used as input by the softmax layer to calculate the final probabilities, thus this layer contains the most information related to the final deepfake classification. In addition, [25] found that the last layer of a neural network is the most tainted layer from adversarial attacks. The output for our machine learning methods is the probability that the input sample was poisoned. Any sample with a poisoned probability beyond a threshold is marked as *poisoned*.

4.4.3 Deep Neural Network Based Defense

Similar to our machine learning-based defense outlined in Section 4.4.2, we employed a deep learning-based approach that utilizes the 2048-dimensional feature vector generated near the end of the exit flow in XceptionNet. Figure 4.14 outlines our model. We apply an approach similar to a convolutional neural network by gradually using smaller layers. We use a total of ten hidden layers, with each layer decreasing the number of neurons per layer by a power of two. Each hidden layer is a dense layer using batch normalization and the Rectified Linear Unit (ReLU) activation function. The final layer is a softmax layer with two neurons. The first neuron represents the probability of the input image being *benign* while the second neuron represents the probability of the input image is a *poisoned* sample.

4.4.4 Convolutional Neural Network Based Defense

Our final proposed defense mechanism is a CNN-based approach. Figure 4.15 outlines our classification pipeline. First, images are used as input to XceptionNet, as is performed during the deepfake detection process. We retain the feature maps present at the end of the entry flow in XceptionNet to be used as input to the discriminator. Our CNN-based discriminator then outputs the probability of the sample being a poisoned image.

Through an empirical study, we have found that the feature maps generated after the entry flow yield the greatest results for our CNN-based discriminator. There are

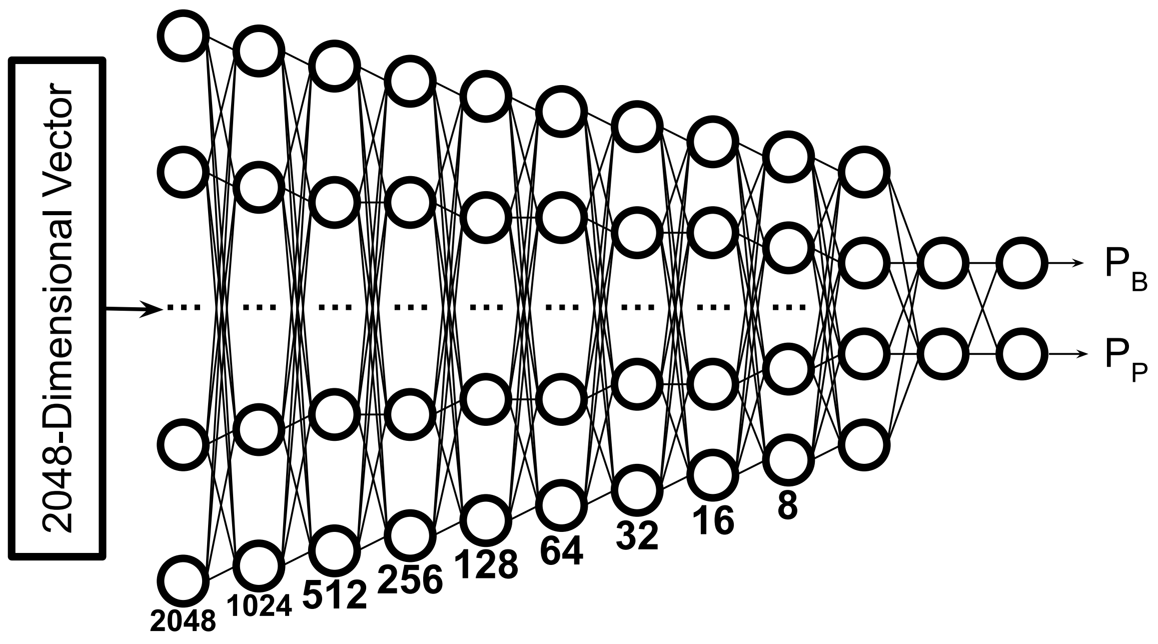


Figure 4.14: Deep Neural Network Based Discriminator

multiple possible reasons for this. First, the earlier layers of a CNN contain higher-level features (such as the positioning of edges) compared to later layers which contain more abstract features [87]. We believe that due to artifacts, these earlier layers are well designed to find deepfakes and thus the majority of our poisoned images.

4.5 DEFENSE PERFORMANCE STUDY

In this section, we present the experiments comparing each of our discriminator's effectiveness in determining poisoned samples.

4.5.1 Experimental Settings

All experiments were conducted on a desktop computer with an Intel i9 processor with 10 cores and 20 threads @3.7GHz and 128 GBs of RAM. Experiments involving neural networks used an NVIDIA GeForce RTX 2080 Ti graphics card with 11 GBs of GDDR6 memory. Due to the size of our datasets, the entire dataset could not be loaded into memory, thus a generator was required to load data into memory in

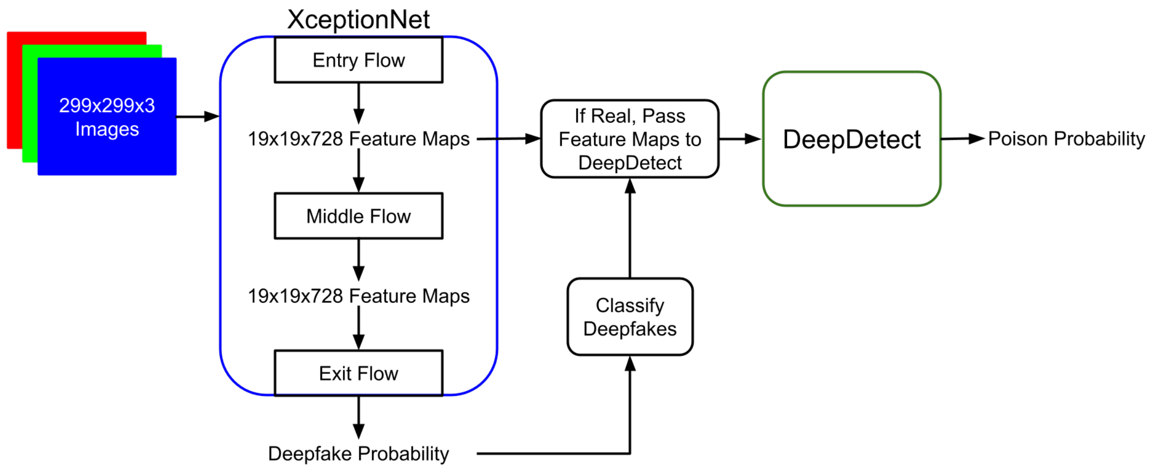


Figure 4.15: CNN-Based Discriminator Pipeline

batches. This was our only major bottleneck in terms of training time.

We used each dataset from Section 4.1.3 and the corresponding trained XceptionNet network from Section 4.3.2 to train and test our discriminators. We used the same training, validation, and testing splits for our discriminators as used by our XceptionNet networks as described in Section 4.3.1. Only the samples classified as *real* by XceptionNet are passed to our discriminators for training, validation, and testing. To compensate for imbalanced classes, each dataset was trimmed such that there were the same number of *benign* and *poisoned* samples for each training, validation, and testing partition.

The effectiveness of each discriminator is evaluated using two metrics (i) *poison* detection recall rate and (ii) *benign* recall rate. In the following definitions, TP stands for “true positive”, FP stands for “false positive”, TN stands for “true negative”, and FN stands for “false negative”.

Definition 4.5.1 (Poison Detection Recall Rate)**.** The poison detection recall rate measures the percentage of correctly identified *poisoned* photos against the total number of *poisoned* photos.

$$PoisonDetectionRecallRate = \frac{\text{Correctly Identified Poisoned Photos}}{\text{All Poisoned Photos}} = \frac{TP}{TP + FN}$$

Definition 4.5.2 (Benign Recall Rate)**.** The benign recall rate measures the percentage of correctly identified *benign* (not *poisoned*) photos against the total number of *benign* photos.

$$BenignRecallRate = \frac{\text{Correctly Identified Benign Photos}}{\text{All Benign Photos}} = \frac{TN}{TN + TP}$$

The remainder of Section 4.5.1 describes the specific configuration for each discriminator.

4.5.1.1 Outlier Based Defense

For our outlier-based defense strategy, we set n to 20, thus 20% of samples that XceptionNet was least confident was *real* was removed.

4.5.1.2 Support Vector Machine Discriminator

We used the sklearn SVC version of the SVM algorithm from scikit learn [122]. The radial basis function kernel was used to transform the data before generating the hyperplane.

4.5.1.3 Random Forest Discriminator

We used the random forest classifier library from scikit learn for our random forest discriminator [123]. Each random forest discriminator consisted of 100 trees with no max depth limit. The maximum number of features per tree was set to 45. Gini impurity was used to determine the best split for each decision tree.

4.5.1.4 DNN Based Discriminator

For our DNN-based discriminator, during training, a dropout rate of 0.4 was used. We utilized the Adam optimization algorithm with a learning rate of 0.0002, β_1 of 0.9, β_2 of 0.999, ϵ of 10^{-8}, and zero decay. Categorical cross-entropy was used as our loss function. Each network is trained for a total of 5 epochs, keeping only the epoch that generated the lowest validation loss. Each batch contained 32 samples.

4.5.1.5 CNN Based Discriminator

Our CNN-based discriminator is trained using the same approach as our DNN-based discriminator.

4.5.2 Experimental Results

Unless otherwise noted, the results described in this section consist of models trained using three XceptionNet networks and tested on four other XceptionNet networks. We wanted our discriminators to be diverse enough to work on a multitude of different deepfake detectors with each detector trained using different data. By utilizing multiple trained deepfake detectors to train each discriminator, a discriminator learns to recognize a broader and more diverse set of poisoned images. By validating and testing on detector networks that the discriminator was not trained upon, we show that our discriminators are robust enough to maintain a high recall rate despite changing conditions.

4.5.2.1 Outlier Based Defense Results

Since outlier detection does not require training data, the average result of applying outlier detection to all seven XceptionNet networks is displayed in Figure 4.16. The

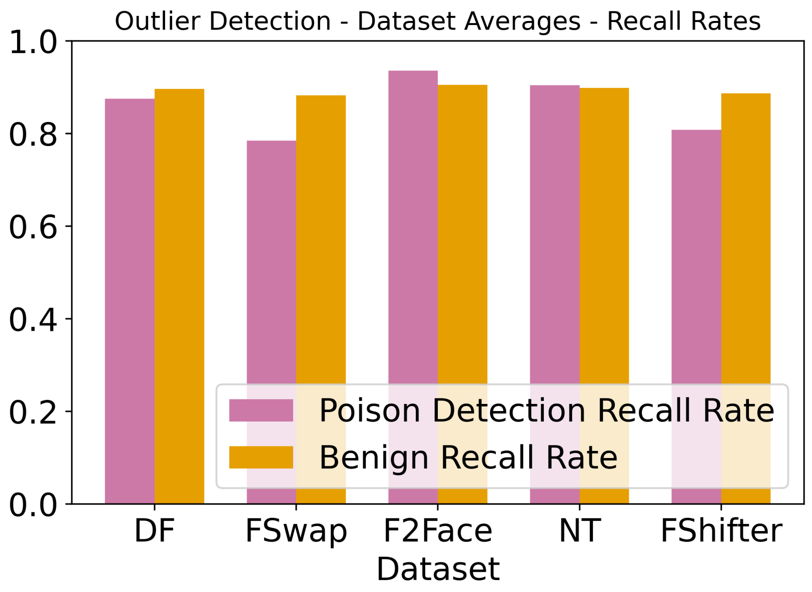


Figure 4.16: Outlier Detection Average Recall Rates

mean poison detection rate and benign recall rate across all datasets were 0.8620 and 0.8941, respectively. Considering no training data is required for outlier detection, these are admirable results, however, there was a high degree of false positives. The average precision was 0.5362. In general outlier detection performed better on the facial reenactment datasets than the identity swapping datasets.

Our outlier detection method is our simplest and possibly fastest method. It was also able to achieve satisfactorily, but far from optimal results. As our current detector stands, it is equally as likely to remove benign images as it is to keep poisoned images. Including the fact that it was our worst-performing discriminator, outlier detection has an even larger flaw. Assuming the detector had a perfect recall rate, the user would still require knowing how many poisoned images there are in the dataset. Given a constant stream of data, a security provider would be unable to use this detector. To make it functional, the security provider would have to group the images into batches and tag the $n\%$ least confident samples as possibly being poisoned and require manual review later. This would be an arduous effort requiring much human interaction. The

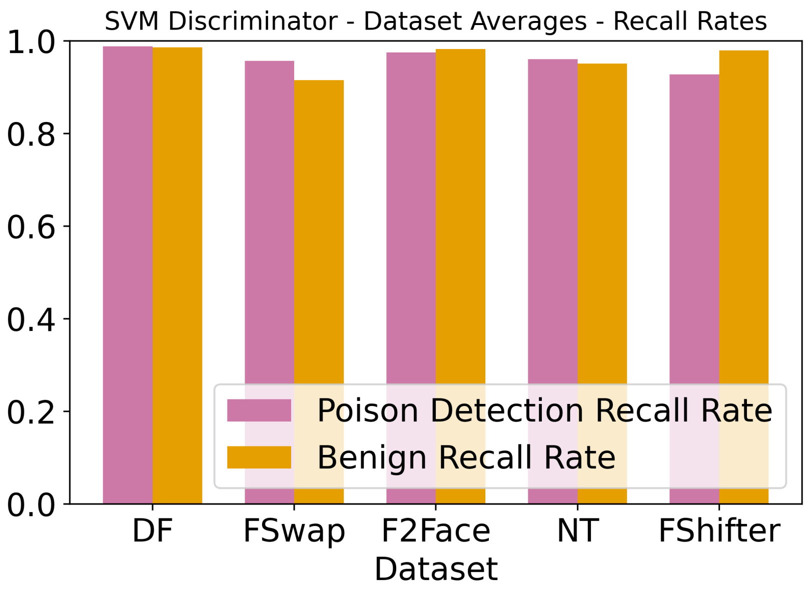


Figure 4.17: SVM Discriminator Average Recall Rates

adversary would be able to work around this criteria by poisoning an entire batch of images. If only the lowest $n\%$ are removed, then the other $(1 - n)\%$ would still be accepted.

4.5.2.2 Support Vector Machine Discriminator Results

Figure 4.17 highlights the results from the SVM discriminator. The mean poison detection rate and benign recall rate across all datasets were 0.9619 and 0.9632, respectively. Our SVM discriminator has a low false-positive rate, with an average precision of 0.9639. Both identity swapping and facial reenactment datasets perform equally as well.

4.5.2.3 Random Forest Discriminator Results

The results of the random forest discriminator are shown in Figure 4.18. Our random forest discriminator performs similarly as well to our SVM discriminator, with an average poison detection recall rate of 0.9404 and an average benign recall rate of 0.9816

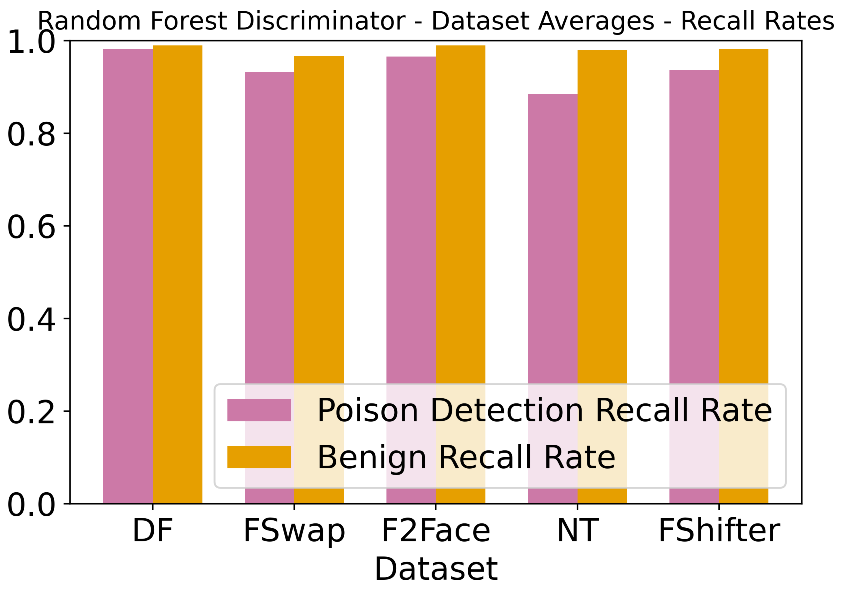


Figure 4.18: Random Forest Discriminator Average Recall Rates

across all datasets. The random forest discriminator precision was better than the SVM discriminator, with a precision of 0.9809. Compared to the SVM discriminator, our random forest discriminator tended to favor classifying benign samples over poisoned samples. In a scenario where a low false-positive rate is required, this would be the preferred discriminator among the two, however, in most security applications a high poison recall rate is more important than a high precision. On the other hand, a random forest classifier can classify images much quicker since each decision tree can be executed in parallel.

4.5.2.4 DNN Discriminator Results

Figure 4.19 presents our results for our DNN-based discriminator. The average poison detection recall rate was 0.9574 and the average benign recall rate was 0.9441 across all datasets. With an average precision of 0.9483, our DNN-based discriminator had the lowest precision out of all the machine learning-based discriminators. Unlike the random-forest discriminator, our DNN-based discriminator did not generally tend to

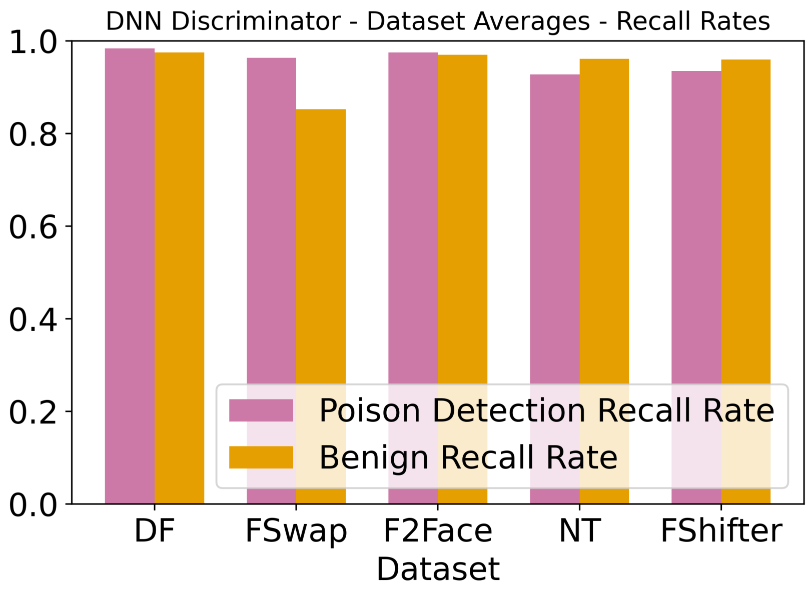


Figure 4.19: DNN Discriminator Average Recall Rates

favor one classification class over another, with both recall rates being near equal in most cases.

