Improving Security in Facial Biometrics: Views from both Attacker Side and Defender Side

by

NGUYEN Hong Huy

Dissertation

submitted to the Department of Informatics
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

The Graduate University for Advanced Studies, SOKENDAI
March 2022

Acknowledgements

First, I am deeply grateful to my advisor, Professor Isao Echizen, and my sub-advisor, Professor Junichi Yamagishi, for their continuous support and guidance during all stages of my work. Their encouragement greatly helped me to overcome many difficulties along the way.

Second, I would like to thank Prof. Vincent Nozick of the Université Paris-Est Marne-la-Vallée, Prof. Minoru Kuribayashi of Okayama University, and Prof. Sébastien Marcel of the Idiap Research Institute for their valuable support during our joint projects.

Third, I would like to express my gratitude to the members of my advisory committee, Prof. Shin'ichi Satoh, Prof. Satoshi Ikehata, Prof. Yinquiang Zheng, and Prof. Koichi Itoin, for their invaluable advice and guidance.

Fourth, I would like to thank the Japan Ministry of Education, Culture, Sports, Science and Technology (MEXT) for their scholarship support and the Japan Society for the Promotion of Science and the Japan Science and Technology Agency for funding our projects.

Fifth, I am fortunate to have great lab mates and friends who have shared a lot with me in both research and daily life. Special thanks go to Dr. Hoang-Quoc Nguyen-Son, Dr. Ngoc-Dung T. Tieu, Dr. Fuming Fang, Dr. Trung-Nghia Le, Dr. Tiago de Freitas Pereira, Dr. Amir Mohammadi, Dr. Minh-Duc Vo, and Dr. Tri-Phuc Nguyen. I also received invaluable support from the Graduate University for Advanced Studies, SOKENDAI, the National Institute of Informatics, Japan, and our labs' secretaries (Ms. Yumiko Seino, Ms. Miki Nihei, Ms. Mai Saito, and Ms. Makiko Kuwahara).

Sixth, I would like to thank my father, my mother, my wife, and my daughter for their constant love and trust. Without them, I would not have been able to finish my Ph.D. degree here in Japan.

Abstract

Biometric authentication is becoming widely used, especially in handheld devices. The face is commonly used as a biometric identifier because of its contactless property. Thanks to recent technological advances in both hardware and software, facial biometric identification is well on the way to replacing traditional password authentication, which is inconvenient. Unfortunately, these advanced technologies can also be used to attack biometric authentication systems. For example, deep neural networks can be used to generate realistic images, videos, and speech. Synthetic and manipulated images and videos created in this way are called "deepfakes." They can be used to deceive face authentication systems (by attacking the integrated face recognition system) besides being used to create fake news and to impersonate or harass individuals. As a result, dealing with deepfakes is a vital task in facial biometrics. The countless battles between attackers and defenders have resulted in continuous improvements that make both sides stronger. This philosophy is the principal motivation for the work reported in this thesis. By standing on both the attacker and defender sides, we can simultaneously identify crucial problems in facial biometrics and provide solutions to make it more secure.

From the attacker side, we discuss the robustness of face recognition systems under a wolf attack using generated images that could match multiple enrolled user templates. Since being introduced in 2007 for fingerprints, wolf attacks have been widely used against fingerprint- and finger-vein-based authentication systems. Motivated by the use of an evolutionary algorithm along with a generative adversarial network to generate wolf partial fingerprints ("master prints"), we have enhanced the algorithm so that it can generate high-resolution high-quality master faces. Our experimental results demonstrated that, even with limited resources and using only pretrained models available on the Internet, attackers are able to initiate master face attacks. Our generated master faces can be used to attack various types of face recognition systems in white-box, gray-box, and even black-box scenarios. Another contribution is that we identify the limitations of face recognition systems by analyzing the distributions of the face embedding spaces of the systems, then suggest some improvements.

From the defender side, we first present a detector that works with various types of computer-generated and manipulated images and videos, commonly known as "deepfakes." The proposed detector uses a capsule network, which is an upgraded version of the traditional convolutional neural network (CNN). The performance of traditional CNNs can be improved by increasing their depth and/or their width, adding more internal connections, or fusing several features or predicted probabilities from multiple CNNs. Consequently, they become bigger, consume more memory and computation power, and require more training data. Thanks to the use of a dynamic routing algorithm, our capsule-network-based detector – namely Capsule-Forensics – has fewer parameters than traditional CNNs with similar performance. To further the understanding of the Capsule-Forensics detector, we visualize the activation of its components as a means to improve its explainability.

Next, we discuss locating manipulated regions (*i.e.*, performing segmentation), which is important when dealing with fake images and videos. We designed a CNN that uses the multi-task learning approach to simultaneously detect manipulated images and videos and locate the manipulated regions. The information gained by performing one task is shared with the other task, thereby enhancing the performance of all tasks. A semi-supervised learning approach is used to improve the network's generalizability. The network includes an encoder and a Y-shaped decoder. Activation of the encoded features is used for binary classification. The output of one branch of the decoder is used for segmenting the manipulated regions while that of the other branch is used for reconstructing the input, which helps improve overall performance. With this design, fine-tuning the network using only a small amount of data enables it to deal with unseen attacks effectively.

Contents

Abbreviations

AI	**A**rtificial **I**ntelligence
CMA-ES	**C**ovariance **M**atrix **A**daptation **E**volution **S**trategy
CNN	**C**onvolutional **N**eural **N**etwork
DB	**D**ata**b**ase
DF	**D**eep**f**ake
DFD	**D**eep**f**ake **D**etection
DFDC	**D**eep**f**ake **D**etection **C**hallenge
EER	**E**qual **E**rror **R**ate
eKYC	**E**lectronic **K**now **Y**our **C**ustomer
DNN	**D**eep **N**eural **N**etwork
FFHQ	**F**lickr-**F**aces-**H**igh-**Q**uality
FMR	**F**alse **M**atching **R**ate
FNR	**F**alse **N**egative **R**ate
FPR	**F**alse **P**ositive **R**ate
FR	**F**ace **R**ecognition
GAN	**G**enerative **A**dversarial **N**etwork
HOG	**H**istogram of **O**riented **G**radients
HTER	**H**alf **T**otal **E**rror **R**ate
LBP	**L**ocal **B**inary **P**atterns
LFW	**L**abeled **F**aces in the **W**ild
LVE	**L**atent **V**ariable **E**volution
ML	**M**achine **L**earning
RF	**R**andom **F**orest
PAI	**P**resentation **A**ttack **I**nstrument
PIN	**P**ersonal **I**dentification **N**umber

SIFT	**S**cale-**I**nvariant **F**eature
SVC	C-**S**upport **V**ector **C**lassification
SVM	**S**upport **V**ector **M**achine
VAE	**V**ariational **A**uto**e**ncoders
WAP	**W**olf **A**ttack **P**robability
WGAN	**W**asserstein **G**enerative **A**dversarial **N**etwork
WGAN-GP	**W**asserstein **G**enerative **A**dversarial **N**etwork – **G**radient **P**enalty

Chapter 1

Introduction

"Si vis pacem, para bellum" — a Latin adage translated as, *"If you want peace, prepare for war."* Besides real wars, this philosophy is applicable in many other areas, including security and privacy, and recently, in machine learning. It is also the basis of this thesis, in which we focus on facial biometrics from both the human and machine perspective. From the machine perspective, a face recognition system is a means for a machine to "understand" human identities via faces. From the human perspective, machines are used to generate or manipulate facial images and videos that are as natural as possible to serve the needs of humans. Deepfake videos ("deepfakes") are an example of computer-manipulated videos that are maliciously used by attackers. Face recognition systems and deepfakes are the two main topics discussed in this thesis.

This Introduction chapter is organized as follows. We first provide background information related to the two main topics. Then, we describe our motivation for doing the work described here. We next summarize our main contributions. Finally, we outline this thesis.

1.1 Background

Passwords should be strong, which can make them difficult to remember, and should be changed regularly to ensure security. Personal identification numbers and unlock patterns are more convenient than passwords, but the user is still required to remember them, and people nearby may be able to steal a peek at them. An even more convenient method is biometric authentication, which uses a biometric trait unique to the user, eliminating the need to remember anything. This advantage has led to the widespread usage of biometric authentication on many portable devices including laptops and smartphones.

The two most commonly used biometric traits for authentication are a fingerprint and the face [1]. Since smartphones using this type of authentication may have a digital wallet (or e-wallet) for making e-payments, they are a prime target for attackers. An attacker may attempt to unlock such a device by performing a presentation attack [2]. For example, the attacker might attempt a presentation attack in which a printed facial image of the victim (known as a *presentation attack instrument, or PAI*) is displayed in front of the smartphone's camera to unlock the smartphone. Another scenario is that the attacker hijacks the smartphone's camera and injects a fake video of the victim to fool the Electronic Know Your Customer (commonly known as eKYC) system's user verification process [3]. This enables the attacker to avoid the PAI artifacts that can be used to identify an attack.

Besides its use for authentication, the face has been the primary way for people to recognize each other since the beginning of humankind. Advances in computer graphics and artificial intelligence (AI) now enable machines to generate realistic media and seamlessly manipulate them, including facial images and videos. Recent studies demonstrated that synthetic and manipulated media are hard to spot with the human eye [4–6]. Examples of high-quality computer-generated and deepfake images are shown in Fig. 1.1. In addition to beneficial applications (such as content creation and virtual avatar rendering), computer graphics and AI have been misused by attackers to generate fake images and videos for malicious purposes. Deepfake images and videos [7], typical examples of such threats, have attracted particular attention from society.

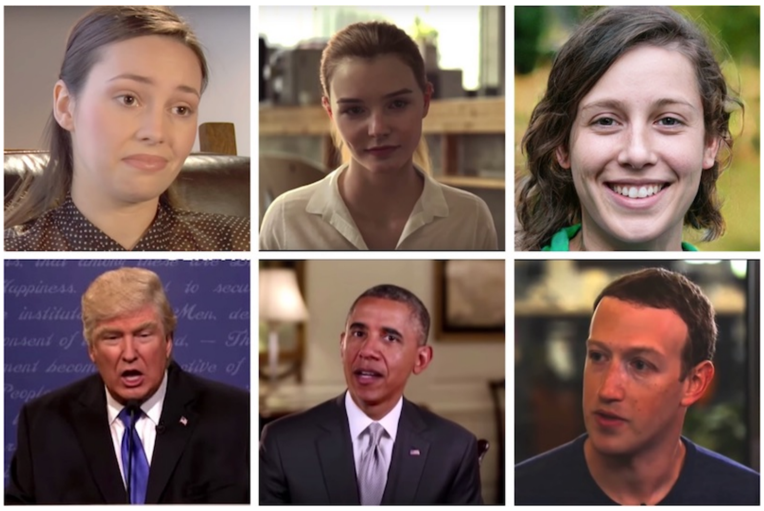

FIGURE 1.1: Example computer-generated facial images and face-swapping ones. Images in top row, left to right, were fully computer-generated and obtained from the Digital Emily Project [15], fully computer-generated and obtained from Dexter Studios [16], and generated using StyleGAN [9]). Images in bottom row, left to right, were manipulated using deepfake [7], Face2Face [17], and Neural Textures [18] methods.

"Deepfakes" was initially defined as "synthetic media in which a person in an existing image or video is replaced with someone else's likeness"[1]. This definition has since expanded to include "synthetic media applications." Attackers can now use applications based on advanced techniques to synthesize unreal faces [8, 9], swap faces [7], manipulate facial attributes [10, 11], and perform facial reenactment [7] and lip-syncing [12].

Besides attacking machines via face recognition systems, as mentioned above, deepfakes can be used to impersonate and harass individuals and to create fake news. State-of-the-art deepfake methods require only a few samples of the target person (*e.g.* one or more portrait photos or a short video) for reference [13, 14]. Due to the popularity of social networks, it is usually easy to obtain a person's facial image or video online, increasing the ability of the attackers. Consequently, deepfakes are a serious threat that must be immediately addressed.

Along with these rapid advances in deepfake methods, several countermeasures have been developed to detect deepfakes. Like local binary pattern methods [19] and histogram of oriented gradient [20] descriptors used in presentation attack detection, the handcrafted steganalysis-based method developed by Fridrich and Kodovsky [21] effectively detects deepfakes. However, finding effective features is challenging and time-consuming. The defender side thus trails in the race.

[1] https://mitsloan.mit.edu/ideas-made-to-matter/deepfakes-explained

Thanks to convolutional neural networks (CNNs) and their automatic feature extraction ability, the defender's tasks have become selecting a suitable input domain, an effective CNN architecture, and an appropriate training strategy. The input domain could be an image (or video frame) in the RGB space (raw or preprocessed) [22–24] or frequency domain [25], a video sequence [26–28], or a video with voice [29]. The architecture could be a single network architecture [22, 23], a two-stream network architecture [24, 25, 30], or a network ensemble [31]. The training strategy could be straightforward optimization [22], transfer learning [23, 32], multi-task learning [33], or knowledge distillation [34].

1.2 Motivation

Research on security measures, as on real-world battlefield measures, involves never-ending competition between the attackers and the defenders. Both sides are continuously improving their abilities to become stronger. This philosophy has recently been applied to the training of generative adversarial networks (GANs) [35], enabling them to generate realistic media, including deepfake images and videos. On the other side, adversarial training helps to improve the robustness of the trained models against adversarial attacks [36–38]. Motivated by this philosophy, we frame this thesis as a competition between the attacker side and the defender side. Since we focus on both people and machines, we use the term "facial biometrics" to represent both the appearance and biometric characteristics of human faces. We address several major problems in facial biometrics that are either unsolved or only partially solved.

From the attacker side, we use face recognition systems as the main target. Such systems are used by machines to "understand" a person's identity from their face. Besides such external threats as presentation and deepfake attacks, face recognition systems have internal problems. One of them is their ability to distinguish between different identities. The well-known face morphing attack [39], for example, can fool both people and machines. Similarly, fingerprint and vein recognition systems have trouble handling wolf attacks [40] as well the recently developed master print attack [41] in which systems are made to mistakenly accept maliciously crafted input. The question we ask is whether face recognition systems have similar problems and what are their causes. Please note that we do not have a malicious purpose in asking this question. Our purpose is to identify the problems of face recognition systems so that the community will be aware of them and we together can find solutions.

From the defender side, we use deepfakes as the main target. Although several CNN-based detectors have been developed and have achieved remarkable results, there are still problems. One problem is their strong need for computational and data resources. The performance of a CNN is normally improved by increasing its depth [42], its width [43], and/or the number of inner connections [44]. Another approach is to increase the number of CNNs, *e.g.*, by adding one or more streams [24, 25, 30] or by using an ensemble of CNNs [31]. Consequently, the detectors have become larger and thus consume more memory and computation power. In addition, bigger models require more training data, which is not always available, especially when new deepfake methods appear. Another problem is the difficulty of “explaining” deepfake detection results. Besides giving the probability of whether the input is real or fake, it is necessary to segment the manipulated regions or highlight the regions containing deepfake artifacts. These additional “explanations” provide clues to the user for use in confirmation or further verification, especially in critical applications in journalism and law enforcement. A third problem is the limited generalizability of deepfake detection and segmentation methods. Since most of them are CNN-based and learn from training data, they may suffer from a mismatch between the training and test data distributions, resulting in poor performance on unseen deepfake methods. The rapid advances in deepfake generation methods have made generalizability important to ensure the robustness of countermeasure methods.

