Investigating Robustness of Deepfake Detectors under Feature Collision Data Poisoning Attacks

Alexander Murphy

MInf Project (Part 1) Report
Master of Informatics
School of Informatics
University of Edinburgh

2024

Abstract

Deepfakes are increasingly used to perform malicious activities. Machine Learning based deepfake detectors try to counter these by distinguishing real content from deepfakes. Due to the need for large datasets when training these detectors, the web is often scraped for data which leaves an opportunity for an adversary to inject harmless looking data, that actually has been engineered to poison a detector trained on it, called a poison attack. We investigate the robustness of deepfake detectors against a family of poison attacks called feature collision attacks. For our experiments we choose the feature collision attack Poison Frogs, which we choose due to its performance being a good indicator of the effectiveness of the whole family of feature collision attacks. We use the FaceForensics++ dataset, due to the large variety of original data and deepfake techniques it contains, and evaluate on three different models in order to generalize our results to deepfake detection as a whole. We perform three different versions of the attack in which we aim to make a detector misclassify a target deepfake as real, with the first investigating the vulnerability in relation to number of poisons, then in relation to the distance between poisons and target used and lastly the vulnerability of networks when finetuning only the last layer of the network, instead of training all layers. For each experiment we compare target classification before and after poisoning. We conclude that the networks we poisoned are robust when training all layers of the network, with no significant change in the target classification caused by the number or distance of poisons, but are potentially vulnerable when just finetuning the last layer, making this the first work to discover the vulnerability of finetuned deepfake detectors against feature collision attacks. We compare the results we get to the existing literature on data poisoning against deepfake detectors, as well data poisoning in general. To the best of our knowledge this is the first published clean-label data poisoning attack, as well as the first published feature collision attack, performed on deepfake detectors.

Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Alexander Murphy*)

Acknowledgements

I want to thank my supervisor Pavlos for his great advice and valuable feedback throughout the year.

I want to thank my parents, grandparents and my whole family for supporting me throughout, even Daniel.

I want to thank Sarah for motivating me to do my best.

Table of Contents

Chapter 1

Introduction

In this dissertation, we investigate the vulnerability of deepfake detection neural networks to feature collision poisoning attacks. We perform these attacks in various settings on three different deepfake detectors to comprehensively evaluate their susceptibility to such attacks. Poison attacks have the ability to manipulate machine learning systems by creating carefully crafted poison data which makes it important to research the vulnerability of systems that need high trust, such as deepfake detectors. This introduction provides a brief overview of deepfakes, deepfake detection, security of machine learning and data poisoning. We outline our research goals, contributions and dissertation structure.

1.1 Deepfakes and Deepfake Detection

Deepfakes are artificially generated images, videos or audio, that are used for impersonation or changing the meaning of what a person originally said [48]. They are generated using a variety of machine learning techniques and have become more common with a growing availability of computational resources. This proliferation of deepfakes on the internet and social media can lead to a variety of harmful effects on society and individuals in the form of misinformation, financial fraud or faked sexual content [48] [13]. In Section 2.3 we discuss deepfakes in more detail.

Deepfake detection is the practice of detecting deepfake content, which can be used to counter the spread of misinformation and impersonation of people online. Especially since humans overestimate their ability to detect whether content is a deepfake, it is important to try tackling their spread [45]. State-of-the-art deepfake detection is done with machine learning based models, especially convolutional neural networks [48], of which we use three different ones in our experiments. In Section 2.3 we explain deepfake detection in detail. We use the FaceForensics++ [63] dataset for training because it contains multiple deepfake generation methods applied to a wide variety of real data, which we explain more in Section 3.1.

1.2 Security of Machine Learning

Security of machine learning has become important with the proliferation of AI systems, especially relating to bias and fairness in AI, as well as the potential dangers of artificial general intelligence (AGI) [79] [77]. Governments, including the US and EU, have created legislation related to the safety of AI, highlighting the need for safe AI [58] [20].

In addition to problems such as bias, fairness and the control of AGI, there are types of adversarial attacks that can be performed by adversaries aiming to maliciously exploit machine learning systems. These attacks are similar to classical computer security attacks where the attacker exploits a vulnerability to achieve a specific goal. The attacker can target different parts of the ML pipeline, such as data collection, training or inference [56]. These attacks have been shown to be successful on commercial ML systems such as self driving cars, showing their practicality and potential dangers [26]. We discuss adversarial attacks in detail in Section 2.4.

1.3 Data Poisoning on Deepfake Detectors

Deepfake detectors can also be victims of adversarial attacks and there has been some published work on the robustness of detectors [62] [37]. The adversarial attack we perform is data poisoning, where a dataset is injected with poison data, in an attempt to decrease classifier performance, either overall or on specific targets, when trained on the poisoned dataset [28]. Most data poisoning techniques have only been done on tasks such as general image recognition and have been shown to be effective on a large range of models, datasets and attack settings within image recognition [28] [72] [66] [4] [80] [27] [75]. One of the most popular data poisoning techniques uses feature collisions, which create poisons in relation to the representation of inputs in feature space as we explain in Section 3.3. We use a targeted feature collision attack called Poison Frogs [66] in our experiments, which is the first published feature collision attack and has been shown to give a good indication of the effectiveness of other feature collision attacks [65]. In our experiments we aim to poison a network such that it misclassifies a randomly chosen deepfake as real, while keeping overall network accuracy unchanged.

1.4 Research Goals

The main aim of this dissertation is to evaluate the robustness of deepfake detectors against feature collision data poisoning attacks. We want to evaluate whether detectors are vulnerable in the finetuning setting (where only one layer is trained) and in the end-to-end setting (where all layers of the network are trained). In addition to that, we want to identify what effect the number of poisons used, and the distance between poisons and the target in feature space, has on the attack. Lastly, we want to identify what network access an adversary might need for a successful attack.

We measure the effectiveness of attacks, by comparing the probability of the deepfake target being classified as real before and after poisoning, with a large increase in the

probability being a successful attack and no change being an unsuccessful attack. We also monitor the change in overall network accuracy, which should stay the same, as we only target one specific image.

To gain a general understanding on the vulnerability of deepfake detectors against feature collision attacks we choose Poison Frogs [66] which gives a good indication of the effectiveness of feature collision attacks as a whole [65]. We choose three different models to evaluate on and perform attacks in both finetuning and end-to-end scenarios, training them on FaceForensics++, which contains data created from a wide distribution of real images and multiple deepfake generators. This setup gives us the ability to generalise our results and answer our research goals.

To the best of our knowledge, there is only one published work on data poisoning against deepfake detectors, by Russell [62], which uses a simpler attack without ”clean labels”. There also is one other type of adversarial attack done by Hussain et al. [37], which isn't a data poisoning attack. We want to add to this literature on robustness of deepfake detectors, as well as use their results to draw stronger conclusions for our own work.

1.5 Contributions

We make the following contributions in this dissertation:

- To the best of our knowledge we conduct the first published feature collision poison attack on deepfake detectors, which we perform on three different models in various settings. It also is the first attack on deepfake detectors which does not assume access to the data labelling function.
- We show that deepfake detectors can be vulnerable against feature collision attacks in finetuning scenarios, while being more robust during end-to-end training. In addition to that we show that number of poisons and their distance from the target do not have a significant impact on our experiment's effectiveness and provide explanations for why this could be.
- We create the latest overview of the current landscape around data poisoning on deepfake detectors by combining our research with previously published work.
- We develop custom code to extract and process a 1.7TB dataset, train three models and run the poison attack on the Eddie cluster, implementing various optimizations to speed up experiments by about 10x from other code found online.

1.6 Dissertation Structure

In Chapter 2 we provide an overview of the background knowledge that is required to understand the dissertation, starting with an introduction to machine learning methods, moving on to deepfake generation and detection, and lastly, adversarial attacks in machine learning with a focus on data poisoning, as well as defenses against them.

We present our methodology in Chapter 3, starting with in depth comparisons of deepfake detection datasets and an overview of the FaceForensics++ dataset that we will be using. We explain the Xception-Full, Xception-Face and Meso-Face models that we attack, including training regimes and results. Finally, we present data poisoning techniques, with a focus on feature collisions and the Poison Frogs attack that we are using.

In Chapter 4 we describe each experiment we perform in detail. In total we perform attacks on three different classifiers in end-to-end and finetuning settings as well as an investigation of the amount and closeness of poisons.

In Chapter 5 we discuss the results of all our experiments, which show that the networks are robust in an end-to-end setting but more vulnerable when finetuning, and compare these results to already existing work.

Chapter 6 is the conclusion of the dissertation with key takeaways and suggestions for potential future work.

Chapter 2

Background

This chapter provides the necessary background that is needed for the later parts of this dissertation. We give a brief overview of machine learning, neural networks and convolutional neural networks before stepping into the realm of deepfake generation and detection. Finally, we give an overview of adversarial attacks and data poisoning methods used today as well as defenses against them.

2.1 Machine Learning and Neural Networks

Machine learning (ML) is the process of a computer program learning from some given data, such that it improves its performance on some specific task. There are a variety of ML techniques such as support vector machines, decision trees, k-nearest neighbors and neural networks. In this dissertation we focus on neural networks.

Neural networks transform their inputs by applying a sequence of affine transformations under a non-linearity [49]. These affine transformations are specified by weight and bias matrices, which are learned by minimizing an error or loss function measuring how well the network is performing. One can minimize the loss by using algorithms such as Stochastic Gradient Descent which calculates the gradients of the loss in respect to the weights and changes the weights in the direction of decreasing loss [49].

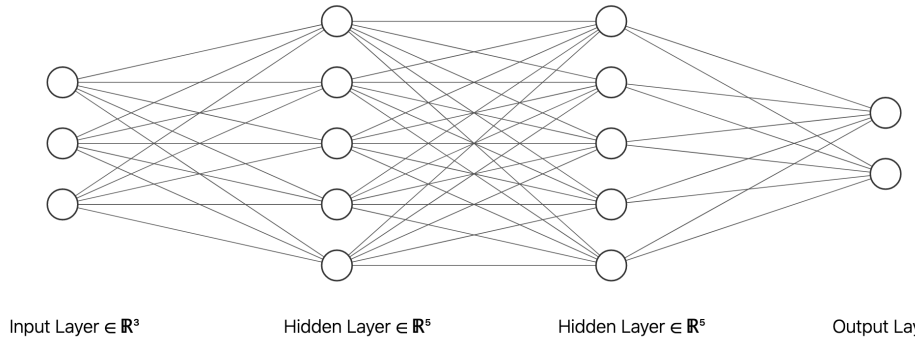


Figure 2.1: A fully connected neural network with 3 input nodes, 2 hidden layers with 5 nodes and 2 output nodes.

In Figure 2.1 we can see a neural network with 3 input nodes, 2 hidden layers with 5 nodes, and one output layer with 2 nodes. The lines in-between layers represent the weights which can be stored in matrices. The nodes apply a non-linear transformation to their inputs.

$$\mathbf{y} = f_3\left(\mathbf{W}^{(3)} f_2\left(\mathbf{W}^{(2)} f_1\left(\mathbf{W}^{(1)}\mathbf{x}\right)\right)\right) \tag{2.1}$$

$$\mathcal{L}(f,(\mathbf{X},\mathbf{y})) = \frac{1}{M}\sum_{m=1}^{M}(y^{(m)} - f(\mathbf{x}^{(m)}))^2 \tag{2.2}$$

$$\mathbf{W}_{t+1}^{(l)} = \mathbf{W}_{t}^{(l)} - \eta\frac{\partial\mathcal{L}}{\partial\mathbf{W}_{t}^{(l)}} \tag{2.3}$$

In Equation 2.1 we can see the matrix multiplications behind the neural network. $\mathbf{W}^{(i)}$ are the weights of the ith layer and f_i are the ith non-linear transformations applied to the input of the layer. The input given to the network is $\mathbf{x}$ and it returns the output $\mathbf{y}$.

In Equation 2.2 we can see an example of mean square error loss, which we want to minimize. f is the network we are calculating the error for, where $\mathbf{X}$ is the matrix containing all M input vectors $\mathbf{x}^{(m)}$ and $\mathbf{y}$ is the vector containing their corresponding labels $y^{(m)}$.

In Equation 2.3 we see the formula for gradient descent, with which we minimize the loss $\mathcal{L}$ by changing the weights $\mathbf{W}^{(i)}$ in the direction of decreasing loss, where η is the learning rate which can be tuned in a hyperparameter search.

2.2 Convolutional Neural Networks

Convolutional neural networks (CNNs) are a type of neural network, often used for image classification or signal processing. CNNs use convolutional layers to detect spatial relationships in data, such as edges or textures. Convolutional layers have filters, which are grids of learnable weights on which we can backpropagate. Those filters learn features such as edge or texture detection, as well as more abstract ones. A window of a specified size then travels across the input, moving by a chosen stride each time. CNNs also often consist of pooling layers in between convolutional layers, which reduce the dimensions of the convolutional layer's output and provide a degree of translation invariance. In most CNN architectures, multiple convolutional and pooling layers are then applied in sequence. For classification tasks, the output of the last convolutional layer is then flattened and passed to fully connected layers, similar to the neural network described in the previous section [49].

2.3 Deepfake Generation

Deepfakes are synthetic images, videos or audio [48]. Often, they are used for harmless purposes, but they can also be utilised to spread misinformation, impersonate people

and change original the original intentions of people's media content in a harmful way. For this dissertation we focus on image and video deepfakes, though audio deepfakes are an interesting topic with potential for future research [1].

State of the art deepfake generation is mostly done with deep learning models such as Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs) [48] [40]. Using datasets of faces, they can learn to generate new faces, or overlay certain faces or facial expressions onto a target. In Appendix A.1 we provide some more detail on GANs and VAEs, though they are not required to understand this dissertation.

In literature there are five commonly defined types of video/image deepfakes: face-swap, lip-syncing, facial reenactment, facial attribute manipulation and face synthesis [48] [40]. They differ in that some of them overlay a face onto another person (face-swap), while others manipulate certain parts of a target face (lip-syncing, facial reenactment, facial attribute manipulation), or synthesize an entirely new image (face synthesis). We show examples of face-swap, facial reenactment, facial attribute manipulation and face synthesis in the Appendix.

2.3.1 Deepfake Detection

The process of differentiating genuine media from deepfakes is called deepfake detection. Deepfake detection can be used to mitigate the spread of deepfakes and help verify whether videos or images are real. There are different methods used for deepfake detection including machine learning, statistical measurements, and blockchain methods, of which nowadays machine learning is the most common and effective [53]. Nearly all machine learning based detectors use CNNs and often achieve over 95% accuracy on deepfake detection datasets [60][48]. In this dissertation we will exclusively be targeting CNNs which seem to be the most likely models to be used in a production setting.

2.4 Adversarial Attacks in Machine Learning

An adversarial attack in the context of ML can target various stages of the ML pipeline, such as data collection, the training process or inference done by a trained model [56]. According to Vassilev et al. [73] adversarial attacks can have three main objectives: availability breakdown (make the model unavailable), integrity violations (make the model mispredict) or privacy compromise (gain information about training data). In this dissertation we examine a data poisoning attack that falls under the integrity violation category and targets the data collection and training process of the ML pipeline.

Integrity violation attacks can target all stages of the ML pipeline [56]. We can differentiate between inference phase and training phase attacks [56] [73]. Inference phase attacks tamper with the data that the model infers on after being trained. An example is putting specially crafted noise on stop signs to classify them as speed limits [26].