4.5.2.5 CNN Discriminator Results

Due to the success of our CNN-based discriminator, we conduct a variety of experiments to verify the robustness of this method. We test the multiple different detector configurations described in Section 4.3 using our CNN-based discriminator.

Single Synthetic Media Type Detection Networks Our CNN-based discriminator outperformed our other discriminators in every metric measured with an average poison detection recall rate of 0.9859, an average benign recall rate of 0.9937, and an average precision of 0.9937. The individual dataset results are shown in Figure 4.20. The highest poison detection recall rate was 0.9936 and the highest benign recall rate was 0.9963, both achieved by the DeepFakes dataset.

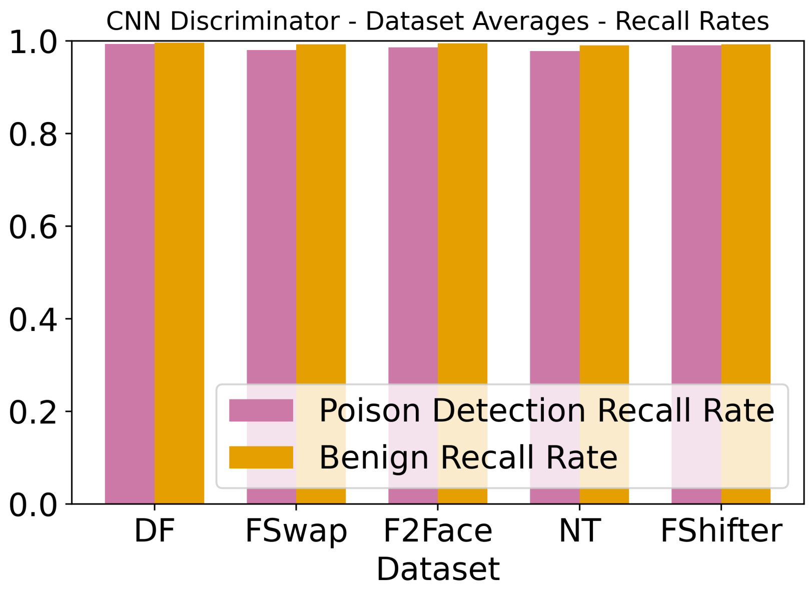


Figure 4.20: CNN Discriminator Average Recall Rates

Percentage of Poisoned Labels The discriminator results for the corresponding section in 4.3.2.3 are given in Figure 4.21. For each discriminator, the number of poisoned identities varied. These results show that the discriminator's recall rates are not directly dependent on the number of poisoned identities. Only varying the number of poisoned identities for the FaceShifter dataset is reported. These results, however, are closely related to the average results for the CNN-based discriminator on multiple datasets. The average poison detection recall rate given the varying number of poisoned identities was 0.9844, the average benign recall rate was 0.9955 and the average precision was 0.9955.

Information Retention One of the goals of our discriminator is to prevent deepfake detectors from being trained with poisoned data. To accomplish this, an immutable discriminator that has been trained on known poisoned data is necessary. The discriminator must function over multiple updates of the deepfake detector. To verify the plausibility of this system, we conducted the following experiment. First,

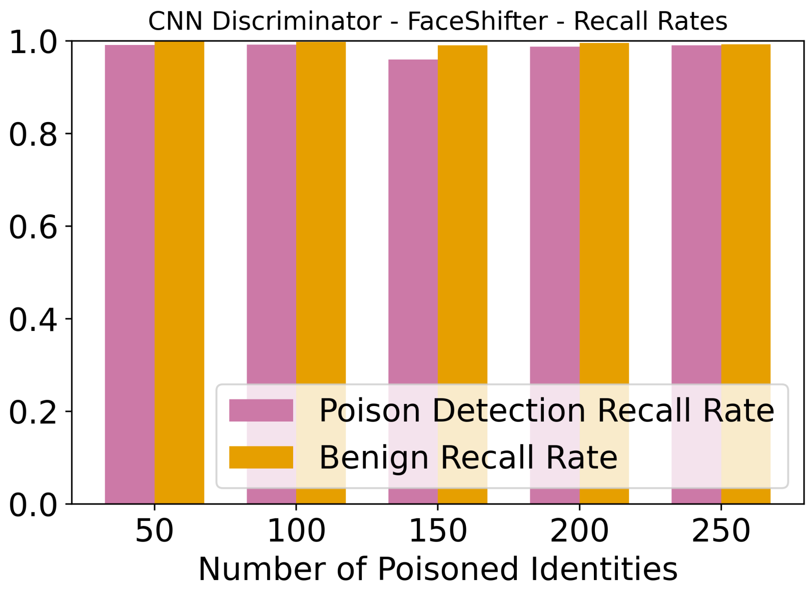
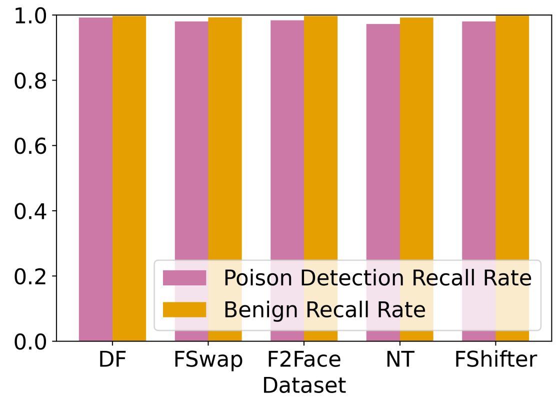
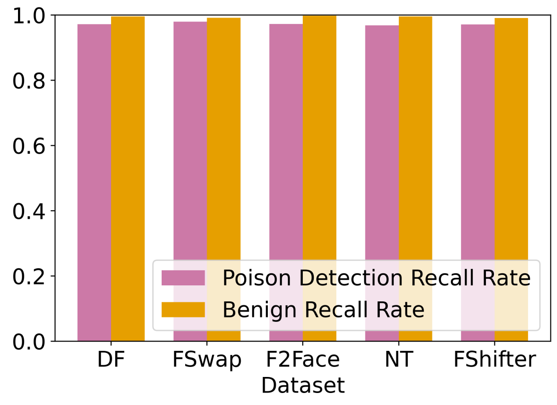


Figure 4.21: CNN Discriminator Average Recall Rates for FaceShifter Dataset Varying the Number of Poisoned Identities

we trained XceptionNet with half of the identities in a dataset, i.e., we split each dataset into two sets with 500 identities each with no overlap in identities. Both of these splits will contain poisoned identities. Let us call this trained network *Detector-A*. Second, we trained our CNN-based discriminator using the data generated from *Detector-A*. We will call this trained CNN-based discriminator *Disc-A*. Third, we update *Detector-B* by training it with the full dataset, i.e. training it with all 1,000 identities. This represents when a new type of synthetic image has been released and needs detection or when current synthetic image generation methods improve. We'll name this updated XceptionNet *Detector-B*. Finally, *Disc-A* is used as a discriminator to detect the poisoned samples used to train *Detector-B*. The results from the final phase of this experiment show how much information XceptionNet retrains between data poisoning attacks. As long as the recall rates do not decrease drastically, our proposed system can be used to help prevent deepfake detectors from being corrupted by label flipping data poisoning attacks.



(a) CNN-based discriminator trained and tested using half of identities in a dataset



(b) CNN-based discriminator tested using the full dataset

Figure 4.22: Effectiveness of Discriminator when varying the testing datasets

Figure 4.22 demonstrates the results of *Disc-A*. In Figure 4.22a, the results of *Disc-A* being trained on *Detector-A* are shown. Across all datasets, *Disc-A* had an average poison detection recall rate of 0.9819 and an average benign recall rate of 0.9954. Next, Figure 4.22b gives the results from *Disc-A* being verified using the data generated by *Detector-B*. An average poison detection recall rate of 0.9727 and an average benign recall rate of 0.9944 was achieved across all datasets. This resulted in a 0.9417% decrease in average poison detection rate after XceptionNet was updated using new poisoned data. With this very minor decrease of less than 1%, we can reliably say that our discriminator retrains the ability to detect the label flipping data poisoning attack across an update of the deepfake detector as long as the data follows the same distribution. There is an even smaller decrease of 0.1003% in the benign recall rate, allowing the defense group to not be overly concerned about false positives over time.

The corresponding data poisoning results for *Detector-A* and *Detector-B* can be found in Appendix B.3.

4.5.2.6 Classification Time Comparison

Our two machine learning methods (SVM and random forest) performed similarly as well, however, our random-forest discriminator has more advantages than our SVM discriminator. Given the parallel nature of a random forest-based method, classification can occur much quicker compared to a one-versus-all hyperplane-like method that SVM uses. Decision trees are one of the quickest machine learning methods available, only requiring binary decisions with a max number of binary decisions being the height of the tree. With a random-forest discriminator, we take this advantage a step further, by training many trees under a decreased feature set. This allows the total height of a tree to be diminished, thus decreasing the time required for classification.

Compared to our CNN-based discriminator, a random forest discriminator is much quicker to train. However, these two discriminators use data from different points in the detector. Our random-forest discriminator uses the 2048-dimensional feature vector found at the very end of the neural network, just before the final soft-max layer. Our CNN-based discriminator uses data from a considerably earlier point, at the end of the entry-flow. Given the hardware, our CNN-based discriminator can run in parallel to XceptionNet, thus allowing poison detection to occur nearly simultaneously with synthetic image detection. Thus, as long as the hardware is available, our CNN-based discriminator is not only faster than all of our other discriminators but also produces near-optimal results.

4.5.2.7 Synthetic Media Generation Method

Our label flipping attack had an average poison success rate of 0.9803 with the identity swapping datasets and an average poison success rate of 0.9870 with the facial reenactment datasets. With this minute difference, we can assume that the attack is equally successful on identity swapping and facial reenactment media. Our primary discriminator, the CNN-based method, achieved an average poison detection recall rate of 0.9883 on the identity swapping datasets and an average poison detection recall rate of 0.9823 on the facial reenactment datasets. Our CNN-based discriminator outperformed all of our other discriminators in both of these metrics. This indicates that our CNN-based discriminator is our best all-round discriminator independent of the type of synthetic media generation tool used.

4.6 CONCLUSION

In this chapter, we proposed a label flipping data poisoning attack against synthetic media detectors. This attack only requires access to the label portion of the training dataset. We applied this attack to XceptionNet, a thoroughly researched deepfake detector. This attack accomplished a poison success rate of 0.9888 when a single dataset was poisoned. We showed that our attack was successful under a variety of circumstances, such as diversifying the types of synthetic media within the dataset or varying the number of poisoned identities in a dataset. To the best of our knowledge, this is the first label flipping data poisoning attack against deepfake detectors, thus, we also present several discriminators to help mitigate this attack. We compare and contrast the pros and cons of each discriminator. Overall, our CNN-based discriminator is the most alluring of our poison detection methods. Not only does it achieve the highest poison detection and benign recall rates, at 0.9936 and 0.9963, respectively, but we also proposed a pipeline to reduce the additional detection time to near zero. Our extensive experimental results demonstrated the potency of our proposed attack method. Fortunately, our CNN-based discriminator is also highly effective at rooting out the attack, thus protecting deepfake detectors during necessary updates.

Chapter 5

CONCLUSION

With the improvements made in facial authentication in recent years, it is increasingly being used as a login method to web services and other applications. With the rise of facial authentication, adversaries will desire to take advantage of different aspects of the system. We seek to improve facial authentication for the benefit of the consumer.

To demonstrate a possible attack avenue, in this dissertation, we introduced a novel data poisoning attack, called *replacement data poisoning*. We applied this attack against the stat-of-the-art facial recognition framework FaceNet. We showed how this attack can allow an adversary to log into a target's account by replacing a few photos at training time.

To combat our attack, we developed a two-phase DNN based detector, DEFEAT. Our DEFEAT architecture was able to achieve over 90% accuracy defending against our proposed attack. We extensively compared our proposed DEFEAT architecture against other possible detectors.

This dissertation also encompasses another facial recognition task. Deepfakes are a malicious type of media meant to defame or otherwise harm someone publicly. Deepfake detectors exist to diminish the possibility of these damaging images going unchallenged. Deepfake detectors exist to catch the bad, but they themselves can be corrupted. To help mitigate this possibility, we designed a viable data poisoning attack method against deepfake detectors and then showed how to defend against such an attack.

In more detail, we proposed a label flipping data poisoning attack against deepfake detectors. To the best of our knowledge, this is the first label flipping data poisoning attack conducted on deepfake detectors. We demonstrated how devastating this type of attack is to deepfake detectors with a poison success rate of 0.9888 when a single dataset was poisoned. We also showed the implications of our attack under a variety of circumstances, such as diversifying the types of synthetic media that the deepfake detector could detect.

To mitigate this concerning attack, we designed several different discriminator methods. Our most advanced method, a CNN-based approach, achieves a poison detection recall rate of 0.9936 and a benign recall rate of 0.9963. Our proposed technique adds nearly zero additional detection time to the deepfake detector.

Chapter A

DEFEAT

A.1 DATASETS

This section demonstrates samples from the datasets used in Section 3.4.

A.1.1 FEI Dataset

Figure A.1 demonstrates an example class from the FEI dataset [92]. Each class consists of 14 images with the person facing in different directions. The 14th image is taken in dark lighting. Since this represents our ideal office setting, it is discarded. There are a total of 200 classes in this dataset.

A.1.2 LFW Dataset

Figure A.2 demonstrates examples from various classes from the LFW dataset [91]. Each class consists of a various number of photos taken with various lighting conditions in a variety of scenarios. For example, a large number of photos from this dataset consist of politicians from around the world. Another major category is tennis players during gameplay. For our experiments, we only considered classes with at least 10 photos. Prior to pruning, there are 5,749 different classes. After pruning, 158 classes remained.