1.3 Contributions

From the attacker side, we are the first in the literature to demonstrate the existence of master faces, which can match the faces of different individuals by the face recognition systems. A master face attack is stronger than a face morphing attack since it does not require knowledge about the victim. By improving the latent variable evolution algorithm used in a master print attack and using a powerful facial generation model (StyleGAN [9]), we can generate high-quality master faces that can attack several face recognition systems in white-box, gray-box, and even black-box scenarios. By analyzing the distributions of the face embedding (identity) spaces, we identify the limitations of several current face recognition systems and suggest several improvements.

From the defender side, we make two contributions. The first contribution is adapting the capsule network – an upgraded version of a CNN that is naturally designed for computer vision – to enable it to work with deepfake detection, including master face detection. Besides the dynamic routing algorithm, the addition of a feature extractor to address the data shortage problem (common in this task) and statistical pooling layers (which work well with deepfake artifacts), the proposed Capsule-Forensics network has high detection performance without sacrificing computational resources and memory. Through visualization, we improve the explainability of the network, which is lacking in most previous work on deepfake detection.

The second contribution is implementing a multi-task learning CNN (the Y-shaped autoencoder) that can simultaneously perform deepfake detection and segmentation. The inclusion of the self-supervised task (reconstruction of the input image) and the sharing of common knowledge between the three tasks enables our proposed Y-shaped autoencoder to adapt well to new deepfake methods by using a limited amount of fine-tuning data. It outputs both predicted probability and segmentation maps, which improves the explainability of the result. Furthermore, if either the detection or segmentation task fails, the other task can compensate for its failure or warn the user of the need for further consideration of the result.

In the appendices, we describe three additional contributions. The first one is asserting the possibility of enhancing computer-generated (CG) facial images to fool spoofing detectors, which are usually integrated into face authentication systems. Unlike the traditional viewpoint of computer graphics, which focuses on the rendering phase, we enhance the rendered images by using a proposed enhancer CNN namely H-Net. It can perform black-box attacks that degrade the accuracies of three spoofing detectors. The second contribution is a modular discriminator for discriminating CG images and photographic images. It uses a probabilistic patch aggregation strategy to deal with high-resolution images and outperformed a state-of-the-art method, achieving accuracy up to 100%. The final contribution are two methods for correcting adversarial images and their labels. Adversarial attacks are increasingly targeting deepfake detection and segmentation methods, hence detecting such attacks and correcting them is important. Our proposed method demonstrated promising performance – correcting nearly 90% of adversarial images while minimally affecting bona fide images.

1.4 Thesis' Outline

The rest of this thesis is organized as follows:

- Chapter 2 provides a literature review of the topics related to this thesis.
- Chapter 3 discusses the master face attack on face recognition systems.
- Chapter 4 introduces Capsule-Forensics, a novel deepfake detection network.
- Chapter 5 presents an approach that combines deepfake detection and segmentation.
- Chapter 6 summarizes this thesis and discusses future work.

Chapter 2

Literature Review

In this chapter, we describe background topics related to the thesis. First, we give an overview of face recognition systems and of commonly used methods for attacking them. Then, we introduce deep generative models, which can generate non-existent facial images for use in attacking face recognition systems and manipulate images and videos. Following that, we give a brief history of image and video manipulation. Since the deepfake phenomenon, a rising star in image and video manipulation, is the main target of this thesis, we then introduce related work on deepfake generation and detection, some widely used deepfake datasets, and the challenges in deepfake detection.

2.1 Face recognition systems

Face recognition (FR) is a process of matching a human face from a captured image or video frame against the enrolled faces stored in a database. An FR system is visualized in Fig. 2.1. An FR system can operates in either of two ways: *verification* and *identification*. In verification, the system tries to prove whether the probe (query face) has the true identity. In identification, the system tries to find the true identity of the probe from a set of enrolled identities. "Recognition" is commonly used to indicate either verification or identification.

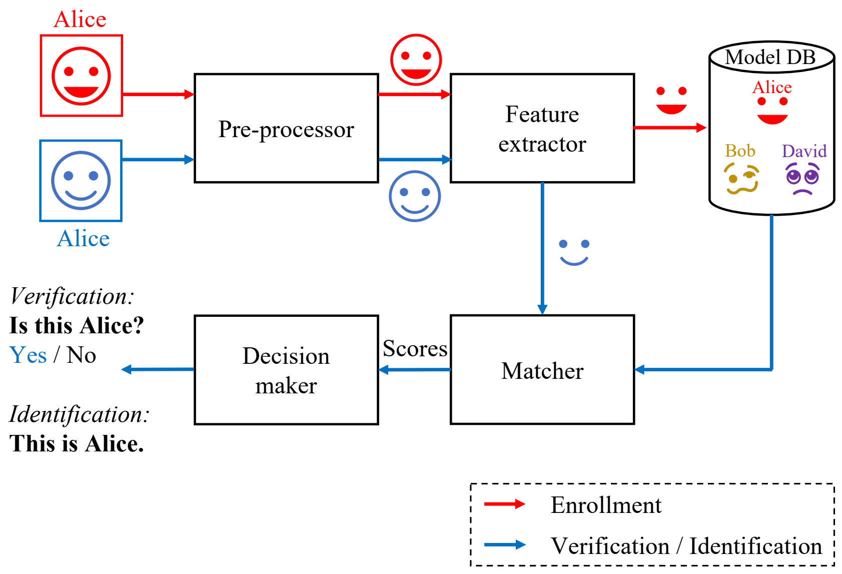

Figure 2.1: Overview of face recognition with enrollment phase and verification/identification phase.

Like other biometric systems, there are two phases in a FR system:

- Enrollment: Storage of a facial image of a user in the model database. The stored biometric data is usually called a model or template.
- Recognition: Comparison of the probe with the enrolled model(s) and return the similarity score(s). The similarity score(s) is (are) then used to judge whether the probe and the model are from the same person.

There are usually four modules in an FR system:

- Preprocessor: Pre-processes the input image to extract the face with the properties most suitable for recognition. The input face is typically detected, cropped, and aligned in this phase.
- Feature extractor: Extracts the features suitable for recognition. Most state-of-the-art FR systems use a CNN for feature extraction.
- Matcher: Compares the probe's feature set to the target's feature set (for verification) or all feature sets (for identification) in the model database and returns the similarity score(s).
- Decision maker: In verification, given the similarity score, decides whether the probe and the target model are from the same identity on the basis of a predefined

threshold. In identification, which model is the best representation of the probe's identity is decided on the basis of the similarity scores.

Recent developments in CNNs and the release of large databases (*e.g.*, the CASIA-WebFace database [45] and the MS-Celeb database [46]) have substantially improved the performance of FR systems and enabled them to work effectively in heterogeneous domains [47]. Most state-of-the-art FR systems [47–49] make use of a network architecture that achieved high performance in the ImageNet Challenge [50], such as the VGG network architecture [51] and the inception network architecture [52]. Parkhi *et al.* trained the VGG-16 network on a custom-built large-scale database [53] to create the VGG-Face network. Wu *et al.* proposed a lightweight CNN that has ten times fewer parameters than the VGG-Face network [54]. The inception architecture was used by de Freitas Pereira *et al.* to build heterogeneous FR networks [47] and by Schroff *et al.* to build the FaceNet network [48]. Sandberg re-implemented FaceNet as an open-source system [55]. Taigman *et al.* introduced the DeepFace architecture in which explicit 3D face modeling is used to improve facial alignment and a CNN is used to extract face representation [56]. Unlike previous methods, which use discriminative classifiers, the generative classifier proposed by Tran *et al.* (DR-GAN) learns a disentangled representation [57].

More recent approaches focus on optimizing the embedding distribution. Deng *et al.* proposed using the additive angular margin loss (ArcFace) instead of the commonly used cosine distance loss to improve the discriminative power of the FR model and to stabilize the training process [49]. Duan *et al.* argued that the distribution of the features plays an important role and therefore proposed using a uniform loss to learn equidistributed representations for their UniformFace FR system [58].

2.2 Attacks on Face Recognition Systems

According to Ratha *et al.* [59], there are nine possible attacks against biometric systems, as shown in Fig. 2.2. The presentation attack (attack 1 in the figure) is one of the most common attacks. Another common attack is 2, which is carried out at the logical level (hereinafter "logical attack"), to modify or inject digital biometric traits into the system.

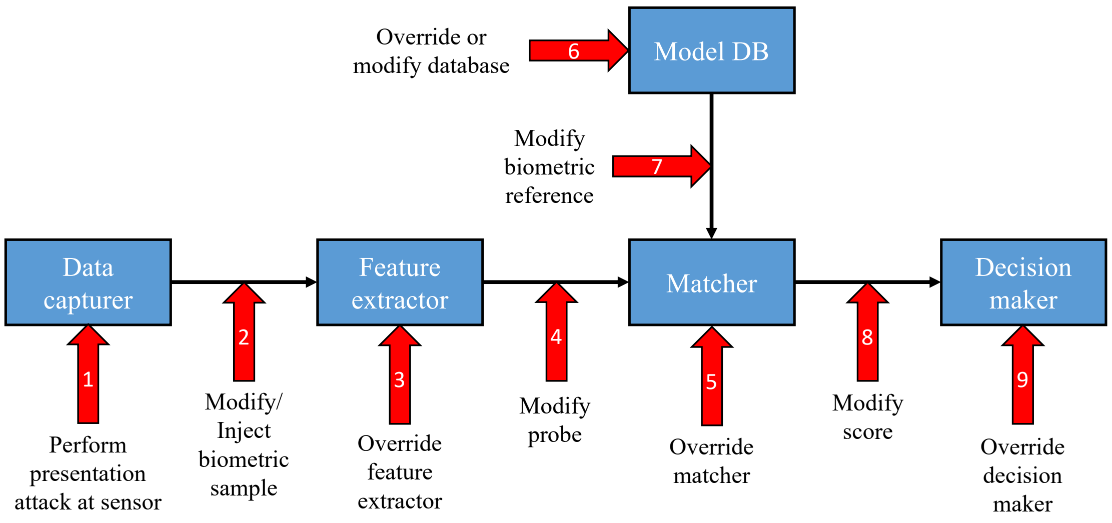
Override or modify database
6
Model DB
Modify biometric reference
7
Data capturer
Feature extractor
Matcher
Decision maker
1
Perform presentation attack at sensor
2
Modify/ Inject biometric sample
3
Override feature extractor
4
Modify probe
5
Override matcher
8
Modify score
9
Override decision maker

FIGURE 2.2: Possible attacks against biometric systems [59]. The preprocessor is integrated into the feature extractor in this figure.

A wolf attack [40] or a master biometric attack [41] can be carried out as a presentation attack or a logical attack.

2.2.1 Presentation attack and detection

FR systems are vulnerable to presentation attacks, which present an artifact or human characteristic to the biometric (facial) capture subsystem to interfere with the intended operation of the biometric (FR) system[1]. A photo attack is a presentation attack in which the attacker displays a photograph of the victim to the sensor of the FR system. This photograph can be printed on paper or displayed on a device screen (*e.g.*, that of a smartphone, tablet, or laptop) [60]. A replay attack is a presentation attack in which a victim's video is played instead of a photograph being displayed [60].

A presentation attack detector can be integrated into an FR system to mitigate presentation attacks [60]. Handcrafted image-based approaches often use descriptors [61–64] such as a color local binary pattern [19], a histogram of oriented gradients [20], and a scale-invariant feature transform [65]. Advances in deep learning have enabled CNNs to automatically extract features [66, 67]. Liveness detection is another commonly used approach, for example, detecting eye blinking [68].

[1]ISO/IEC CD 30107-1 definition. Accessed at `https://www.iso.org/obp/ui/#iso:std:iso-iec:30107:-1:ed-1:v1:en:term:3.5`

The generalization of presentation attacks is a big concern because it can degrade the performance of detectors in real-world conditions. Domain adaptation improves cross-database detection performance [69, 70]. Fatemifar *et al.* proposed combining multiple one-class classifiers for anomaly detection [71]. Mohammadi *et al.* leveraged the variability present in multiple FR datasets to model common nuisance factors that cause domain shift [72].

2.2.2 Wolf attack and master biometric attack

A "wolf sample" is an input sample that can be falsely accepted as a match with multiple user templates ("enrolled subjects") in a biometric recognition system [40]. Wolf samples could be either biometric or non-biometric. A wolf sample is used in a wolf attack against a biometric recognition system. Wolf attacks were initially used to target fingerprint recognition systems [59]. Their success is theoretically measured using the wolf attack probability (WAP)–the maximum probability of a successful attack with one wolf sample [40]. To mitigate wolf attacks, Inuma *et al.* [73] presented a principle for the construction of secure matching algorithms for any biometric authentication systems that calculates the entropy of the probability distribution of each input value.

A master biometric attack is a wolf attack in which the sample looks like an actual biometric trait. Two example traits are partial fingerprint images [41] and facial images [74]. They are generated by GANs using the latent variable evolution (LVE) algorithm to maximize the false matching rate (as a result, WAPs are also maximized). A master print attack [41] targets partial fingerprint recognition systems using small sensors with limited resolution while a master face attack targets FR systems, which require higher resolution images [74].

2.2.3 Latent variable evolution

Evolution algorithms are commonly used in artificial intelligence applications to approximate complex, multimodal, and non-differentiable functions since they do not require any assumption about the underlying fitness landscape. The covariance matrix adaptation evolution strategy (CMA-ES) is a powerful strategy designed for non-linear and non-convex functions [75]. Bontrager *et al.* used CMA-ES with a pre-trained generative adversarial network to perform interactive evolutionary computation to improve the quality of generated samples [76]. This strategy was used in subsequent work for the LVE algorithm to maximize the WAP of generated partial fingerprint images [41]. We modified the LVE algorithm scoring method [74] so that it could work smoothly with high-resolution facial images generated by StyleGAN.

Given n random initial vectors $\mathcal{Z} = \{\mathbf{z}_1, \mathbf{z}_2, ..., \mathbf{z}_n\}$, a generation model $\mathcal{G}$, a scoring function $\mathcal{F}$, and m enrolled temples $\mathcal{T} = \{\mathbf{t}_1, \mathbf{t}_2, ..., \mathbf{t}_m\}$. The LVE algorithm runs in a loop in which at first, n samples are generated by $\mathcal{G}$ using $\mathcal{Z}$. Each sample is then matched with m templates in $\mathcal{T}$ to obtain a mean score s. An evolution algorithm (*e.g.*, CMA-ES) takes the set of the mean scores s to evolve n new latent vectors $\mathcal{Z}'$ for the next loop.

2.3 Deep Generative Models

Image generation is a major topic in deep learning research, and the face is a common target. There are two major approaches to image generation: using variational autoencoders (VAEs) [77] and using GANs [35]. In the beginning, they could only generate small images with low quality. VAEs tended to generate blurry images while GANs were difficult to train. Subsequent improvements in GANS (Wasserstein GAN (WGAN) [78] and WGAN gradient penalty (WGAN-GP) [79]) resolved the training problem. GANs then began to be used to generate master prints [41].

Recently improved versions of both VAEs [80, 81] and GANs [8, 9, 82, 83] can generate high-resolution images. By gradually adding more layers during training in order to output larger images, Karras *et al.* were able to generate 1024×1024 pixel images with their progressive GAN [8]. In subsequent work, they combined the ideas of progressive training and style transfer to create a better disentanglement network called StyleGAN [9]. Unlike traditional GANs, which directly use a latent vector for generating images, StyleGAN uses a mapping network to transfer this latent vector into intermediate style vectors used for synthesizing images. Controlling these intermediate style vectors changes the facial attributes. With the abilities of strong disentanglement and high-quality facial image generation, StyleGAN and its subsequent version [83] are the best methods for generating master faces [74].