[1] OpenAI announced their Voice Engine a few days before submission of this dissertation, capable of mimicking the voice of a person with just 15 seconds of audio [55].

Training phase attacks focus on the data the model receives while it is training, which is what we focus on in this dissertation and expand on in Section 2.5.

We also differentiate between white-box and black-box attacks [73] [56]. In white-box attacks the adversary has some knowledge of the training data or the model, such as architecture, hyperparameters or weights. In black-box attacks the attacker does not have access to specific information about the target ML pipeline. The adversary may have knowledge about common model architectures found in the domain or can craft inputs for the model and gain information by observing its outputs [56] [73].

2.5 Data Poisoning

Data poisoning is a type of adversarial attack that takes advantage of the training data of a model. Data created by the adversary is inserted into the training set, which, when trained on, makes the model misclassify [72].

We can distinguish between targeted and untargeted poison attacks [72]. Untargeted poison attacks aim to degrade the overall performance of a model. Conversely, targeted poison attacks aim to change classification for just a target input while leaving overall performance unchanged.

We can also differentiate between clean-label and non-clean-label attacks. Clean-label attacks have no access to the labelling function in the ML pipeline, which means that poison data is assigned a label that a human would normally assign. Non-clean-label attacks do have access, and can thus decide on the poison's labels, no matter what it looks like to a human [72].

In the next chapter we go into an in-depth explanation of different techniques used in data poisoning and explain the feature collision poisoning attack that we will be using in this dissertation.

2.6 Defenses against Data Poisoning

There are a variety of techniques to defend against data poisoning and, according to Goldblum et al. [28], can be categorized into 5 different types of defense: Identifying Poisoned Data, Identifying Poisoned Models, Repairing Poisoned Models after Training, Preventing Poisoning during Training and Defenses for Federated Learning. "Identifying Poisoned Data" aims to detect poisons before training on them, with popular techniques being outlier detection on either the raw data or latent space of models [28]. "Identifying Poisoned Models" is a type of defense that tries to detect whether a model already is poisoned, with some defenses trying to reconstruct backdoor triggers in order to find out whether they are poisoned. "Repairing Poisoned Models" aims to repair poisoned networks, by removing backdoor triggers for example. Some of those defenses rely on already knowing which triggers are built into the model, while some do not have that requirement. "Preventing Poisoning" focuses on the training process and employs training strategies that are robust to poisoning. Lastly, "Defenses for Federated Learning" is a class of defense for federated learning systems. Federated learning is the

process of training a model with many different clients, of which some can potentially be malicious. This is a setting which requires specially crafted defenses, as there are no guarantees on data, training methods, or weight updates from any of the clients. In Table 2.1 we list a few common defense methods for each type of defense (adapted from Goldblum et al. [28]).

Defense Type	Defense Method
Identifiying Poisoned Data	Outliers in Input Space [23]
Identifiying Poisoned Data	Latent Space Signatures [12]
Identifiying Poisoned Data	Predictions Signatures [18]
Identifiying Poisoned Models	Trigger Reconstruction [14]
Identifiying Poisoned Models	Trigger-agnostic Detection [14]
Repairing Poisoned Models	Patching Known Triggers [59]
Repairing Poisoned Models	Trigger-agnostic Backdoor Removal [16]
Preventing Poisoning	Randomized Smoothing [61]
Preventing Poisoning	Majority Voting [38]
Preventing Poisoning	Differential Privacy [34]
Preventing Poisoning	Input Pre-processing [10]
Defenses for Fed. Learning	Robust Federated Aggregation [9]
Defenses for Fed. Learning	Robust Federated Training [7]
Defenses for Fed. Learning	Post-Training Defenses [74]

Table 2.1: A list of different defense methods, adapted from Goldblum et al. [28]

Some techniques require full access to any data which is a trade-off with privacy concerns, while other techniques sacrifice accuracy for the sake of security. Especially in Federated Learning, defenses rely on full access to weight updates which is a concern for data privacy [28]. There also is the issue of the computational resources required for some defense methods. Defenses proposed by Huang et al. [35] or Xu et al. [76] require the training of multiple poisoned and clean networks, which is massively resource-intensive and only feasible to implement for strong defenders. The defense proposed by Guo et al. [32] requires a dataset that is guaranteed to be clean from poisons, which might not be possible in many scenarios. Thus, current defenses have numerous weaknesses and vulnerabilities, still needing to be addressed.

Chapter 3

Datasets, Models and Attacks

In this chapter we first describe the datasets used for deepfake detection, followed by a detailed overview of the one we use for our experiments. We then provide an overview of networks commonly used in deepfake detection and describe the networks we use for our experiments. Finally, we present popular poison attacks as well as the Poison Frogs [66] attack that we are using.

3.1 Datasets

In this section we first give an overview of the general datasets commonly used for deepfake detection tasks. We then provide a detailed overview of the FaceForensics++ (FF++) [63] dataset that we use in our experiments.

3.1.1 Datasets for Deepfake Detection

There exist a variety of different datasets for deepfake generation and detection [48] [60] [40] [53] [5]. We distinguish between real datasets, with no deepfake methods applied to the content, and fake datasets, with deepfake methods applied to the content. There also are datasets containing both real and fake data where the real data often is the source of the fake data.

Table 3.1 summarizes popular datasets used in deepfake detection. There is variation in the size of the datasets, the way data is stored (image or video), the number of different people (identities) in the dataset and in the case of fake datasets, the number of deepfake methods applied to the dataset.

There are some important factors to consider when deciding on a dataset. Since we want to generalise the results of our poison attacks as much as possible, we want to maximize the amount and diversity of the data, the number of identities (number of unique people) in the dataset, as well as the number of deepfake methods applied. In addition to that, using a dataset that has been used in the context of adversarial attacks on deepfake detection before, would help us combine knowledge gained from our results with those of others.

Dataset	Type	Size	Identities	Methods
CASIA-WebFace [78]	Real	494,414 images	10,575	/
CelebA [47]	Real	202,599 images	10,177	/
VGGFace [57]	Real	2.6 million images	2,622	/
MegaFace [42]	Real	1 million images	690,000	/
Microsoft Celeb [33]	Real	10 million images	750	/
VGGFace2 [11]	Real	3.31 million images	9000	/
Flickr-Faces-HQ [41]	Real	70,000 images	/	/
DeepFake-TIMIT [43]	Both	640 videos	43	2
DeeperForensics1.0 [39]	Both	60,000 videos	1000	1
Celeb-DF [46]	Both	5,229 videos	59	1
DFFD [21]	Fake	300,000 images	/	1
iFakeFaceDB [52]	Fake	87,000 images	/	1
DFDC [24]	Both	124,000 videos	960	4
Vox-DeepFake [25]	Both	2.2 million videos	4000	3
FaceForensics++ [63]	Both	5000 videos	1000	2

Table 3.1: A list of datasets used in deepfake detection. Type: Real, Fake or Both. Size: number of images/videos in dataset. Identities: number of unique identities in the dataset. Methods: number of deepfake methods used to create data (face-swap, lip-syncing, puppet-master/facial reenactment, face synthesis and facial attribute manipulation). Different algorithms that belong to the same method are counted as only one method.

3.1.2 FaceForensics++

From Table 3.1 we can deduce that FF++ is the best dataset for our use case. It is 1.7TB large, just under the 2TB limit we have on the cluster we are using. It also has 1000 unique identities, making for a diverse set of people to train on. Not mentioned in Table 3.1, but also speaking for FF++, is that the resolution of the videos is high, with videos having a resolution from 640x480 up to 1920x1080. This makes it a more realistic dataset as most media circulated on the internet is of similar quality. In addition to that, it was used by the only other published poisoning attack [62] and the only published inference time adversarial attack [37] performed against deepfake detectors, to our knowledge.

FF++ contains 1000 original videos on which four different deepfake generation algorithms have been applied, resulting in 4000 deepfake videos. The original videos are all videos where a face is clear and unobstructed, though there can be multiple faces. They are taken from YouTube and contain news reports, home videos and vlogs, meaning there is a variety of different content. The deepfake generation algorithms applied are Deepfakes [1], Face2Face [71], FaceSwap [44] and NeuralTextures [70] (see Appendix for examples). The first three are different types of face-swap generation methods, which means they overlay a source person's face or expression onto a target person. NeuralTextures is a facial reenactment generation method, which changes the facial expressions of a target person to those of a source person. The target and source

identities are all from the real videos in the dataset. Having three different face-swap algorithms and one facial reenactment algorithm improves the diversity of our data and ability to generalise with the FF++ dataset even more. The videos come in three different qualities: raw, high and low. Due to storage and computing constraints we choose to go with high, which gets slightly lower classification accuracies on average than raw (99.26% vs 95.73%) but a lot higher than low (81.00%). The low accuracy for compressed videos comes from the fact that artifacts are introduced that make it harder to distinguish from real videos and deepfakes [63]. We extracted all frames from the videos, resulting in a total of 1,830,531 fake images with 509,128, 509,128, 406,138 and 406,137 images for Deepfakes, Face2Face, FaceSwap and NeuralTextures respectively. We also have 509,128 real images. Due to details in the FaceSwap and NeuralTextures algorithms we do not have a corresponding fake image for each real image.

We use the train-val-test splits provided with the FF++ dataset. Since the deepfakes are generated from the original data, we do not want a deepfake in a different split than the real video it was generated from, to prevent information about the test/val set from leaking to the training set. For example, having a real video in the train set and its corresponding fake in the test set might make the model classify the deepfake as real, since it will look similar to the real video in the train set. The FF++ given splits account for this and there is no ”cross contamination”. 720 of the real videos and all their corresponding fake videos are in the train split while validation and test both have 140 real videos and their corresponding fakes.

We used existing scripts for image extraction which we then modified to run extraction in parallel, which sped up extraction from taking multiple days to only a few hours. Because of the sheer size of the dataset we sometimes create corrupted images during extraction with the FFmpeg library. We created tests that are run after extraction, to automatically detect corrupted images and re-extract them. We also want to point out that we have accessed and used the dataset in line with the FaceForensics Terms of Use [2].

3.1.3 Limitations of FaceForensics++

While FF++ does contain two different deepfake generation methods (face-swap and facial reenactment) and four different generation algorithms, there are no lip-syncing, face synthesis or facial attribute manipulation deepfakes in the dataset. It also is limited to videos from YouTube, which might introduce compression artifacts or other properties specific to the YouTube platform. Lastly, it was released in 2019, meaning that there are some newer deepfake generation methods that have been released since, which might generate more realistic deepfakes, more likely to be used in the wild.

3.2 Models

In this section we first provide an overview of deepfake detection in relation to FF++. We then give detailed descriptions of the Xception [66] and Meso [3] models that we use for the remainder of the dissertation. We train the networks and provide the exact

training regime and results. These are necessary since we need to verify our training regimes are correct in order to correctly train on the poisoned datasets.

3.2.1 Deepfake Detectors on FF++

There have been a lot of CNN based deepfake detectors trained on the FF++ dataset [6] [64] [3] [53]. They all are able to achieve over 90% accuracy, with Meso [3] achieving over 95% accuracy. Rössler et al. [63] also train an Xception [17] model on the FF++ dataset, which is able to get over 99% accuracy on the raw FF++ data. In addition to that, Meso and Xception have both been the target of Russell [62] in their poisoning attack and Hussain et al. [37] in their inference time attack on deepfake detectors. Because of this we will be using Meso and Xception in our poisoning attacks.

3.2.2 Xception and Meso

Rössler et al. [63] trained two Xception networks [17] on their own FF++ dataset: Xception-Full, trained on full images, and Xception-Face, trained on face crops. Xception-Face achieves over 99% on the raw FF++ data and over 95% accuracy on the high-resolution data that we will be using. The original Xception proposed by Chollet [17] was trained on ImageNet [22] and thus we will be using those weights to initialise our network. The Xception network trained on ImageNet is called a feature extractor in this case, since we take advantage of the features that Xception has learned when training on ImageNet. Using a feature extractor is a common practice and decreases the amount of data needed and improves model performance [49]. We then proceed with a combination of finetuning and end-to-end training of this pre-trained Xception (our feature extractor).

MesoInception4, or Meso, originally proposed by Afchar et al. [3], is another CNN that achieves over 95% accuracy on the raw FF++ dataset and 98% on face-swap videos that Afchar et al. [3] generated themselves. For this dissertation we will only train it on FF++ data and no additional data created by Afchar et al. [3]. Meso is only trained on face-crops and we will not evaluate on a version trained on full images. Meso uses inception modules [69], which stack the output of several convolutional layers, with the aim of increasing the function space in which the network is optimized. It is a lot shallower than Xception at only 21 layers deep compared to 71.

Since there is no publicly available code to train Xception, we had to create training, image processing and data loading code from scratch. Because of the large size of the dataset, we faced a lot of problems with speed, memory management and dataloading. We improved the inference code published by Rössler et al. [63] which results in 10x speed ups in inference time and we updated it to work with PyTorch 2.2 from the original code using PyTorch 1.0.1. We plan on releasing a separate repository with our updated Xception code in order to let people use up-to-date libraries and ease further research.

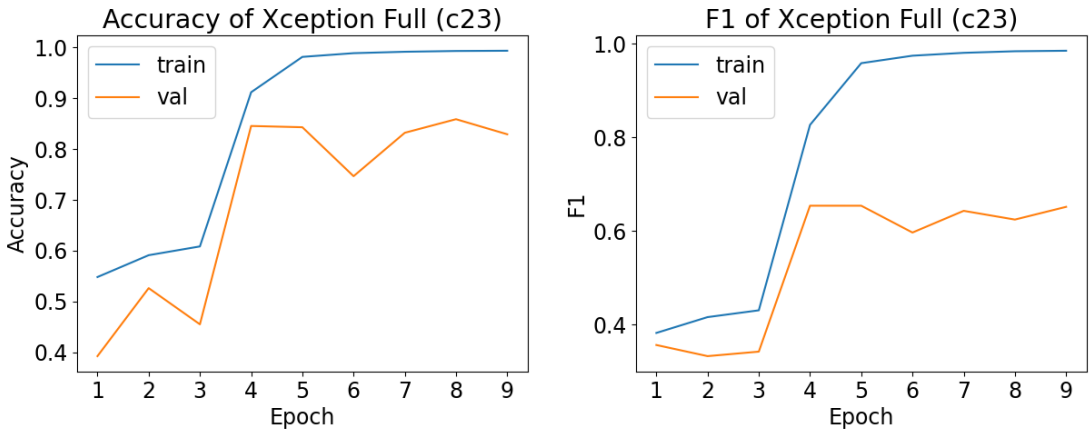
We also create code to train Meso on FF++ from scratch. The original Meso implementation was made with TensorFlow, thus we created a PyTorch 2.2 compatible version

and implemented training and inference code for Meso on FF++, which we also plan to release for others to access.

3.2.3 Training of Xception-Full

We first train Xception-Full, which uses the full images as input, without a face crop. We resize images to 299x299 and z-score normalize them. We use the same parameters and training regime as Rössler et al. [63] do, which is as follows: We start off with the original Xception [17], pre-trained on ImageNet [22]. We then proceed by replacing the output layer with a layer that has two outputs, one for *real* and one for *fake*. The training process can be separated into a finetuning part and an end-to-end training part. We start off training by freezing the weights of all layers except the last fully connected layer. We train for three epochs, after which we unfreeze all weights and train for another six epochs. Rössler et al. [63] do not explain why they only train the last fully connected layer and then move on to train the whole network. We theorize that it might lead to the fully connected layer stabilizing, leading to faster convergence when training on all layers.