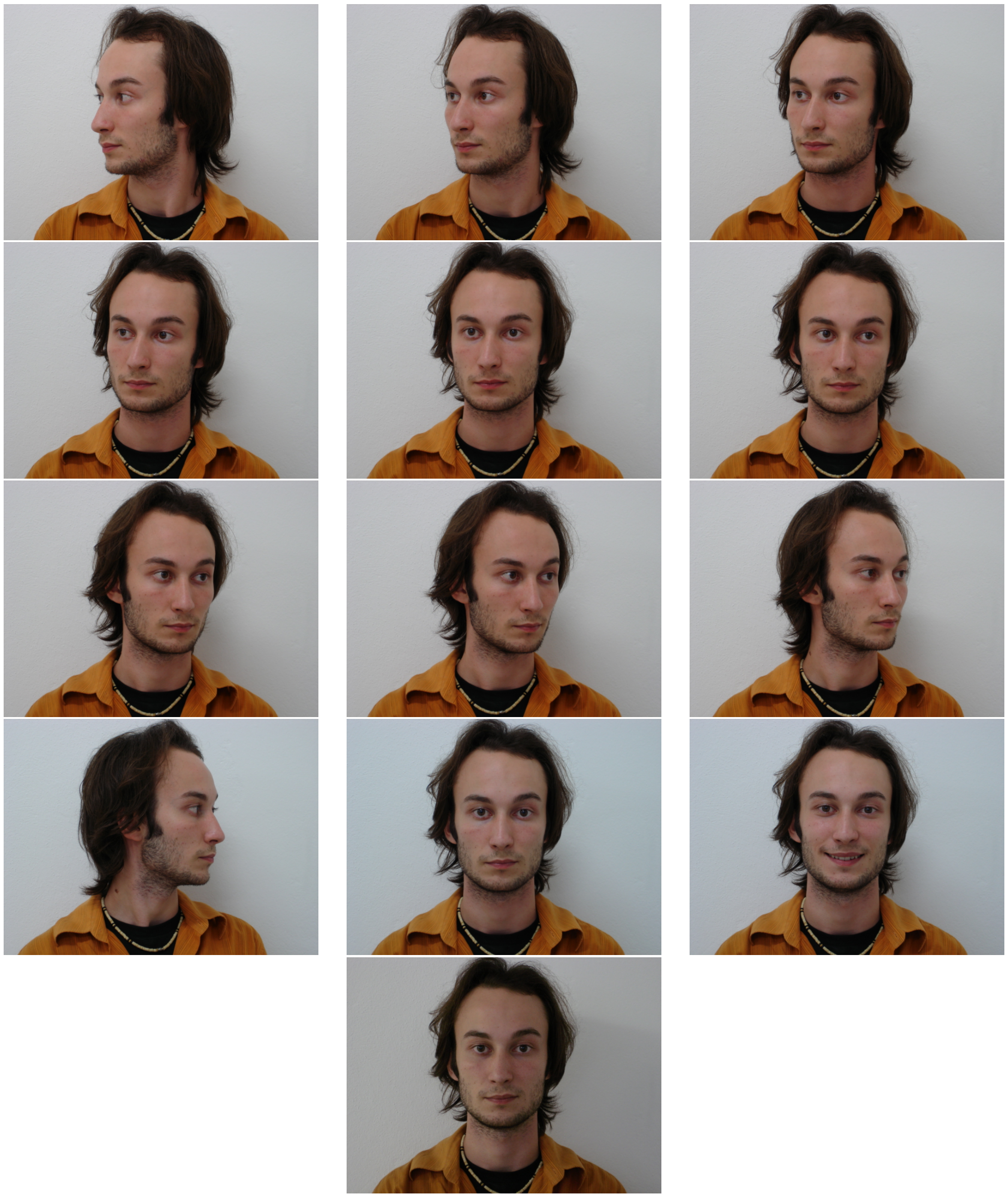


Figure A.1: Example of a single class in the FEI dataset

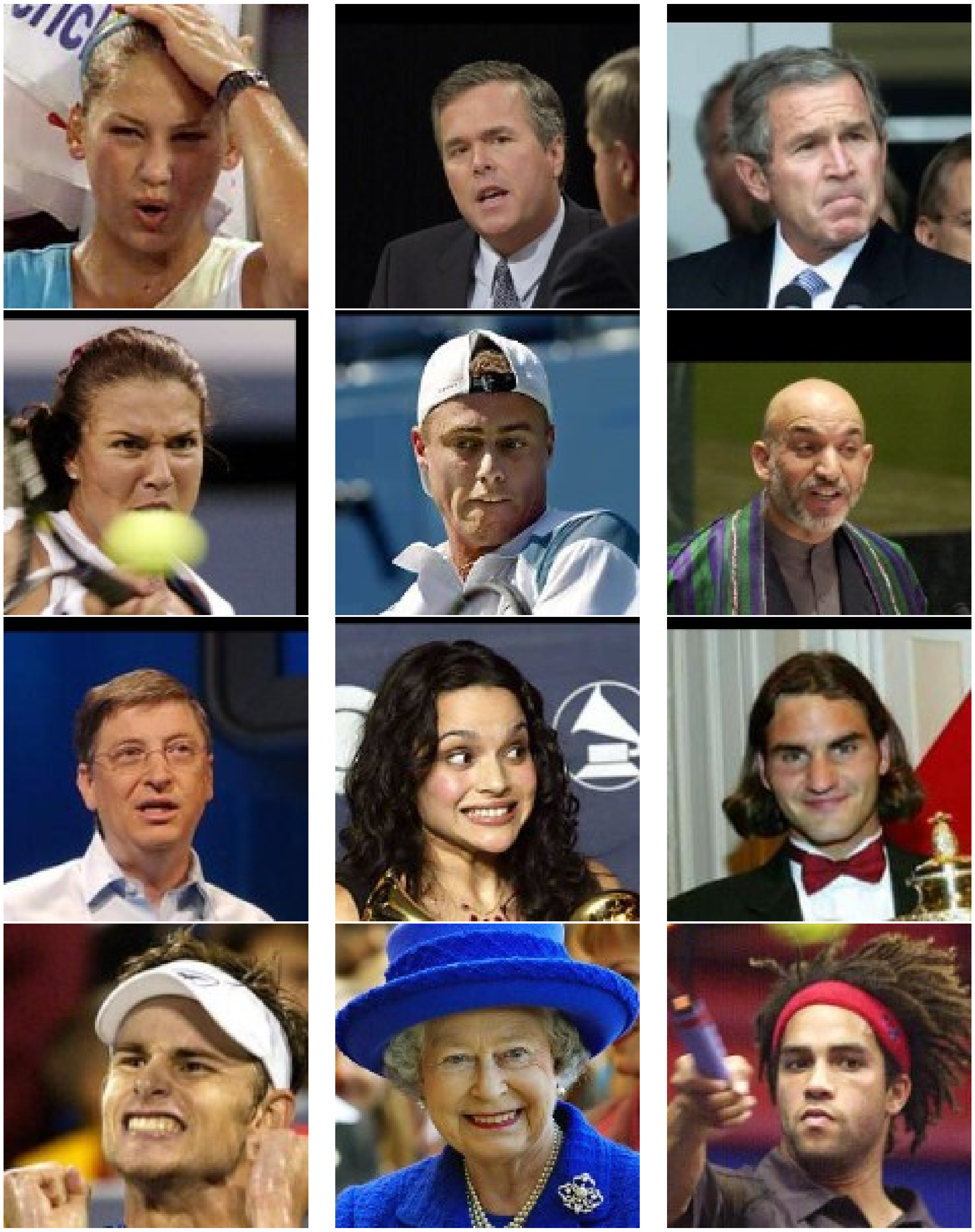


Figure A.2: Examples from various classes in the LFW dataset

A.2 STATISTICS-BASED DISCRIMINATOR

A.2.1 Principle Component Analysis

In Section 3.3.2, we explored reducing the dimensionality of FaceNet's facial embedding from 128 dimensions down to 2 dimensions to visualize the clustering. We reduced the dimensionality via Principal Component Analysis (PCA). In this section, we explore some additional plots that we found interesting.

A.2.1.1 Random Attack Strategy

By applying the random attack strategy on the LFW dataset, we obtained Figure A.3. Nine PCA plots were generated, where each plot consisted of five random users, three from the pristine group, two from the target group. The corresponding attacker for each target is also plotted. From these nine plots, six plots were chosen to present here by their ability to demonstrate various considerations we had to make when choosing our defense strategies. Figure A.3a highlights how the targeted and attacked classes can intermingle, as demonstrated by users 4 and 5. When the attacker and target embeddings cluster, the difficulty of using FaceNet's embedding increases tremendously. Our goal with using a deep neural network with DEFEAT is to be able to use the subtle differences between embeddings for detection. Figure A.3b demonstrates an extreme version of target-attacker intermingling. User 5 in this example has nearly overlapping target-attacker embeddings. Figure A.3c forms four separate clusters, with two users target and attack sets becoming intermingled. User intermingling in addition to target-attacker intermingling adds additional complexity for the detector to discern. Figure A.3d takes this clustering approach further by only having 3 clusters for 5 users. Users 3, 4, and 5 are entangled with one another. Although this cluster is quite wide, a clustering method such as DBSCAN would likely consider this a single cluster under many configurations. Figure A.3e shows a close to ideal case where each user and target-attacker pair are separated in space. Finally, Figure A.3f demonstrates how genuine users can become intermingled without their attacker counterparts.

A.2.1.2 Optimal Attack Strategy

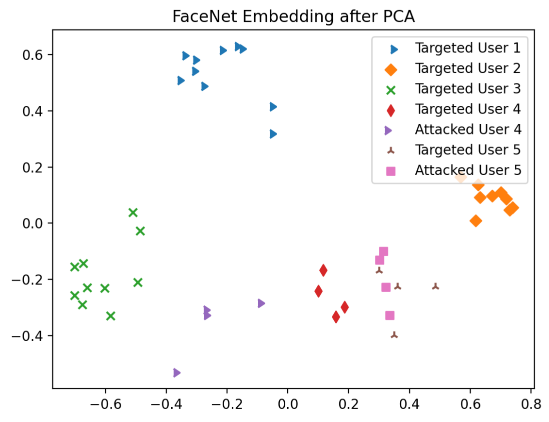
Figure A.4 was generated by applying the optimal attack strategy on the LFW dataset using the same method described above. Since the goal of the optimal attack strategy is to bring together similar-looking people into target-attack pairs, clusters should be easier to form compared to the random attack strategy. This does appear to be true, as demonstrated by figure A.4a, where 3 clusters are formed, with one cluster being large and tight-knit compared to Figure A.3d. In Figure A.4b, user 4 forms an elongated cluster. In addition, user 3 has embeddings that contain outliers that are incorporated into user 5's attacked cluster and user 4's cluster. With outliers for a pristine user, the facial authentication model is likely to be less accurate. Our discriminator must also account for such possibilities. Figure A.4c demonstrates that separate clusters can form even in our ideal attack scenario. Finally, Figure A.4d highlights that separate users can cluster in addition to target-attack pairs.

A.3 DEFEAT VS STATISTICS-BASED DISCRIMINATOR

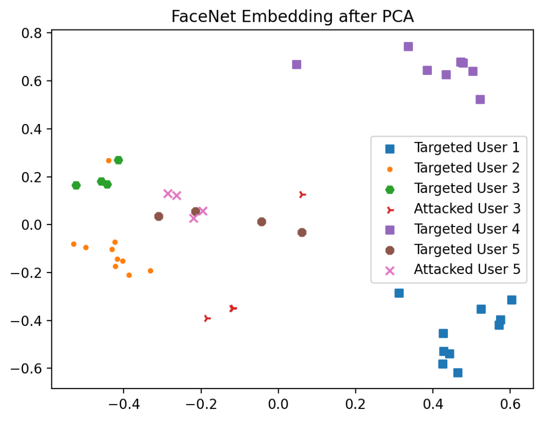
In this section, we present some additional findings comparing DEFEAT to our statistics-based discriminator, including use cases where the statistics-based discriminator is superior to DEFEAT.

A.3.1 FEI Results

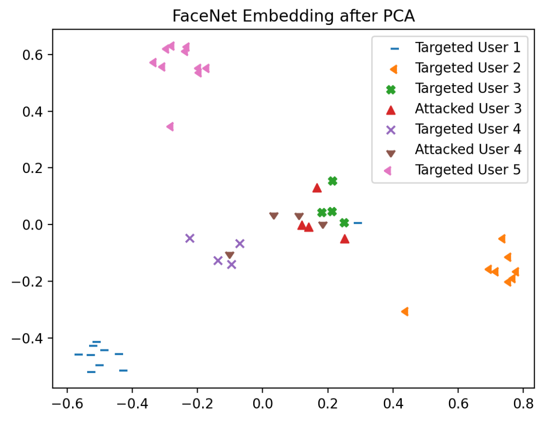
In this section, the corresponding results from Section 3.4 using the FEI dataset are presented. The LFW dataset represents the more difficult, real-world scenario, dataset, thus the FEI dataset was saved for the appendix.



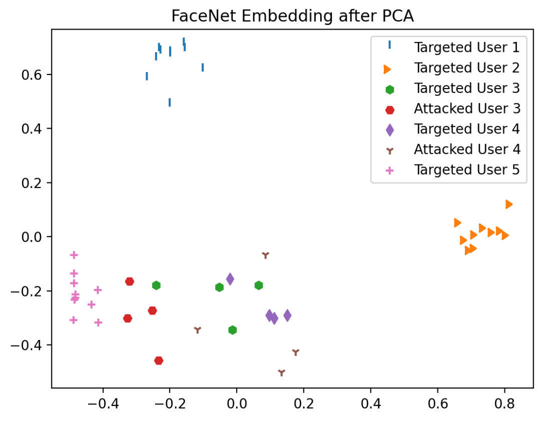
(a) Target-Attack Intermingling



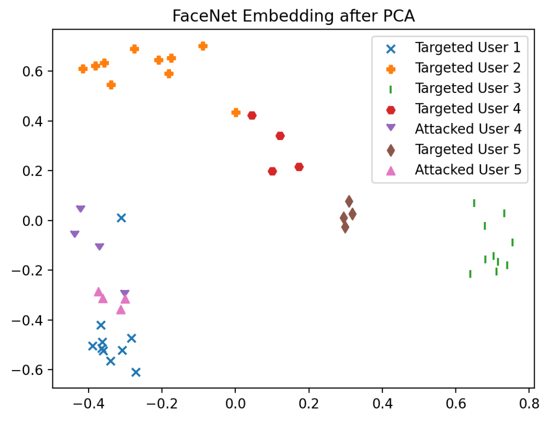
(b) Target-Attack Overlapping



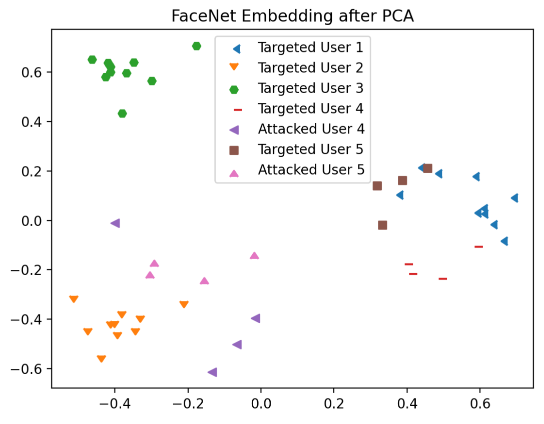
(c) Four Clusters



(d) Three Clusters

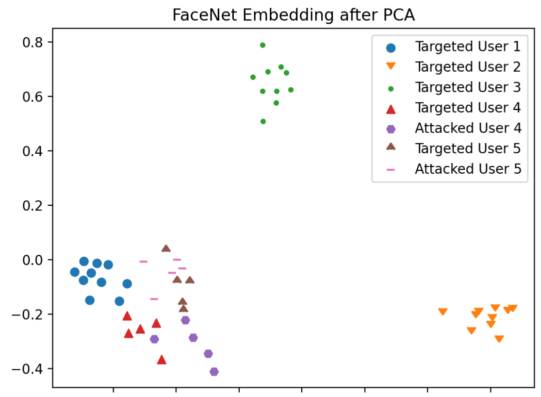


(e) Separated Clustering

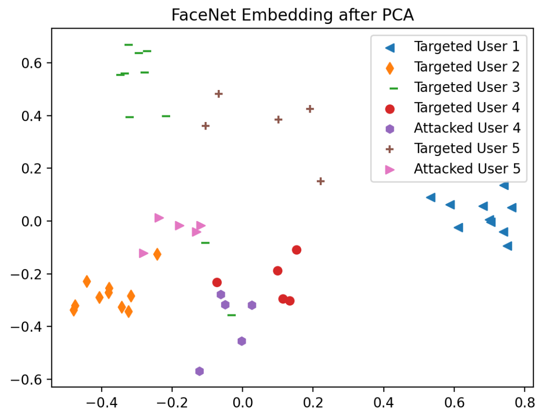


(f) Combined Intermingling

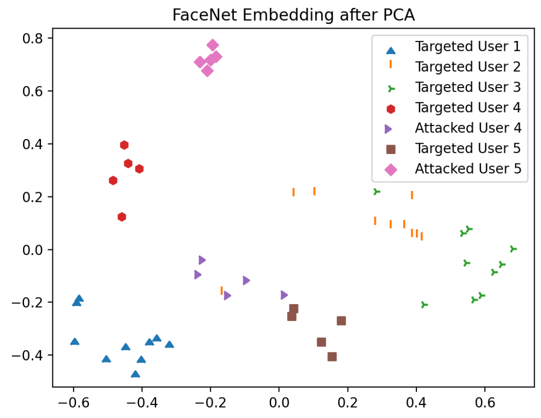
Figure A.3: FaceNet Embeddings PCA Plots Using the Random Attack Strategy



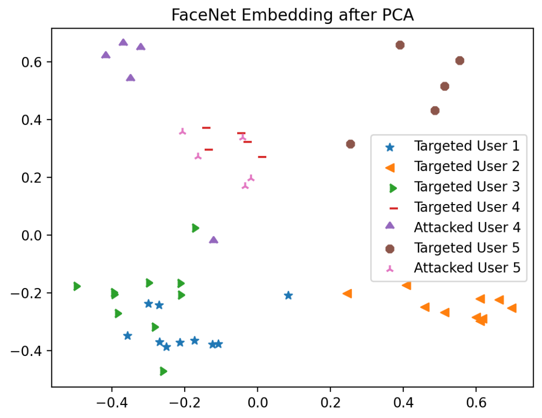
(a) Large Cluster



(b) Outliers

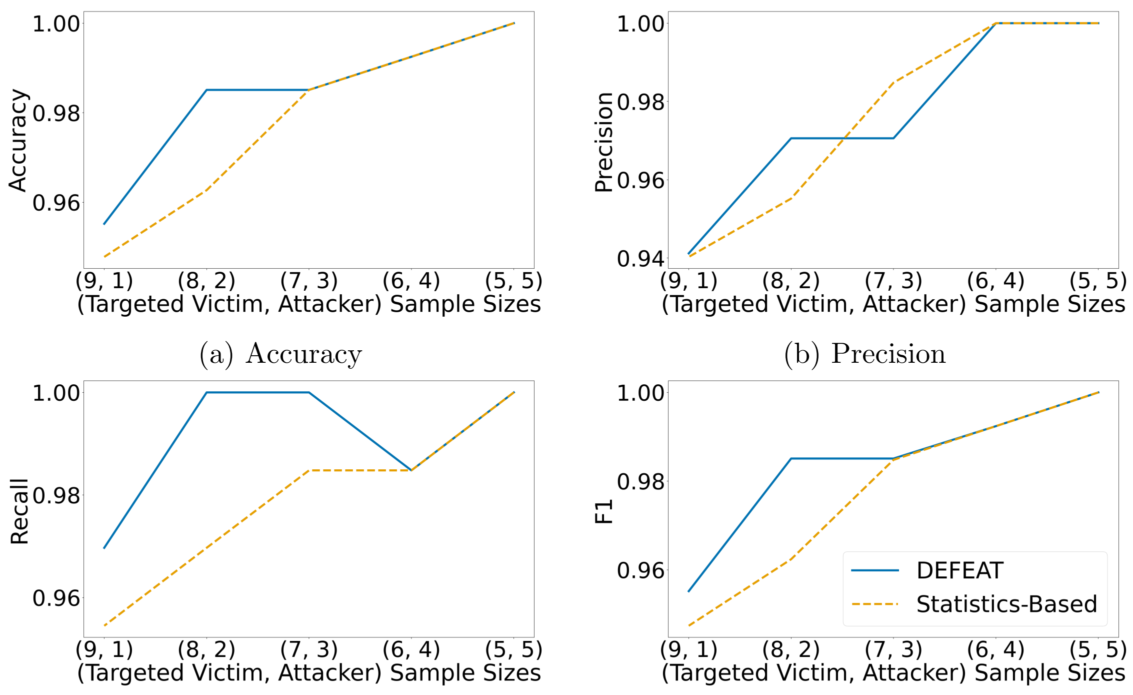


(c) Separate Clusters



(d) Targets Clustering

Figure A.4: FaceNet Embeddings PCA Plots Using the Optimal Attack Strategy

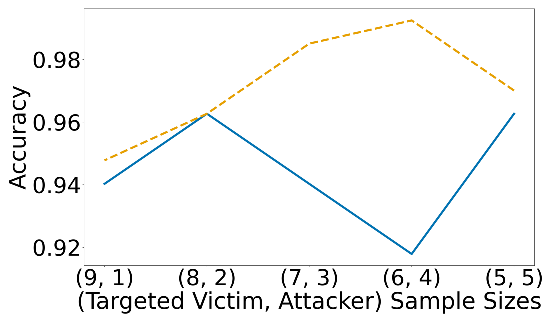


(a) Accuracy (b) Precision (c) Recall (d) F1

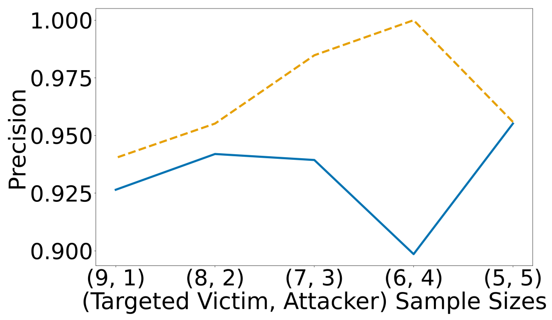
Figure A.5: Random Attack on the FEI Dataset - Varying the Number of Injected Photos - Two Inputs

A.3.1.1 Effect of the Number of Injected Photos - FEI

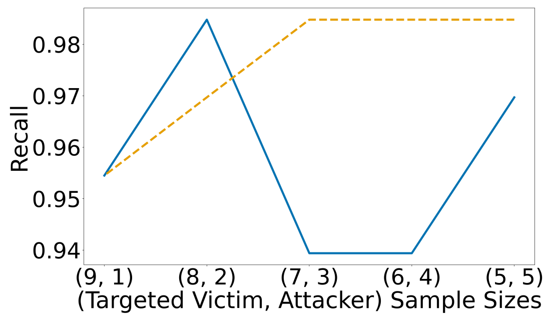
Figure A.5 and Figure A.6 highlights the random and optimal attack strategies on the FEI dataset, respectively. Given the random attack strategy, both discriminators performed nearly equally. Given the optimal attack strategy, however, the statistics-based discriminator outperforms DEFEAT. This is in stark contrast to the results presented in Section 3.4, where DEFEAT has far superior results on the LFW dataset given the optimal attack strategy. Although DEFEAT still performed well ($> 90\%$ in all cases), it is better able to discern when embeddings from difficult settings are used compared to the simplistic FEI dataset setting. The variety of lighting and background conditions highlights deep learning's ability to glean insightful information given difficult situations; however, in a simple setting, the raw statistics-based classifier appears to be better.