2.4 A Brief History of Image and Video Manipulation

Ever since the invention of photography, people have been interested in manipulating photographs, mainly to correct problems in the photos or to enhance them. Technology has advanced far beyond these basic manipulations and can now be used to change the identities of the subjects or alter their emotions. The advent of deep learning has enabled high-quality manipulated images and videos to be easily created. Moreover, the popularity of social media has enabled massive amounts of data, including personal information, news reports, images, and videos, to be created and shared. The consequence is that people with malicious intent can easily make use of these advanced technologies and data to create fake images and videos.

The requirements for manipulating or synthesizing videos were dramatically simplified when it became possible to create forged videos from only a short video [17, 84] or even from a single ID photo [13] of the target subject. Suwajanakorn *et al.*'s mapping method [12] has enhanced the ability of manipulators to learn the mapping between speech and lip motion. State-of-the-art natural speech synthesizers can be used with Suwajanakorn's method to create a fake video of any person speaking anything. Deepfakes [7] exemplify this threat – an attacker with a personal computer and an appropriate tool can create videos of a person impersonating any other person. Deepfake videos have been posted on YouTube with the challenge being to spot them. In this thesis, we use the term "deepfake" to refer to this family of manipulation techniques, not to a particular one.

2.5 Deepfake Generation

The term "deepfake" originated when deep learning began to be used for face swapping [7]. The original deep fake method used two autoencoders that swapped the latent features of two input images so that the corresponding identities were exchanged. This method is illustrated in Fig. 2.3. Face swapping and manipulation have since become

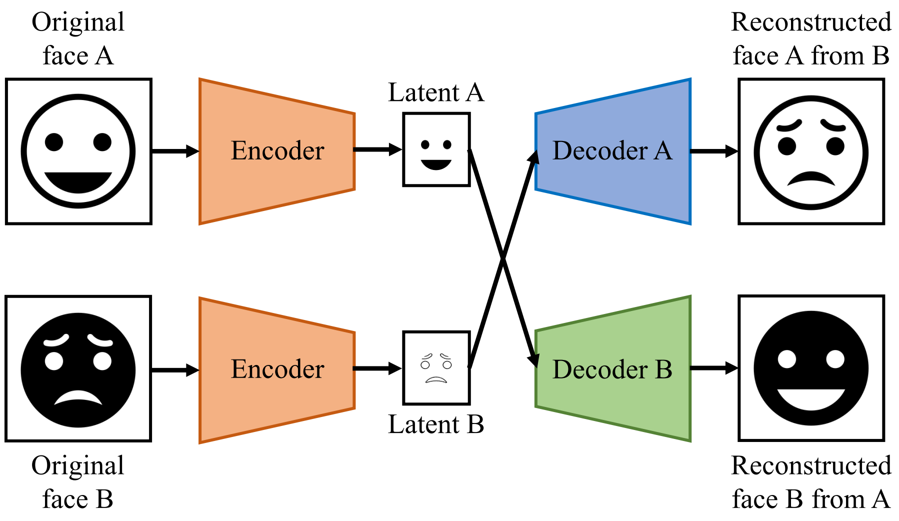

FIGURE 2.3: Overview of original deepfake method, which uses two autoencoders to swap faces.

quite common, and any deep-learning-based tool serving this purpose is now referred to as a deepfake tool by the media and academics. Manipulated speech is referred to as "deepfake audio." In this thesis, we use the term deepfake to indicate any kind of media computer-generated or manipulated using advanced methods like deep learning. Deepfakes can be classified into five categories: entire face synthesis (*e.g.*, GAN images), attribute manipulation (*e.g.*, changing hair or skin color, altering facial expression), face swapping, facial reenactment, and speaking manipulation. We consider images, videos, and audio with benign manipulations not to be deepfakes; such manipulations include changing the contrast, brightness, or sharpness, slightly adding or removing benign noise, and applying affine transforms. Detection of deepfake images, videos, and audio is called "deepfake detection."

Recent achievements have demonstrated that deepfakes can reach a photo-realistic level. Thies *et al.* demonstrated that expression transfer for facial reenactment can be performed in real time [17]. Kim *et al.* demonstrated the transfer of a head pose along with facial movements from an actor to another person [84]. Similarly, Tripathy *et al.* devised a lightweight face reenactment method using a GAN [85]. Nirkin *et al.* presented a face-swapping method that does not require training on new faces [86], unlike early deepfake methods [7]. Thies *et al.* combined the traditional graphics pipeline with learnable components to deal with imperfect 3D contents [18]. Besides facial reenactment and face swapping, facial attributes such as skin and hair color, bangs, mustache, glasses, and emotions can be modified by GANs [10, 11, 87].

Work on deepfakes has gone beyond only the visual part. Suwajanakorn *et al.* presented a method for learning the mapping between speech and lip movements in which speech can also be synthesized, enabling creation of a full-function spoof video [12]. Fried *et al.* demonstrated that speech can be easily modified in any video in accordance with the intention of the manipulator while maintaining a seamless audio-visual flow [88]. Averbuch-Elor *et al.* addressed a different task – converting still portraits into motion pictures expressing various emotions [13]. This work greatly simplified the requirements for attackers: simply acquire a picture of the victim (usually a profile picture on a social network or an ID photo). Zakharov *et al.* followed up by improving the quality of videos generated using only a few input images [14]. Vougioukas *et al.* raised the bar by introducing a method for animating a facial image from an audio track containing speech [89].

2.6 Deepfake Detection

The handcrafted steganalysis-based method developed by Fridrich and Kodovsky [21] was used in early efforts to detect deepfake images. Noise residuals extracted using handcrafted linear and nonlinear high-pass filters are fed into an ensemble classifier. This approach was later implemented in a CNN by Cozzolino *et al.* [90]. Transfer learning is a common choice when a CNN pretrained on the ImageNet dataset [50] is used [23, 32]. Our previous work [32] used part of a pretrained VGG-19 network [51] as the feature extractor for our modular network while Rössler *et al.* finetuned the XceptionNet network [91] on a deepfake dataset. Afchar *et al.* utilized inception modules [92] to build a lightweight network [22] while Wang *et al.* utilized a dilated residual network [93]. Bayar and Stamm presented a new convolutional layer that helps a CNN adaptively learn manipulation detection features [94]. Zhou *et al.* proposed using a two-stream network in which one stream takes RGB input and the other takes steganalysis features and uses a triplet loss [30]. Dang *et al.*'s two-stream network [24] uses the RGB domain while Qian *et al.*'s two-stream network [25] uses the frequency domain. Kim *et al.* used knowledge distillation to improve the generalizability of detectors [34].

Videos provide more information than images for detection, especially when they contain sound. Li *et al.* used eye blinking as a feature to detect deepfakes [26] while Agarwal *et al.* used facial expressions and movements [27]. Sabir *et al.* used a recurrent neural network to additionally learn the temporal information [28]. Korshunov and Marcel used several approaches for lip-syncing and dubbing detection to detect fake videos [29].

In addition to binary classification, another major branch in digital media forensics is locating manipulated regions in images. Besides "pure" segmentation-based approaches [33, 95, 96], binary classification approaches using a sliding window to locate manipulated regions are also applicable [23, 32]. From a different viewpoint, Li *et al.* introduced a method called "face X-ray" to detect the blending boundary between real and fake regions [97]. They noted that blending methods have not been advancing as rapidly as manipulation methods; therefore, focusing on blending methods makes the detector more robust against unseen manipulations.

2.7 Deepfake Datasets

Several standardized datasets have been constructed to support deepfake detection, as shown in Table 2.1. The DeepfakeTIMIT [98] and the FaceForensics datasets [99] were early datasets for deepfake detection. Besides the deepfake method [7], the FaceForensics dataset included the Face2Face [17] and FaceSwap [23] methods. Subsequently, the Neural Textures [18] and FaceShifter [100] methods were added to the FaceForensics

Table 2.1: Commonly used deepfake datasets

Dataset	**Year**	**No. of real videos**	**No. of fake videos**	**No. of faces/image**	**Manipulation methods**
DeepfakeTIMIT [98]	2018	320	320	1	Deepfake [7]
UADFV [26]	2018	49	49	1	Deepfake [7]
FaceForensics++ (FF++) [23, 99]	2019	1,000	5,000	1	+ Deepfake [7] + Face2Face [17] + FaceSwap [23] + NeuralTextures [18] + FaceShifter [100]
Google DFD [101]	2019	363	3,068	1	Deepfake [7]
Facebook DFDC [31]	2020	23,654	104,500	~ 1	Various
Celeb-DF [102]	2020	590	5,639	1	Deepfake [7]
DeeperForensics [103]	2020	1,000 (from FF++)	10,000 (augmented)	1	DeepFake-VAE [103]
WildDeepfake [104]	2020	0	707	1	No information
Face Forensics in the Wild (FFIW) [105]	2021	10,000	10,000	3.15	+ DeepFaceLab [106] + FaceSwap [23] + FaceSwap-GAN [107]
OpenForensics [5]	2021	45,473 (images)	70,325 (images)	2.90	+ Adversarial latent autoencoder [108] + InterFaceGAN [109]

dataset, forming the FaceForensics++ one [23]. Google contributed to the FaceForensics++ dataset with the Google Deepfake Detection (DFD) dataset [101]. The Facebook Deepfake Detection Challenge dataset (DFDC) [31] was the first large-scale dataset in this area. The WildDeepfake dataset [104] is a challenging dataset containing various deepfake videos created using unknown methods on the Internet. Unlike previous datasets, which mainly had one face per video frame, the Face Forensics in the Wild dataset [105] and the OpenForensics dataset [5] contain videos and images with around three faces per frame or image. Additional datasets include the Albany Deep Fake Video (UADFV) [26], Celeb-DF [102], and DeeperForensics [103] datasets.

2.8 Challenges in Deepfake Detection

There are several challenges in deepfake detection. Since deepfakes have altered faces, most deepfake detection methods need to first detect and crop the face. The success of this step depends on the performance of the face detection method. Most state-of-the-art deepfake datasets have annotated face regions, so researchers may assume that cropped faces are available without considering the face detector's performance. Another challenge is the generalizability of the detector when an advanced deepfake technique is introduced. Moreover, a large amount of appropriate training data may not be available when a new attack appears, so detectors using large networks may be difficult to train. Another challenge is gaining user trust by convincing them to accept the detection results. This requires visualizing the learned features and/or focused regions of the detectors.

The performance of general CNNs can usually be improved by increasing their depth, their width, and/or the number of inner connections. Multiple CNNs are commonly used for deepfake detection, especially in competitions [31, 110]. Fusion is often used in the multiple-CNN approach, including feature aggregation (feature fusion) and output fusion (ensembling). Consequently, these networks get bigger with more parameters, consuming more memory and computation power. Since a larger number of parameters usually requires more training data, dealing with new attacks is difficult.

Chapter 3

Generating Master Faces to Attack Face Recognition Systems

Chapter 3 introduces the method used to generate facial images for use in attacking face recognition systems. It is GAN-based, as mentioned in the previous chapter, and utilizes an evolutionary algorithm to generate master faces, *i.e.*, faces that can match the faces of different individuals.

3.1 Introduction

Biometric authentication systems may be vulnerable to presentation attacks [2] in which a printed facial image of the victim (known as a *presentation attack instrument, or PAI*) is displayed in front of the smartphone's camera. The probability of a presentation attack succeeding is higher if the PAI matches multiple enrolled templates. In the facial domain, the creation of PAIs by blending together two or more faces is called *face morphing* [39]. The morphed face should match all source faces when used against a face recognition (FR) system and possibly even fool a human observer. This ability has made morphing a commonly used attack against automated border control systems in which the attacker "borrows" the identity of the victim to enter or exit a location [39]. The face morphing

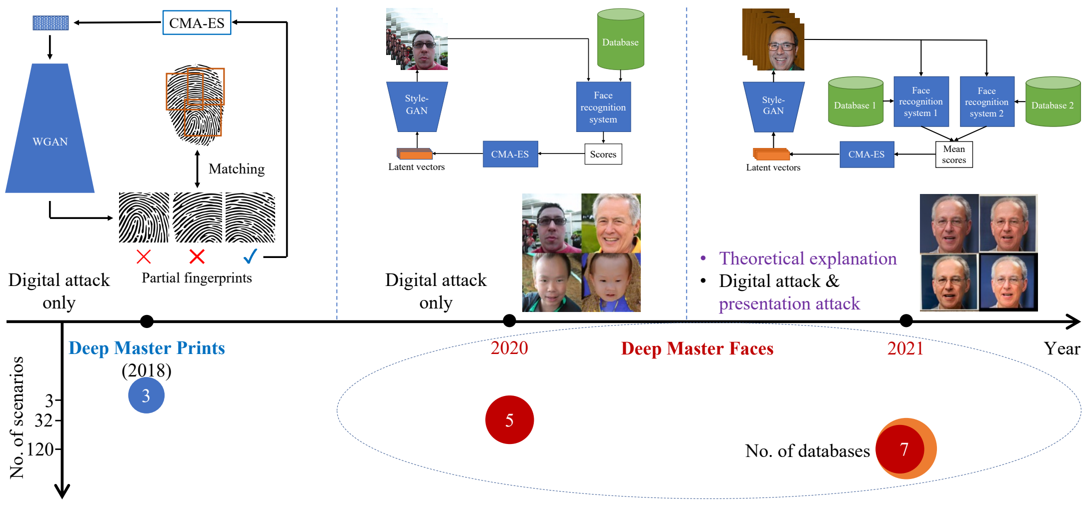

FIGURE 3.1: Stages in master biometrics research. First stage was partial master fingerprints as proposed by Bontrager *et al.* [76]. Next stage was our preliminary work on master faces [74]. Our following stage builds upon previous work and introduces extensions in algorithm, analysis, visualization, and test scenarios.

approach is limited by the requirement that target faces be available. Another approach is to generate a "master biometric" sample [41, 74]–a kind of "wolf sample" that matches multiple enrolled templates in a biometric recognition system [40]. This approach was first developed by Bontrager *et al.* [41] for the fingerprint domain.

The stages in master biometrics research are shown in Fig. 3.1. Our contributions can be summarized as follows:

- We are the first to generate master faces that can match multiple faces with different identities. This ability means that FR systems are vulnerable to a master face attack. Unlike the face morphing approach, the attacker's advantage in this "master face" approach is that it does not require any information about the victim.

- We analyze the effect of using multiple databases (DBs) and/or multiple FR systems for the latent variable evolution (LVE) algorithm used to generate master faces. Some DB/FR system combinations boosted overall attack performance while others did not due to intra-component conflicts. Knowledge of the successful combinations is critical to understanding under which conditions strong master faces can be generated and to appropriately assessing the potential risks.

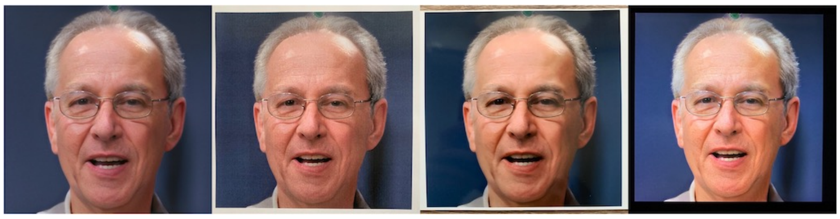

FIGURE 3.2: Original master face generated using two face recognition systems (top left) and its PAI forms printed on plain paper (top right), photo paper (bottom left), and displayed on a 13-inch Apple MacBook Pro screen (bottom right).