For the whole training process we use a batch size of 32, learning rate of 0.0002, cross entropy loss (a weight of 4 for *real* images and 1 for *fake* images) and an Adam optimizer with default parameters ($\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 10^{-8}$).



(a) Accuracy of Xception-Full

(b) F1 of Xception-Full

Figure 3.1

We select the model that performs best on the validation set, using accuracy and F1 as evaluation metrics. This is at epoch 4 of training, where it has high accuracy and also a high F1 score on the validation set as shown in Figure 3.1b. We observe overfitting after epoch 4, also making us choose the weights from epoch 4. It is important to use F1 score, since FF++ is very unbalanced, with four times as many *fake* images than *real* ones.

Predicted/Actual	Real	Fake
Real	24228	15317
Fake	11214	118125
Accuracy	68.36%	88.52%

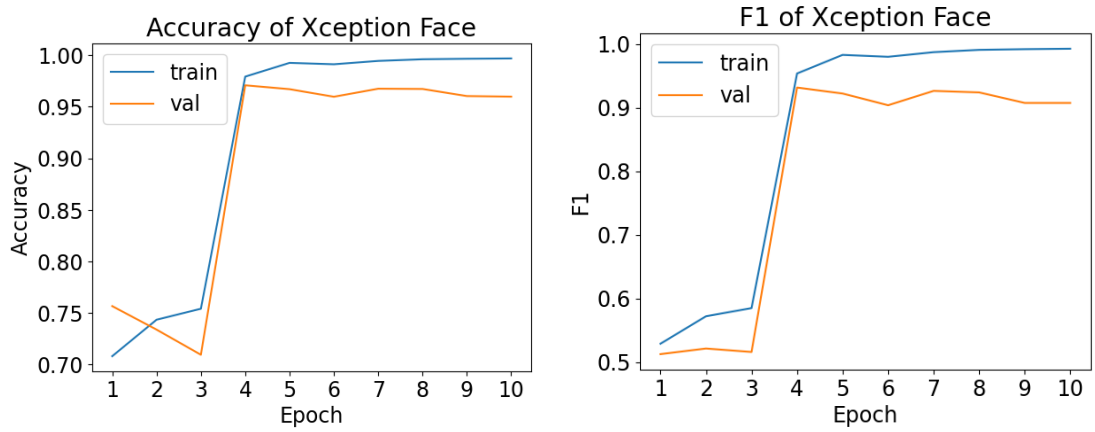
Table 3.2: Xception-Full test set confusion matrix

Evaluating Xception-Full on the test set and writing down the results in a confusion matrix (see Table 3.2) shows that a large proportion of *fake* images are correctly classified compared to a smaller proportion of *real* images. The overall accuracy we get on the test set is 84.29% which is in line with what Rössler et al. [63] get.

3.2.4 Training of Xception-Face

We train Xception-Face with the same regime as Xception-Full. The only change that we make is to train only on a cropped face in the image, for which we use the same face detection library as Shafahi et al. [66].

We again select the model that has the highest accuracy and high F1, which we can see in Figure 3.2 is at epoch 4. By checking the confusion matrix in Table 3.3 we can see that both *real* and *fake* images get classified correctly most of the time, with a better performance for *fake* images. The test set accuracy of Xception-Face is 94.53%, which is in line with what Rössler et al. [63] get.



(a) Accuracy of Xception-Face

(b) F1 of Xception-Face

Figure 3.2

Predicted/Actual	Real	Fake
Real	31187	4981
Fake	4255	128461
Accuracy	87.99%	96.27%

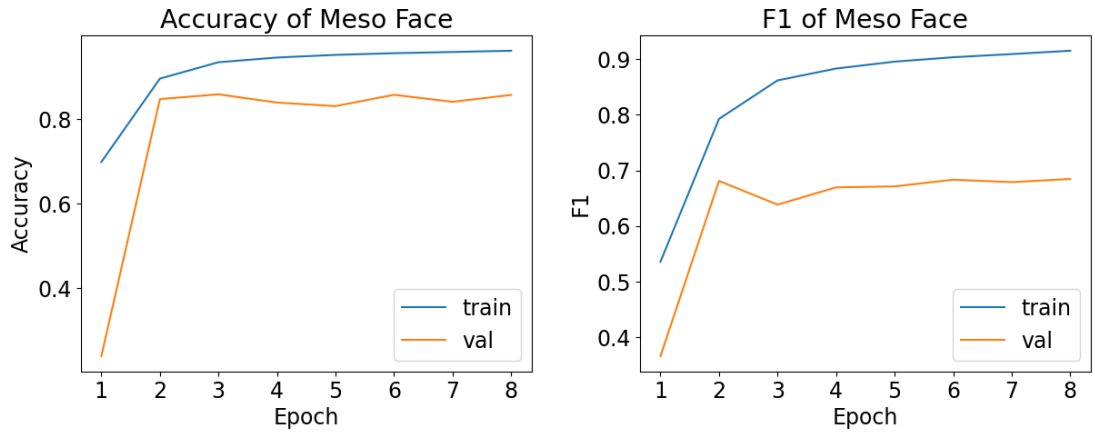
Table 3.3: Xception-Face test set confusion matrix

The improved performance in comparison to Xception-Full can be attributed to focusing on the face region, which is the main region affected during deepfake generation. This means the data given to Xception-Face is a lot more useful than the full images used by Xception-Full which contain a lot of irrelevant information.

3.2.5 Training of Meso-Face

The third model we train is Meso-Face, which is trained on the same face crops as Xception-Face with the only difference being that they are scaled to 256x256. We use the same training regime as Afchar et al. [3] and Shafahi et al. [66]. We train the network all layers unfrozen for the whole duration. We train for a total of 8 epochs using a batch size of 76. The learning rate starts off at 0.001 and every 1000 iterations we divide it by 10 until it reaches a value of 10^{-6}. We use cross entropy loss (a weight of 4 for *real* images and 1 for *fake* images) and an Adam optimizer with default parameters ($\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 10^{-8}$).

We again select the model with high accuracy and high F1 on the validation set, which is at epoch 6. We do note, that there only is very slight improvement on the validation set and high improvement on the training set after epoch 2, which is a sign of overfitting. In Table 3.4 we see that just like the other models, Meso classifies *fake* images more accurately than *real* ones. The test set accuracy of Meso-Face is 84.92%, which is in line with what Rössler et al. [63] report.



(a) Accuracy of Meso-Face

(b) F1 of Meso-Face

Figure 3.3

Predicted/Actual	Real	Fake
Real	23866	13884
Fake	11576	119558
Accuracy	67.34%	89.60%

Table 3.4: Meso-Face test set confusion matrix

3.2.6 Limitations of our models

There are limitations to the models we are using. First, not all of them perform very well on the FF++ dataset, with Xception-Full especially only getting 85% accuracy. All our models also have a slight bias to classifying *fake* images more accurately than *real* ones. In our case this means that it probably is harder to fool a detector into believing a *fake* image is actually *real*, but nevertheless it should be mentioned. We also only use deep models that are CNNs. While CNNs are state of the art and the most common, there are other techniques such as Recurrent Neural Networks or Generative Adversarial Networks used to detect deepfakes which might differ in their robustness against feature collision data poisoning [48].

3.3 Data Poisoning Attacks

As explained in Section 2.5, data poisoning attacks create poison data, that when inserted into a network's training set, cause it to misclassify either a target input or all inputs. There are different methods used in data poisoning attacks, with the most prominent ones being label flipping, backdoor triggers, feature collisions and bilevel optimization [28]. Current state of the art attacks use feature collisions, bilevel optimization or backdoor triggers [19]. All three methods of attack create new poison instances that need to be trained on, although backdoor triggers also require modification of input instances at inference. In Table 3.5 we show some of the most popular attacks, with a summary of whether they are clean-label, targeted and which settings they work for.

3.3.1 Label Flipping

Label flipping is the simplest method used in data poisoning attacks. Label flipping attacks change labels of training instances, while leaving the actual training instances unchanged [28]. Since they require access to the labelling function, label flipping attacks are not clean-label. An advantage of label flipping is that no artifacts are introduced into training data, which may happen with other data poisoning methods. Label flipping attacks can be targeted or untargeted. Often, they require a large amount of training data to be poisoned in order to be successful. Russell [62] use label flipping in their data poisoning attack on deepfake detectors.

Attack Type	Targeted	Clean-label	Black/White-box	Type
Label Flipping [75]	Both	No	White-box	LF
BadNet [30]	Yes	No	White-box	BD
Chen et al. [15]	Yes	No	Black-box	BD
Poison Frogs [66]	Yes	Yes	Both	FC
Convex Polytope [80]	Yes	Yes	Both	FC
Bullseye Polytope [4]	Yes	Yes	Both	FC
String Ray [68]	Yes	Yes	Both	FC
Black Card [31]	Yes	Yes	Both	FC
Witches' Brew [27]	Yes	Yes	Both	BO
Meta Poison [36]	Yes	Yes	Black-box	BO
Biggio et al. [8]	No	No	White-box	BO
Muñoz-González et al. [51]	No	No	Both	BO

Table 3.5: Capabilities of different data poisoning attacks. Targeted: Only misclassify a chosen input. Clean-label: Whether it needs access to the labelling function. Black/White-box: Whether it needs full access to the network weights or architecture. There isn't a specific definition of when something is white-box, but we explain in detail which access is needed for the attack we will perform. Type: LF - Label Flipping, BD: Backdoor Trigger, FC - Feature Collisions, BO - Bilevel Optimization

3.3.2 Backdoor Triggers

Backdoor trigger attacks insert poisons into the training set, such that at inference, any inputs containing a certain type of trigger are classified in some chosen way [28]. Backdoor trigger attacks are a combination of data poisoning and inference time attacks, since embedding a trigger requires access to instances at inference time. In the case of deepfake detection for example, any *fake* image could be classified as *real* if it contains a certain trigger e.g. in the form of a watermark.

3.3.3 Bilevel Optimization

Bilevel optimization attacks consider two functions that need to be optimized, where one function sets a constraint on another. Data poisoning in general can be described as a bilevel optimization problem, but only some attacks try optimizing this problem directly. In Equation 3.1 and 3.2 we can see the general formulation of the bilevel optimization problem for data poisoning, with notation by Goldblum et al. [28].

$$\min_{(x_p,y_p)\in C} \mathcal{L}(F(x^t,\theta'),y^{adv}) \tag{3.1}$$

$$\text{subject to } \theta' = \operatorname{argmin}_\theta \mathcal{L}(F(\{x_p\}\cup X_c,\theta),\{y_p\}\cup Y_c) \tag{3.2}$$

Equation 3.1 shows that we want to find the poison x_p with its corresponding label y_p which is in the allowable set of poisons C, that minimizes the loss between the target

instance x^t and the label we want it to be classified as, y^{adv} by the poisoned classifier defined by the weights θ'. In Equation 3.2 we show that we want the poisoned weights θ' to be the weights that minimize the loss between the clean train set X_c combined with the poison x_p and their labels Y_c and y_p. This equation can be easily adapted for the case of multiple poisons. The set C is defined depending on the specific bilevel optimization attack that is being used.

A less formal explanation of this is that we want to find a poison, such that after training on the poison and the clean data, our target instance x^t will be classified as the chosen label y^{adv}. This equation can easily be adapted to the case of multiple poisons instead of one.

There are a lot of different attacks that fall into the bilevel optimization category [27] [36] [8] [51]. They have been shown to be very effective in transfer learning and end-to-end training scenarios. A drawback of them is that they are very computationally expensive [28].

3.3.4 Feature Collision Attacks

Feature collision attacks take advantage of the feature space of a trained neural network or a feature extractor, by creating poison instances that are similar to the target instance in feature space. Feature space can be the representation of the input at any layer of the neural network, though in most feature collision attacks it is defined as the penultimate layer [66] [80] [4]. The reasoning behind creating poisons that are close to the target in feature space, is that even if they look differently to the human eye, a neural network would think of them as similar. This way we can have the same effect as flipping labels of instances very close to the target, without actually having to falsify any labels. Especially in the case of transfer learning, one often just finetunes the last fully connected layer of a network. Thus, if one creates poisons that are close to the target image in the penultimate layer of the network, the input of the layer being finetuned will always receive poisons that are close to the target, changing the classification boundaries around this target.

Formally we can define feature collision as follows (for consistency purposes we will adapt notation from Goldblum et al. [28]):

$$x_i^p = \text{argmin}_{x_i} ||f(x_i) - f(x^t)||_2^2 + \beta ||x_i - b_i||_2^2 \tag{3.3}$$

In Equation 3.3 x_i^p refers to the ith poison being created. f is the feature space transformation and b_i is the ith base. β is a parameter to regulate how close the poison should be to the base instance. In other words, we want to find a poison that is close to the target in feature space, while being close to the base instance (so that the assigned label will be that of the base instance).

A drawback of feature collision attacks is that they do not work as well as bilevel optimization attacks in the case of attacking an unknown model architecture or when end-to-end training a model [28]. Nevertheless, they are a lot more computationally efficient and thus easier for adversaries to use.

3.4 Poison Frogs

Poison Frogs is the first published feature collision attack, originally presented by Shafahi et al. [66]. The exact poisoning pipeline used by Poison Frogs is illustrated in Figure 3.4 It creates poisons by adding perturbation to base instances that cause the perturbed base instances to be close to the target instance in feature space. This feature space is taken from either a feature extractor, in the case of finetuning, or from an already trained model in the case of end-to-end training. Shafahi et al. [66] perform attacks in a transfer learning scenario, where training consists of finetuning a pre-trained model and an end-to-end training scenario. Extensions of Shafahi et al. [66]'s Poison Frogs attack are the Convex Polytope [80] and Bullseye Polytope [4] attacks. They create perturbed base instances, that additionally to being close to the target in feature space, also form a convex polytope surrounding the target, leading to an increased success rate.

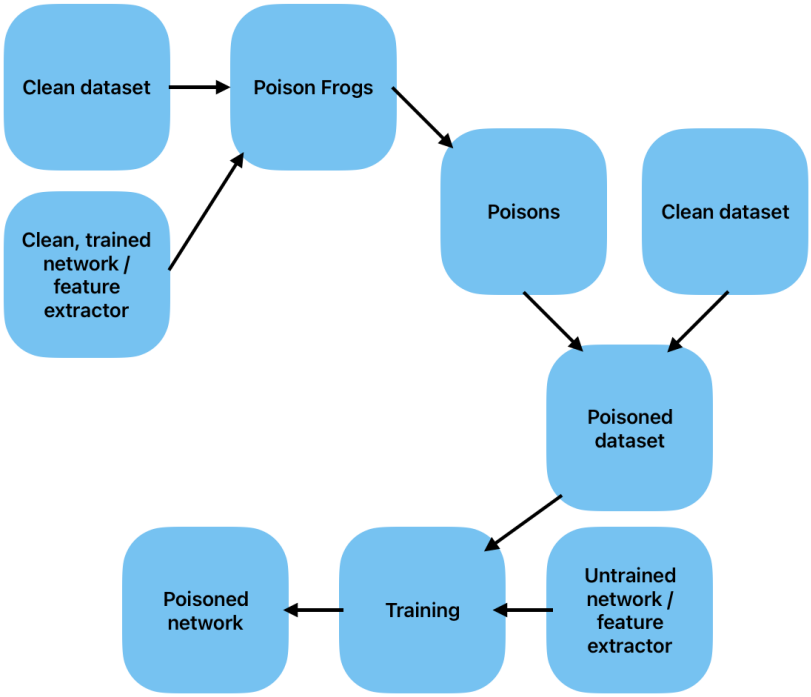


Figure 3.4: The poisoning pipeline for Poison Frogs. We start with a clean dataset and model, with which we create poisons. We then insert the poisons into the clean training data. Training on the poisoned data results in a poisoned model.