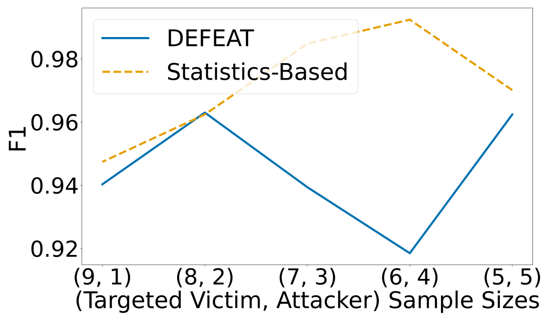
(a) Accuracy



(b) Precision



(c) Recall



(d) F1

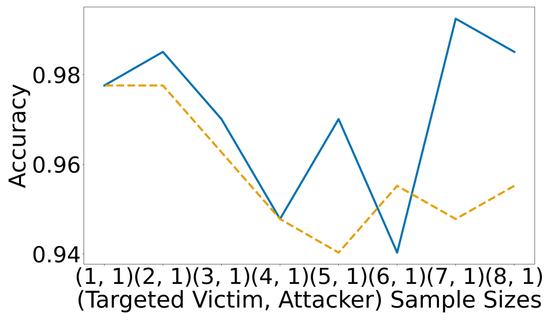
Figure A.6: Optimal Attack on the FEI Dataset - Varying the Number of Injected Photos - Two Inputs

A.3.1.2 Effect of Number of Training Photos per New User - FEI

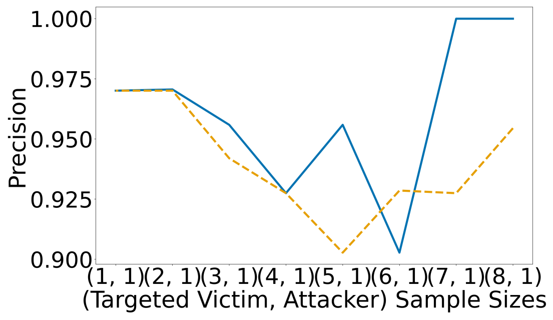
Figure A.7 and Figure A.8 showcases our random and optimal attack strategies on the FEI dataset given a constant *attacking* size partition and varying *targeted* size partition, respectively. In both attack scenarios, DEFEAT generally outperforms the statistics-based classifier, however, both classifiers performs nearly equally. These restuls nearly mirror the results found in Section 3.4.

A.3.2 3-Input

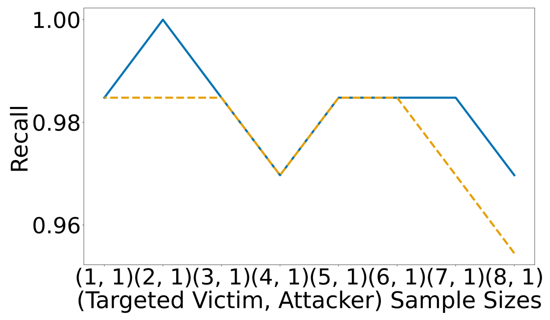
In this section, we discuss the results from DEFEAT's three input counterparts. In Section 3.4 we explored supplying DEFEAT with two image embeddings from FaceNet. When one embedding is from the *targeted* set and the other from the *attacking* set, the first phase of DEFEAT would label the pair as *infected*. In the three input variants, as long as at least one input is from the *targeted* dataset and at



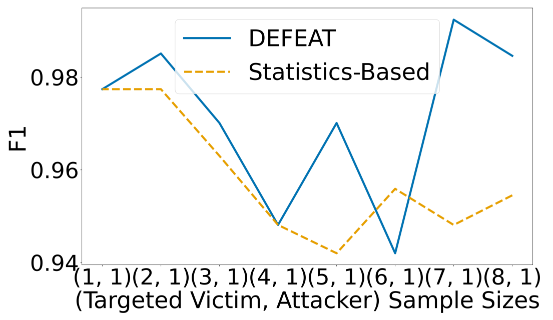
(a) Accuracy



(b) Precision

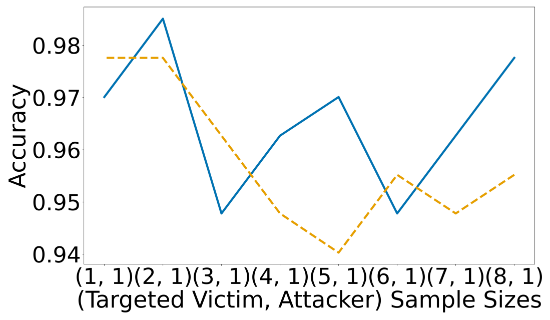


(c) Recall

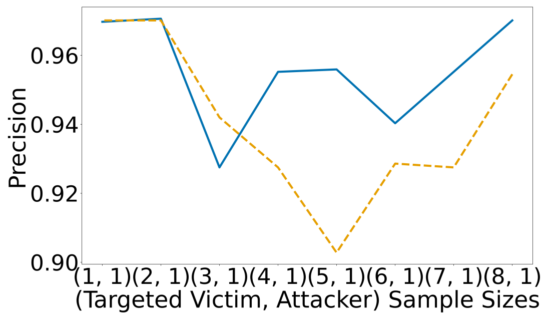


(d) F1

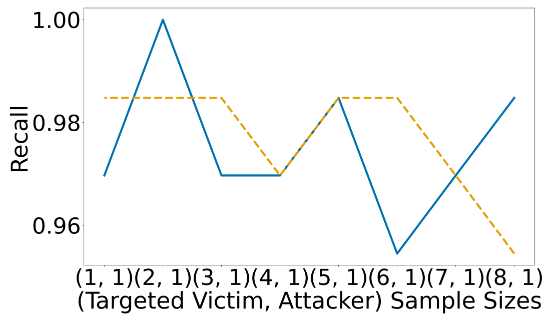
Figure A.7: Random Attack on the FEI Dataset - Varying the Number of Training Photos - Two Inputs. The x-axis represents the number of photos per label. For infected labels, a single photo is being injected for all cases.



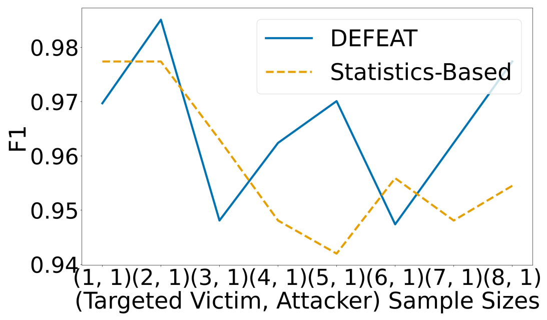
(a) Accuracy



(b) Precision



(c) Recall



(d) F1

Figure A.8: Optimal Attack on the FEI Dataset - Varying the Number of Training Photos - Two Inputs. The x-axis represents the number of photos per label. For infected labels, a single photo is being injected for all cases.

least one input is from the *attacking* dataset, the input triplet is labeled as *infected*. DEFEAT's second phase is unaffected by this change.

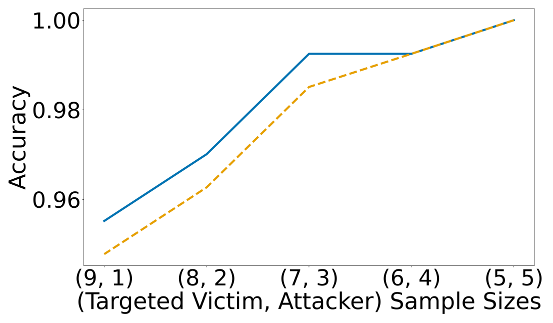
A.3.2.1 Effect of the Number of Injected Photos

Figure A.9 and Figure A.10 highlights the random and optimal attack strategy on the FEI dataset, respectively. Given the random attack strategy, the two discriminators perform nearly equally. For the optimal attack, however, the statistical discriminator outperforms DEFEAT in multiple experiments by over 4%. Both classifiers achieve above 90% accuracy in all trials, however, it is surprising that the statistics-based classifier does better than DEFEAT. When images are taken in a more controlled setting, DEFEAT's first phase does not improve the results by much, if at all. However, given more complex images, DEFEAT's DNN can glean more insightful information, as shown by the corresponding LFW figures.

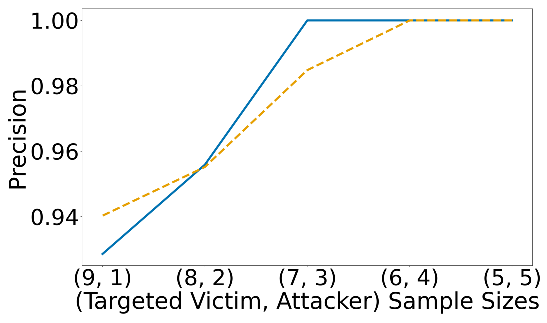
Figure A.11 and Figure A.12 shows the random and optimal attack strategy on the LFW dataset, respectively. DEFEAT largely outperformed the statistics-based classifier in most trials. Given the optimal attack strategy, DEFEAT had over 15% higher accuracy than the statistics-based classifier. This shows that the first phase of DEFEAT can better classify embeddings from more difficult images than raw statistical measurements are able to.

A.3.2.2 Effect of the Number of Training Photos per New User

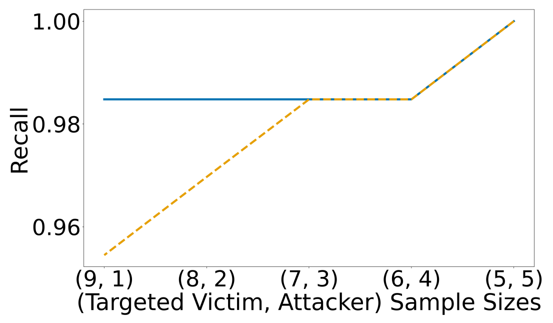
Figure A.13 and Figure A.15 showcases our random attack strategies on the FEI and LFW dataset given a constant *attacking* size partition and varying *targeted* size partition, respectively. DEFEAT outperforms the statistical-based discriminator as the ratio of *attacking* to *targeted* decreases. This implies that the three input variants of our DNN can glean more information than the raw statistics disclose as the number of differential information decreases. The same can be seen in Figure A.16



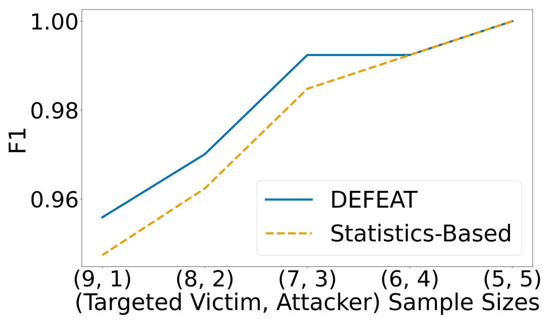
(a) Accuracy



(b) Precision

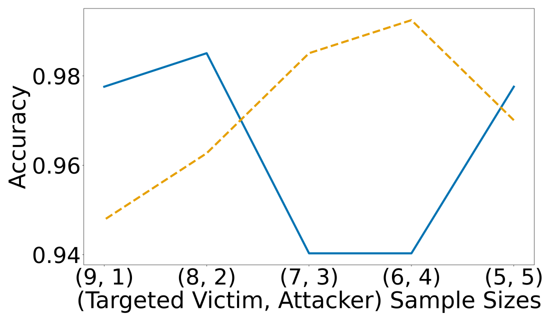


(c) Recall

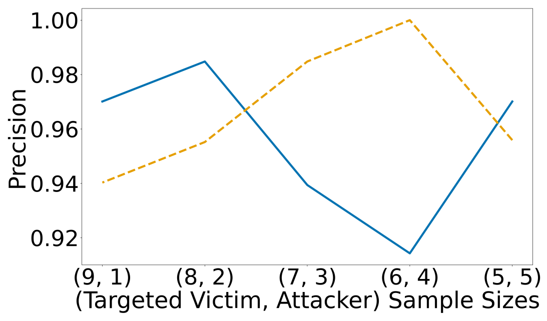


(d) F1

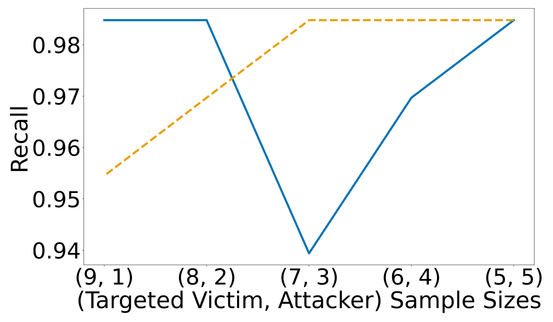
Figure A.9: Random Attack on the FEI Dataset - Varying the Number of Injected Photos - Three Inputs



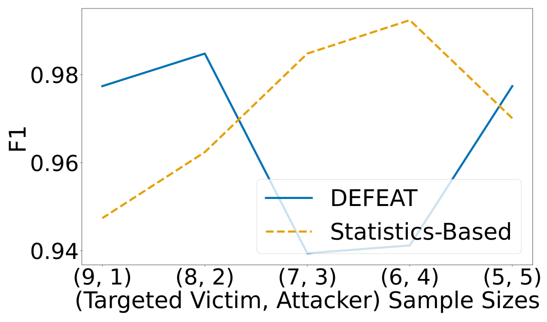
(a) Accuracy



(b) Precision

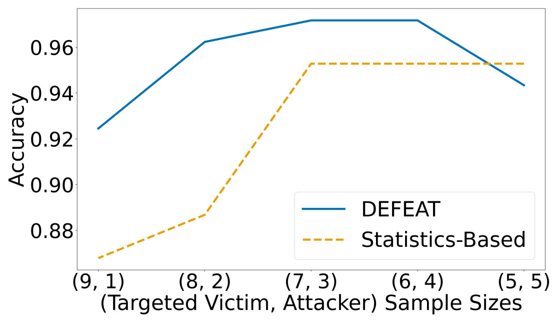


(c) Recall

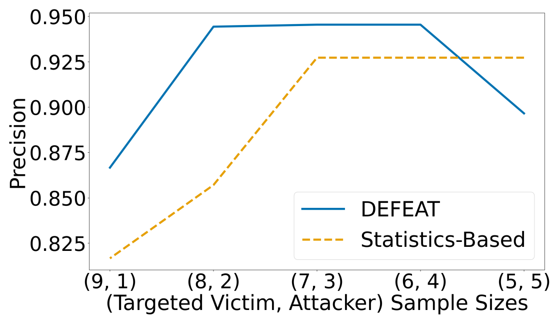


(d) F1

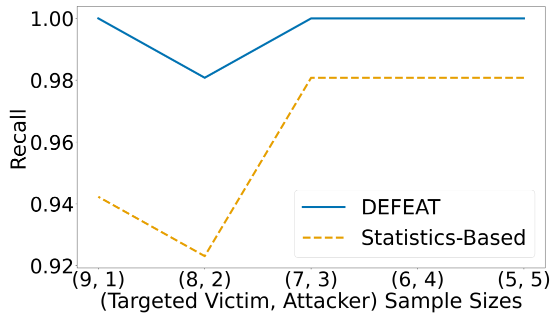
Figure A.10: Optimal Attack on the FEI Dataset - Varying the Number of Injected Photos - Three Inputs



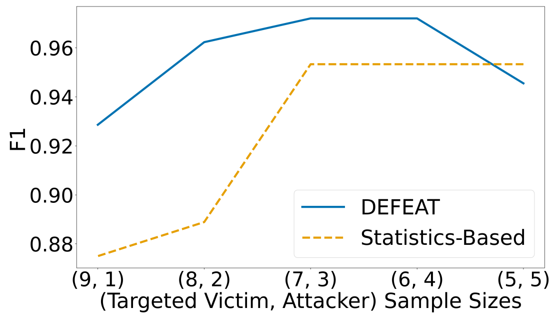
(a) Accuracy



(b) Precision

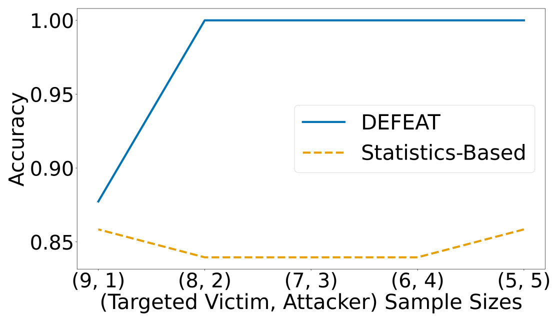


(c) Recall

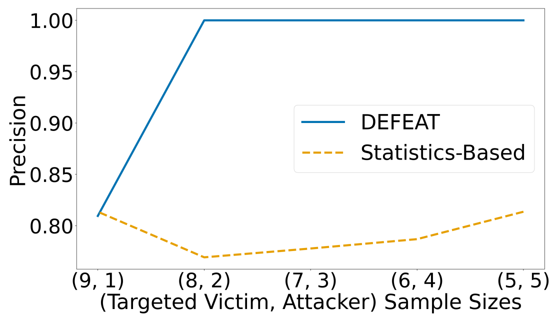


(d) F1

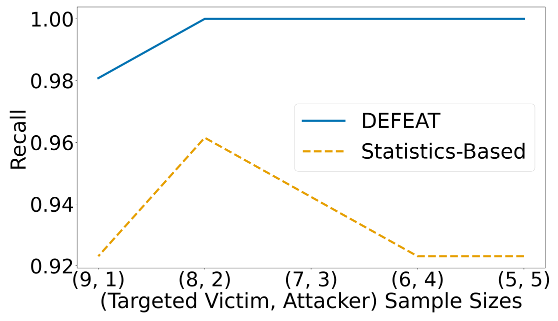
Figure A.11: Random Attack on the LFW Dataset - Varying the Number of Injected Photos - Three Inputs