- We introduce visualization in the face embedding (identity) space to obtain more insights into master faces. The gained insights are valuable to improve the robustness of the FR systems.
- To demonstrate the actual threat posed by the existence of master faces, we evaluated master face attacks by performing presentation attacks using printed images and the corresponding digital images displayed on a computer screen. Three of the PAIs we used are shown in Fig. 3.2.

The rest of the chapter is organized as follows. First, we discuss the existence of master faces and introduce an improved LVE algorithm using multiple databases and/or FR systems in section 3.2. Our experiments are covered in two sections: we first discuss generating master faces and their analysis in section 3.3, and then discuss using master faces to perform presentation attacks in section 3.4. Next, in section 3.5, we discuss ways to reduce the risk of master face attacks. Finally, we summarize the key points and make some closing remarks in section 3.6.

3.2 Deep Master Faces

3.2.1 Existence of master faces

Before describing the proposed master face generation algorithm, we briefly explain why master faces exist. For a typical FR system (or biometric recognition systems in general), there are four phases (Fig. 3.3): pre-processing the input, extracting its features, matching them with those of the enrolled subject(s) in the model database, and making a decision. The feature extractor plays the role of a mapping function. It maps the facial image domain to the identity domain. The objective when training the

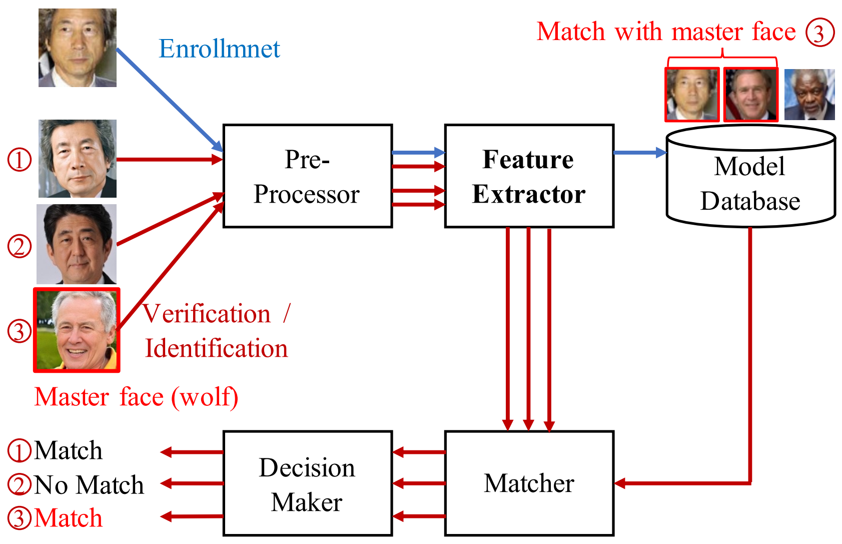

FIGURE 3.3: Operation of typical FR system. There are two phases: enrollment (blue path) and verification/identification (red path). The master face (face 3) was falsely matched with the two faces of two enrolled subjects. Best viewed in color.

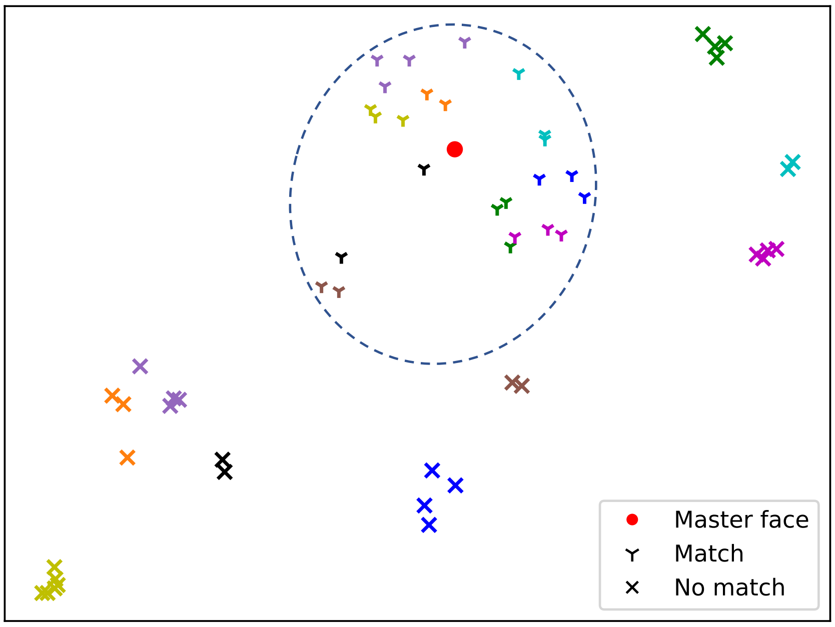

FIGURE 3.4: UMAP visualization of identity space containing embeddings of a master face and of "match" and "no-match" faces of 18 enrolled subjects. For each cluster (match or no match), symbols with the same color correspond to the same subject. Best viewed in color.

feature extractor is to optimize the mapping function so that the mappings of the same-identity faces are close together in the identity space and vice versa. Since this is an optimization problem, the solution is simply an approximation. Furthermore, there is no guarantee that the mapping function will work well on new data due to the possible lack of generalizability.

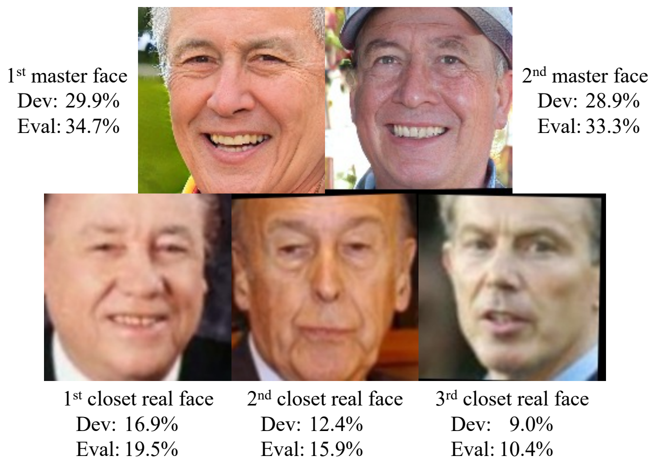

FIGURE 3.5: The first and second master faces generated using the LVE algorithm and the three real faces closest to the first master face and their corresponding false matching rate. The two master faces were generated using the training set of the LFW - Fold 1 database and the Inception-ResNet-v2 based FR system trained on the CASIA-WebFace database. The false matching rates were calculated on the development and evaluation sets of the Labeled Faces in the Wild (LFW) - Fold 1 database.

Master faces may exist because the identity (embedding) space used by FR systems is not uniformly distributed, resulting in dense areas in this space. If we generate an identity corresponding to a point in a dense area, it may falsely match several nearby faces in the identity space. The LVE algorithm aims to find such a position in a dense area in the identity space after several evolutions. To intuitively and empirically show this, we visualize the identity space and one of the master faces generated in this work using uniform manifold approximation and projection (UMAP) [111] in Fig. 3.4. The master face generated by our algorithm (described in the next section) is at such a position (red dot) surrounded by many embeddings. All faces from these surrounding embeddings are falsely matched with the master face by the FR system. The no-match embeddings are scattered far from the master face and lie in less dense areas. We explain how to generate such master faces in the next section.

To verify our hypothesis of dense areas in the identity space, we searched for the real faces that were closest to a generated master face and checked whether they had wolf characteristics like this master face. These real faces were chosen from the facial database used to generate the master face. We used the cosine distance between two embeddings for selection. The result, which is shown in Fig. 3.5, confirms our hypothesis. However, the real wolf faces had lower false matching rates (FMRs) than the master faces. Therefore, using synthesized master faces rather than real faces to carry out wolf attacks should increase the success rate.

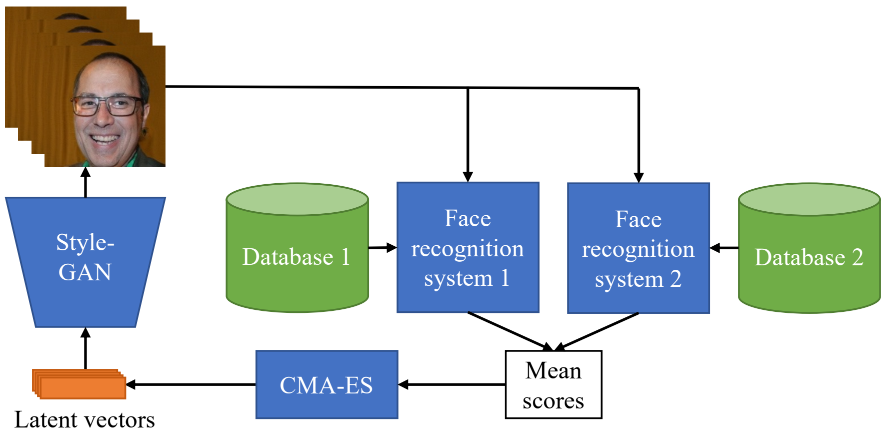

FIGURE 3.6: Overview of extended latent variable evolution algorithm. Latent vectors are fed into StyleGAN [9] to generate facial images. One or more surrogate FR system(s) then calculates mean score for each image on the basis of the subjects in one or more database(s). For example, for the *combination 3* setting described in Table 3.3, **database 1** is LFW - Fold 1, **database 2** is MOBIO, **FR system 1** is Inception-ResNet-v2 network (trained on MS-Celeb database), and **FR system 2** is DR-GAN network. The CMA-ES [75] algorithm uses these scores to generate new latent vectors.

3.2.2 Latent variable evolution with multiple databases and/or face recognition systems

We use more than one database and/or FR system to generate master faces, which requires support from the LVE algorithm. The extended LVE algorithm is illustrated in Fig. 3.6 and is formalized in Algorithm 1. First, m latent vectors $\{\mathbf{z_1}, ..., \mathbf{z_m}\}$ are initialized randomly. They are then fed into a pretrained StyleGAN network to generate m faces. Two face matching functions, $FaceMatching^{(1)}(\cdot,\cdot)$ and $FaceMatching^{(2)}(\cdot,\cdot)$ (corresponding to two FR systems), calculate the similarity between the generated faces and all subject faces in databases $E_j^{(1)}$ and $E_j^{(2)}$, respectively. Two m-dimension mean score vectors, $\mathbf{s^{(1)}}$ and $\mathbf{s^{(2)}}$, are obtained from the results of $FaceMatching^{(1)}(\cdot,\cdot)$ and $FaceMatching^{(2)}(\cdot,\cdot)$. The mean $\mathbf{s}$ of these two vectors is used to select the best local master face F_b among the m generated faces. Finally, $\mathbf{s}$ is fed into the CMA-ES algorithm to generate new latent vectors $\{\mathbf{z_1}, ..., \mathbf{z_m}\}$. This process is repeated n times. The final (global) best master face is chosen from among the n best master faces $\mathcal{F}$ obtained in the n iterations.

To generate another master face, all faces matching the previously generated master face(s) in the training database(s) need to be removed, as shown in Algorithm 2. This prevents the new master face from overlapping the previous master face(s). An example of a second master face is shown in Fig. 3.5 along with the first master face, the real wolf face, and their corresponding FMRs The FMR of the second master face is lower than that of the first one, and this usually holds for any subsequent master faces.

Algorithm 1 Latent variable evolution.

```
$m \leftarrow 22$                                                    ▷ Population
procedure RunLVE($m, n$)
    $\mathcal{F} = \{\}$                                             ▷ Master face set
    $\mathcal{S} = \{\}$                                             ▷ and corresponding score set
    $\mathcal{Z} = \{\mathbf{z_1} \leftarrow rand(), ..., \mathbf{z_m} \leftarrow rand()\}$    ▷ Initialize
    for $n$ iterations do                                            ▷ Run LVE algorithm $n$ times
        $F \leftarrow StyleGAN(\mathcal{Z})$                         ▷ Generate $m$ faces $F$
        $\mathbf{s^{(1)}} \leftarrow 0, \mathbf{s^{(2)}} \leftarrow 0$    ▷ Initialize scores $\mathbf{s^{(1)}}, \mathbf{s^{(2)}} \in \mathbb{R}^m$
        for face $F_i$ in faces $\mathbf{F}$ do
            for face $E_j^{(1)}$ in data $\mathbf{E^{(1)}}$ do
                $s_i^{(1)} \leftarrow s_i^{(1)} + FaceMatching^{(1)}(F_i, E_j^{(1)})$
            $s_i^{(1)} \leftarrow \frac{s_i^{(1)}}{|\mathbf{E^{(1)}}|}$    ▷ Mean scores of 1st system
            for face $E_j^{(2)}$ in data $\mathbf{E^{(2)}}$ do
                $s_i^{(2)} \leftarrow s_i^{(2)} + FaceMatching^{(2)}(F_i, E_j^{(2)})$
            $s_i^{(2)} \leftarrow \frac{s_i^{(2)}}{|\mathbf{E^{(2)}}|}$    ▷ Mean scores of 2nd system
            $s_i = \frac{s_i^{(1)} + s_i^{(2)}}{2}$                  ▷ Mean scores of both systems
        $F_b, s_b \leftarrow GetBestFace(\mathbf{F}, \mathbf{s})$
        $\mathcal{F} \leftarrow \mathcal{F} \cup \{F_b\}$            ▷ Append best master face
        $\mathcal{S} \leftarrow \mathcal{S} \cup \{s_b\}$            ▷ and its corresponding score
        $\mathcal{Z} \leftarrow$ CMA_ES($\mathbf{s}$)
    return $\mathcal{F}, \mathcal{S}$
$F_b, s_b \leftarrow GetBestFace(\mathcal{F}, \mathcal{S})$          ▷ Final (best) master face
```

Algorithm 2 Database refining.

```
$\mathcal{M} = \{M_1, ..., M_n\}$                                   ▷ Previous master faces
procedure Refine_Database($\mathcal{M}$, $\mathbf{E}$)
    $\mathbf{E}' = \{\}$                                            ▷ Initialize refined database
    for face $E_i$ in data $\mathbf{E}$ do
        keep ← true
        for face $M_j$ in $\mathcal{M}$ do
            if isMatch($E_i$, $M_j$) is true then
                keep ← false
        if keep is true then
            $\mathbf{E}' \leftarrow \mathbf{E}' \cup \{E_i\}$
    return $\mathbf{E}'$
```

3.3 Generating Master Faces

To evaluate the risks and threats of a master face attack, we designed several settings for the LVE algorithm and several attack scenarios that cover white-box, gray-box, and black-box attacks. For white-box attacks, both the architecture of the target FR system and its training database are known while for gray-box attacks, only one of them is known. For black-box attacks, there is no information about the target FR system. Attackers may use more than one FR system for the LVE algorithm to increase the probability of their attack being a white-box or gray-box attack. They can also use more than one database for the LVE algorithm to better approximate the distribution of the model database of the target FR system.

This section is organized as follows: We first briefly describe the FR systems and the databases we used in our experiments. Then, we describe our generation of master faces using several combinations of single and multiple FR systems with single and multiple facial databases when running the LVE algorithm. Next, we analyze the generation processes and the generated master faces as well as explain their properties. Finally, we evaluate the false matching performance of the generated master faces for several scenarios, including black-box, gray-box, and white-box attacks.