In the case of end-to-end training a network (meaning all layers are trained during training), we choose to take the feature space from a clean trained network, while in the case of finetuning a network (only last fully connected layer is trained), we choose to take the feature space from the pre-trained network, also called feature extractor. In this dissertation the feature space is the layer before the fully connected layer of the network. Shafahi et al. [66] use this layer because it is the input of the layer that will be finetuned in the finetuning case, since an input will have the same feature space representation

before and after finetuning. They do the same for end-to-end training, thus we will keep their setup and also use the layer before the fully connected layer of the network.

3.4.1 Why Poison Frogs?

We choose Poison Frogs as our attack for a variety of reasons. First of all, other feature collision attacks such as Convex Polytope [80] and Bullseye Polytope [4] require a lot of compute to create poisons, taking multiple days for large networks [4]. Bilevel optimization attacks, while very effective, also take multiple days for even the fastest attacks to generate poisons [28]. When finetuning networks based on pre-trained models, Poison Frogs just needs access to the pre-trained model, making it a realistic attack setting, especially compared to the label flipping attack by Russell [62]. Because other feature collision attacks are based on Poison Frogs, an effective Poison Frogs attack would imply that all feature collision attacks would also be effective, making our attack a good lower bound on the effectiveness of feature collision attacks. Schwarzschild et al. [65] evaluated different poisoning attacks, including feature collision attacks, in various settings. Their study showed that nearly all cases where Poison Frogs wasn't effective, other feature collision attacks weren't effective either, which means that we can at least give an indication of whether other feature collision attacks would be effective. For all these reasons, we choose Poison Frogs as the attack we will be using.

3.4.2 Inner Workings of Poison Frogs

The Poison Frogs attack starts off by choosing our target instance t. We then select n base instances $b_1, ..., b_n$, which are images from the test set and in the base class. For each of these base instances we find poisons $x_1^p, ..., x_n^p$ by optimizing Equation 3.3 which consists of two different parts. First, we want as small as possible distance between the feature space representation of the poison and the target, since we want to change decision boundaries near the target. Secondly, we want as small distance between the poison and the original base instance, since we want to add very little perturbation as to not raise suspicion. A successful attack is when the target x^t will be classified as the base class after retraining the classifier with the poisons. In our case we will focus on choosing targets from the *fake* class and base instances from the *real* class.

Algorithm 1 Poison Creation

Inputs: target x^t, base b, learning rate λ
Define $L(x) = ||f(x) - f(x^t)||_2^2$
for i $<$ maxIters **do**
    Forward-Step: $\widehat{x_i} = x_{i-1} - \lambda \nabla_x L(x_{i-1})$
    Backward-Step: $x_i = (\widehat{x_i} + \lambda\beta b)/(1 + \beta\lambda)$
end for

Algorithm 1 shows how Poison Frogs optimizes Equation 3.3. It applies a forward-backward splitting algorithm [29] to optimize it. A forward-backward splitting algorithm is used when trying to minimize a function which consists of two separate components. In our case it is the distance of poison and target in feature space, and the distance between poison and base. The forward step of the algorithm moves the poison

image closer to the target image in feature space. It does this by applying gradient descent to minimize the distance of poison and target in feature space. The backward step then adjusts for the poison to be closer to the base image. This is done by adding a very transparent version of the base image to the poison. By repeating this procedure, the poison ends up close to the target in feature space, while also being close to the base image to the human eye. In Figure 3.5 we illustrate how poisons move in feature space and their normal representation space during the attack.

Just as Shafahi et al. [66], we use the Frobenius Norm (default PyTorch matrix norm) to measure distance between images and Euclidean Norm when measuring distance in feature space, since the feature space will be represented by a vector (see appendix for Euclidean and Frobenius Norm).

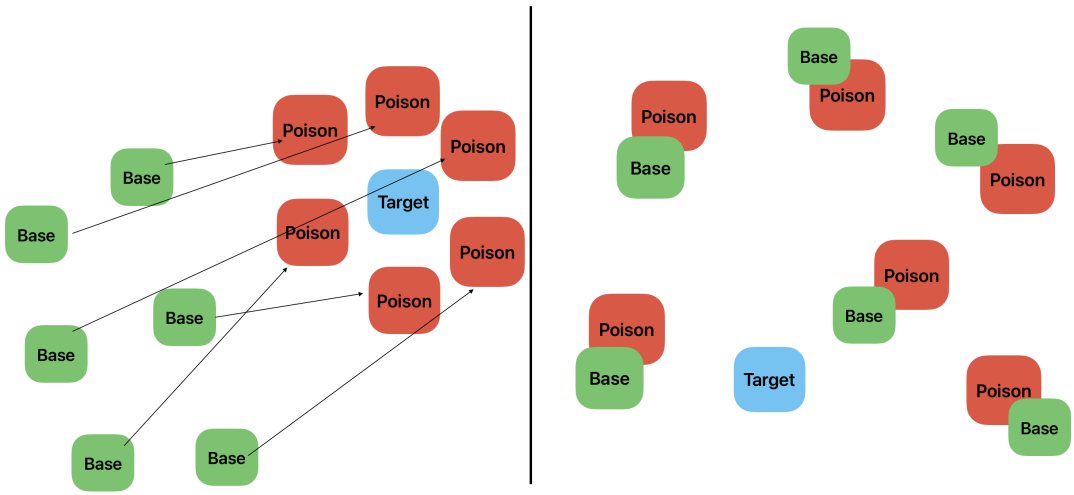


Figure 3.5: This figure illustrates how poisons move in space, compared to their base instance, with the left side showing feature space and the right side showing the normal pixel representation of an image. We can see that the poisons move far away from their bases in feature space, close to the target. However, their pixel representation doesn't move much and poisons are near their bases.

3.4.3 Limitations of Poison Frogs

There are limitations that come with using Poison Frogs as the attack used in our experiments. First of all, limiting ourselves to one attack limits the amount of generalization we can do about the effectiveness of other poisoning attacks, especially ones that do not use feature collisions. Secondly, Poison Frogs is weaker in the end-to-end training setting compared to other feature collision attacks like Bullseye Polytope [4] and bilevel optimization attacks [65]. This is because the feature space changes a lot when training all layers of a network and we do not have any guarantees of the poisoned network's feature space being similar to the clean network's with which we created the poisons. Thus, we can only create lower bounds on how effective poison attacks are in the end-to-end training setting.

Chapter 4

Experiments

In this chapter we present the experiments that we perform with the goal of determining the robustness of deepfake detectors against feature collision poison attacks. In all experiments we choose a *fake* image as the target, which we want the poisoned network to classify as *real* (*real* being the base class that we create the poisons from). We perform three experiments on up to three different networks as outlined in Table 4.1. We evaluate the performance of the attacks by comparing the probability of the poisoned classifier and the unpoisoned classifier from Chapter 3, assigning the *real* class to the target. In order to gain confidence in our results we repeat each experiment eight times, as this is the most we can do since each repeat requires training a whole network. There is a big range in how often poison attacks are repeated in literature with Russell [62] only doing 7 repeats, but Shafahi et al. [66] doing 1099.

Model	Poisons	Iterations	Max Distance	Setting
Xception-Full	20	2000	/	End-to-end
Xception-Full	50	2000	/	End-to-end
Xception-Full	50	8000	100	End-to-end
Xception-Face	50	8000	600	End-to-end
Meso-Face	50	8000	50	End-to-end
Xception-Full	50	8000	100	Finetuning
Xception-Face	50	8000	600	Finetuning

Table 4.1: Summary of all experiments. Poisons: Number of poisons generated. Iterations: Number of iterations done to create poison. Max Distance: Maximum distance a poison can be from the target, in feature space. Setting: End-to-end: Same training as Chapter 3. All layers unfrozen for at least some epochs. Finetuning: Only last fully connected layer is trained with poisoned dataset, others are frozen.

The first attack will be on Xception-Full where we create 20 and 50 poisons with 2000 iterations per poison, using the feature space from the trained Xception-Full in Chapter 3 to create the poisons. We train the whole network on the clean + poisoned data, with

the same training regime as in Chapter 3. Since this means all layers will be trained, this setting is called end-to-end as noted by Shafahi et al. [66]. We choose to investigate the number of poisons, since they have a big effect in previously performed feature collision attacks as well as other poisoning attacks [66] [62], which makes us want to investigate the specific effect of more poisons when attacking deepfake detectors.

We then move on to performing an attack in the same end-to-end setting on Xception-Full, Xception-Face and Meso-Face. For this attack we use 8000 iterations and set a maximum distance that the poison images can be from the target image in feature space. We explain the details behind this in Section 4.2. We choose to investigate the distance between poisons and target because it has been shown to influence the effectiveness of attacks, as close poisons have a stronger effect on classification boundaries near the target [4]. There even are extensions of Poison Frogs such as Bullseye Polytope [4] or Convex Polytope [80] which try positioning poisons in the most ideal position which partially consists of being near the target.

Finally, we perform an attack in a finetuning setting on Xception-Full and Xception-Face, where we only train the last fully connected layer (with all other layers frozen) instead of the whole network. We create poisons on the feature extractor that Xception-Full and Xception-Face are trained on, leaving all other parameters the same as for the previous experiment. This attack is more effective due to the feature space not being able to change during training, which we explain in depth in Section 4.3 [66] [65]. Because of this effectiveness, we choose to experiment in a finetuning setting to find out how vulnerable deepfake detectors are in this worst-case setting.

4.1 Baseline Attack

In our baseline attack we target Xception-Full with Poison Frogs in an end-to-end setting. We perform the attack with 20 poisons and 50 poisons, repeating them 8 times each, since this is the most we can reasonable do (even with our optimized training code, which took training from weeks down to a day). Having few repeats limits how sure we can be of the effectiveness of our attack, but we include standard errors in our results to reflect our uncertainty.

4.1.1 Attack Description

We attack the Xception-Full network trained in Section 3.2 with the assumption that the attacker can either access the weights of a trained Xception-Full or replicate the training process. In this attack we want to find out the effect of changing the number of poisons. Thus, we choose 20 and 50 poisons, as Shafahi et al. [66] use between 10-70 poisons in their end-to-end attacks. We select one *fake* image from the test set as our target and *real* images from the test set as our bases. We make sure to only select target images that are classified as *fake* with 99% probability by the network trained in Section 3.2, to make sure we aren't targeting easily misclassifiable images. We choose different ones for each repeat of the experiment. The goal of the attack is to classify the target image as *real* after retraining the network with the poisoned + clean data.

We use the default parameters used by Shafahi et al. [66] for the number of iterations and β: 2000 iterations per poison and aβ of $0.1 \cdot 2048^2/(299 \cdot 299)^2$ (feature space dimension = 2048, base dimension = 299 x 299). We choose 2000 iterations because this is in the range of iterations used by Shafahi et al. [66] (1000-12,000) and because through inspection of poisons (see Figure 4.4) we can see that poisons don't change a lot after 2000 iterations. We choose a learning rate of 0.0001 by running a hyperparameter search on values that are common in literature and settling on the value that results in low distance to target and base. Shafahi et al. [66] do not mention why they choose this formula to select beta, and we notice that the beta value is extremely small but manual inspection of poisons reveals that no obvious perturbations are visible (see Figure 4.3). In addition to that, changing the beta value in preliminary experiments did not seem to significantly impact the perturbations or poison distances for us.

The feature space function used to create the poisons is the output of the layer immediately preceding the final fully connected layer of the Xception-Full network trained in Section 3.2. This choice of feature space is motivated by a similar reasoning to that of Shafahi et al. [66]. The goal is to generate poisons that end up near the target instance in the feature space of the trained network. This allows the poisons to influence the decision boundaries around the target, causing the trained network's predictions to change.

We opt to use the feature space from the network trained in Section 3.2 because we are retraining all layers, rather than just fine-tuning the final layer using a pre-trained network. Since the feature space will evolve during the full training process, we want the poisons to be positioned near the final feature space, not just the initial one. Both Shafahi et al. [66] and Aghakhani et al. [4] also leverage the feature space of the fully trained network when performing end-to-end training attacks.

Once we created our poison images, we add them to the training set. We train Xception-Full from the pre-trained network on ImageNet, in exactly the same manner as we did in Section 3.2. We select the epoch with the highest accuracy to evaluate the effectiveness of the poison attack.

4.1.2 Attack results

For 20 poisons the average classification of the target image by the unpoisoned classifier assigns a 0.00052 probability to the *real* class. The average of the poisoned classifiers assigns a 0.00621 probability to the *real* class, with a standard error of 0.00565. Even though the average probability of the *real* class being assigned after poisoning increased slightly, this isn't a significant increase.

For the 50 poisons the average classification of the target image by the unpoisoned classifier assigns a 0.00019 probability to the *real* class. The average of the poisoned classifiers assigns a 0.00043 probability to the *real* class, with a standard error of 0.00035. As with the 20 poisons, even though we do have a small increase in the probability of the *real* class being assigned after poisoning, this isn't a meaningful increase and thus both attacks seem to be ineffective.

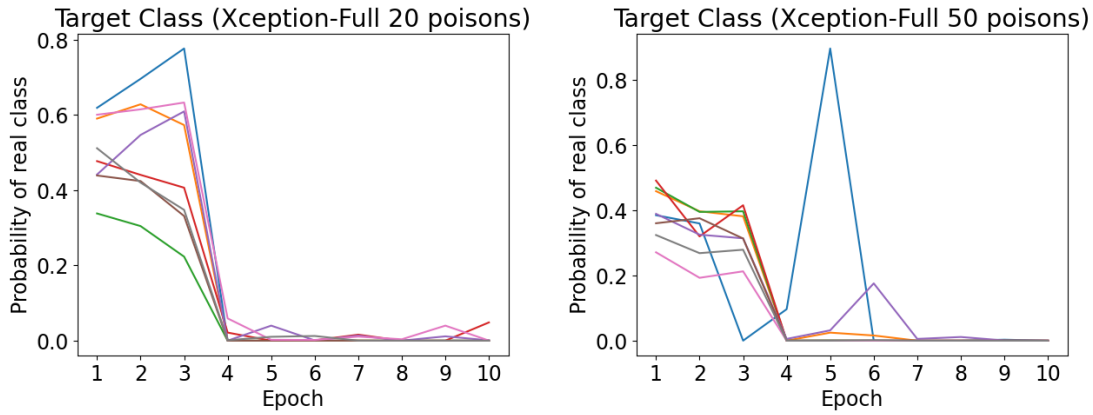


Figure 4.1: Probability that the target is classified as *real*, at each epoch of training the clean + poisoned data. Left: Xception-Full 20 poisons, Right: Xception-Full 50 poisons

In Figure 4.1 we can see that during the finetuning stage the target is classified as *real* by a significant amount. This drops immediately when all layers are unfrozen at epoch 4. The misclassification in the first 3 epochs doesn't happen due to the poisoning attack, since Figure 4.2 shows that the network is only about 50% accurate during the finetuning stage, and thus misclassification is to be expected. After epoch 3 the accuracy increases and the target isn't being misclassified anymore.