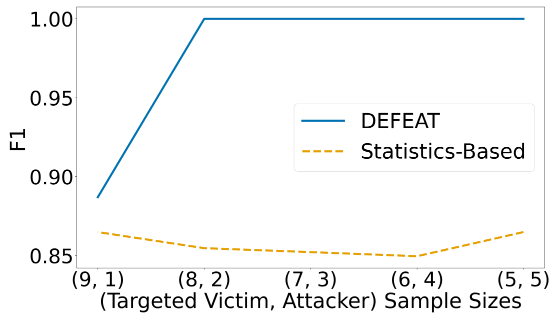
(a) Accuracy



(b) Precision

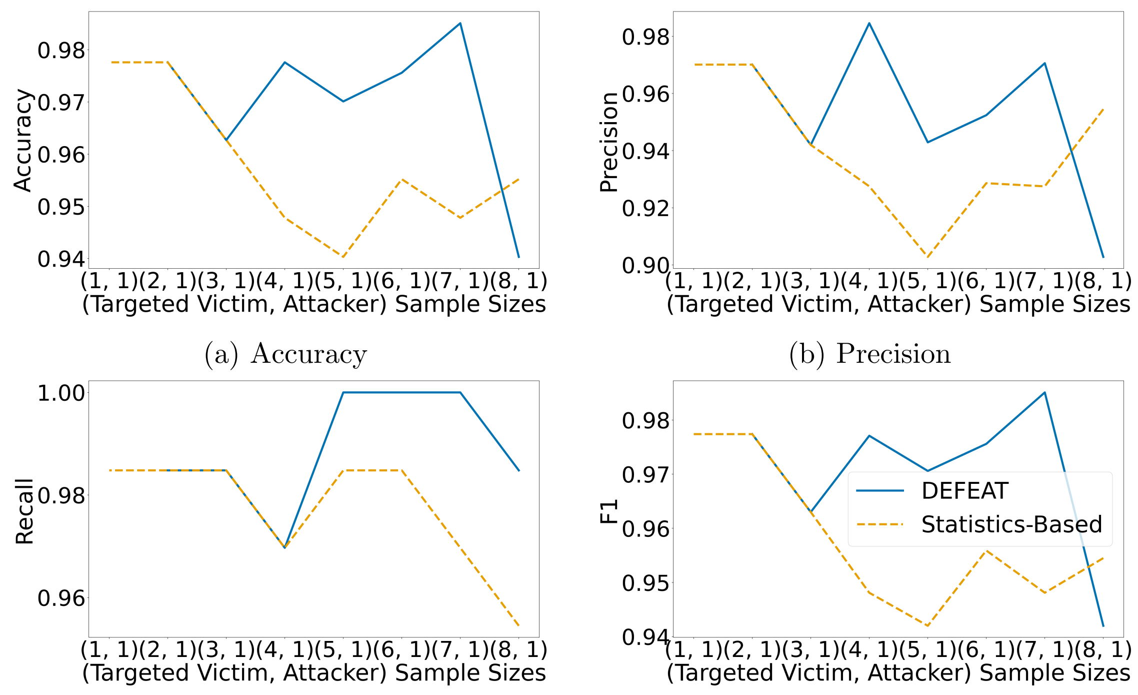


(c) Recall



(d) F1

Figure A.12: Optimal Attack on the LFW Dataset - Varying the Number of Injected Photos - Three Inputs



(a) Accuracy (b) Precision

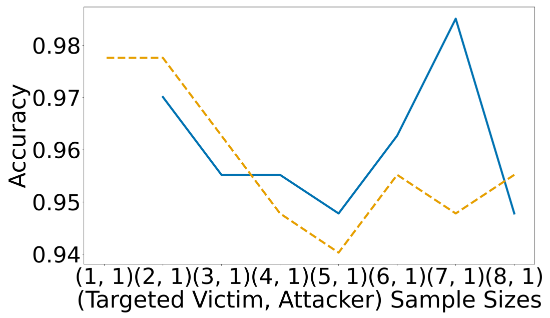
(c) Recall (d) F1

Figure A.13: Random Attack on the FEI Dataset - Varying the Number of Training Photos - Two Inputs. The x-axis represents the number of photos per label. For infected labels, a single photo is being injected for all cases.

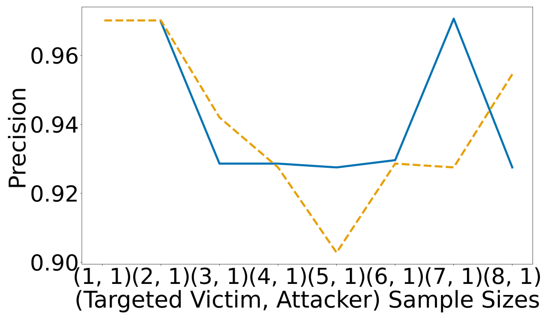
which shows the optimal attack variant on the LFW dataset. This is seen to a lesser extent in Figure A.14, where the two discriminators are generally closely related, with DEFEAT improving compared to the statistical-based discriminator as the *attacking* to *targeted* ratio decreases. Note, there are no results for DEFEAT in any of the aforementioned figures for the $(1, 1)$ case. Since three inputs are required for the three-input variant of DEFEAT, at least three samples are required. In addition, none of our facial authentication discriminators can learn via one-shot learning.

A.3.2.3 Effect of the Choice of Second Phase Classifier for DEFEAT

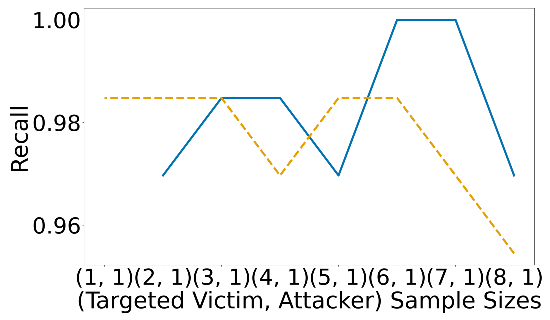
Figure A.17 highlights the differences between the KNN and SVM classifiers. Similar to the two input results, the KNN outperforms the SVM classifier, with one exception. As with the two input experiments, KNN is superior given the optimal attack;



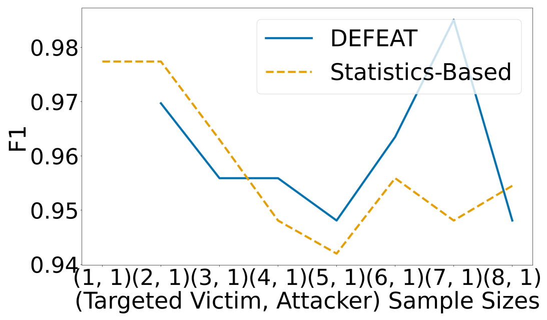
(a) Accuracy



(b) Precision

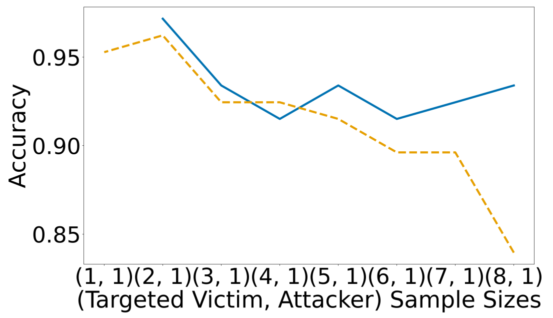


(c) Recall

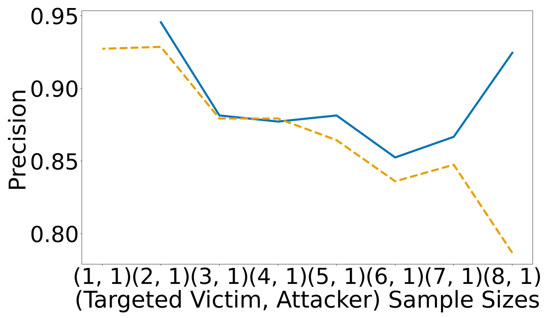


(d) F1

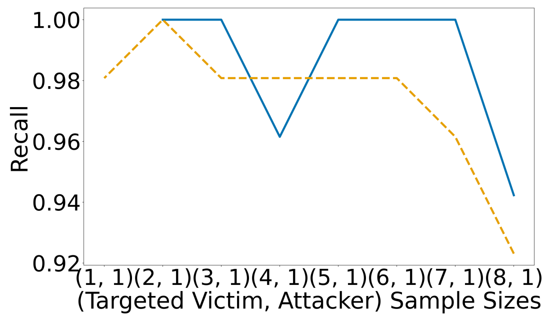
Figure A.14: Optimal Attack on the FEI Dataset - Varying the Number of Training Photos - Three Inputs. The x-axis represents the number of photos per label. For infected labels, a single photo is being injected for all cases.



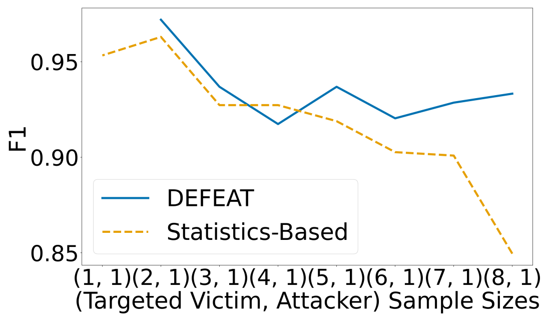
(a) Accuracy



(b) Precision

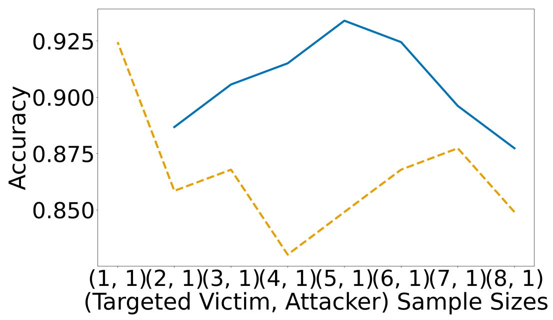


(c) Recall

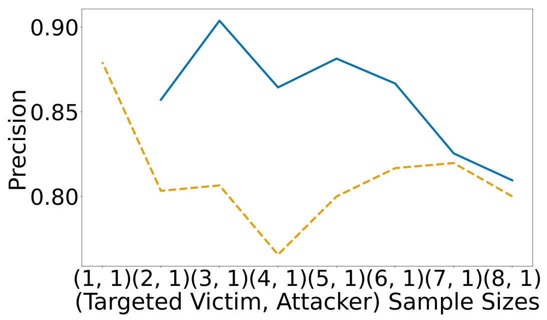


(d) F1

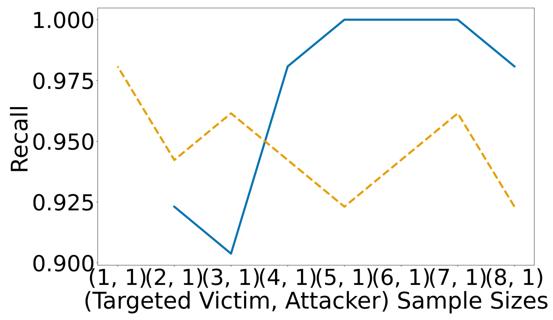
Figure A.15: Random Attack on the LFW Dataset - Varying the Number of Training Photos - Three Inputs. The x-axis represents the number of photos per label. For infected labels, a single photo is being injected for all cases.



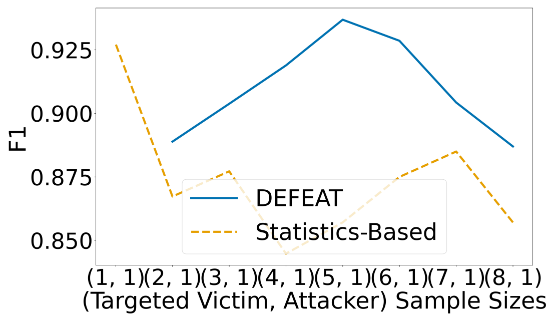
(a) Accuracy



(b) Precision

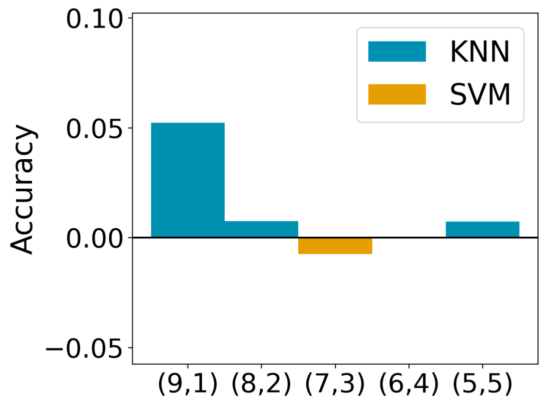


(c) Recall

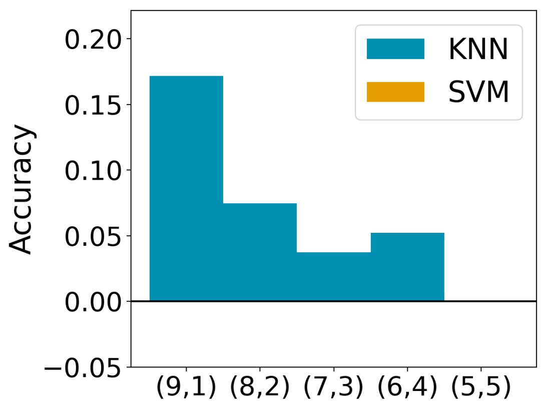


(d) F1

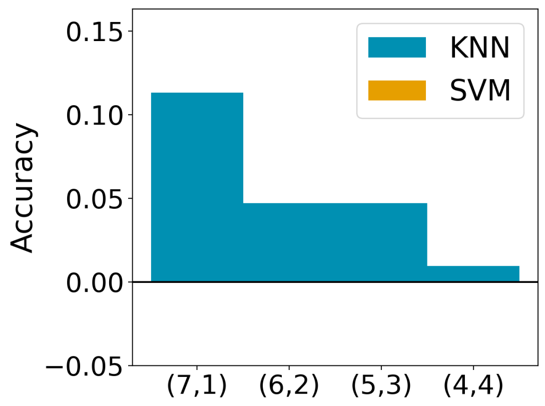
Figure A.16: Optimal Attack on the LFW Dataset - Varying the Number of Training Photos - Three Inputs. The x-axis represents the number of photos per label. For infected labels, a single photo is being injected for all cases.



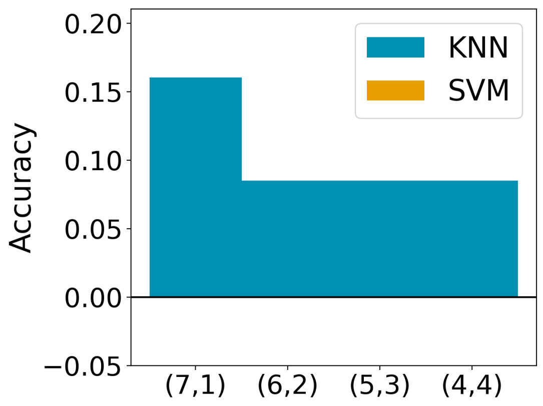
(a) FEI Random Attack Strategy



(b) FEI Optimal Attack Strategy



(c) LFW Random Attack Strategy



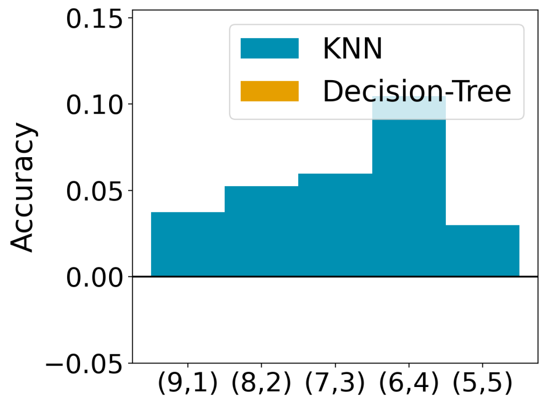
(d) LFW Optimal Attack Strategy

Figure A.17: DEFEAT Second Phase Classifier - KNN vs SVM - Three Input. Positive blue values represent when the KNN classifier outperforms the SVM classifier. Negative orange values represent when the SVM classifier outperforms the KNN classifier.

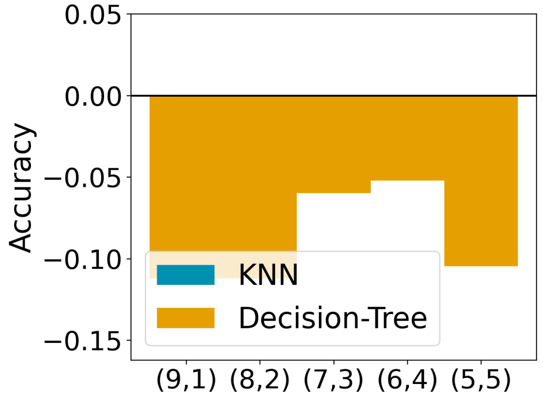
however, given the random attack, the SVM classifier outclasses the KNN classifier in a single experiment.

Figure A.18 are for the KNN and decision tree classifiers. Similar results were found as in the two input case.

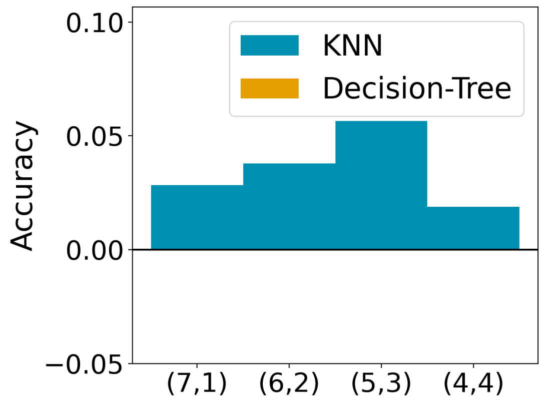
Finally, Figure A.19 are for the SVM and decision tree classifiers. In this case, the SVM classifier outperformed the decision tree classifier in a single optimal attack case. The rest of the experiments follow the two-input case.