3.3.1 Experiment materials

3.3.1.1 Face recognition systems

We used five mainstream publicly available high-performance FR systems in our experiments:

- Inception-ResNet-v2 based FR systems: one trained on the CASIA-WebFace database [45] and one trained on the MS-Celeb database [46] by de Freitas Pereira *et al.* [47].
- Open-source version of FaceNet [48] implemented and trained on the MS-Celeb database [46] by Sandberg [55].
- DR-GAN [57] trained on a combination of the Multi-PIE database [112] and the CASIA-WebFace database [45].

Table 3.1: Details of all databases used in our experiments.

Database	Year	No. of images	Resolution
Flickr-Faces-HQ [9]	2019	70,000	1024 × 1024
CASIA-WebFace [45]	2014	494,414	256 × 256
MS-Celeb [46]	2016	10,490,534	Up to 300 × 300
Multi-PIE [112]	2009	755,370	3072 × 2048
LFW [114]	2007	13,233	Various
MOBIO [116]	2012	30,326	Various
IJB-A [117]	2015	5,712	Various

- ArcFace [49] trained on the MS-Celeb database [46].

We used the two Inception-ResNet-v2 based FR systems and DR-GAN for generating master faces and all of the FR systems for evaluating master face attacks[1]. All FR systems were pretrained and obtained from the Bob toolbox [113].

3.3.1.2 Databases

Seven databases were used for four different purposes:

- *Training StyleGAN:* Flickr-Faces-HQ (FFHQ) database [9].
- *Training FR systems:* CASIA-WebFace [45], MS-Celeb [46], and Multi-PIE [112].
- *Running LVE algorithm:* Training set of LFW - Fold 1 database [114] aligned by funneling [115] and both male and female components of training set of MOBIO database [116].
- *Evaluating master faces:* Corresponding development (dev) and evaluation (eval) sets of LFW database [114] and MOBIO database [116] plus dev set of IARPA Janus Benchmark A (IJB-A) database [117][2]. The dev sets were used for threshold selection for the FR systems (which was based on the calculated equal error rates).

Details of all databases are shown in Table 3.1. There are no overlapping subjects between the databases used for training StyleGAN, training the FR systems, and running the LVE algorithm. This point is important to demonstrate that the LVE algorithm can work well with mutually exclusive databases used by its components.

[1]Benchmarks for some of the systems can be found at `https://www.idiap.ch/software/bob/docs/bob/bob.bio.face_ongoing/v1.0.4/leaderboard.html`

[2]There is no eval set for the IJB-A database.

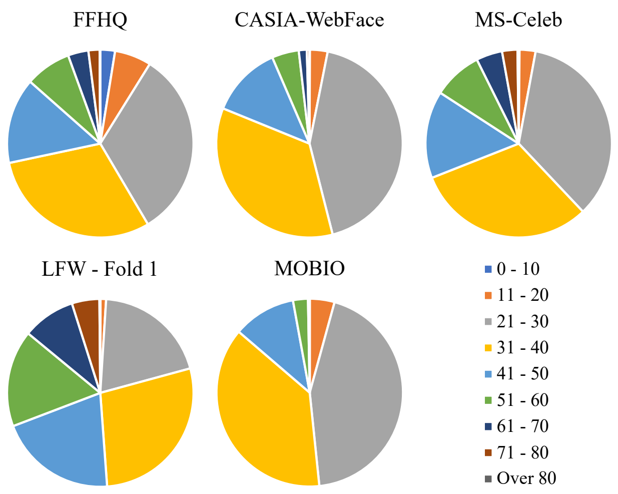

FIGURE 3.7: Estimated age distribution of five databases used for training StyleGAN, FR systems, and generation of master faces. Best viewed in color.

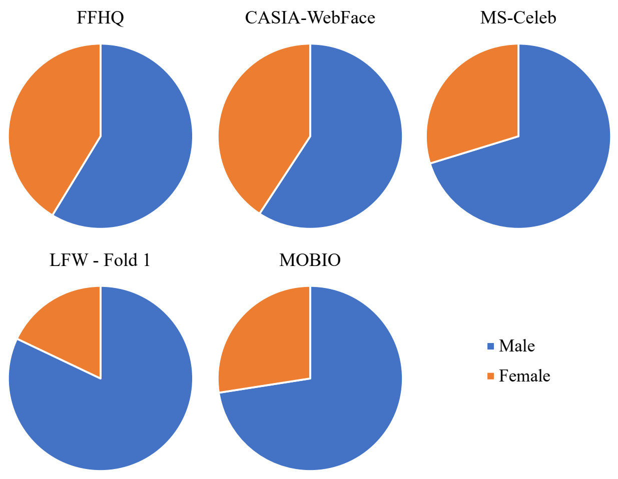

FIGURE 3.8: Estimated gender distribution of five databases used for training StyleGAN, FR systems, and generation of master faces. Best viewed in color.

We used the InsightFace library[3] to estimate the age and gender distributions of the databases used for training StyleGAN and the FR systems and for generating master faces. For the MOBIO database, we used its annotated gender information. We ignored the Multi-PIE database since it contributes only as an additional part of the database for training the DR-GAN FR system. The estimated distributions are shown in Fig. 3.7 and Fig. 3.8 respectively. The ages are dominantly 21 to 40, especially in the CASIA-WebFace, MS-Celeb, and MOBIO databases. The LFW - Fold 1 database is more balanced with a larger proportion of 41 to 60 ages. There are tiny numbers of child faces in all databases except for the MOBIO one, which has none. For gender, there are more male than female faces in all databases. The LFW - Fold 1 and MOBIO databases

[3]https://github.com/deepinsight/insightface

are the most unbalanced, with less than 25% female faces. This may cause bias in the FR systems as well as affect the properties of the generated master faces, as explained in the following section.

3.3.2 Latent variable evolution configurations

Since there are many FR systems and databases, evaluating all possible combinations is impossible with the available computation and time resources. We thus selected a subset with the aim of covering a range as broad as possible. We defined eight settings (Table 3.2) for the LVE algorithm using three FR systems (two versions of Inception-ResNet-v2, one trained on the CASIA-WebFace database and one trained on the MS-Celeb one, and DR-GAN) and two databases (LFW - Fold 1 and MOBIO). There are five settings in which one FR system and one database are used (*single 1* to *single 5*) and three settings in which more than one FR system and/or database is used (*combination 1*, *combination 2*, and *combination 3*).

Each combination setting combined two single settings and was selected on the basis of its reasonable coverage of cases. The main differences among the three combination settings are highlighted in Table 3.3. In the *combination 1* setting, only one database was used with the LVE algorithm, and the databases used for training the FR systems were similar. In the *combination 2* setting, two databases were used with the LVE algorithm, and two FR systems with the same architecture but trained on different databases were used. In the *combination 3* setting, two databases and two FR systems without anything in common were used with the LVE algorithm. We ran 1000 iterations of the LVE algorithm for each of the eight settings.

The generated master faces corresponding to the eight settings are shown in Fig. 3.9. All of them are male faces. One-fourth are child faces, generated using only the Inception-ResNet-v2 based FR system trained on the MS-Celeb database. Half are elder faces generated using only the Inception-ResNet-v2 based FR system trained on the CASIA-WebFace database or only the DR-GAN FR system trained on the combination of the CASIA-WebFace and Multi-PIE databases, or a combination of these two FR systems. The rest (one-fourth) are middle-aged faces, generated using the combinations of the two FR systems in the previous two cases (one in each case).

TABLE 3.2: Settings for running LVE algorithm. "Single" means using only one database and one FR system. "Combination" means using more than one database and/or FR system. For each FR system, we show both its network architecture (top row) and its training database (bottom row).

No.	**Setting**	**Database 1**	**FR System 1 (FR Training DB)**	**Database 2**	**FR System 2 (FR Training DB)**
1	*Single 1*	LFW - Fold 1	Inception-ResNet-v2 (CASIA-WebFace)		
2	*Single 2*	LFW - Fold 1	DR-GAN (CASIA-WebFace & Multi-PIE)		
3	*Single 3*	MOBIO	Inception-ResNet-v2 (MS-Celeb)		
4	*Single 4*	LFW - Fold 1	Inception-ResNet-v2 (MS-Celeb)		
5	*Single 5*	MOBIO	DR-GAN (CASIA-WebFace & Multi-PIE)		
6	*Combination 1* (No. 1 & 2)	LFW - Fold 1	Inception-ResNet-v2 (CASIA-WebFace)	LFW - Fold 1	DR-GAN (CASIA-WebFace & Multi-PIE)
7	*Combination 2* (No. 1 & 3)	LFW - Fold 1	Inception-ResNet-v2 (CASIA-WebFace)	MOBIO	Inception-ResNet-v2 (MS-Celeb)
8	*Combination 3* (No. 4 & 5)	LFW - Fold 1	Inception-ResNet-v2 (MS-Celeb)	MOBIO	DR-GAN (CASIA-WebFace & Multi-PIE)

TABLE 3.3: Comparison between three combination settings for LVE algorithm. For FR systems, we compared their architectures and their training databases.

Setting	**Database 1 vs. Database 2**	**FR System 1 vs. FR System 2**	
		Architectures	**Training DBs**
Combination 1	Same	Different	Similar
Combination 2	Different	Same	Different
Combination 3	Different	Different	Different

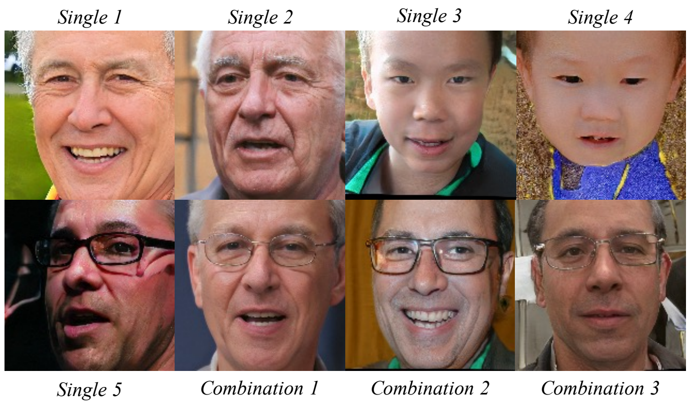

FIGURE 3.9: Master faces generated using eight settings specified in Table 3.2.

3.3.3 Running latent variable evolution

To understand what happened while running the latent variable evolution algorithm, we performed the T-SNE visualization [118] of the embeddings of the intermediate master faces generated using the *single 1* setting. The result is shown in Figure 3.10. Initially, the CMA-ES algorithm was unsure about the optimal direction. After finding some clues, it began generating master faces that jumped around the best master face (the red dot) and came closer and closer to it.

We tried to match the obtained master face generated using the *single 1* setting with all enrolled faces in the dev and eval sets of the LFW - Fold 1 database. A score histogram for the master face is plotted in Figure 3.11 along with those for the genuine faces and the zero-effort imposter faces from the original test design of the database. The master face scores moved away from the zero-effort imposter scores in the direction of the genuine face scores with about 30–35% overlap. This means that the master face matched 30–35% of the enrolled faces, which is significant.

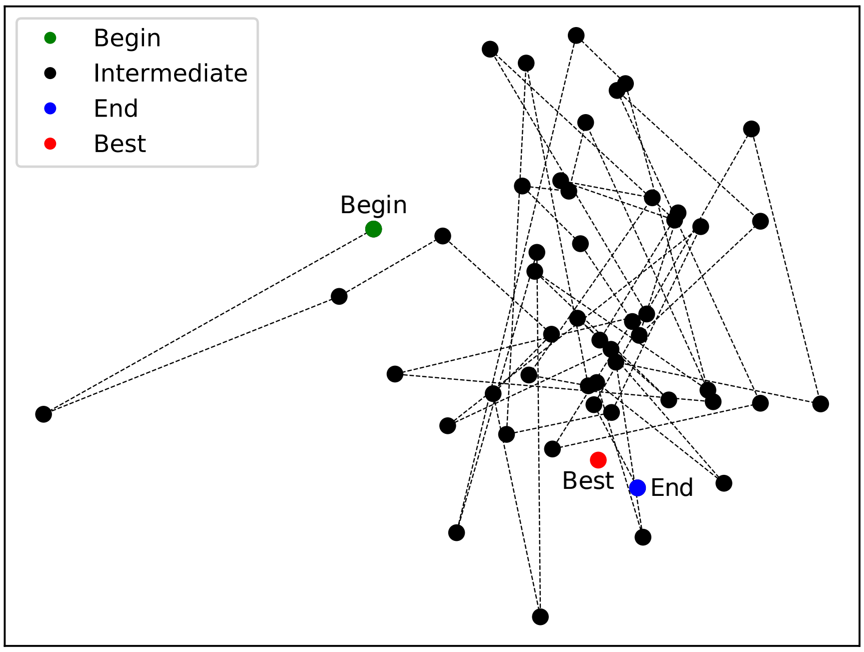

FIGURE 3.10: T-SNE visualization of master faces obtained every 20 iterations (1000 in total) on LFW - Fold 1 database [114]. Green dot represents master face at beginning; it is connected by dashed lines with intermediate master faces (black dots) that end at blue dot. Red dot represents best master face, created at 989 th iteration; therefore, it does not overlap any black dot.

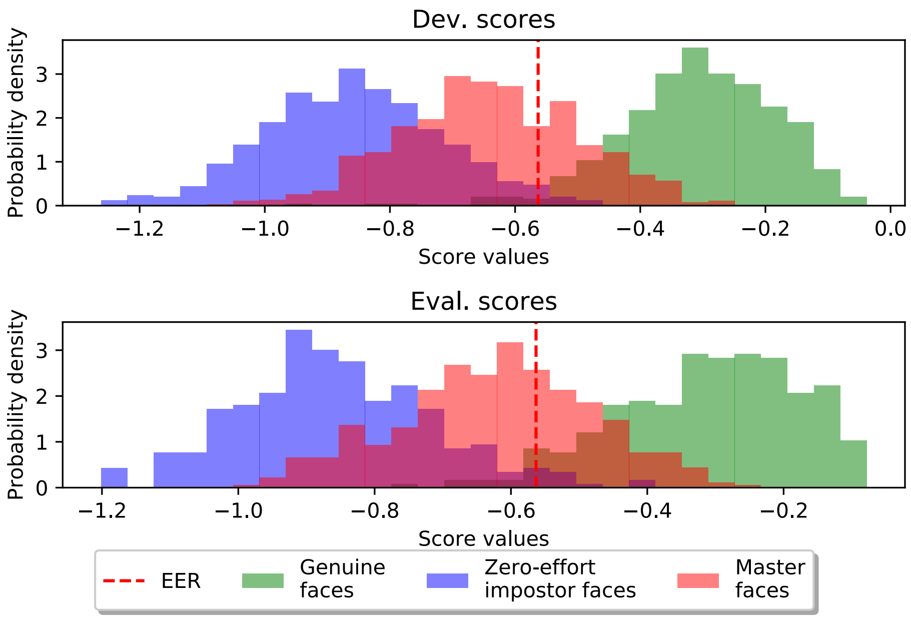

FIGURE 3.11: Histogram of scores for genuine faces, zero-effort imposter faces, and master face generated using LFW - Fold 1 database [114] calculated using Inception-ResNet-v2 based FR system [52].