For the 50 poison case we see similar behaviour, with a big spike at epoch 5 for one target. This might just be random behaviour, since the network isn't always accurate. It is also interesting to note that the targets seem to be tighter together during the finetuning period, all between 0.3 and 0.5 probability of being *real*.

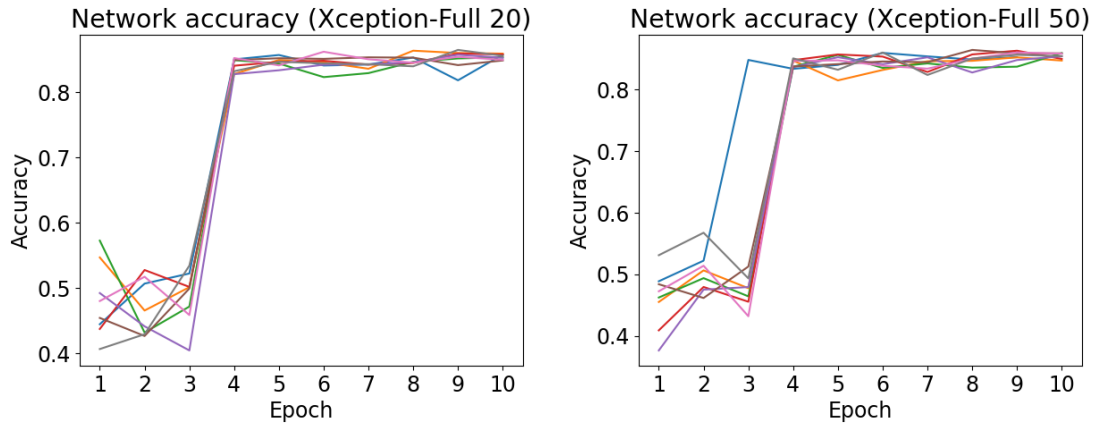


Figure 4.2: Overall validation accuracy per epoch of networks during poisoning. Left: Xception-Full 20 poisons, Right: Xception-Full 50 poisons

In Figure 4.3 we show a sample base image and its corresponding poison. It shows that there is a small amount of adversarial noise around the face of the person. This poison was selected for illustrative purposes with most other poisons showing nearly no noise at all. The distance between the poison and target in feature space is 170.26, compared

to the distance between the base and target in feature space is 1750.5, which is about 10 times more. The distance between poison and base image is just 2.0434, a bit more than the average for all poisons of 1.4664.

Figure 4.3: Xception-Full baseline sample base (left) and poison (right). The poison has slight perturbation as can be seen in the lower pictures, which are crops of the upper pictures.

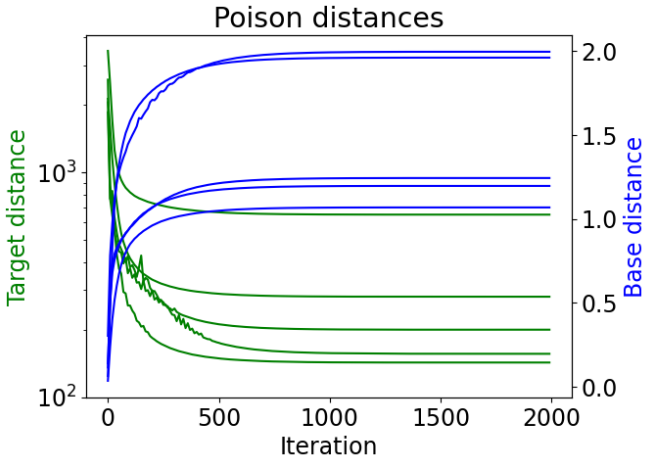


Figure 4.4: Distances of poison from target instance in feature space (green). Distances of poison from base instance (blue).

In Figure 4.4, for five randomly selected poisons, we can see the distance of the target

from the poison in feature space and the distance of the base from the poison instance, as we create our poisons. In all five of the poisons we show, we can see that the distance between target and poison in feature space decreases a lot, while the distance between base and target only increases very slightly. This means that the poison attack is creating poisons in the correct way, as described by Shafahi et al. [66].

Since the baseline attack seems to be ineffective, we skip evaluation on Xception-Face and Meso-Face. Due to time and compute constraints, we think it will be more insightful to directly move on towards a stronger attack.

4.2 Closer poisons

We saw in the previous section that more poisons generated with the same parameters didn't have a significant impact in the effectiveness of the poisoning attack. We now investigate the effect of poisons that are closer to the target in feature space.

4.2.1 Attack setup

We use a similar approach as in the previous section but make a few changes to create closer poison images. We select 50 poisons instead of 20, since the previous results were inconclusive and literature suggests that attacks with more poisons are more effective [66] [62]. We decide on a maximum distance that poisons can have to the target in feature space. The maximum distance gives us control over only selecting the closest poisons for the final poisoning and discarding poisons that are too far from the target. Since a poison with a small distance from the target in feature space will have a greater chance of changing the decision boundary near the target location in feature space, we want to minimise this distance.

For Xception-Full we decide on a maximum distance of 100, for Xception-Face of 600 and Meso-Face of 50. We choose the max distance values by inspecting the distances to the target from the previous experiment and run some preliminary experiments on Xception-Face and Meso-Face, from which we select the distance that about the top 10% of closest poisons fall under. We notice that the distance between poisons and target in feature space is a lot higher in Xception-Face than Xception-Full (600 vs 100 max distance). This is a sign of Xception-Face being able to distinguish the *real* class from the *fake* class better, which is reflected in Xception-Face's higher accuracy. Due to Meso-Face's different architecture we select a lower max distance of 50, since the feature space is a smaller vector in the Meso network compared to Xception. Since a lot of poisons only reach the maximum distance after a considerably higher number of iterations, we increase the number of iterations from 2000 to 8000 iterations. To accommodate the higher number of iterations we halve the poison learning rate every 1000 iterations, similar to learning rate scheduling when training a network. All other parameters are left the same as in the previous attack.

Shafahi et al. [66] or Aghakhani et al. [4] do broadly mention that nearer poisons should be more effective but to the best of our knowledge this is a novel modification and

there aren't any other feature collision attacks that have a maximum distance for poison selection.

4.2.2 Attack results

For Xception-Full the average probability of the *real* class being assigned to the target before poisoning is 0.00461. After poisoning the average probability of the *real* class being assigned is 2.49292e-5 with a standard error of 1.24040e-5.

For Xception-Face the average probability of the *real* class being assigned to the target before poisoning is 0.00046. After poisoning the average probability of the *real* class being assigned is 0.00149 with a standard error of 0.00137.

For Meso-Face the average probability of the *real* class being assigned to the target before poisoning is 0.00496. After poisoning the average probability of the *real* class being assigned is 0.00603 with a standard error of 0.00515.

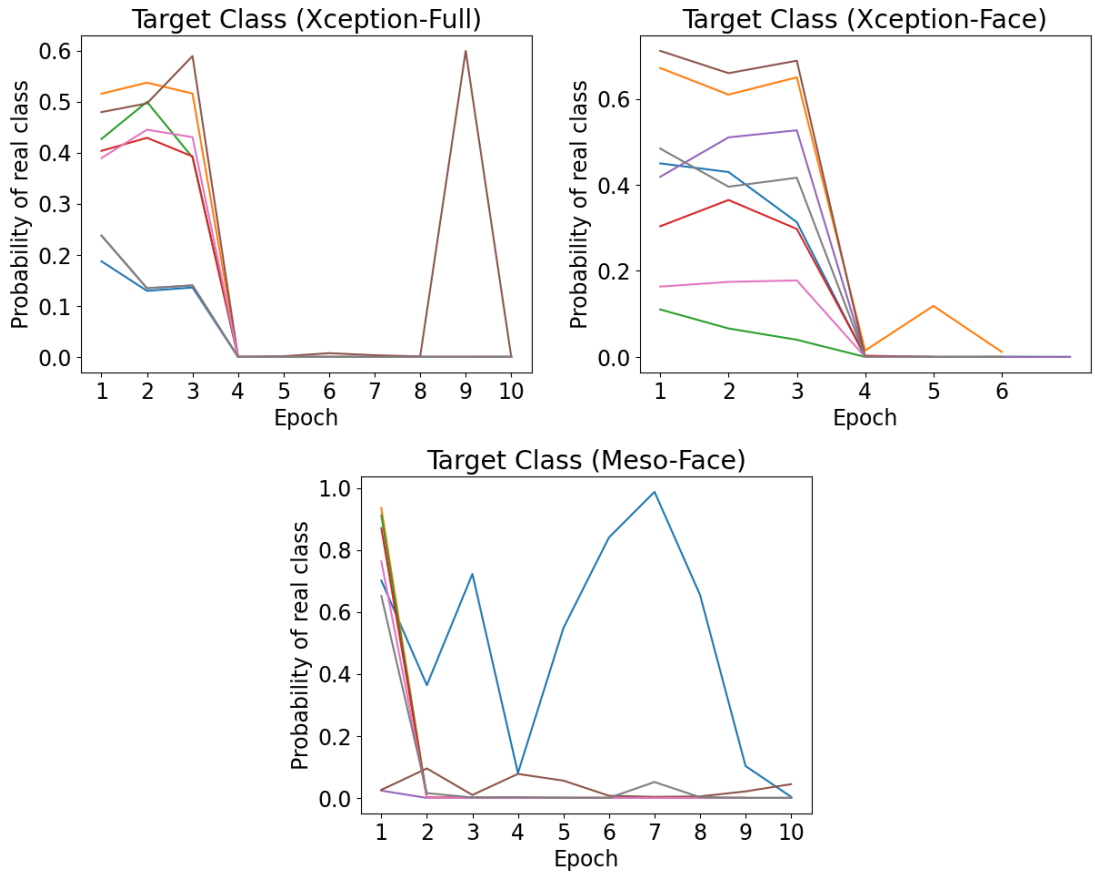


Figure 4.5: Probability that the target is classified as *real*, at each epoch of training the clean + poisoned data. Top left: Xception-Full, top right: Xception Face, Bottom: Meso-Face

Compared to our first attack, the closer poisons did not significantly improve the performance of our attack. For all 3 networks the poison attack didn't significantly

change the classification of the target class. Similarly to the previous experiment, we can see in Figure 4.5 that Xception-Full and Xception-Face both assign the *real* class to the targets at around 0.5 probability for the first 3 epochs while accuracy is low, but then drop off sharply once all layers of network are trained and accuracy increases (see Figure 4.6). Meso-Face assigns the *real* class with a very high probability to a lot of targets after the first epoch, but also a very low probability for a few targets. After the first epoch nearly all targets are classified as *fake*, except one that oscillates between being assigned as *real* and *fake*. Again, this can be explained by the accuracy increasing after 2 epochs, and Meso-Face never having 100% accuracy on the data. The oscillating behaviour is different from the spikes we've seen in the Xception networks, but it is inconclusive whether this is caused due to poisoning or general network misclassification.

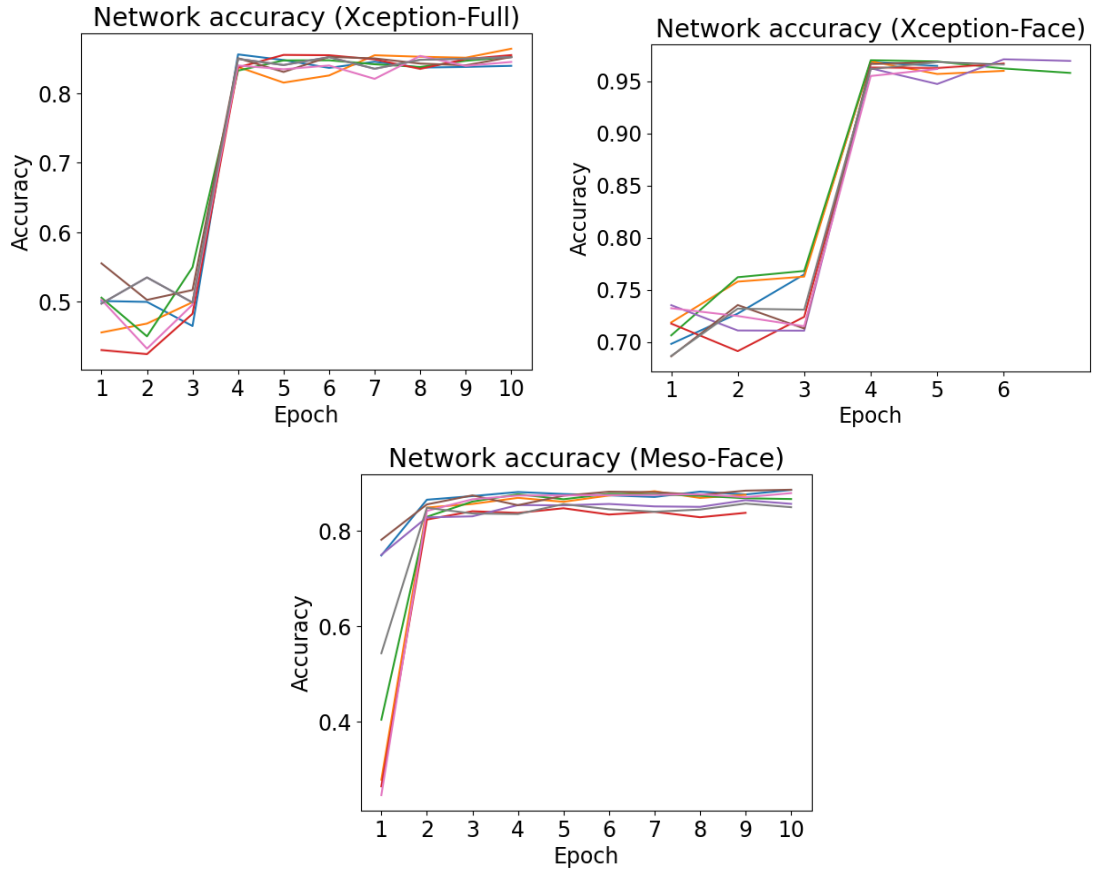


Figure 4.6: Overall validation accuracy per epoch of networks during poisoning. Top Left: Xception-Full, Top Right: Xception-Face, Bottom: Meso-Face

In Figure 4.7 we can see how the poison distances from the target in feature space and the distances from the bases changes during creation. We can see that Meso-Face is closer to the target in feature space in addition to also creating fewer adversarial perturbations in the image than Xception-Full. This is probably caused due to the area of the cropped face image that Meso-Face can add perturbations to being a lot smaller than for the full image that Xception-Full can add perturbations to. The average distances between poison and base are 8.6260, 21.9381 and 3.9869 for Xception-Full, Xception-Face and Meso-Face, meaning that they all are quite subtle.