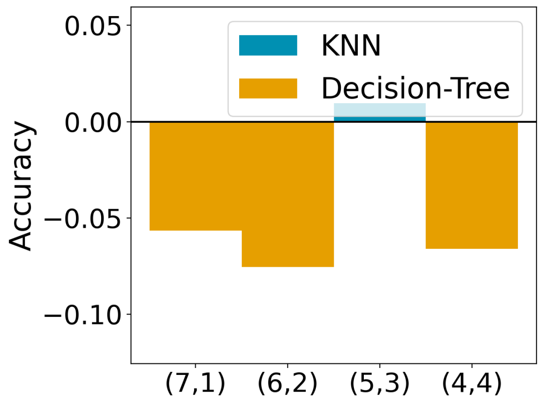
(a) FEI Random Attack Strategy



(b) FEI Optimal Attack Strategy

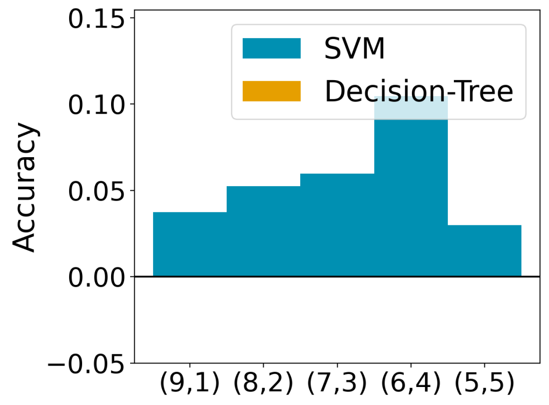


(c) LFW Random Attack Strategy

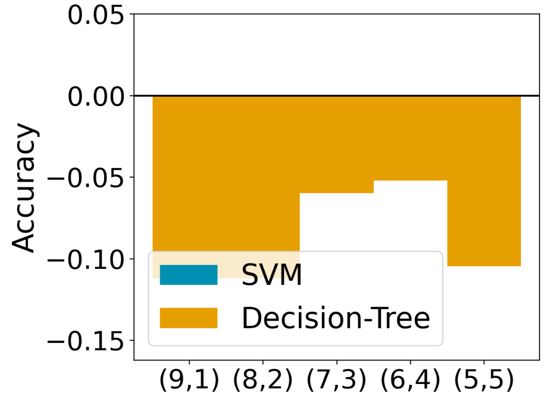


(d) LFW Optimal Attack Strategy

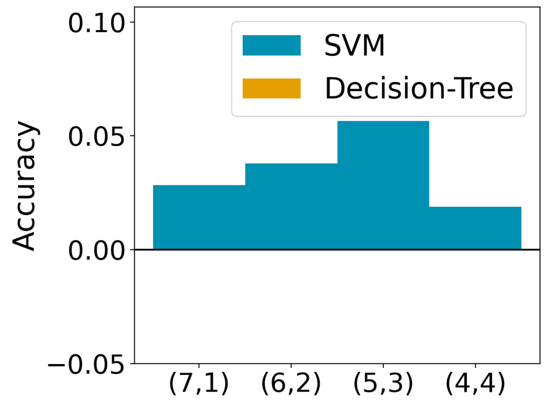
Figure A.18: DEFEAT Second Phase Classifier - KNN vs Decision Tree - Three Input. Positive blue values represent when the KNN classifier outperforms the decision tree classifier. Negative orange values represent when the decision tree classifier outperforms the KNN classifier.



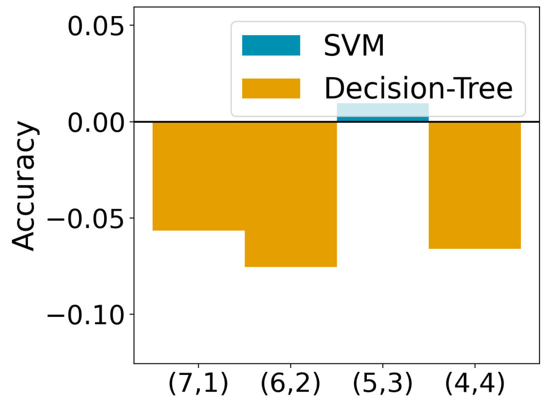
(a) FEI Random Attack Strategy



(b) FEI Optimal Attack Strategy

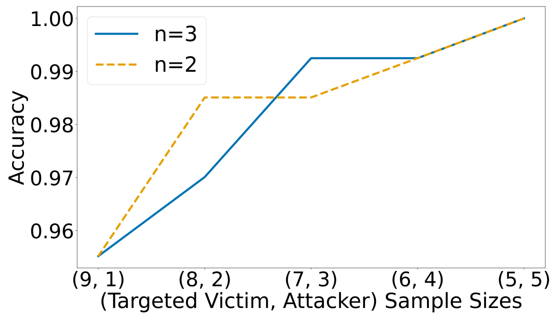


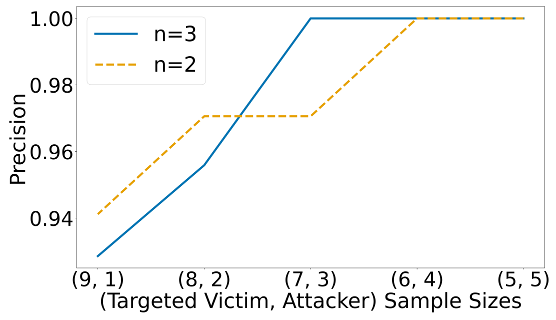
(c) LFW Random Attack Strategy

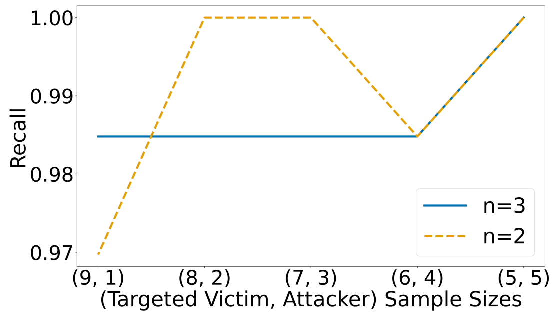


(d) LFW Optimal Attack Strategy

Figure A.19: DEFEAT Second Phase Classifier - SVM vs Decision Tree - Three Input. Positive blue values represent when the SVM classifier outperforms the decision tree classifier. Negative orange values represent when the decision tree classifier outperforms the SVM classifier.


(a) Accuracy


(b) Precision


(c) Recall

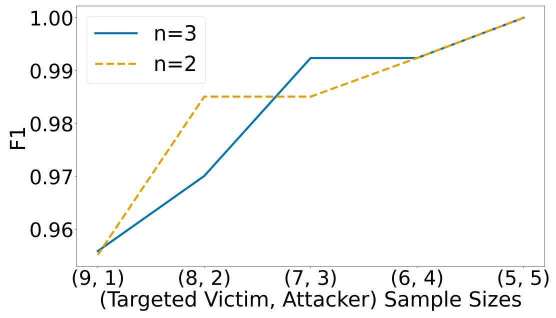
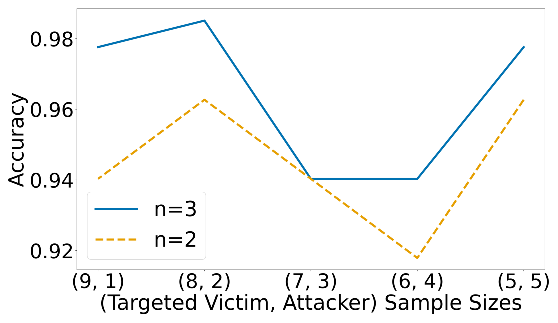

(d) F1

Figure A.20: Random Attack on the FEI Dataset - Comparing the Number of Inputs for DEFEAT

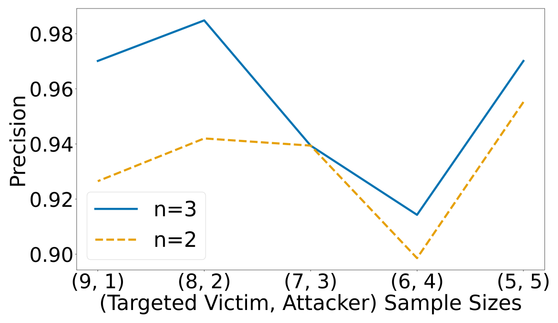
A.3.3 Effect of Number of Inputs to DEFEAT

We tested supplying DEFEAT with a varying number of photos as input, specifically, we tested if using two images as input or three images as input leads to superior results. We also test if a single image input was useful, but this showed to be as good as a random guess.

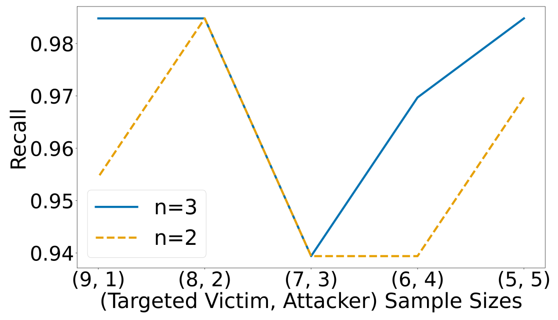
Figure A.20 and Figure A.21 show two and three inputs on the FEI dataset for the random and optimal attack strategy, respectively, while Figure A.22 and Figure A.23 are show the LFW dataset given the random and optimal attack strategy, respectively. In general, both attack strategies performed equally well. Supplying more information to DEFEAT's DNN did not seem to measurably affect results in any of the cases tested, thus we went with the less computationally expensive two input model for the majority of our work.



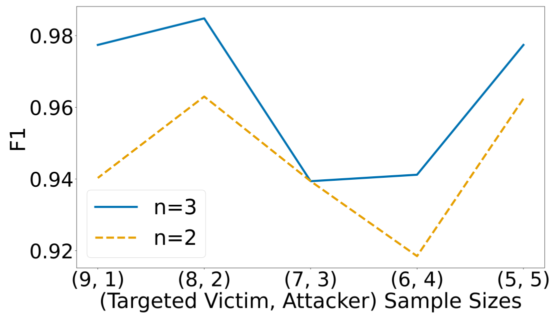
(a) Accuracy



(b) Precision

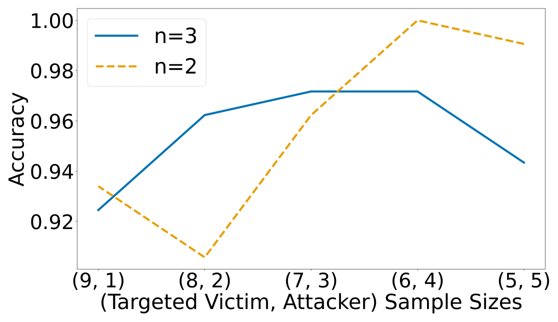


(c) Recall

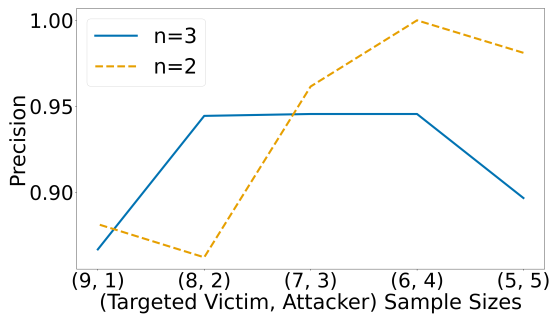


(d) F1

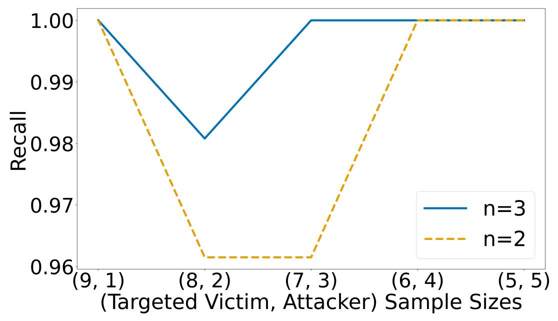
Figure A.21: Optimal Attack on the FEI Dataset - Comparing the Number of Inputs for DEFEAT



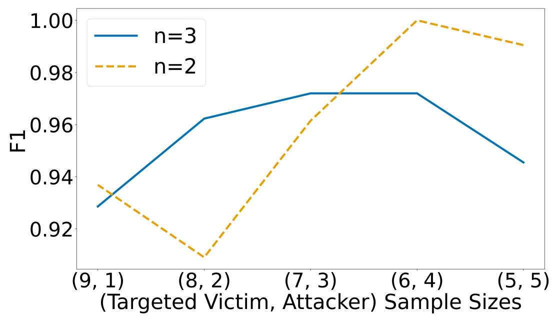
(a) Accuracy



(b) Precision

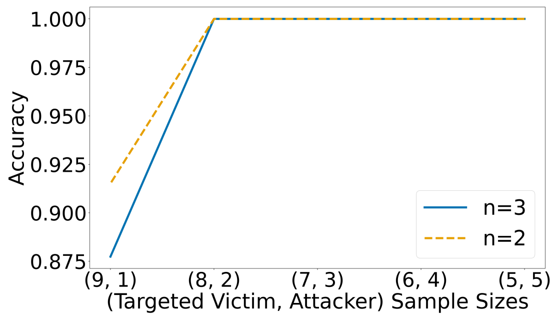


(c) Recall

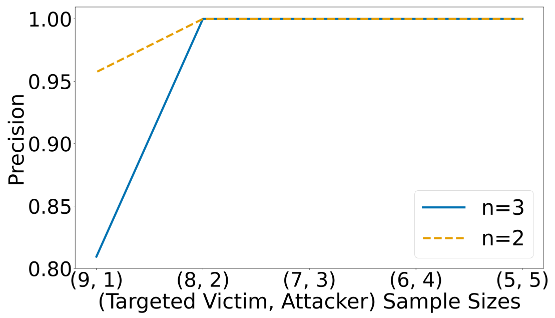


(d) F1

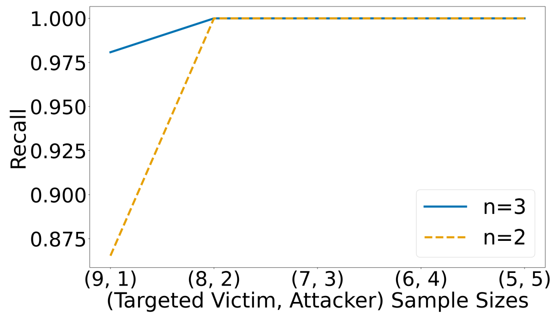
Figure A.22: Random Attack on the LFW Dataset - Comparing the Number of Inputs for DEFEAT



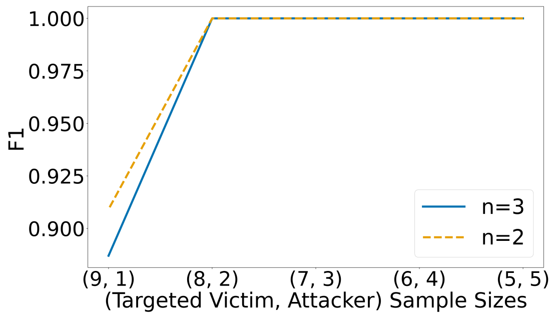
(a) Accuracy



(b) Precision



(c) Recall



(d) F1

Figure A.23: Optimal Attack on the LFW Dataset - Comparing the Number of Inputs for DEFEAT

Chapter B

DEEPFAKE DETECTION

B.1 DEEPFAKE DETECTION LITERATURE REVIEW

In this section, a thorough review of current deepfake detection methods are discussed.

To detect deepfakes, [124] used *photo response non uniformity* (PRNU) analysis. They used a small dataset consisting of 10 authentic and 16 deepfake videos lasting from 20 to 40 seconds each. To perform the photo-response analysis, they split each frame of a video into eight different groups. The average PRNU pattern was measured for each group resulting in a correlation score for each video. They found that deepfakes tended to have a lower mean normalized correlation score compared to benign samples. Both deepfakes and benign samples tended to have similar variances in normalized cross-correlation scores.

MesoNet is a widely used deepfake detector that uses the "mesoscopic" properties of images [42]. Mesoscopic properties consist of features of medium size, i.e. not small image noise nor the entire face (at once). The authors propose two different detection networks, Meso-4 and MesoInception-4, which are based on Inception modules. Both networks only consist of four convolutional layers. They evaluated their networks on their deepfake dataset consisting of 175 videos compressed using the H.264 codec at different compression levels. MesoInception-4 had a 0.917 classification score on a per-image basis, and a 0.984 classification score on a per video basis.

In [107], capsule forensics are applied to deepfake detection by using a network consisting of the VGG-19 network followed by a capsule-forensics CNN. Capsule networks are similar to traditional CNNs but are pose independent, so the face does not have to be centered and aligned. They added a pre-processing step consisting of adding Gaussian random noise to prevent overfitting. They used the same dataset used by MesoNet [42] and achieved 95.93% and 99.23% accuracy on the frame and video level, respectively. They also tested their network on the Face2Face partition of the FaceForensics dataset and achieved 99.33%, 96.0%, and 83.33% accuracy on the raw, high quality, and low-quality videos, respectively. Their difficulty in classifying low-quality videos highlights the challenge that compression adds to video manipulation detection.

In [44], the authors used the inconsistency in head poses to detect deepfakes, specifically, the difference between head position and facial feature position. Using dlib [125], a machine learning toolkit, they extracted 68 facial landmarks. Out of these landmarks, they focused on the points consisting of the eyebrows, the nose, and on the two ends of the mouth to determine the central face region. Those points, in addition to those encompassing the outline of the face, were used to determine the whole face. They used the difference in the direction that these two sets of points faced to determine if an image was a deepfake or not. They used the UADFV dataset [126], which is a rather small dataset, consisting of 49 real and 49 fake videos, and achieved an AUROC curve of 0.890 using an SVM classifier.

A CNN followed by an RNN for classification is used in [127]. The RNN was used to remember temporal differences between deepfakes and benign videos, an aspect most deepfake generators do not account for. They used the Inception V3 network as their CNN and a 2048-wide LSTM unit for the RNN. A 512 fully connected layer followed by a softmax layer was used for generating classification probabilities. They used their dataset to train their detector, which consisted of 600 deepfake videos. They achieved 96.7%, 97.1%, and 97.1% accuracy by using 20, 40, and 80 frames, respectively.

Similar to [127], [128] also used a CNN followed by a RNN for detection. They achieved 96.9% accuracy on FaceForensic++'s low-quality deepfake dataset by using 5 frames at a time for the RNN. Their RNN process allows for the network to be trained end-to-end on videos as opposed to training a network a single frame at a time and then averaging the prediction probabilities.

Unlike other deepfake detectors, [106] trained a network using only real images. They tested four different CNN models, VGG16, ResNet50, ResNet101, and ResNet152. To train these networks they used 24,442 JPEG images consisting of only real images. To simulate a deepfake, they applied various methods that produce artifacts. The idea behind this is that deepfake detectors use artifacts to detect deepfakes. Since deepfakes are expensive to produce in quantity and quality, they applied three different techniques to heavily reduce the time and computing power to produce images with similar artifacts. They first aligned the faces using different scales (make the image bigger or smaller). They then applied Gaussian blur. Finally, the images were warped back to their original dimensions. In addition, they split each image into separate (10) RoIs (regions of interest) to be classified. They then average the predictions from the RoIs to determine if the image is a deepfake. They tested their network on the UADFV [126] and DeepfakeTIMIT [103] datasets. They achieved a maximum of 97.4% on the UADFV dataset, 99.9% on low-quality DeepfakeTIMIT, and 93.2% on the high-quality DeepfakeTIMIT (in this dataset, the high-quality partition is more difficult).

[110] used biological signals for their deepfake detector, specifically, they used a combination of Photoplethysmogram (PPG) maps, spatial coherence, and temporal consistency. Power spectrum density was used to explore image behavior in the frequency domain. They achieved 96.25% accuracy on the faceswap dataset in FaceForensics++ on a per video basis.