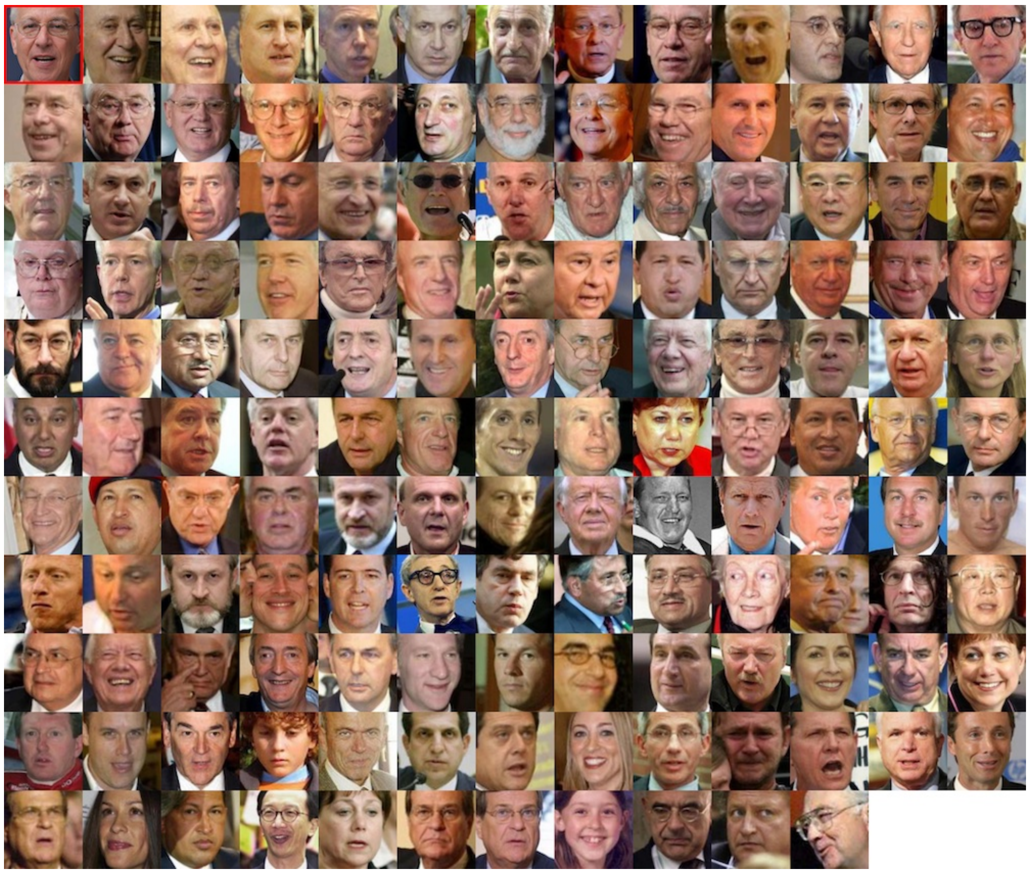

FIGURE 3.12: Master face (top left) generated using *combination 1* setting and all matched faces from eval set of the LFW - Fold 1 database [114] sorted from closest to farthest match. Inception-ResNet-v2 based FR system [47] was used in this case.

3.3.4 Master face analysis

The master face generated using the *combination 1* setting and the faces it matched using the Inception-ResNet-v2 based FR system [47] on the eval set of the LFW - Fold 1 database [114] are shown in Fig. 3.12. The master face matched those of persons of both genders, of multiple races (White, Black, and Asian), and of multiple ages (from children to elders). In many cases, the facial angles and lighting conditions differed from those of the master face. The subjects are both wearing and not wearing glasses (eyeglasses or sunglasses). A typical master face can match about 10 to 50 identities. Since the LFW database is unbalanced (as shown in Figs. 3.7 and 3.8), a large portion of the matched faces are male. Furthermore, since the master face falls in the elder cluster (discussed below), most of the matched faces are those of elders.

To better understand these results, we ran the uniform manifold approximation and projection (UMAP) dimension reduction algorithm on the embedding spaces of three

FR systems and then applied a kernel density estimation method to the reduced spaces to form the density maps. We did that from both age and gender perspectives. We used two Inception-ResNet-v2 based FR systems (CASIA-WebFace version and MS-Celeb version) and the ArcFace FR system to perform the embedding space density estimation. Among them, the two Inception-ResNet-v2 FR systems were used on both the attacker side and the defender side while the ArcFace FR system was used only on the defender side. The estimated densities are shown in Fig. 3.13. We also included the positions of the intermediate master faces' and the optimized master faces' embeddings in the plots.

From the age perspective, young faces (less than 30 years old) are separated from the elder faces (more than 60 years old), while the remaining faces (30 to 60 years old) are scattered throughout both the young and elder faces. From the gender perspective, the male and female faces are somewhat separated. To maximize the false matches, the LVE algorithm placed the master face in a dense area near the border of a cluster, which increased the probability of matching diverse faces. Since there are more male than female faces in all databases, the probability of placement in a dense area in the male cluster was higher than that of placement in the female one. However, since they were only somewhat separated, the master faces could match both male and female faces (with more male face matches, as shown in Fig. 3.12).

For age, the selected dense area could be in a young cluster, a middle-aged cluster, or a elder cluster. Since the training data for the FR systems was unbalanced in terms of age with only a few samples for young and elder faces, these systems may not accurately recognize young and elder faces. The CASIA version of the Inception-ResNet-v2 based FR system may perform poorly on elder male faces, resulting in the generation of elder male master faces. Interestingly, the master face generated using the *combination 1* setting also lies at the centroid of the ArcFace FR system, which is used only on the defender side. For this case, dense areas also exist even if we use the angular margin loss in training.

On the other hand, the MS-Celeb version of the Inception-ResNet-v2 based FR system performed poorly on young male faces, resulting in the generation of boy master faces. For *combination 2* (not fully shown in Fig. 3.13 due to limited space), we observed that the 30- to 60-year-old faces were scattered in the embedding spaces of both of these FR

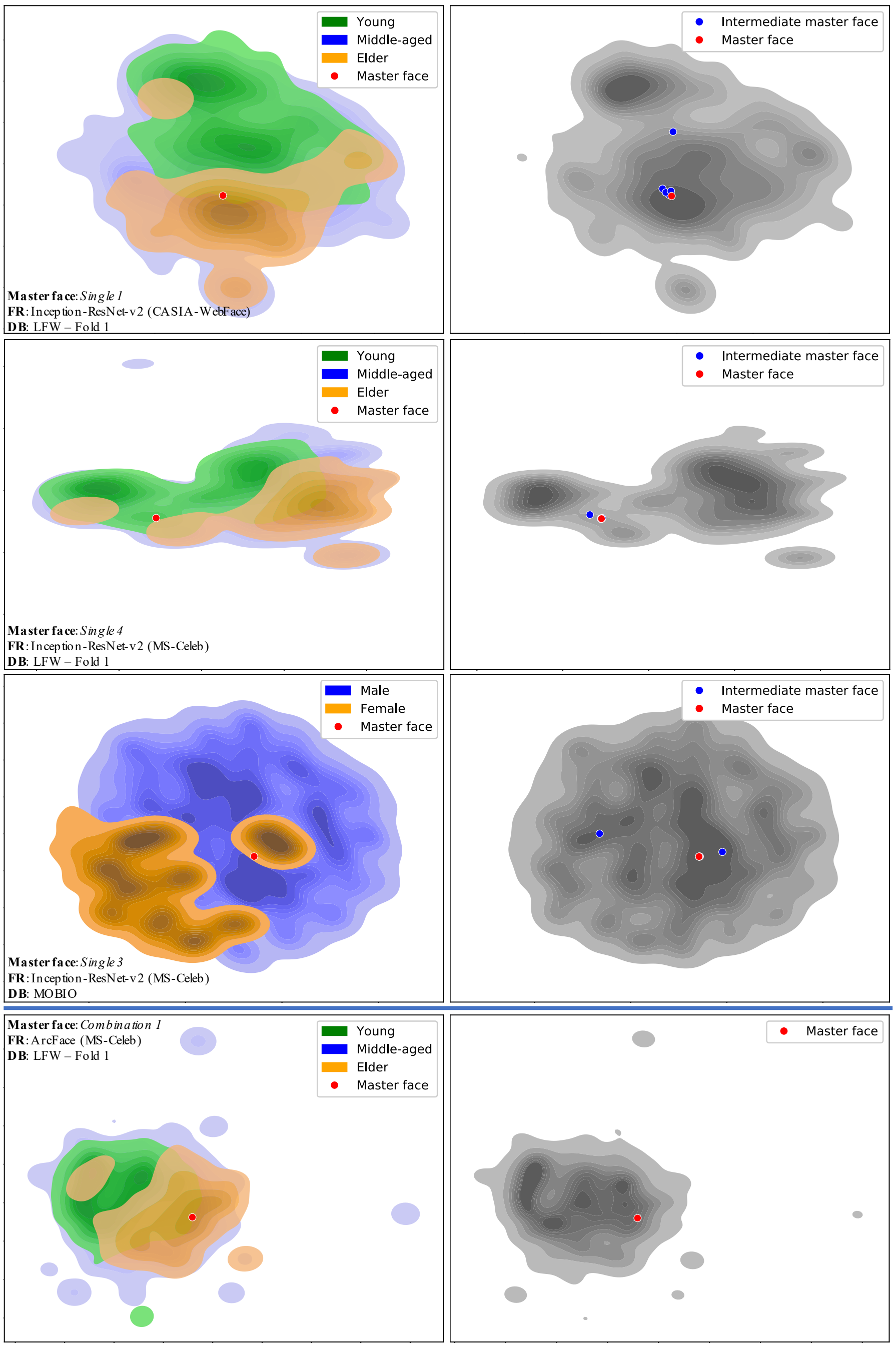

FIGURE 3.13: Estimated densities of ages (rows 1, 2, and 4) and genders (row 3) of embedded faces extracted by Inception-ResNet-v2 based FR systems trained on CASIA-WebFace database (row 1) and MS-Celeb database (rows 2 and 3) and by ArcFace FR system (row 4). Plots on left show estimated densities per class while those on right show estimated densities of all embeddings. Two Inception-ResNet-v2 based FR systems were used on both attacker and defender sides while ArcFace system was used only on defender side. We also included the embeddings of five intermediate master faces generated during running of LVE algorithm (blue dots) and of optimized master face (red dot). These embeddings were extracted from the entire training set of the LFW - Fold 1 database (rows 1, 2, and 4) and of the MOBIO database (row 3). Corresponding LVE settings (see Table 3.2 for more detail) used to generate master faces are shown in figures on left, along with information about target FR system (denoted as FR) and database (denoted as DB). Best viewed in color.

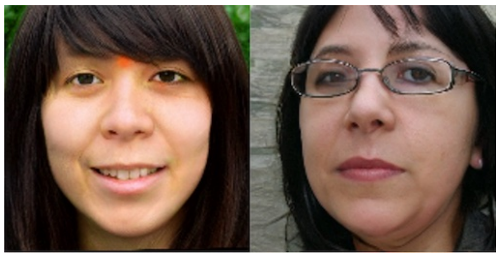

FIGURE 3.14: Two female master faces generated using only female part of MOBIO database and Inception-ResNet-v2 based FR system (MS-Celeb version) and DR-GAN FR system, respectively.

systems; it seems that an "average" middle-aged face is the optimal solution according to the proposed LVE algorithm. To further verify the effects of the clusters on the properties of the master faces, we generated two master faces using only the female part of the MOBIO database and the Inception-ResNet-v2 FR system (MS-Celeb version) and the DR-GAN FR system. Both master faces are female, as shown in Fig 3.14.

3.3.5 False matching rate analysis

Next, we evaluated the performances of attacks using master faces. The greater the number of enrolled subjects that match the generated master face, the higher the FMR. Hence, we compared the FMRs between two tests:

- *Normal test:* One side of the test pairs included either a genuine or zero-effort imposter face defined by the test protocols of the database used.
- *Master face test:* The master face was paired with the faces of all the enrolled subjects.

First, we show how the FMRs measured on the master face set changed during the LVE optimization. As shown in in Fig. 3.15, the FMRs became higher in six of the eight settings. For the two remaining settings (*combination 2* and *3*), the FMR of one of their component FR systems also became higher while that of the other one remained almost zero. In these two cases, two different databases were used with the LVE algorithm, and the algorithm tried to maximize the similarities between the master face and all faces in database 1 as calculated by component FR system 1 as well as to maximize the similarities between the master face and all faces in database 2 as calculated by component FR system 2. This task is difficult, even if the two FR systems share the

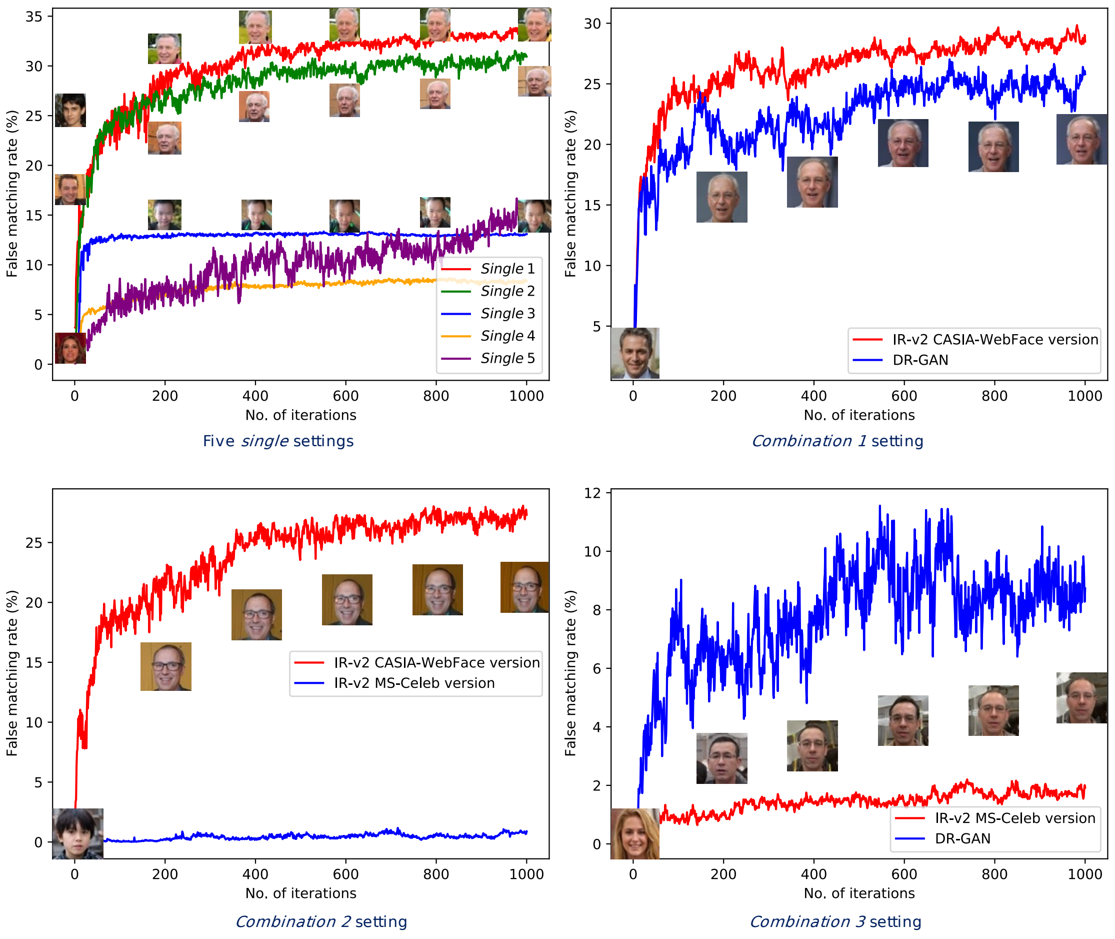

FIGURE 3.15: FMRs of each FR system when running the LVE algorithm using five *single* settings and three *combination* settings. There are two Inception-ResNet-v2 (IR-v2) FR systems, one trained on CASIA-WebFace database and one trained on MS-Celeb database. We included intermediate master faces generated using three *single* settings 1, 2, and 3, and three combination settings. Best viewed in color.

same architecture, as they do in the *combination 2* setting. Since the LFW and MOBIO databases have different distributions, finding a master face that matches the face of many subjects in both of them is challenging. The LVE algorithm focused on only one database (the LFW database) and ignored the other (the MOBIO database, which has higher variability in terms of pose and illumination conditions than the LFW database). Moreover, the Inception-ResNet-v2 based FR system trained on the MS-Celeb database was harder to fool when it was run with the LVE algorithm compared with its CASIA-WebFace version. In contrast, although two FR systems were used in the *combination 1* setting, they shared the same database, so the algorithm was able to fool both of them.