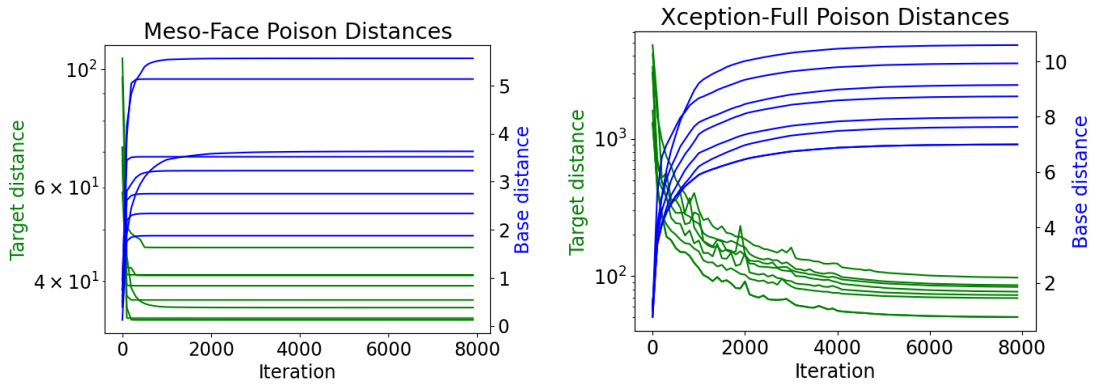


Figure 4.7: Distance of poison from target in feature space (green). Distance of poison from base (blue). Left: Meso-Face. Right: Xception-Full.

4.3 Attacks in a finetuning setting

From the previous attack we learned that Poison Frogs isn't effective in the end-to-end training setting. Since Xception-Full and Xception-Face both use a pre-trained network as a feature extractor, we perform an attack on them to show how effective the attack is in a finetuning setting, where we freeze all layers except the last one. Following Shafahi et al. [66], we select the output of this feature extractor as our feature space.

We choose to train a new Xception-Full for 5 epochs and a new Xception-Face for 3 epochs, because we observe that the networks do not seem to improve in accuracy after that, when just finetuning the last layer. We use these two clean finetuned network as our baseline to evaluate the difference in classification compared to the poisoned networks. The clean finetuned networks have lower accuracy, with the finetuned Xception-Full having an accuracy of 53.78% and the finetuned Xception-Face having an accuracy of 72.68%. This is to be expected, as there are fewer weights that the network can adjust, compared to training end-to-end.

In this scenario, the adversary is not required to know the training regime for how we finetune the detector. The only requirement is to know which feature extractor we use for finetuning, making it a powerful attack since a lot of publicly available and common networks are employed as pre-trained feature extractors.

Since we create poisons that are near the target in feature space, we should expect better results in a finetuning setting, where previous layers cannot change the feature space representation of our target and poisons over time. Conversely, leaving all layers trainable, as in the previous experiments, will change the feature space representation of our inputs, as weights in the network change with each backpropagation. Shafahi et al. [66] also get better results on finetuned networks than end-to-end trained networks.

4.3.1 Attack Results

For the clean Xception-Full the average probability of the *real* class being assigned to the target before poisoning is 0.50967. After poisoning, the average probability of the

real class being assigned is 0.53568 with a standard error of 0.07408.

For Xception-Face, the average probability of the *real* class being assigned to the target before poisoning is 0.31205. After poisoning, the average probability of the *real* class being assigned is 0.37471 with a standard error of 0.00389.

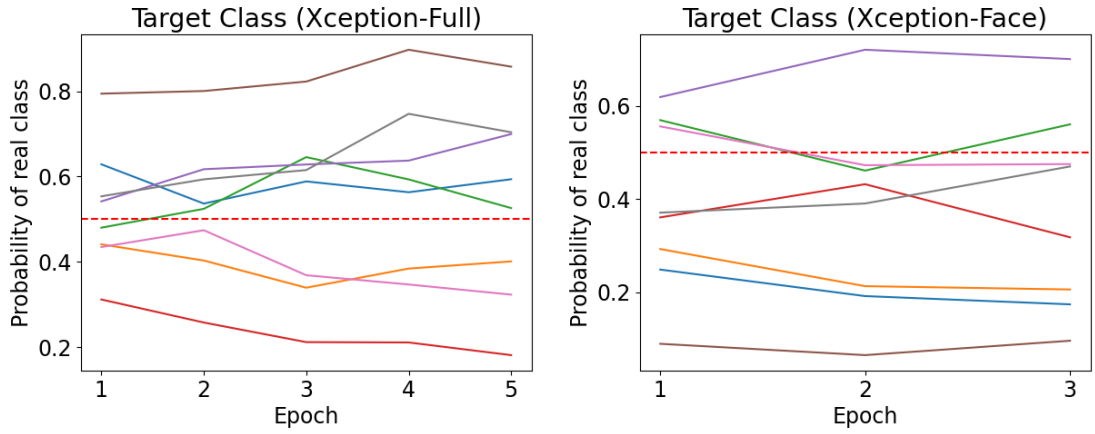


Figure 4.8: Probability that the target is classified as *real*, at each epoch of training the clean + poisoned data. Left: Xception-Full, Right: Xception-Face. Red dotted line = 0.5.

As we can see in Figure 4.8, during the whole finetuning process the targets are classified as a mix of *real* and *fake*, for both Xception-Full and Xception-Face. Figure 4.9 shows that Xception-Full doesn't improve much when finetuning over more epochs, with an accuracy of around 0.5. Xception-Face starts off with an accuracy of around 0.7 at epoch 1 and doesn't improve by a significant amount during finetuning.

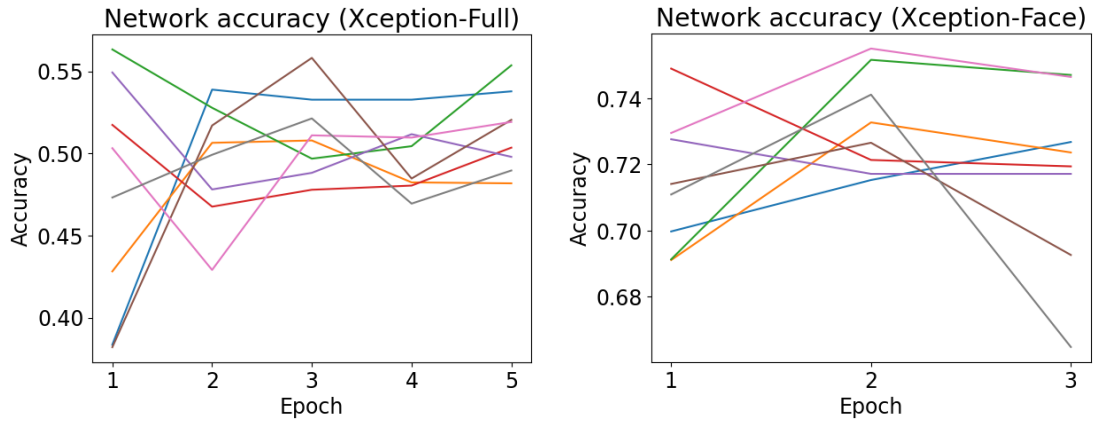


Figure 4.9: Overall validation accuracy per epoch of networks during poisoning. Left: Xception-Full, Right: Xception-Face

In Figure 4.10 we can see the distance of poisons from the target in feature space and the distance between poisons and the base images. For both Xception-Full and Xception-Face, the distance between poison and target in feature space drops considerably, with Xception-Face even having distances of below 50, which is a lot less than when using

the end-to-end trained network's feature space which had average distances of about 600. This is because the feature extractor isn't trained to differentiate between *fake* and *real* images, thus not keeping them apart in feature space.

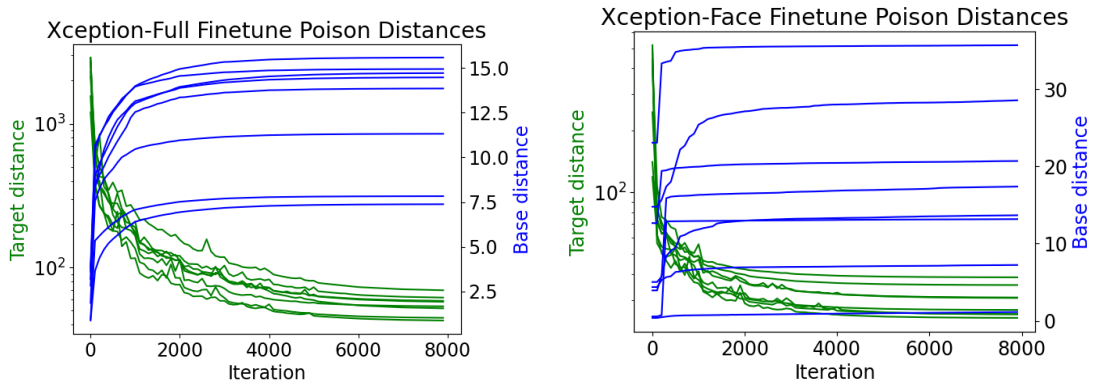


Figure 4.10: Distance of poison from target in feature space (green). Distance of poison from base (blue). Left: Xception-Full. Right: Xception-Face.

We can see in Figure 4.11 that the poisons created for Xception-Face still are subtle and not noticeable to the human eye, even as the distance from the bases is over 30 for some of them. This means that the clean-label property still holds even in the finetuning setting.

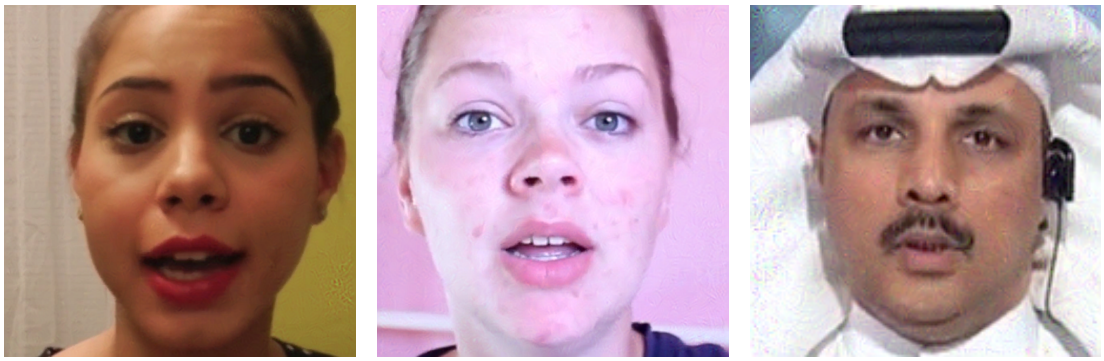


Figure 4.11: Faces of poisons created for Xception-Face in a finetuning setting.

Chapter 5

Evaluation, Limitations and Comparison to Existing Work

In this chapter we evaluate the experiments that we have performed in the previous chapter and compare them to each other. In addition to that we compare our findings to previous work done in the fields of data poisoning and also data poisoning on deepfake detectors specifically.

5.1 Evaluation of experiments

We first performed a baseline attack on Xception-Full in an end-to-end training setting, with 20 and 50 poisons. We then moved on to the next attack, which we performed on Xception-Full, Xception-Face and Meso-Face, with poisons closer to the target. Our last attack was on Xception-Full and Xception-Face in a finetuning setting where we left all layers frozen during training except the last fully connected layer.

Experiment	Before	After	Increase	Err
Xception-Full Baseline 20	0.00052	0.00621	0.00569	0.00538
Xception-Full Baseline 50	0.00019	0.00043	0.00023	0.00039
Xception-Full Closer Poisons	0.00461	2.49292e-5	-0.00458	0.00431
Xception-Face Closer Poisons	0.00046	0.00149	0.00103	0.00094
Meso-Face Closer Poisons	0.00496	0.00603	0.00106	0.00088
Xception-Full Finetuning	0.50967	0.53568	0.02601	0.02985
Xception-Face Finetuning	0.31205	0.37471	0.06266	0.03274

Table 5.1: Summary of all experiments. Before: Average probability of target image being classified as *real* on clean classifier. After: Average probability of target image being classified as *real* on poisoned classifier. Increase: Average increase in target being predicted as *real* after poisoning. Err: Standard error of mean of increase.

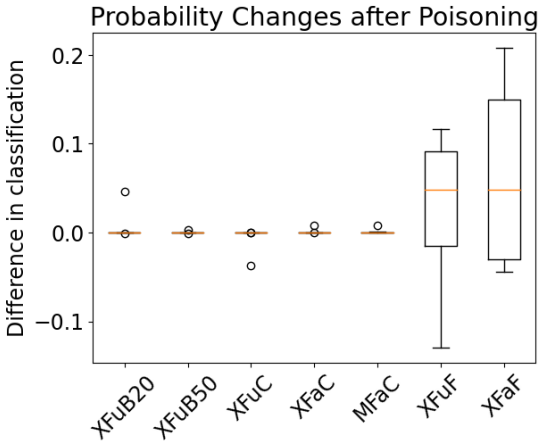


Figure 5.1: A boxplot showing the differences of classification before and after poisoning from 8 repeats of each experiment. XFuB20: Xception-Full Baseline 20 poisons, XFuB50: Xception-Full Baseline 50 Poisons, XFuS: Xception-Full Closer Poisons, XFaS: Xception-Face Closer Poisons, MFuS: Meso-Face Closer Poisons, XFuF: Xception-Full Finetune, XFaF: Xception-Face Finetune

5.1.1 Baseline Attack

Firstly, we performed a baseline attack on Xception-Full in an end-to-end setting, with 20 and 50 poisons, where the classifier was first finetuned on just the last fully connected layer for 3 epochs and then trained on all layers for the remaining 7 epochs. We concluded that this attack was not effective, as there was no significant increase in the probability of the target deepfake being classified as *real*. The average increase in the probability of the target being classified as *real* for 8 repeats of the experiment was just 0.00569.

We have three explanations for this attack not being effective. First of all, the feature space used to generate poisons is from a trained network. The networks we poison are not guaranteed to converge to the same exact feature space, thus rendering our attack less effective. Second of all, we have an extremely large training set, compared to previous work done using the Poison Frogs attack. This makes the number of poisons we need to create extremely high, which is computationally intensive. Lastly, having only two classes, also means that there are a large number of instances for each class, exacerbating the need of a lot of poison instances to cause noticeable changes in the network's decision boundaries.

5.1.2 Closer Poisons

We then moved on to the next experiment, which we performed on Xception-Full, Xception-Face and Meso-Face. The modification we made in this attack was to only allow poisons if they were under a selected distance from the target in feature space. We also selected 50 poisons, even though for us the effect of increasing the number of poisons was inconclusive, since literature suggests that more poisons strengthen attacks [66]. We concluded that this experiment also was not very effective, since the average

increases in the probability of the target being assigned the *real* class were -0.00458, 0.00103, 0.00106 for Xception-Full, Xception-Face and Meso-Face respectively.

We have a few explanations for why this attack wasn't effective. The simplest one is that the feature space of the poisoned network is very different from the one that we created the poisons from, thus not changing the decision boundaries near our target. A less likely explanation, but interesting to explore in future work, is that the closer poisons created a tighter polytope and thus smaller space for the decision boundaries to be changed, making it less likely for the target to end up in the *real* class. Aghakhani et al. [4] explain that centering the target in between all poisons at an equal distance makes attacks more effective, since it increases the space spanned by the poisons where the base class is assigned, as shown by their Bullseye Polytope attack.

5.1.3 Finetuning

The last attack was performed on Xception-Full and Xception-Face. The key difference to the previous attack was that we created our poisons based on the feature extractor that Xception-Full and Xception-Face are initially trained from and that we exclusively finetuned the last layer for all epochs, leaving all other layers frozen. As depicted in Figure 5.1 the finetuning attack was the most effective one out of all that we performed, with the attack being more effective on Xception-Face than Xception-Full. For Xception-Full the standard error on the increase in classification is too high to draw conclusion on whether it is a significant increase. For Xception-Face we have an average increase of 0.06266 with a standard error of 0.03274, which likely makes this attack effective, although more experiments are needed to increase the certainty of the effectiveness.