In addition to the FaceForensics++ dataset, the paper also released a state-of-the-art detection model based on XceptionNet [41]. Xception was first proposed in [120] as a new module to replace Inception modules by decoupling the cross-channel correlations (for example, RGB channels) and spatial correlations (for example, how the left and right eyes are related). [41] used the network proposed in [120], pre-trained on ImageNet. The final layer was replaced with two outputs, one representing the probability of a genuine image, the other representing the probability of a deepfake image (a softmax layer was used). XceptionNet was able to achieve 90.29% precision on the low-quality faceswap dataset partition of FaceForensics++. Using all of the available partitions of the FaceForensics++ dataset, XceptionNet achieved 99.26%, 95.73%, and 81.00% accuracy on the raw, high-quality, and low-quality datasets, respectively.

[108] achieved high accuracy results despite using a small feature set and not using deep learning. Their method relied on transforming images to a lower dimensionality by first using the power output from a Discrete Fourier Transformer. They then applied azimuthal averaging to reduce the dimensionality. They feed this input to one of three different machine learning methods, a logistic regression classifier, SVM, or k-means clustering model. Using the DeepFakeDetection dataset from FaceForensics++ (the second generation deepfake dataset), they were able to achieve 90% and 81% accuracy per video using an SVM classifier and logistic regression classifier, respectively (they did not mention which compression level they used from the DeepFakeDetection dataset). On a per-frame basis, their SVM classifier achieved 85%, 82%, 77%, and 66% using 2000, 1000, 200, and 20 training samples, respectively. This highlights how important abundant training data is for training most deepfake detectors. Their logistic regression classifier on the other hand achieved nearly the same accuracy with 20 and 2000 samples, the accuracy is 76% and 78% accuracy, respectively. Using their dataset, they were able to achieve 100% accuracy by using

a small (roughly 100 elements) feature vector as input to their model. In addition, they were able to achieve this by only using 20 samples for training.

Unlike most deepfake detectors, [70] trained a generic fake image detector that could be applied to deepfakes. They used ResNet-50 as their base classifier and trained it using samples generated from ProGAN [129] that outputted fake samples from up to 20 different classes, such as car, cat, church, and horse. The real dataset used was LSUN [130], which was also used to train ProGAN. They tested multiple different variations of their network. The tested training ResNet-50 using a variety of classes, specifically, 2, 3, 8, 16, and 20 classes. They also tested applying different post-processing strategies during training, specifically, Gaussian blur, JPEG compression, or both. In general, they found that using 16 or more classes provided the best results for fake image detection with diminishing results after 16. In addition, augmenting the training samples did increase detection accuracy, except for two cases, one of which was with deepfakes. Their deepfake detector performed best when no augmentation was used during training, achieving 98.2 average precision using the FaceForensics++ faceswap dataset.

The authors of [39] tested XceptionNet [41] and the capsule network from [107] on four datasets including FaceForensic++. In addition, the authors tested to see which region of the face provided the highest predictive capabilities, specifically, they used the eyes, nose, mouth, or the rest of the face. According to the authors, the deepfakes from the first generation are an almost solved problem, with both networks achieving at least a 99.4% AUROC curve on the two tested datasets from the first generation. In general, XceptionNet was shown to outperform the Capsule Network on three of the four datasets. Just using a person's eyes was shown to provide similar AUC to using the entire face. The rest of the face (the face not including eyes, nose, and mouth) was shown to have the least amount of predictive power. An interesting find was that the detectors generally focused on a single eye instead of both eyes to determine if an image was fake.

Instead of focusing on detecting a fake image, [131] detected the use of blending techniques commonly used by face manipulation. They found a mask that would outline where the background and foreground images meet. Based on the existence of this mask, a probability was calculated. A two-phase network was used, where the first phase used HRNet [132] to generate the blending boundary. A custom network was followed to generate the probability blending. It is a common problem for deepfake detectors to work well on the dataset it was trained with, however, since face X-ray detected the presence of blending instead of artifacts, face X-ray found success in detecting datasets it was not trained on. [131] found much greater success than XceptionNet did on unseen datasets.

B.2 DATASETS

Figure B.1 highlights the difference in quality and function between the different synthetic face generation methods present in the FaceForensics++ dataset. These specific datasets were chosen since they have been religiously tested. In addition, to measure the effectiveness of our attack, we required the knowledge of both the source and the target identities used to generate the synthetic images. This would not be necessary to perform our attack, however, since the adversary would already have a target in mind and could poison only the samples that the adversary was interested in. If the adversary did not know the samples they wanted to attack but had access to the sample itself, they could perform facial recognition on deepfake samples to find the source face.

B.3 DATA POISONING ATTACK - INFORMATION RETENTION

This section is the corresponding data poisoning attack for the discriminator from Section 4.5.2.5. Figure B.2 demonstrates the data poisoning attack results from

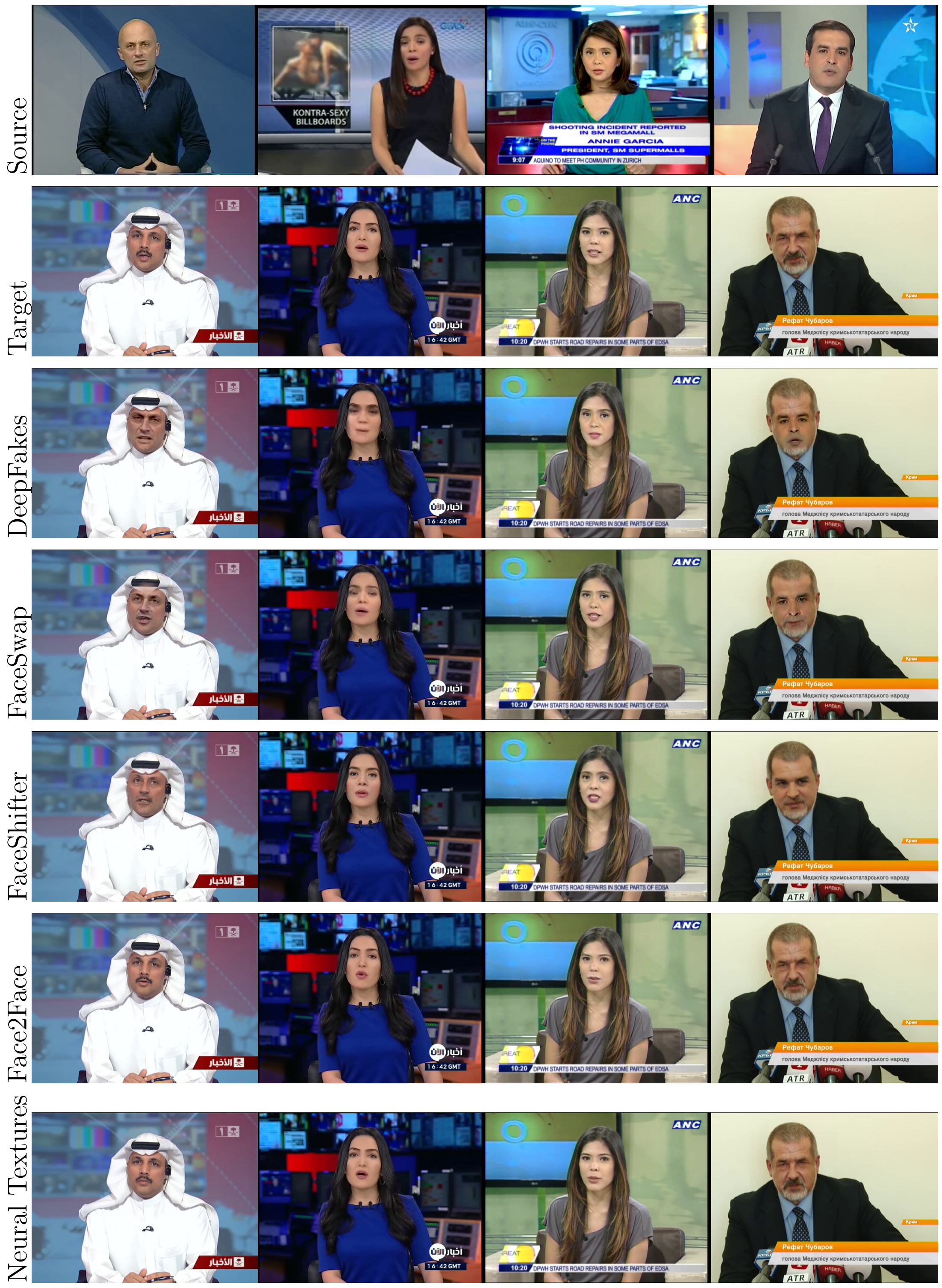
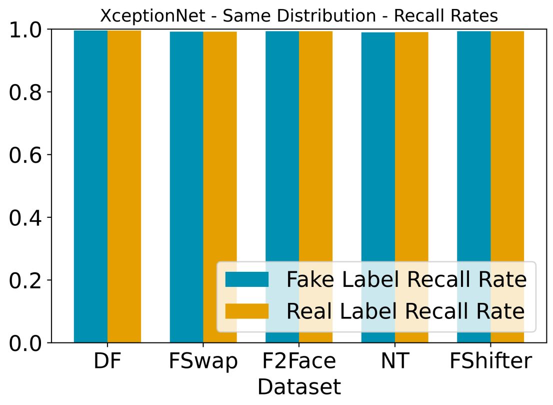
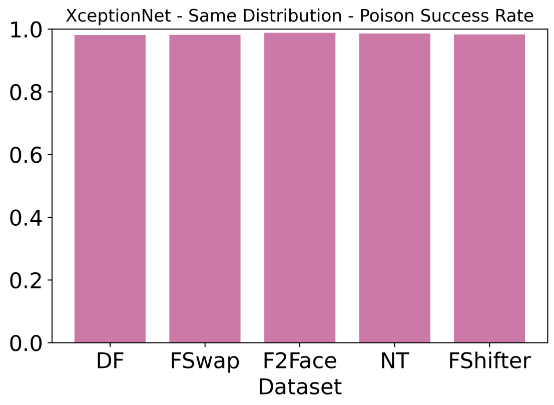


Figure B.1: FaceForensics++ Datasets Samples

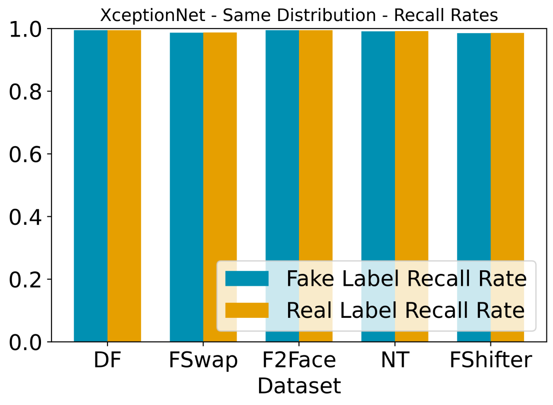
Detector-A and *Detector-B*. Figure B.2a and Figure B.2b shows the attack results for *Detector-A*, where 500 identities were used to train XceptionNet. The label flipping data poisoning attack achieved a fake label recall rate of 0.9932, a real label recall rate of 0.9937, and a poison success rate of 0.9843. Figure B.2c and Figure B.2d gives the attack results for *Detector-B*, when the full dataset was used to update XceptionNet from the *Detector-A* state. After the update, XceptionNet accomplished a fake label recall rate of 0.9910, a real label recall rate of 0.9937, and a poison success rate of 0.9845. Updating XceptionNet lead to a 0.2227% decrease in the fake label recall rate, a 0.0016% decrease in the real label recall rate, and a 0.0246% increase in poison success rate. The minute difference between *Detector-A* and *Detector-B* can be contributed to random network variation. The very slight decrease in network performance and increase in attack power may be contributed to the larger dataset size found with *Detector-B*. More research would be necessary to confirm these results however.



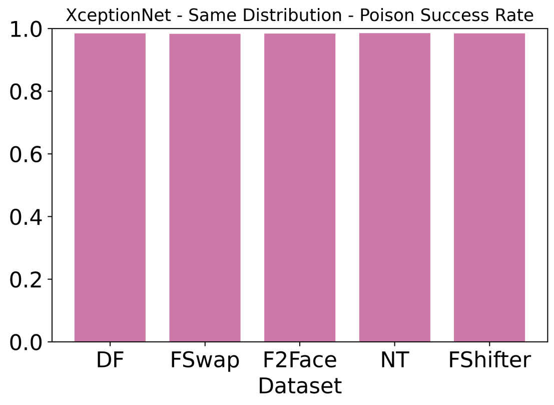
(a) Half of Dataset - Recall Rates



(b) Half of Dataset - Poison Success Rate



(c) Full Datase - Recall Rates



(d) Full Dataset - Poison Success Rate

Figure B.2: XceptionNet Average Recall Rates

BIBLIOGRAPHY

[1] D. Lin, N. Hilbert, C. Storer, W. Jiang, and J. Fan. “UFace: Your universal password that no one can see”. In: *Computers & Security* 77 (2018), pp. 627–641.

[2] A. Tagat. *Online fraud: too many accounts, too few passwords.* TechRadar. July 2012. URL: `http://www.techradar.com/us/news/internet/online-fraud-too-many-accounts-too-few-passwords-1089283`.

[3] A. Walling. *Top 10 Facial Recognition APIs & Software of 2020.* URL: `https://rapidapi.com/blog/top-facial-recognition-apis/`.

[4] URL: `https://facex.io/`.

[5] F. Schroff, D. Kalenichenko, and J. Philbin. “Facenet: A unified embedding for face recognition and clustering”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2015, pp. 815–823.

[6] I. Masi, Y. Wu, T. Hassner, and P. Natarajan. “Deep face recognition: A survey”. In: *2018 31st SIBGRAPI conference on graphics, patterns and images (SIBGRAPI).* IEEE. 2018, pp. 471–478.

[7] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. “Intriguing properties of neural networks”. In: *arXiv preprint arXiv:1312.6199* (2013).

[8] M. Sharif, S. Bhagavatula, L. Bauer, and M. K. Reiter. "Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition". In: *Proceedings of the 2016 acm sigsac conference on computer and communications security.* 2016, pp. 1528–1540.

[9] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard. "Deepfool: a simple and accurate method to fool deep neural networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2016, pp. 2574–2582.

[10] A. J. Bose and P. Aarabi. "Adversarial attacks on face detectors using neural net based constrained optimization". In: *2018 IEEE 20th International Workshop on Multimedia Signal Processing (MMSP).* IEEE. 2018, pp. 1–6.

[11] J. H. Metzen, T. Genewein, V. Fischer, and B. Bischoff. "On detecting adversarial perturbations". In: *arXiv preprint arXiv:1702.04267* (2017).

[12] W. Xu, D. Evans, and Y. Qi. "Feature squeezing: Detecting adversarial examples in deep neural networks". In: *arXiv preprint arXiv:1704.01155* (2017).

[13] G. Cohen, G. Sapiro, and R. Giryes. "Detecting adversarial samples using influence functions and nearest neighbors". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.* 2020, pp. 14453–14462.

[14] D. Meng and H. Chen. "Magnet: a two-pronged defense against adversarial examples". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security.* 2017, pp. 135–147.

[15] B. Nelson, M. Barreno, F. J. Chi, A. D. Joseph, B. I. Rubinstein, U. Saini, C. A. Sutton, J. D. Tygar, and K. Xia. "Exploiting Machine Learning to Subvert Your Spam Filter." In: *LEET* 8 (2008), pp. 1–9.

[16] D. Lowd and C. Meek. “Adversarial learning”. In: *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*. 2005, pp. 641–647.

[17] G. L. Wittel and S. F. Wu. “On Attacking Statistical Spam Filters.” In: *CEAS*. 2004.

[18] B. Biggio, B. Nelson, and P. Laskov. “Poisoning attacks against support vector machines”. In: *arXiv preprint arXiv:1206.6389* (2012).

[19] L. Muñoz-González, B. Biggio, A. Demontis, A. Paudice, V. Wongrassamee, E. C. Lupu, and F. Roli. “Towards poisoning of deep learning algorithms with back-gradient optimization”. In: *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*. 2017, pp. 27–38.

[20] B. Wang and N. Z. Gong. “Stealing hyperparameters in machine learning”. In: *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2018, pp. 36–52.

[21] T. Orekondy, B. Schiele, and M. Fritz. “Knockoff nets: Stealing functionality of black-box models”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 4954–4963.

[22] T. Lee, B. Edwards, I. Molloy, and D. Su. “Defending against model stealing attacks using deceptive perturbations”. In: *arXiv preprint arXiv:1806.00054* (2018).

[23] M. Juuti, S. Szyller, S. Marchal, and N. Asokan. “PRADA: protecting against DNN model stealing attacks”. In: *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE. 2019, pp. 512–527.

[24] A. Paudice, L. Muñoz-González, A. Gyorgy, and E. C. Lupu. “Detection of adversarial training examples in poisoning attacks through anomaly detection”. In: *arXiv preprint arXiv:1802.03041* (2018).

[25] B. Wang, Y. Yao, S. Shan, H. Li, B. Viswanath, H. Zheng, and B. Y. Zhao. "Neural cleanse: Identifying and mitigating backdoor attacks in neural networks". In: *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2019, pp. 707–723.

[26] G. Goswami, N. Ratha, A. Agarwal, R. Singh, and M. Vatsa. "Unravelling robustness of deep learning based face recognition against adversarial attacks". In: *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

[27] T. Seals. "ASUS Home Router Bugs Open Consumers to Snooping Attacks". In: *ThreatPost* (2020). URL: `https://threatpost.com/asus-home-router-bugs-snooping-attacks/157682/`.

[28] L. Pascu. "Acronis reports critical flaws in GeoVision biometric devices, man-in-the-middle attack risks". In: *BiometricUpdate* (2020). URL: `https://www.biometricupdate.com/202006/acronis-reports-critical-flaws-in-geovision-biometric-devices-man-in-the-middle-attack-risks`.

[29] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, and D. Song. "Robust physical-world attacks on deep learning visual classification". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 1625–1634.

[30] *Google Sign-In JavaScript client reference*. URL: `https://developers.google.com/identity/sign-in/web/reference`.

[31] *Facebook Login*. URL: `https://developers.facebook.com/docs/facebook-login/`.

[32] R. Mitz. *The fight to stay ahead of deepfake videos before the 2020 US election*. URL: `https://www.cnn.com/2019/06/12/tech/deepfake-2020-detection/index.html`.

[33] J. Vincent. *Watch Jordan Peele use AI to make Barack Obama deliver a PSA about fake news.* Apr. 2018. URL: https://www.theverge.com/tldr/2018/4/17/17247334/ai-fake-news-video-barack-obama-jordan-peele-buzzfeed.

[34] O. Schwartz. "You thought fake news was bad? Deep fakes are where truth goes to die". In: *The Guardian* (Nov. 2018). URL: https://www.theguardian.com/technology/2018/nov/12/deep-fakes-fake-news-truth.

[35] A. Escalante. *Research Finds Social Media Users Are More Likely To Believe Fake News.* URL: https://www.forbes.com/sites/alisonescalante/2020/08/03/research-finds-social-media-users-are-more-likely-to-believe-fake-news/.