Table 3.4: FMRs of normal tests and corresponding master face tests using master faces generated using five *single* settings and three *combination* settings. For each FR system, we show both its network architecture (top row) and its training database (bottom row). Within each cell, number at upper left is FMR for normal test from development set of target database, that at upper right is FMR for normal test from evaluation set of target database, that at lower left is FMR for master face test from development set of target database, and that at lower right is FMR for master face test from evaluation set of target database. Gray cells indicate that surrogate database(s) used by attackers when running LVE algorithm and target database(s) were different while white cells indicate that they were the same. Numbers in blue indicate that surrogate FR system(s) and target FR system(s) were identical in both architecture(s) and training database(s). Numbers in bold indicate successful master face attacks. Best viewed in color.

Target DB	**Target FR System**	***Single 1***		***Single 2***		***Single 3***		***Single 4***		***Single 5***		***Combi. 1***		***Combi. 2***		***Combi. 3***	
LFW - Fold 1	Inception-ResNet-v2	2.3	3.3	2.3	3.3	2.3	3.3	2.3	3.3	2.3	3.3	2.3	3.3	2.3	3.3	2.3	3.3
	(CASIA-WebFace)	**29.9**	**34.7**	**15.4**	**19.7**	2.4	2.3	0.4	0.4	0.8	0.2	**26.6**	**29.7**	**23.4**	**27.3**	**5.1**	4.4
	Inception-ResNet-v2	0.5	0.3	0.5	0.3	0.5	0.3	0.5	0.3	0.5	0.3	0.5	0.3	0.5	0.3	0.5	0.3
	(MS-Celeb)	0.1	0.8	1.0	1.1	**7.3**	**5.7**	**10.0**	**8.1**	1.1	0.6	1.2	1.7	2.0	2.3	2.3	0.8
	FaceNet (Inception-v1)	0.7	0.3	0.7	0.3	0.7	0.3	0.7	0.3	0.7	0.3	0.7	0.3	0.7	0.3	0.7	0.3
	(MS-Celeb)	0.0	1.1	0.3	1.1	0.4	0.8	0.5	1.5	0.4	0.2	1.3	0.6	0.8	1.3	1.2	0.6
	DR-GAN	3.3	3.7	3.3	3.7	3.3	3.7	3.3	3.7	3.3	3.7	3.3	3.7	3.3	3.7	3.3	3.7
	(CASIA-WebFace)	**6.2**	**8.1**	**30.1**	**33.3**	1.1	0.8	2.0	0.8	4.3	3.6	**27.3**	**27.8**	3.5	2.8	**6.0**	**5.5**
	ArcFace	14.3	12.3	14.3	12.3	14.3	12.3	14.3	12.3	14.3	12.3	14.3	12.3	14.3	12.3	14.3	12.3
	(MS-Celeb)	11.6	13.6	**21.3**	**26.3**	2.4	2.5	4.9	2.5	9.7	7.6	**22.1**	**23.5**	14.3	15.3	13.6	14.6
MOBIO	Inception-ResNet-v2	1.9	2.1	1.9	2.1	1.9	2.1	1.9	2.1	1.9	2.1	1.9	2.1	1.9	2.1	1.9	2.1
	(CASIA-WebFace)	2.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.4	0.0	0.0	0.0	**4.8**	**5.2**
	Inception-ResNet-v2	1.0	0.4	1.0	0.4	1.0	0.4	1.0	0.4	1.0	0.4	1.0	0.4	1.0	0.4	1.0	0.4
	(MS-Celeb)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	FaceNet (Inception-v1)	0.8	0.5	0.8	0.5	0.8	0.5	0.8	0.5	0.8	0.5	0.8	0.5	0.8	0.5	0.8	0.5
	(MS-Celeb)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.7	0.0	0.0	0.0	0.0	0.0	0.0
	DR-GAN	2.3	1.3	2.3	1.3	2.3	1.3	2.3	1.3	2.3	1.3	2.3	1.3	2.3	1.3	2.3	1.3
	(CASIA-WebFace)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.4	**12.1**	0.0	0.0	0.0	0.0	**4.8**	**6.9**
	ArcFace	8.2	7.8	8.2	7.8	8.2	7.8	8.2	7.8	8.2	7.8	8.2	7.8	8.2	7.8	8.2	7.8
	(MS-Celeb)	0.0	0.0	2.4	1.7	0.0	0.0	0.0	0.0	0.0	0.0	4.8	1.7	0.0	0.0	4.8	3.4
IJB-A	Inception-ResNet-v2	10.1		10.1		10.1		10.1		10.1		10.1		10.1		10.1	
	(CASIA-WebFace)	**37.5**		**15.2**		**13.4**		9.8		2.7		**20.5**		**25.0**		2.7	
	Inception-ResNet-v2	3.1		3.1		3.1		3.1		3.1		3.1		3.1		3.1	
	(MS-Celeb)	0.0		1.8		**20.5**		**22.3**		0.0		1.8		0.9		0.0	
	FaceNet (Inception-v1)	5.7		5.7		5.7		5.7		5.7		5.7		5.7		5.7	
	(MS-Celeb)	4.5		2.7		**9.8**		**8.0**		1.8		6.2		4.5		4.5	
	DR-GAN	10.8		10.8		10.8		10.8		10.8		10.8		10.8		10.8	
	(CASIA-WebFace)	4.5		**16.1**		7.1		**14.3**		5.4		**17.9**		5.4		1.8	
	ArcFace	10.7		10.7		10.7		10.7		10.7		10.7		10.7		10.7	
	(MS-Celeb)	0.0		3.6		0.9		0.9		2.7		0.9		0.9		0.9	

Two rules for designing settings for the LVE algorithm can be inferred from these results:

- Using more than one database for running the LVE algorithm is difficult as the algorithm may prioritize the database that is less challenging.
- Using more than two FR systems is OK. They can have the same or different architectures, trained on similar or different databases.

Table 3.4 shows the FMRs for the normal tests and the corresponding master face tests using master faces generated using five *single* settings and three *combination* settings. Each cell has four numbers, the FMRs for the normal test (upper part) and the master face test (lower part), from the development set (left) and the evaluation set of the target database (right). Gray cells indicate that the surrogate database(s) used by attackers when running the LVE algorithm and the target database(s) were different while gray cells indicate that they were the same. Numbers in bold indicate successful master face attacks. There are several observations regarding the FMRs of the attacks using the master faces generated using the *single* and *combination* settings shown in Table 3.4 in connection with the FMR curves shown in Fig. 3.15:

- All FR systems are vulnerable to master face attacks. Some systems are easier to fool than others.
- With the *combination 1* setting, the master face had the attack abilities of the master faces generated using the corresponding single settings (*single 1* and *single 2*). In this case, there was no conflict.

TABLE 3.5: Summary of successful attack ratios using **five** *single* settings and **three** combination settings. Numerators are number of successful attacks; denominators are total number of attack cases. Note that for *combination 3*, some attacks fall into two settings: "Same Arch. - Different DB" and "Different Arch. - Same DB." Numbers for overlapped cases are shown red inside parentheses.

Target FR (Architecture - Training DB)	***Single* Settings**		***Combination* Settings**	
	Known target DB	**Unknown target DB**	**Known target DB**	**Unknown target DB**
Same Arch. - Same DB	**7/10**	**6/15**	**10/20**	**3/10**
Same Arch. - Different DB	0/6	**1/9**	**3/6 (3/4)**	0/4 (0/1)
Different Arch. - Same DB	**4/14**	**3/21**	**3/24 (3/4)**	0/6 (0/1)
Different Arch. - Different DB	**2/20**	**1/30**	**2/4**	0/6
Overall success ratio	0.26	0.15	0.30	0.12
	0.19		0.24	
	0.21			

- With the *combination 2* and *combination 3* settings, in which conflict occurred, their master faces were lacking some attack abilities of the master faces generated using the corresponding single settings. This is clearly seen for the *combination 2* setting, for which six attacks that were successful in the single settings failed.
- With the *combination 3* setting, for which the two component databases and FR systems differed, five attacks that had been successful were no long successful, and there were six newly successful ones. Moreover, the FMRs of the successful attacks were not as high as the those of the single setting. Although conflict still occurred in this case, it was less severe than in the *combination 2* setting.

The above observations provide valuable clues for effectively designing the LVE algorithm. Using only one database is a safe way to avoid conflicts when running the LVE algorithm. Although there negative side effects due to conflicts, using both different databases and different FR systems may result in unpredictable successful attacks when the single setting fails.

Table 3.5 summarizes the number of successful attacks using both *single* and *combination* settings. An attack is successful if the master face's FMR is higher than the normal test set's FMR. Recall that there were **five** *single* settings and **three** *combination* settings. Moreover, the *combination* settings used more than one database and/or one FR system, and there were only three databases and five FR systems used for evaluation. As a result, the total number of black/gray-box attacks (attacks on different architecture, different database) with these settings was less than that of attacks with the *single* settings.

The overall success ratio of master face attacks was 21%. White-box attacks had the highest success ratios, followed sequentially by gray-box and black-box attacks. The success rate for the *combination* settings (24%, overlapped cases removed) is higher than that for the *single* settings (19%). This means that, although the generation process is more difficult for the *combination* settings, when the attacks are successful, the master faces have stronger attack ability. Regarding black-box attacks (both target database and FR system are unknown), since we had only a limited number of scenarios (6 in total for *combination* settings compared with 30 for *single* settings), it is hard to conclude whether master faces generated using *combination* settings can successfully perform black-box attacks. The main point of using a *combination* setting is to increase the chance of an attack being a gray-box or white-box (if lucky) attack by using multiple databases and FR systems for guessing and approximating the target system.

In reality, attackers can mix several databases to create a single large database with increased generalizability. There are not many public FR system architectures; therefore, attackers can prepare in advance several master faces for each one using a mixed database.

3.4 Presentation Attacks

Finally, we evaluated the risk and threats of presentation attacks using master faces on FR systems. For master face candidates, we chose one generated using the *single 2* setting and another generated using the *combination 1* setting. For digital attack candidates, we chose two attack scenarios in the IJB-A database [117] in which the two master faces were falsely accepted by the Inception-ResNet-v2 based FR system [47] (CASIA-WebFace version) and the DR-GAN FR system [57]. We compared the FMRs of these two digital attacks with those of the corresponding presentation attacks.

3.4.1 Experiment design

To simulate simple presentation attacks like the one shown in Fig. 3.16, we needed to prepare PAIs and cameras. For the PAIs of each of the two selected master faces, we used three kinds of materials:

- Color photos printed on plain A4 paper.
- Color photos printed on 127 mm × 178 mm photo paper.

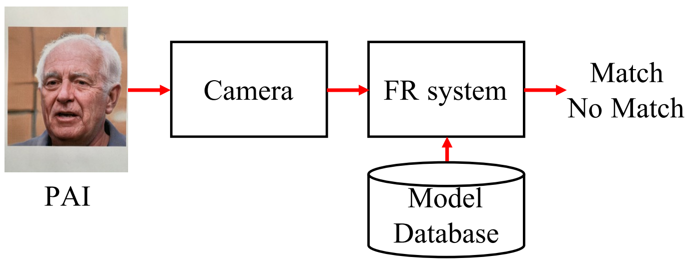

FIGURE 3.16: Overview of presentation attack on FR system.

- Color photos displayed on the screen of an Apple 13-inch MacBook Pro 2017.

For the cameras, we used two types:

- the rear camera in an iPhone XR.
- a Canon EOS 60D DSLR camera with a Canon EF 40mm F2.8 STM lens.

For simplicity, we used these cameras to take photos of the PAIs under normal room conditions. We adjusted the position of the cameras such that they were relatively perpendicular to the surface of the PAIs so they could capture the displayed PAIs as much as possible without loosing any contents. This condition is close to that of real-world presentation attacks. Three example PAIs are shown in Fig. 3.2.

3.4.2 Results

The FMRs of the attacks using PAI master faces are shown in Table 3.6 along with those of attacks using digital master faces and those of the normal dev set of the IJB-A database. The attacks were successful in 19 of the 24 cases, demonstrating that PAI master faces can be effective in real-world attacks. In eight cases, the FMRs were higher

Table 3.6: FMRs of master face PAI attacks on dev set of IJB-A database [117] using two settings: *Single 2* and *combination 1*. First line in each row shows result for Inception-ResNet-v2 based FR system [47] trained on CASIA-WebFace database, and second line shows result for DR-GAN FR system [57]. Numbers in **bold font** indicate successful attacks although there was degradation in the FMR in some cases. Numbers in **red** indicate the increment of the FMRs in the presentation attack compared with those in digital attacks.

Camera	Plain Paper	Photo Paper	MacBook Screen	Digital Master Face	Normal Dev Set
	Setting: *Single 2*				
iPhone XR	**17.9 (+2.7)**	**15.2 (0.0)**	**13.4 (−1.8)**	15.2	10.1
	18.8 (+2.7)	11.6 (−4.5)	**19.6 (+3.5)**	16.1	10.8
Canon 60D	**17.9 (+2.7)**	**18.8 (+3.6)**	**19.6 (+4.4)**	15.2	10.1
	20.5 (+4.4)	11.6 (−4.5)	**15.2 (−0.9)**	16.1	10.8
	Setting: *Combination 1*				
iPhone XR	**18.8 (−1.7)**	**19.6 (−0.9)**	**15.2 (−5.3)**	20.5	10.1
	13.4 (−4.5)	11.6 (−6.3)	8.9 (−9.0)	17.9	10.8
Canon 60D	**17.9 (−2.6)**	**19.6 (−0.9)**	**20.5 (0.0)**	20.5	10.1
	13.4 (−4.5)	8.0 (−9.9)	**18.8 (+0.9)**	17.9	10.8

than those of attacks using digital master faces. This is attributed to the distribution of PAI master faces being closer to the distribution of faces in the facial databases (which contain faces also captured with a camera) thanks to the camera processing. The lower rate in the other cases is attributed to artifacts from the PAI materials playing a bigger role than the effect of the camera processing. All of the PAI attacks using plain paper were successful while seven of the eight PAI attacks using a computer screen were successful. The attacks using photo paper, which easily reflects light, had the worst performance. Those using photos taken with the iPhone camera were more successful than those using ones taken with the Canon camera. This is attributed to the Canon camera being able to capture more detailed PAI artifacts.