The main reason for this attack not being as effective on Xception-Full is that the classification accuracy of the finetuned Xception-Full is about 50%, making it equivalent to guessing between the two classes. Since it cannot train fit its clean data, it is not able to train on the poisons effectively either, and thus there is no significant change in decision boundaries around the target, which are mostly randomly assigned. Conversely, Xception-Face reaches an accuracy of about 72% when finetuned, which still isn't as good as Xception-Face trained in its normal training regime, but better than Xception-Full, and thus can fit the poison data better.

The attack is more effective in a finetuned setting than the end-to-end setting, because the feature space used to create the poisons is the same as the feature space during training, both being the Xception trained on ImageNet. This allowed the poisons to stay close to the target during the whole training process and change the classification boundaries near the target to match that of the poisons, in this case the *real* class.

5.2 Limitations of our work

There are a few limitations to our work. First, the number of experiments we performed often was not large enough to get an exact bound on how effective the attacks were. We were limited by the amount of compute, since it requires training a whole network for each experiment. Ideally, we would repeat each experiment more than 8 times.

We also only performed attacks with up to 50 poisons. Considering how large the dataset is, the fraction of poisons we used compared to the dataset size is very small. Doing attacks with a few hundred poisons might very well be more effective than 50, especially when attacking the finetuned Xception-Face network. In works by Shafahi et al. [66] or Russell [62] more poisons are used than in our case. Often poison attacks poison between 0.1% up to 30% of the whole training set, which would be tens of thousands of poisons for FF++ [19].

We only performed one feature collision attack, namely Poison Frogs. Even though this does deliver a good lower bound on the effectiveness of other feature collision attacks and is a good indication for which networks are vulnerable and which ones are not, it is worth using other attacks such as Convex Polytope [80] or Bullseye Polytope [4], especially since they generally perform better when training all layers of a network like we did in our first two experiments.

Using multiple datasets and more classifiers would also generalize the results more. Especially using a classifier that has only been finetuned on the last layer, but still achieves high accuracy, since the attacks in a finetuning setting seemed to be the most promising, but limited by network accuracy.

5.3 Comparison to previous work

The only other published data poisoning attack done on deepfake detectors was performed by Russell [62]. They also used the FF++ dataset in their experiments and attacked Xception-Face and Meso-Face. Unlike our clean-label Poison Frogs attack, they performed a label flipping attack, which needs access to the labelling function. They also didn't attack Xception-Full, which seemed the most robust to poison attacks in our experiments.

Previous work done by Russell [62] has shown that deepfake detectors are vulnerable to label flipping attacks, with between 50 - 250 poison instances. There are a few key differences between our approach and theirs. First of all, they train networks only on one type of deepfake at once. Since there are four different generation algorithms in FF++, they poison four versions of Xception-Face and Meso-Face. Their label flipping algorithm also specifically takes advantage of the fact that the deepfakes in FF++ have a source face and a target person, where the source face is transferred onto the target person. They create poisons by flipping the labels of deepfakes where the source face is equal to the target image that we are trying to classify as *real*.

Overall, our attack is more realistic for an adversary to perform, since the poisons generated would be labelled by a human. We also generated poisons based on feature space, which for the finetuning case is a publicly available pre-trained network. Because our poisons are engineered to be close to the target in feature space, we would have fewer poisons required than Russell [62], where the label flipping method often chooses images to flip which aren't very similar to the target in the network.

Hussain et al. [37] performed an inference time attack, that added perturbations to specific target deepfakes which then got classified as *real*. Our attack differs in that it

doesn't require any modification to the target image. In addition to that, their attack required access to the exact weights of the trained network that is used for inference, which we don't need. They also targeted Xception-Face and Meso-Face, but didn't target Xception-Full, which seems to be more robust since it can't fit data as well as the networks using face crops. It would be of interest to show whether Xception-Full also is more robust against inference time attacks, like it is against feature collision poison attacks.

Chapter 6

Conclusion

In this section we will summarise our findings from the previous chapters, as well as suggest future work to be undertaken.

6.1 Summary of our findings

We performed the feature collision attack, Poison Frogs [66], on three different deepfake detectors in end-to-end and finetuning settings. To the best of our knowledge, they were the first published feature collision poisoning attacks performed against deepfake detectors. In addition to that, it was the first poison attack performed against deepfake detectors, which did not assume access to the labelling function, and thus is the most realistic attack performed to date.

The main research question we had, was to find out how robust deepfake detectors are against feature collision data poisoning attacks. We conclude, that especially for finetuned deepfake detectors, using publicly available pre-trained models as feature extractors, there is a high risk and they definitely are vulnerable against feature collision attacks, as seen from our experiments in the finetuning setting. Especially models that are able to fit their data very well are of high risk, since they will fit poisons easily, as we saw when we compared Xception-Full with Xception-Face.

Deepfake detectors trained in an end-to-end setting are more robust to feature collision data poisoning attacks, with neither of our attacks set in an end-to-end setting being effective on any of the three models. Nevertheless, there is a need for performing experiments with more poisons, as previous work done by Russell [62] suggests that multiple hundred poisons can poison deepfake detectors with a label flipping attack, thus potentially working with a feature collision attack as well. In addition to that, there are feature collision attacks such as Bullseye Polytope by Aghakhani et al. [4], which are designed for the end-to-end setting and can potentially poison deepfake detectors.

In summary, deepfake detectors are vulnerable against feature collision data poisoning attacks, but end-to-end training or finetuning on non-publicly available pre-trained models increase the robustness of deepfake detectors.

6.2 Future work and ideas

There are a few interesting things to be explored in future work. First of all, since the finetuning attacks seemed to be the most promising, targeting fully finetuned detectors which can achieve high accuracy without training all layers is something to be explored. These attacks should be more effective since the detector will be able to fit the data, including the poisons, a lot better than the Xception networks we attacked.

Modifying Poison Frogs to work better in the end-to-end setting also is worth looking into. Either by implementing watermarks in the poisons as suggested by Shafahi et al. [66] or by using multiple feature spaces from different fully trained networks. Zhu et al. [80] show that this increases the effectiveness of poison attacks in the end-to-end setting, by reducing the impact of the changing feature space during training.

Increasing the number of poisons used in attacks should be looked into as well. Since we were limited by compute, we only used up to 50 poisons. Other attacks often poison between 0.1% up to 30% of the whole training set, which would be tens of thousands of poisons for FF++ [19]. Russell [62] poisons a few hundred samples in their label flipping attacks against deepfake detectors. This might mean that feature collision attacks also would work with a few hundred poisons.

Another thing to consider is to improve poisons in a different way than we did, when we made them closer to the target in feature space. Souri et al. [67] for example generate base instances using diffusion methods, from which they then create the poisons. This has been shown to drastically reduce the number of poisons required for a successful attack, compared to the normal Poison Frogs attack.

In addition to that, other feature collision poisoning attacks like Bullseye Polytope [4] or Black Card [31] should be tried on deepfake detectors, as they have better performance in end-to-end training settings, where Poison Frogs wasn't effective. Aghakhani et al. [4] also uses poisons using the feature space of multiple trained networks, improving performance in the end-to-end setting, which might improve performance against deepfake detectors. Other types of poison attacks like bilevel optimization attacks should also be tried against deepfake detectors, since they use a different method entirely for poisoning the network, potentially being a lot more effective than feature collisions.

It is also worth noting that defenses against feature collision attacks should be explored in the deepfake detection setting. We presented different defenses in Chapter 2 which can be explored in the future.

Overall, there are a lot of areas that are worth exploring in the future. Data poisoning against deepfake detection is an important issue, and thus it is critical to expand the existing literature on it.

Bibliography

[1] Deepfakes github. https://github.com/deepfakes/faceswap.

[2] Faceforensics, faceforensics++, and deepfakes detection dataset terms of use. https://kaldir.vc.in.tum.de/faceforensics_tos.pdf.

[3] Darius Afchar, Vincent Nozick, Junichi Yamagishi, and Isao Echizen. Mesonet: a compact facial video forgery detection network. In *2018 IEEE International Workshop on Information Forensics and Security (WIFS)*. IEEE, dec 2018. doi: 10.1109/wifs.2018.8630761. URL http://dx.doi.org/10.1109/WIFS.2018.8630761.

[4] Hojjat Aghakhani, Dongyu Meng, Yu-Xiang Wang, Christopher Kruegel, and Giovanni Vigna. Bullseye polytope: A scalable clean-label poisoning attack with improved transferability, 2021.

[5] Zahid Akhtar. Deepfakes generation and detection: A short survey. *Journal of Imaging*, 9(1), 2023. ISSN 2313-433X. doi: 10.3390/jimaging9010018. URL https://www.mdpi.com/2313-433X/9/1/18.

[6] Irene Amerini, Leonardo Galteri, Roberto Caldelli, and Alberto Del Bimbo. Deepfake video detection through optical flow based cnn. In *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pages 1205–1207, 2019. doi: 10.1109/ICCVW.2019.00152.

[7] Sebastien Andreina, Giorgia Azzurra Marson, Helen Möllering, and Ghassan Karame. Baffle: Backdoor detection via feedback-based federated learning, 2021.

[8] Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning attacks against support vector machines, 2013.

[9] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. Byzantine-tolerant machine learning, 2017.

[10] Eitan Borgnia, Valeriia Cherepanova, Liam Fowl, Amin Ghiasi, Jonas Geiping, Micah Goldblum, Tom Goldstein, and Arjun Gupta. Strong data augmentation sanitizes poisoning and backdoor attacks without an accuracy tradeoff, 2020.

[11] Qiong Cao, Li Shen, Weidi Xie, Omkar M. Parkhi, and Andrew Zisserman. Vggface2: A dataset for recognising faces across pose and age, 2018.

[12] Bryant Chen, Wilka Carvalho, Nathalie Baracaldo, Heiko Ludwig, Benjamin

Edwards, Taesung Lee, Ian Molloy, and Biplav Srivastava. Detecting backdoor attacks on deep neural networks by activation clustering, 2018.

[13] Heather Chen and Kathleen Magramo. Finance worker pays out $25 million after video call with deepfake 'chief financial officer'. *CNN*, January 2024.

[14] Huili Chen, Cheng Fu, Jishen Zhao, and Farinaz Koushanfar. Deepinspect: A black-box trojan detection and mitigation framework for deep neural networks. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 4658–4664. International Joint Conferences on Artificial Intelligence Organization, 7 2019. doi: 10.24963/ijcai.2019/647. URL https://doi.org/10.24963/ijcai.2019/647.

[15] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. Targeted backdoor attacks on deep learning systems using data poisoning, 2017.

[16] Xinyun Chen, Wenxiao Wang, Chris Bender, Yiming Ding, Ruoxi Jia, Bo Li, and Dawn Song. Refit: A unified watermark removal framework for deep learning systems with limited data. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, ASIA CCS '21. ACM, May 2021. doi: 10.1145/3433210.3453079. URL http://dx.doi.org/10.1145/3433210.3453079.

[17] Francois Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[18] Edward Chou, Florian Tramèr, and Giancarlo Pellegrino. Sentinet: Detecting localized universal attacks against deep learning systems, 2020.

[19] Antonio Emanuele Cinà, Kathrin Grosse, Ambra Demontis, Sebastiano Vascon, Werner Zellinger, Bernhard A. Moser, Alina Oprea, Battista Biggio, Marcello Pelillo, and Fabio Roli. Wild patterns reloaded: A survey of machine learning security against training data poisoning. *ACM Computing Surveys*, 55(13s):1–39, jul 2023. ISSN 1557-7341. doi: 10.1145/3585385. URL http://dx.doi.org/10.1145/3585385.

[20] European Commission. Ai act. 2024. URL https://digital-strategy.ec.europa.eu/en/policies/regulatory-framework-ai.

[21] Hao Dang, Feng Liu, Joel Stehouwer, Xiaoming Liu, and Anil Jain. On the detection of digital face manipulation, 2020.

[22] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.

[23] Ilias Diakonikolas, Gautam Kamath, Daniel M. Kane, Jerry Li, Jacob Steinhardt, and Alistair Stewart. Sever: A robust meta-algorithm for stochastic optimization, 2019.

[24] Brian Dolhansky, Joanna Bitton, Ben Pflaum, Jikuo Lu, Russ Howes, Menglin Wang, and Cristian Canton Ferrer. The deepfake detection challenge (dfdc) dataset. 2020.

[25] Xiaoyi Dong, Jianmin Bao, Dongdong Chen, Weiming Zhang, Nenghai Yu, Dong Chen, Fang Wen, and Baining Guo. Identity-driven deepfake detection, 2020.

[26] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. Robust physical-world attacks on deep learning visual classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[27] Jonas Geiping, Liam Fowl, W. Ronny Huang, Wojciech Czaja, Gavin Taylor, Michael Moeller, and Tom Goldstein. Witches' brew: Industrial scale data poisoning via gradient matching, 2021.

[28] Micah Goldblum, Dimitris Tsipras, Chulin Xie, Xinyun Chen, Avi Schwarzschild, Dawn Song, Aleksander Madry, Bo Li, and Tom Goldstein. Dataset security for machine learning: Data poisoning, backdoor attacks, and defenses. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(2):1563–1580, 2023. doi: 10.1109/TPAMI.2022.3162397.

[29] Tom Goldstein, Christoph Studer, and Richard Baraniuk. A field guide to forward-backward splitting with a fasta implementation, 2016.

[30] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain, 2019.

[31] Junfeng Guo and Cong Liu. *Practical Poisoning Attacks on Neural Networks*, pages 142–158. 11 2020. ISBN 978-3-030-58582-2. doi: 10.1007/978-3-030-58583-9_9.

[32] Wenbo Guo, Lun Wang, Xinyu Xing, Min Du, and Dawn Song. Tabor: A highly accurate approach to inspecting and restoring trojan backdoors in ai systems, 2019.

[33] Yandong Guo, Lei Zhang, Yuxiao Hu, Xiaodong He, and Jianfeng Gao. Ms-celeb-1m: A dataset and benchmark for large-scale face recognition. In *ECCV 2016*, August 2016. URL `https://www.microsoft.com/en-us/research/publication/ms-celeb-1m-dataset-benchmark-large-scale-face-recognition-2/`.

[34] Sanghyun Hong, Varun Chandrasekaran, Yiğitcan Kaya, Tudor Dumitraş, and Nicolas Papernot. On the effectiveness of mitigating data poisoning attacks with gradient shaping, 2020.

[35] Shanjiaoyang Huang, Weiqi Peng, Zhiwei Jia, and Zhuowen Tu. One-pixel signature: Characterizing cnn models for backdoor detection, 2020.

[36] W. Ronny Huang, Jonas Geiping, Liam Fowl, Gavin Taylor, and Tom Goldstein. Metapoison: Practical general-purpose clean-label data poisoning, 2021.

[37] Shehzeen Hussain, Paarth Neekhara, Malhar Jere, Farinaz Koushanfar, and Julian

McAuley. Adversarial deepfakes: Evaluating vulnerability of deepfake detectors to adversarial examples, 2020.

[38] Jinyuan Jia, Yupei Liu, Xiaoyu Cao, and Neil Zhenqiang Gong. Certified robustness of nearest neighbors against data poisoning and backdoor attacks, 2021.