[36] *Pentagon's Race Against Deepfakes.* URL: https://www.cnn.com/interactive/2019/01/business/pentagons-race-against-deepfakes/.

[37] B. Dolhansky, J. Bitton, B. Pflaum, J. Lu, R. Howes, M. Wang, and C. C. Ferrer. "The DeepFake Detection Challenge Dataset". In: *arXiv preprint arXiv:2006.07397* (2020).

[38] D. Song. *A Short History of Deepfakes.* Sept. 2019. URL: https://medium.com/@songda/a-short-history-of-deepfakes-604ac7be6016.

[39] R. Tolosana, S. Romero-Tapiador, J. Fierrez, and R. Vera-Rodriguez. "DeepFakes Evolution: Analysis of Facial Regions and Fake Detection Performance". In: *arXiv preprint arXiv:2004.07532* (2020).

[40] R. Tolosana, R. Vera-Rodriguez, J. Fierrez, A. Morales, and J. Ortega-Garcia. "Deepfakes and beyond: A survey of face manipulation and fake detection". In: *arXiv preprint arXiv:2001.00179* (2020).

[41] A. Rössler, D. Cozzolino, L. Verdoliva, C. Riess, J. Thies, and M. Nießner. "FaceForensics++: Learning to Detect Manipulated Facial Images". In: *International Conference on Computer Vision (ICCV)*. 2019.

[42] D. Afchar, V. Nozick, J. Yamagishi, and I. Echizen. "Mesonet: a compact facial video forgery detection network". In: *2018 IEEE International Workshop on Information Forensics and Security (WIFS)*. IEEE. 2018, pp. 1–7.

[43] P. Neekhara, S. Hussain, M. Jere, F. Koushanfar, and J. McAuley. "Adversarial Deepfakes: Evaluating Vulnerability of Deepfake Detectors to Adversarial Examples". In: *arXiv preprint arXiv:2002.12749* (2020).

[44] X. Yang, Y. Li, and S. Lyu. "Exposing deep fakes using inconsistent head poses". In: *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2019, pp. 8261–8265.

[45] F. Matern, C. Riess, and M. Stamminger. "Exploiting visual artifacts to expose deepfakes and face manipulations". In: *2019 IEEE Winter Applications of Computer Vision Workshops (WACVW)*. IEEE. 2019, pp. 83–92.

[46] R. Daza, A. Morales, J. Fierrez, and R. Tolosana. "mEBAL: A Multimodal Database for Eye Blink Detection and Attention Level Estimation". In: *Companion Publication of the 2020 International Conference on Multimodal Interaction*. 2020, pp. 32–36.

[47] S. Agarwal, H. Farid, Y. Gu, M. He, K. Nagano, and H. Li. "Protecting World Leaders Against Deep Fakes." In: *CVPR Workshops*. 2019, pp. 38–45.

[48] N. Carlini and H. Farid. "Evading Deepfake-Image Detectors with White-and Black-Box Attacks". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*. 2020, pp. 658–659.

[49] A. Gandhi and S. Jain. "Adversarial perturbations fool deepfake detectors". In: *arXiv preprint arXiv:2003.10596* (2020).

[50] C. Wang. *Deepfakes, Revenge Porn, And The Impact On Women*. Nov. 2019. URL: `https://www.forbes.com/sites/chenxiwang/2019/11/01/deepfakes-revenge-porn-and-the-impact-on-women/`.

[51] Feb. 2021. URL: `https://faceswap.dev/`.

[52] deepfakes. *deepfakes/faceswap*. Oct. 2020. URL: `https://github.com/deepfakes/faceswap`.

[53] A. Shafahi, W. R. Huang, M. Najibi, O. Suciu, C. Studer, T. Dumitras, and T. Goldstein. "Poison frogs! targeted clean-label poisoning attacks on neural networks". In: *Advances in Neural Information Processing Systems*. 2018, pp. 6103–6113.

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

[55] A. Kurakin, I. Goodfellow, S. Bengio, et al. *Adversarial examples in the physical world*. 2016.

[56] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami. "The limitations of deep learning in adversarial settings". In: *2016 IEEE European symposium on security and privacy (EuroS&P)*. IEEE. 2016, pp. 372–387.

[57] J. Su, D. V. Vargas, and K. Sakurai. "One pixel attack for fooling deep neural networks". In: *IEEE Transactions on Evolutionary Computation* 23.5 (2019), pp. 828–841.

[58] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. "Towards deep learning models resistant to adversarial attacks". In: *arXiv preprint arXiv:1706.06083* (2017).

[59] C. Kanbak, S.-M. Moosavi-Dezfooli, and P. Frossard. "Geometric robustness of deep networks: analysis and improvement". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 4441–4449.

[60] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard. "Universal adversarial perturbations". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1765–1773.

[61] S. Sarkar, A. Bansal, U. Mahbub, and R. Chellappa. "UPSET and ANGRI: Breaking high performance image classifiers". In: *arXiv preprint arXiv:1707.01159* (2017).

[62] M. Cisse, Y. Adi, N. Neverova, and J. Keshet. "Houdini: Fooling deep structured visual and speech recognition models with adversarial examples". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. 2017, pp. 6980–6990.

[63] S. Baluja and I. Fischer. "Learning to attack: Adversarial transformation networks". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1. 2018.

[64] J. Hayes and G. Danezis. "Machine learning as an adversarial service: Learning black-box adversarial examples". In: *arXiv preprint arXiv:1708.05207* 2 (2017).

[65] K. R. Mopuri, U. Garg, and R. V. Babu. "Fast feature fool: A data independent approach to universal adversarial perturbations". In: *arXiv preprint arXiv:1707.05572* (2017).

[66] K. Xu, G. Zhang, S. Liu, Q. Fan, M. Sun, H. Chen, P.-Y. Chen, Y. Wang, and X. Lin. "Evading real-time person detectors by adversarial t-shirt". In: *arXiv preprint arXiv:1910.11099* 3 (2019).

[67] A. Kurakin, I. Goodfellow, and S. Bengio. “Adversarial machine learning at scale”. In: *arXiv preprint arXiv:1611.01236* (2016).

[68] Y. Wang, X. Ma, J. Bailey, J. Yi, B. Zhou, and Q. Gu. “On the Convergence and Robustness of Adversarial Training.” In: *ICML*. Vol. 1. 2019, p. 2.

[69] G. W. Ding, Y. Sharma, K. Y. C. Lui, and R. Huang. “Mma training: Direct input space margin maximization through adversarial training”. In: *arXiv preprint arXiv:1812.02637* (2018).

[70] S.-Y. Wang, O. Wang, R. Zhang, A. Owens, and A. A. Efros. “CNN-generated images are surprisingly easy to spot... for now”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. Vol. 7. 2020.

[71] J. Frank, T. Eisenhofer, L. Schönherr, A. Fischer, D. Kolossa, and T. Holz. “Leveraging Frequency Analysis for Deep Fake Image Recognition”. In: *arXiv preprint arXiv:2003.08685* (2020).

[72] V. Chandrasekaran, K. Chaudhuri, I. Giacomelli, S. Jha, and S. Yan. “Exploring connections between active learning and model extraction”. In: *29th USENIX Security Symposium (USENIX Security 20)*. 2020, pp. 1309–1326.

[73] B. Nelson, M. Barreno, F. J. Chi, A. D. Joseph, B. I. Rubinstein, U. Saini, C. Sutton, J. Tygar, and K. Xia. “Misleading learners: Co-opting your spam filter”. In: *Machine learning in cyber trust*. Springer, 2009, pp. 17–51.

[74] J. Steinhardt, P. W. W. Koh, and P. S. Liang. “Certified defenses for data poisoning attacks”. In: *Advances in neural information processing systems*. 2017, pp. 3517–3529.

[75] A. Paudice, L. Muñoz-González, and E. C. Lupu. “Label sanitization against label flipping poisoning attacks”. In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2018, pp. 5–15.

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

[77] J. Clements and Y. Lao. "Backdoor attacks on neural network operations". In: *2018 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*. IEEE. 2018, pp. 1154–1158.

[78] Y. Liu, S. Ma, Y. Aafer, W.-C. Lee, J. Zhai, W. Wang, and X. Zhang. "Trojaning attack on neural networks". In: (2017).

[79] Y. Gao, C. Xu, D. Wang, S. Chen, D. C. Ranasinghe, and S. Nepal. "Strip: A defence against trojan attacks on deep neural networks". In: *Proceedings of the 35th Annual Computer Security Applications Conference*. 2019, pp. 113–125.

[80] K. Liu, B. Dolan-Gavitt, and S. Garg. "Fine-pruning: Defending against backdooring attacks on deep neural networks". In: *International Symposium on Research in Attacks, Intrusions, and Defenses*. Springer. 2018, pp. 273–294.

[81] H. Xiao, H. Xiao, and C. Eckert. "Adversarial Label Flips Attack on Support Vector Machines." In: *ECAI*. 2012, pp. 870–875.

[82] M. Jagielski, G. Severi, N. P. Harger, and A. Oprea. "Subpopulation data poisoning attacks". In: *arXiv preprint arXiv:2006.14026* (2020).

[83] X. Chen, C. Liu, B. Li, K. Lu, and D. Song. "Targeted backdoor attacks on deep learning systems using data poisoning". In: *arXiv preprint arXiv:1712.05526* (2017).

[84] O. Suciu, R. Marginean, Y. Kaya, H. Daume III, and T. Dumitras. "When does machine learning FAIL? generalized transferability for evasion and poisoning attacks". In: *27th USENIX Security Symposium USENIX Security 18)*. 2018, pp. 1299–1316.

[85] S. Shan, E. Wenger, J. Zhang, H. Li, H. Zheng, and B. Y. Zhao. "Fawkes: Protecting privacy against unauthorized deep learning models". In: *29th {USENIX} Security Symposium ({USENIX} Security 20)*. 2020, pp. 1589–1604.

[86] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. "Going deeper with convolutions". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.

[87] M. D. Zeiler and R. Fergus. "Visualizing and understanding convolutional networks". In: *European conference on computer vision*. Springer. 2014, pp. 818–833.

[88] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao. "Joint face detection and alignment using multitask cascaded convolutional networks". In: *IEEE Signal Processing Letters* 23.10 (2016), pp. 1499–1503.

[89] D. Sandberg. *davidsandberg/facenet*. Apr. 2018. URL: `https://github.com/davidsandberg/facenet`.

[90] Q. Cao, L. Shen, W. Xie, O. M. Parkhi, and A. Zisserman. "Vggface2: A dataset for recognising faces across pose and age". In: *2018 13th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2018)*. IEEE. 2018, pp. 67–74.

[91] G. B. Huang, M. Mattar, T. Berg, and E. Learned-Miller. "Labeled faces in the wild: A database for studying face recognition in unconstrained environments". In: 2008.

[92] C. E. Thomaz and G. A. Giraldi. "A new ranking method for principal components analysis and its application to face image analysis". In: *Image and Vision Computing* 28.6 (2010), pp. 902–913.

[93] A. D. Rayome. "Why do so many wireless routers lack basic security protections?" In: *TechRepublic. https://threatpost.com/asus-home-router-bugs-snooping-attacks/157682/* (2019).

[94] BioID. "https://www.bioid.com/". In: ().

[95] V. Nair and G. E. Hinton. "Rectified linear units improve restricted boltzmann machines". In: *Proceedings of the 27th international conference on machine learning (ICML-10)*. 2010, pp. 807–814.

[96] C. E. Shannon. "A mathematical theory of communication". In: *Bell system technical journal* 27.3 (1948), pp. 379–423.

[97] K. Keahey, J. Anderson, Z. Zhen, P. Riteau, P. Ruth, D. Stanzione, M. Cevik, J. Colleran, H. S. Gunawi, C. Hammock, J. Mambretti, A. Barnes, F. Halbach, A. Rocha, and J. Stubbs. "Lessons Learned from the Chameleon Testbed". In: *Proceedings of the 2020 USENIX Annual Technical Conference (USENIX ATC '20)*. USENIX Association, July 2020.

[98] Microsoft. "Face API - v1.0". In: *https://westus.dev.cognitive.microsoft.com/docs / services/ 563879b61984550e40cbbe8d/ operations/ 563879b61984550f30395236* ().

[99] M. Somers. *Deepfakes, explained*. July 2020. URL: `https://mitsloan.mit.edu/ideas-made-to-matter/deepfakes-explained`.

[100] C. Bregler, M. Covell, and M. Slaney. "Video rewrite: Driving visual speech with audio". In: *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. 1997, pp. 353–360.

[101] J. Vincent. *Disney's deepfakes are getting closer to a big-screen debut*. June 2020. URL: `https://www.theverge.com/2020/6/29/21306889/disney-deepfake-face-swapping-research-megapixel-resolution-film-tv`.

[102] R. Metz. *The fight to stay ahead of deepfake videos before the 2020 US election.* June 2019. URL: `https://www.cnn.com/2019/06/12/tech/deepfake-2020-detection`.

[103] P. Korshunov and S. Marcel. "Deepfakes: a new threat to face recognition? assessment and detection". In: *arXiv preprint arXiv:1812.08685* (2018).

[104] F. Marra, D. Gragnaniello, D. Cozzolino, and L. Verdoliva. "Detection of gan-generated fake images over social networks". In: *2018 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR)*. IEEE. 2018, pp. 384–389.

[105] Y. Li, X. Yang, P. Sun, H. Qi, and S. Lyu. "Celeb-df: A new dataset for deepfake forensics". In: *arXiv preprint arXiv:1909.12962* (2019).

[106] Y. Li and S. Lyu. "Exposing deepfake videos by detecting face warping artifacts". In: *arXiv preprint arXiv:1811.00656* (2018).

[107] H. H. Nguyen, J. Yamagishi, and I. Echizen. "Capsule-forensics: Using capsule networks to detect forged images and videos". In: *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2019, pp. 2307–2311.

[108] R. Durall, M. Keuper, F.-J. Pfreundt, and J. Keuper. "Unmasking deepfakes with simple features". In: *arXiv preprint arXiv:1911.00686* (2019).

[109] S. Hussain, P. Neekhara, M. Jere, F. Koushanfar, and J. McAuley. "Adversarial deepfakes: Evaluating vulnerability of deepfake detectors to adversarial examples". In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 2021, pp. 3348–3357.

[110] U. A. Ciftci, I. Demir, and L. Yin. "Fakecatcher: Detection of synthetic portrait videos using biological signals". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2020).

[111] MarekKowalski. *MarekKowalski/FaceSwap*. URL: `https://github.com/MarekKowalski/FaceSwap/`.

[112] J. Thies, M. Zollhofer, M. Stamminger, C. Theobalt, and M. Nießner. "Face2face: Real-time face capture and reenactment of rgb videos". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2387–2395.

[113] J. Thies, M. Zollhöfer, and M. Nießner. "Deferred neural rendering: Image synthesis using neural textures". In: *ACM Transactions on Graphics (TOG)* 38.4 (2019), pp. 1–12.

[114] L. Li, J. Bao, H. Yang, D. Chen, and F. Wen. "Faceshifter: Towards high fidelity and occlusion aware face swapping". In: *arXiv preprint arXiv:1912.13457* (2019).

[115] K. Dale, K. Sunkavalli, M. K. Johnson, D. Vlasic, W. Matusik, and H. Pfister. "Video face replacement". In: *Proceedings of the 2011 SIGGRAPH Asia conference*. 2011, pp. 1–10.

[116] D. Vlasic, M. Brand, H. Pfister, and J. Popovic. "Face transfer with multilinear models". In: *ACM SIGGRAPH 2006 Courses*. 2006, 24–es.

[117] P. Garrido, L. Valgaerts, H. Sarmadi, I. Steiner, K. Varanasi, P. Perez, and C. Theobalt. "Vdub: Modifying face video of actors for plausible visual alignment to a dubbed audio track". In: *Computer graphics forum*. Vol. 34. 2. Wiley Online Library. 2015, pp. 193–204.

[118] J. Thies, M. Zollhöfer, M. Nießner, L. Valgaerts, M. Stamminger, and C. Theobalt. "Real-time expression transfer for facial reenactment." In: *ACM Trans. Graph.* 34.6 (2015), pp. 183–1.

[119] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. "Image-to-image translation with conditional adversarial networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1125–1134.

[120] F. Chollet. "Xception: Deep learning with depthwise separable convolutions". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1251–1258.

[121] *Keras documentation: Xception*. URL: `https://keras.io/api/applications/xception/`.

[122] *sklearn.svm.SVC*. URL: `https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html`.

[123] *sklearn.ensemble.RandomForestClassifier*. URL: `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html`.

[124] M. Koopman, A. M. Rodriguez, and Z. Geradts. "Detection of deepfake video manipulation". In: *Conference: IMVIP*. 2018.

[125] D. E. King. "Dlib-ml: A machine learning toolkit". In: *The Journal of Machine Learning Research* 10 (2009), pp. 1755–1758.

[126] Y. Li, M.-C. Chang, and S. Lyu. "In ictu oculi: Exposing ai generated fake face videos by detecting eye blinking". In: *arXiv preprint arXiv:1806.02877* (2018).

[127] D. Güera and E. J. Delp. "Deepfake video detection using recurrent neural networks". In: *2018 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*. IEEE. 2018, pp. 1–6.

[128] E. Sabir, J. Cheng, A. Jaiswal, W. AbdAlmageed, I. Masi, and P. Natarajan. "Recurrent convolutional strategies for face manipulation detection in videos". In: *Interfaces (GUI)* 3.1 (2019).

[129] T. Karras, T. Aila, S. Laine, and J. Lehtinen. “Progressive growing of gans for improved quality, stability, and variation”. In: *arXiv preprint arXiv:1710.10196* (2017).

[130] F. Yu, A. Seff, Y. Zhang, S. Song, T. Funkhouser, and J. Xiao. “Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop”. In: *arXiv preprint arXiv:1506.03365* (2015).

[131] L. Li, J. Bao, T. Zhang, H. Yang, D. Chen, F. Wen, and B. Guo. “Face x-ray for more general face forgery detection”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 5001–5010.

[132] K. Sun, B. Xiao, D. Liu, and J. Wang. “Deep high-resolution representation learning for human pose estimation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2019, pp. 5693–5703.

VITA

Dalton Russell Cole was born in Kansas City, Missouri. He graduated Summa Cum Laude from Missouri University of Science and Technology in 2016 with a Bachelor of Science in Computer Science and minors in Computer Engineering and Mathematics. Dalton received the Scholarship For Service to obtain his Master's degree and Ph.D. He completed his Master's degree at Missouri S&T in May 2019 and transferred to the University of Missouri - Columbia where he completed his Ph.D. July of 2021 under Dr. Dan Lin. Dalton has accepted a Research and Development position at Sandia National Laboratories in Albuquerque, New Mexico.