[39] Liming Jiang, Ren Li, Wayne Wu, Chen Qian, and Chen Change Loy. Deeperforensics-1.0: A large-scale dataset for real-world face forgery detection, 2020.

[40] Felix Juefei-Xu, Run Wang, Yihao Huang, Qing Guo, Lei Ma, and Yang Liu. Countering malicious deepfakes: Survey, battleground, and horizon. 2022.

[41] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks, 2019.

[42] Ira Kemelmacher-Shlizerman, Steve Seitz, Daniel Miller, and Evan Brossard. The megaface benchmark: 1 million faces for recognition at scale, 2015.

[43] Pavel Korshunov and Sebastien Marcel. Deepfakes: a new threat to face recognition? assessment and detection, 2018.

[44] Marek Kowalski. Faceswap github. https://github.com/MarekKowalski/FaceSwap/.

[45] Nils C. Köbis, Barbora Doležalová, and Ivan Soraperra. Fooled twice: People cannot detect deepfakes but think they can. *iScience*, 24(11):103364, 2021. ISSN 2589-0042. doi: https://doi.org/10.1016/j.isci.2021.103364. URL `https://www.sciencedirect.com/science/article/pii/S2589004221013353`.

[46] Yuezun Li, Xin Yang, Pu Sun, Honggang Qi, and Siwei Lyu. Celeb-df: A large-scale challenging dataset for deepfake forensics, 2020.

[47] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild, 2015.

[48] Momina Masood, Mariam Nawaz, Khalid Mahmood Malik, Ali Javed, Aun Irtaza, and Hafiz Malik. Deepfakes generation and detection: state-of-the-art, open challenges, countermeasures, and way forward. *Applied Intelligence*, 2023.

[49] Kevin P. Murphy. *Probabilistic Machine Learning, An Introduction*. The MIT Press, 2022.

[50] Kevin P. Murphy. *Probabilistic Machine Learning, Advanced Topics*. The MIT Press, 2023.

[51] Luis Muñoz-González, Battista Biggio, Ambra Demontis, Andrea Paudice, Vasin Wongrassamee, Emil C. Lupu, and Fabio Roli. Towards poisoning of deep learning algorithms with back-gradient optimization, 2017.

[52] Joao C. Neves, Ruben Tolosana, Ruben Vera-Rodriguez, Vasco Lopes, Hugo Proenca, and Julian Fierrez. Ganprintr: Improved fakes and evaluation of the state of the art in face manipulation detection. *IEEE Journal of Selected Topics in Signal Processing*, 14(5):1038–1048, August 2020. ISSN 1941-0484. doi: 10.1109/jstsp.2020.3007250. URL `http://dx.doi.org/10.1109/JSTSP.2020.3007250`.

[53] Thanh Thi Nguyen, Quoc Viet Hung Nguyen, Dung Tien Nguyen, Duc Thanh Nguyen, Thien Huynh-The, Saeid Nahavandi, Thanh Tam Nguyen, Quoc-Viet Pham, and Cuong M. Nguyen. Deep learning for deepfakes creation and detection: A survey. *Computer Vision and Image Understanding*, 223:103525, 2022. ISSN 1077-3142. doi: https://doi.org/10.1016/j.cviu.2022.103525. URL https://www.sciencedirect.com/science/article/pii/S1077314222001114.

[54] Sophie J. Nightingale and Hany Farid. Ai-synthesized faces are indistinguishable from real faces and more trustworthy. *Proceedings of the National Academy of Sciences*, 119(8):e2120481119, 2022. doi: 10.1073/pnas.2120481119. URL https://www.pnas.org/doi/abs/10.1073/pnas.2120481119.

[55] OpenAI. Navigating the challenges and opportunities of synthetic voices, 2024. URL https://openai.com/blog/navigating-the-challenges-and-opportunities-of-synthetic-voices.

[56] Nicolas Papernot, Patrick McDaniel, Arunesh Sinha, and Michael P. Wellman. Sok: Security and privacy in machine learning. In *2018 IEEE European Symposium on Security and Privacy (EuroSP)*, pages 399–414, 2018. doi: 10.1109/EuroSP.2018.00035.

[57] Omkar M. Parkhi, Andrea Vedaldi, and Andrew Zisserman. Deep face recognition. In *British Machine Vision Conference*, 2015. URL https://api.semanticscholar.org/CorpusID:4637184.

[58] White House Press. Executive order on the safe, secure, and trustworthy development and use of artificial intelligence. 2023. URL https://www.whitehouse.gov/briefing-room/presidential-actions/2023/10/30/executive-order-on-the-safe-secure-and-trustworthy-development-and-use-of-

[59] Ximing Qiao, Yukun Yang, and Hai Li. Defending neural backdoors via generative distribution modeling, 2019.

[60] Md Shohel Rana, Mohammad Nur Nobi, Beddhu Murali, and Andrew H. Sung. Deepfake detection: A systematic literature review. *IEEE Access*, 10:25494–25513, 2022. doi: 10.1109/ACCESS.2022.3154404.

[61] Elan Rosenfeld, Ezra Winston, Pradeep Ravikumar, and J. Zico Kolter. Certified robustness to label-flipping attacks via randomized smoothing, 2020.

[62] Cole Dalton Russell. Defeat data poisoning attacks on facial recognition applications. 2021.

[63] Andreas Rössler, Davide Cozzolino, Luisa Verdoliva, Christian Riess, Justus Thies, and Matthias Nießner. Faceforensics++: Learning to detect manipulated facial images, 2019.

[64] Ekraam Sabir, Jiaxin Cheng, Ayush Jaiswal, Wael AbdAlmageed, Iacopo Masi, and Prem Natarajan. Recurrent convolutional strategies for face manipulation detection in videos, 2019.

[65] Avi Schwarzschild, Micah Goldblum, Arjun Gupta, John P Dickerson, and Tom

Goldstein. Just how toxic is data poisoning? a unified benchmark for backdoor and data poisoning attacks, 2021.

[66] Ali Shafahi, W. Ronny Huang, Mahyar Najibi, Octavian Suciu, Christoph Studer, Tudor Dumitras, and Tom Goldstein. Poison frogs! targeted clean-label poisoning attacks on neural networks, 2018.

[67] Hossein Souri, Arpit Bansal, Hamid Kazemi, Liam Fowl, Aniruddha Saha, Jonas Geiping, Andrew Gordon Wilson, Rama Chellappa, Tom Goldstein, and Micah Goldblum. Generating potent poisons and backdoors from scratch with guided diffusion, 2024.

[68] Octavian Suciu, Radu Mărginean, Yiğitcan Kaya, Hal Daumé III au2, and Tudor Dumitraş. Technical report: When does machine learning fail? generalized transferability for evasion and poisoning attacks, 2019.

[69] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions, 2014.

[70] Justus Thies, Michael Zollhöfer, and Matthias Nießner. Deferred neural rendering: Image synthesis using neural textures. *CoRR*, abs/1904.12356, 2019. URL http://arxiv.org/abs/1904.12356.

[71] Justus Thies, Michael Zollhöfer, Marc Stamminger, Christian Theobalt, and Matthias Nießner. Face2face: Real-time face capture and reenactment of RGB videos. *CoRR*, abs/2007.14808, 2020. URL https://arxiv.org/abs/2007.14808.

[72] Zhiyi Tian, Lei Cui, Jie Liang, and Shui Yu. A comprehensive survey on poisoning attacks and countermeasures in machine learning. *ACM Comput. Surv.*, 55(8), dec 2022. ISSN 0360-0300. doi: 10.1145/3551636. URL https://doi.org/10.1145/3551636.

[73] Apostol Vassilev, Alinea Oprea, Alie Fordyce, and Hyrum Anderson. Adversarial machine learning a taxonomy and terminology of attacks and mitigations. *NIST AI*, 2023.

[74] Chen Wu, Xian Yang, Sencun Zhu, and Prasenjit Mitra. Mitigating backdoor attacks in federated learning, 2021.

[75] Huang Xiao, Battista Biggio, Blaine Nelson, Han Xiao, Claudia Eckert, and Fabio Roli. Support vector machines under adversarial label contamination. *Neurocomputing*, 160:53–62, July 2015. ISSN 0925-2312. doi: 10.1016/j.neucom.2014.08.081. URL http://dx.doi.org/10.1016/j.neucom.2014.08.081.

[76] Xiaojun Xu, Qi Wang, Huichen Li, Nikita Borisov, Carl A. Gunter, and Bo Li. Detecting ai trojans using meta neural analysis, 2020.

[77] Yifan Yao, Jinhao Duan, Kaidi Xu, Yuanfang Cai, Zhibo Sun, and Yue Zhang. A survey on large language model (llm) security and privacy: The good, the bad, and the ugly. *High-Confidence Computing*, page 100211, 2024. ISSN 2667-2952. doi:

https://doi.org/10.1016/j.hcc.2024.100211. URL https://www.sciencedirect.com/science/article/pii/S266729522400014X.

[78] Dong Yi, Zhen Lei, Shengcai Liao, and Stan Z. Li. Learning face representation from scratch, 2014.

[79] Eliezer Yudkowsky. The ai alignment problem: why it is hard, and where to start. *Symbolic Systems Distinguished Speaker*, 4, 2016.

[80] Chen Zhu, W. Ronny Huang, Ali Shafahi, Hengduo Li, Gavin Taylor, Christoph Studer, and Tom Goldstein. Transferable clean-label poisoning attacks on deep neural nets, 2019.

Appendix A

Deepfake Generation

A.1 Generative Adversarial Networks and Variational Autoencoders

A.1.1 Generative Adversarial Networks

Generative Adversarial Networks (GANs) are networks used to create new data that is similar to a given dataset. A GAN consists of a generator and a discriminator. The generator takes random noise as an input and trains with the aim to generate data that is similar to the one found in the given dataset. The discriminator tries to differentiate the data from the generator and the data from the given dataset. The feedback loop from the generator trying to fool the discriminator and the discriminator trying to differentiate between generated images improves the performance of both [50].

A.1.2 Variational Autoencoders

Variational Autoencoders (VAEs) consist of two components, the encoder and the decoder. The encoder takes an input and maps it into a lower-dimensional latent space, typically defined by a mean and variance. This context vector is then passed to the decoder, which aims to reconstruct the input given to the encoder. During training the VAE has the goal to minimize the difference between the input and output while also keeping the latent space similar to a unit Gaussian distribution, though a regularization term called the Kullback-Leibler divergence.

A.2 Deepfake Examples

Figure A.1: Face-Swap: Source image, target face, deepfake [48]

Figure A.2: Facial reenactment: Source image, target face, deepfake [48]

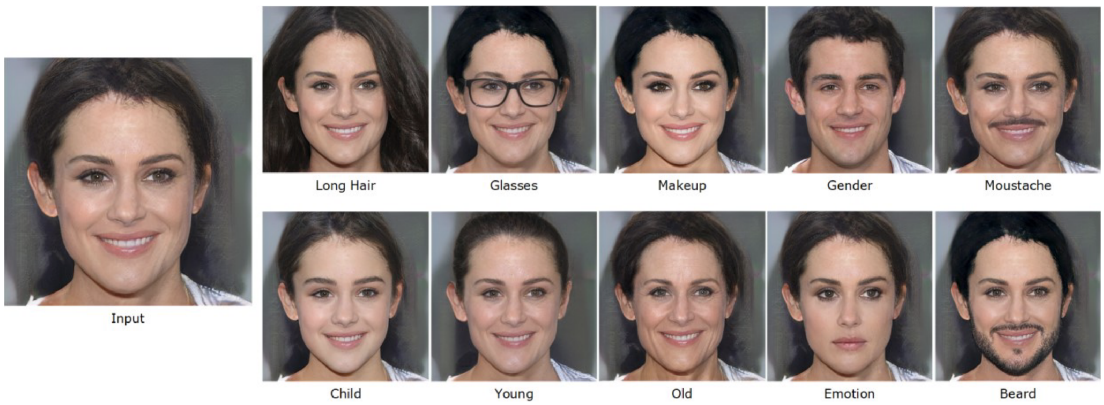


Figure A.3: Facial attribute manipulation [48]

Figure A.4: Synthetically generated faces [54]

Appendix B

Norms

B.0.1 Euclidean Norm

The Euclidean Norm for the vector $\mathbf{x}$ is:

$$\|\mathbf{x}\|_2^2 = \sqrt{\mathbf{x}^T\mathbf{x}} \tag{B.1}$$

B.0.2 Frobenius Norm

The Frobenius Norm of matrix $\mathbf{A}$ is:

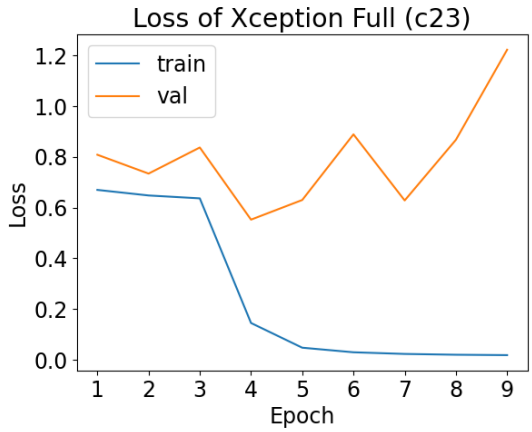
$$\|\mathbf{A}\|_F = \sqrt{\text{tr}(\mathbf{A}\mathbf{A}^H)} \tag{B.2}$$

where $\mathbf{A}^H$ is the conjugate transpose of $\mathbf{A}$. If $\mathbf{A}$ is a real matrix, this is just equal to $\mathbf{A}^T$.

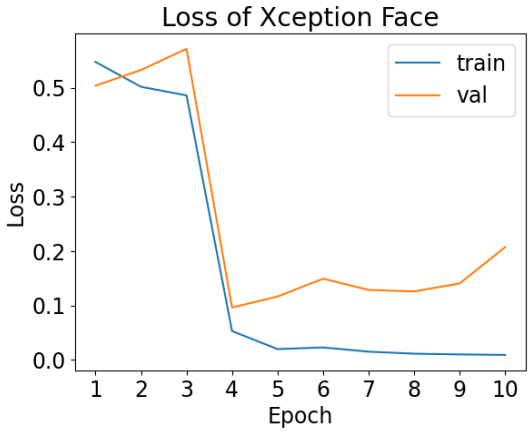
Appendix C

Xception-Full, Xception-Face and Meso-Face

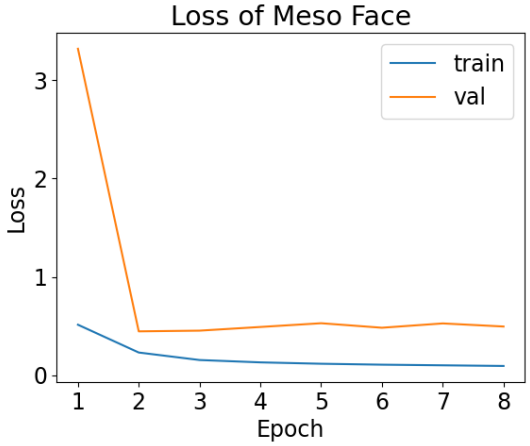
C.1 Training Loss of Xception-Full, Xception-Face and Meso-Face



(a) Loss of Xception-Full



(b) Loss of Xception-Face



(c) Loss of Meso-Face

Appendix D

FaceForensics++

D.1 Sample Poisons

Figure D.1: Samples from FF++. Top to bottom: Real, Deepfakes, Face2Face, FaceSwap, NeuralTextures