3.5 Defense Against Master Face Attacks

What is the main problem of existing FR systems that causes the existence of master faces? We hypothesized that it comes from the distributions of the embedding spaces where the extracted features are not well distributed. This results in the formation of clusters, not only multi-identity clusters but also age and gender ones. There are two possible origins of this problem: (1) the training data and (2) the objective function design. Regarding the training data, as shown in Figs. 3.7 and 3.8, the training data was unbalanced in terms of age and gender. This could affect the distribution of the embeddings for which the FR systems discriminate faces in the majority group better than in the minority one. For example, the 30-60 year-old face embeddings were scattered more uniformly than the others, as shown in Fig 3.13.Simply enlarging the database has a certain effect on the robustness of the FR systems (the MS-Celeb version of the Inception-ResNet-v2 based FR system had fewer successful master face attacks than the CASIA-WebFace version); however, they are still vulnerable. It is thus important to balance the training data.

Regarding the objective function design, the objective functions are mainly designed so that same-identity embeddings stay close together while different-identity ones stay far apart. The introduction of the angular margin loss [49] improves this ability while the uniform loss [58] forces the embeddings to be uniformly distributed. Although these improvements reduce the risk of master face attacks, they mainly focus on identity. Since gender, age, and race are also important [119], the attack is successful in some cases. This suggests that the design of the objective functions used for training the FR systems needs further improvement.

Beside harnessing FR systems, using master face detectors could mitigate master face attacks. Since master faces are generated using a GAN, GAN image detectors [120–122] or deepfake detectors [123] could be used to detect them. Although looking realistic from the human perspective, computer-generated images have different properties than natural ones captured by cameras. Some GAN artifacts may exist in the generated images; therefore, most GAN image detectors focus on detecting their presence. We could also integrate a presentation attack detector [123] with an FR system to prevent master face attacks as well as other traditional presentation attacks using images or videos of the victims. However, generalization of these detectors is still a huge challenge. The StyleGAN used in the LVE algorithm could be replaced with a more advanced facial generator to fool fake image detectors. Although some degree of generalizability has been achieved, performance is still not good enough for real-world applications. Therefore, further research on generalizability is needed.

3.6 Conclusion

We demonstrated, especially in our presentation attack experiment, that master face attacks pose a severe security threat if the FR systems are not properly protected. Our intensive evaluation of the performance of the LVE algorithm using several settings, including both *single* and *combination* settings, has brought to light several properties of master faces as well as of the LVE algorithm. Some of the *combination* settings caused intra-component conflicts while others produced interesting positive results. Being aware of the existence of master faces and their properties is critical to improving the robustness of FR systems. Combining the use of an FR system with a well-designed objective function trained on a large balanced database with a fake image detector could mitigate master face attacks. Since digital attack detectors (GAN image detectors and deepfake detectors) and presentation attack detectors still have difficulty with generalization and since master face attacks continue to improve, these attacks cannot be taken lightly. Future work will focus on designing a better method to generate master faces and one to detect master face attacks.

Chapter 4

Deepfake Detection

This chapter and the following chapter aim to deal with deepfakes, *i.e.*, images and videos generated or manipulated by computers, including deep master faces. In this chapter, we introduce a novel deepfake detection network called “Capsule-Forensics,” which overcomes two of the limitations of traditional CNNs.

4.1 Introduction

To deal with the rapid growth of deepfake methods, several countermeasures have been developed to detect deepfake images and videos. Automatic feature extraction using CNNs has dramatically improved detection performance [22, 23, 28]. Several methods are image-based [22, 23, 30] while others work only on videos [26–28] or on video with voice [29]. Although some video-based methods perform better than image-based ones, they are only applicable to particular kinds of attacks. For example, some of them [26, 27] may fail if the quality of the eye area is sufficiently good or the synchronization between the video and audio parts is sufficiently natural [124]. In this chapter, we limit our scope to image-based methods since our aim is to build a general detector that can work with both generated/manipulated images and videos and does not rely on any particular kind of attack.

Conventionally, the performance of a CNN can be improved by increasing its depth [42], its width [43], and/or the number of inner connections [44]. Another solution is to use multiple CNNs as is done in Zhou *et al.*' s two-stream network [30] or to use feature aggregation (feature fusion) or output fusion (ensemble). The fusion approach has been used in several competitions [31, 110]. This approach not only improves network performance on seen data but also improves network performance on unseen data. This has resulted in CNNs and groups of CNNs becoming bigger and thus consuming more memory and computation power. Moreover, they may need more training data, which are not always available when new attacks emerge. Rather than making the network bigger, we took a different approach: redesign it to make it more efficient in memory usage, detection accuracy, and generalization.

We introduce "Capsule-Forensics" [125], a proof-of-concept capsule network [126] designed especially for detecting manipulated images and videos. We hypothesize that the special design of the network makes it better able to detect deepfakes than a corresponding CNN while keeping the network smaller. This special design includes:

- A feature extractor, which is part of a pretrained image classification CNN, prevents the network from overfitting and improves its performance on both seen and unseen attacks.
- A statistical pooling layer, which is used in each primary capsule of the network, greatly reduces the number of parameters compared with the original capsule network while improving performance on deepfake detection.
- A dynamic routing algorithm produces better fusion than the traditional feature aggregation approach.

To sum up, our contribution is four-fold:

1. We propose "Capsule-Forensics", a novel designed deepfake detection network that focuses on efficiency and memory usage.
2. We provide a theoretical explanation of the Capsule-Forensics network on deepfake detection by verifying our hypothesis that its special design is the reason it performs better than the corresponding CNN version.

3. We visualize the activation of each primary capsule as well as the routing weights and thereby clarify which kind of information these capsules learn and how they agree on the final decision of the entire network. This is a step towards explainability of the Capsule-Forensics network.

4. We introduce small deepfake detection benchmarks that focuses on detection performance, number of parameters, and inference time for both seen and unseen data.

The rest of this chapter is structured as follows. We first briefly introduce the capsule networks and our reason of choosing this kind of architecture for deepfake detection. Next, we describe the proposed Capsule-Forensics network. We also visualize the features the Capsule-Forensics network learns to understand the differences between it and a conventional capsule network, which learns the hierarchical relationships between object parts. Then, we describe several experiments we performed to test our hypothesis that the special design of the network makes it better able to detect deepfakes than a corresponding CNN while keeping the network smaller. Finally, we conclude by discussing the meaning of our results and mentioning future work.

4.2 Capsule Networks

4.2.1 Original capsule network

"Capsule network" is not a new term as it was first introduced in 2011 by Hinton *et al.* [127]. They argued that CNNs have limited ability to learn the hierarchical relationships between object parts and introduced a more robust architecture comprising several "capsules." However, they initially faced the same problem affecting CNNs – limited hardware performance – and the lack of effective algorithms, which prevented practical application of capsule networks. CNNs thus remained dominant in this research field.

These problems were overcome when the dynamic routing algorithm [126] and its variant – the expectation-maximization routing algorithm [128] – were introduced. These breakthroughs enabled capsule networks to achieve better performance and outperform CNNs on object classification tasks [126, 128–131]. The agreements between low- and

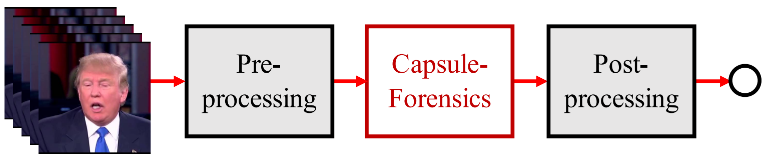

FIGURE 4.1: Capsule-Forensics unit processing.

high-level capsules, which encode the hierarchical relationships between objects and their parts with pose information, enable a capsule network to preserve more information than a CNN while using only a fraction of the data used by a CNN.

4.2.2 Why Capsule-Forensics?

To overcome the weakness of conventional CNNs, we adapted the capsule network concept [126], which was originally designed for computer vision tasks, to make it well suited for deepfake detection. We named our adapted network “Capsule-Forensics.” Its design takes advantage of transfer learning by using part of a pretrained CNN (trained on the ImageNet dataset [50]) as the feature extractor. This helps the network achieve high performance and have better generalizability. The feature aggregation used in conventional CNNs was replaced with a modified version of the dynamic routing algorithm. The use of a statistical pooling layer in each primary capsule reduces the number of parameters while improving performance. The next section describe the processing flow and architecture. We performed several experiments to verify the novelty of this design. The results are presented and discussed in the Evaluation section.

4.3 Proposed Method

4.3.1 Overview

The Capsule-Forensics based method comprises three processing units, as illustrated in Figure 4.1. The task performed in the pre-processing unit depends on the input. If the input is video, the first step is to separate the frames. A face detection algorithm is used to crop the facial area(s). The cropped face(s) are sent to the Capsule-Forensics unit for classification. The detection result(s) are sent to the post-processing unit, which works in accordance with the pre-processing one. If the input is an image, nothing is done

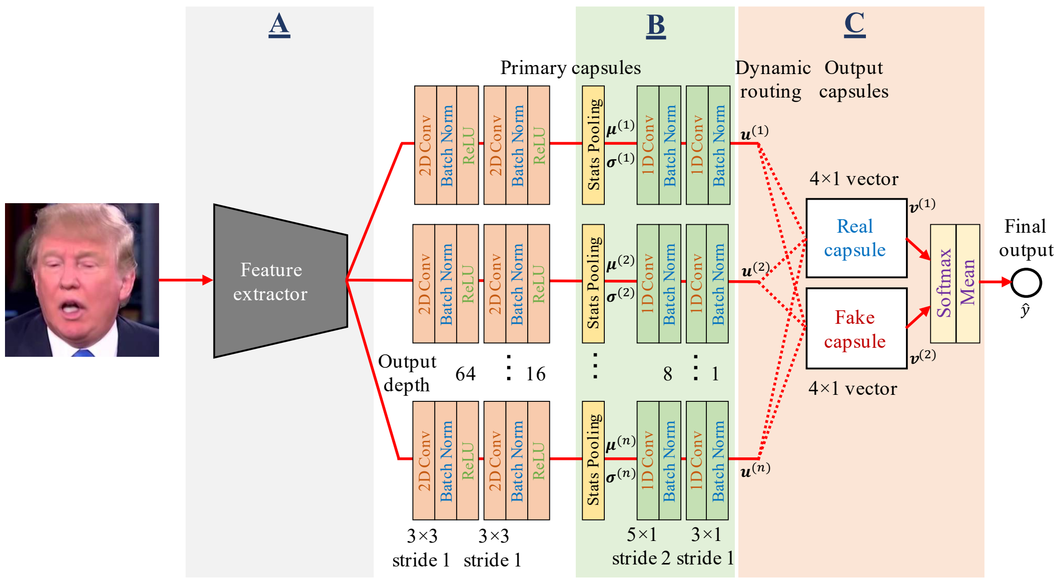

FIGURE 4.2: Capsule-Forensics architecture. Blocks A, B, and C contain tunable hyperparameters.

here. If the input is video, the scores of all frames are averaged. This average score is the final output.

4.3.2 Architecture

The Capsule-Forensics network includes a feature extractor, several primary capsules, and two output capsules (“real” and “fake”), as illustrated in Figure 4.2. For simplification, we use the same architecture for all primary capsules. Since we use random weight initialization, their behaviors are not the same after training. The number of primary capsules is a hyperparameter.

Each primary capsule has three parts: a 2D convolutional part, a statistical pooling layer, and a 1D convolutional part. The statistical pooling layer has been proven to be effective in detecting computer-generated images [32, 132] by learning the statistical differences between the real and computer-generated images. For deepfakes, when a part of a face image is swapped, the swapped face region may have different textures and color patterns. The blending region between the swapped face region and the remaining original face region may also contain artifacts. Thus, the statistics such as mean and variance of each filter are useful for differentiating the swapped region from the original one. Moreover, they help reduce the number of parameters by omitting features that are not useful for deepfake detection.

The mean and variance of each filter are calculated in the statistical pooling layer.

- Mean:
$$\mu_k = \frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} I_{kij}$$
- Variance:
$$\sigma_k^2 = \frac{1}{H \times W - 1} \sum_{i=1}^{H} \sum_{j=1}^{W} (I_{kij} - \mu_k)^2,$$

where k is the layer index, H and W are respectively the height and width of the filter, and I is a two-dimensional filter array.

The output of the statistical layer goes through the following 1D convolutional part. Then it is dynamically routed to the output capsules. The final result is calculated on the basis of the activation of the output capsules. The algorithm is discussed in detail in the next section. For binary classification, there are two output capsules, as shown in Figure 4.2. Multi-class classification could be performed by adding more output capsules, as discussed in section 4.4.3.

The Capsule-Forensics source code has been published at `https://github.com/nii-yamagishilab/Capsule-Forensics-v2`.

4.3.3 Dynamic routing algorithm

Different manipulation methods use different face regions, generating models, and blending algorithms. Therefore, each primary capsule extracts different features depending on the manipulation method, and they may work better on a particular manipulation than on others. Furthermore, since the weights of the primary capsules are initialized differently in training, the capsules learn different features for the same input. These features need to be fused correctly to predict whether the input is real or fake. For a capsule network, this fusion is done dynamically using a dynamic routing algorithm. The "agreement" between all primary capsules is calculated and routed to the appropriate output capsule (real or fake for binary classification). An example of the routing weight vectors is visualized in Figure 4.3. Since the primary capsules may make different judgments and some of them may be wrong, this algorithm is designed to find a consensus. The output probabilities are determined on the basis of the activations of

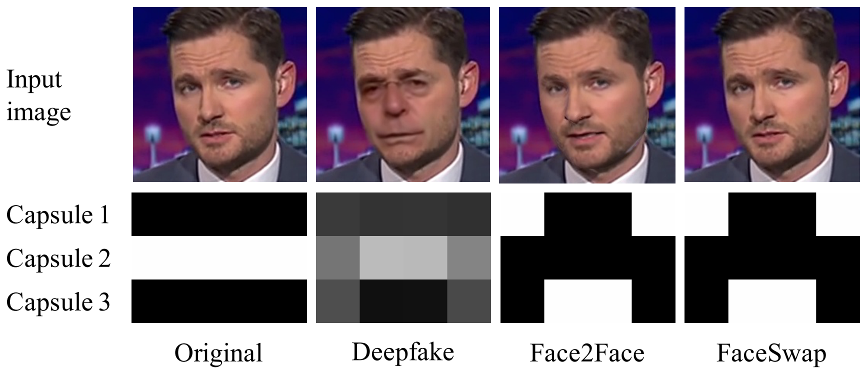

FIGURE 4.3: Visualization of the routing matrix $\mathbf{C}^{(2)\mathsf{T}}$ used to route the outputs of three primary capsules to fake output capsule. Face2Face and FaceSwap methods are graphical based, so their routing weights are similar. Deepfake method is deep learning based, so its routing weights are different from the two graphical-based manipulation methods.

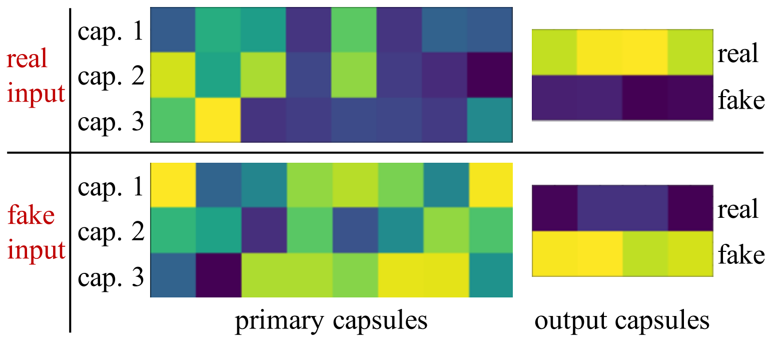

FIGURE 4.4: Average results calculated by primary capsules and output capsules from real and fake images generated with Face2Face method [17]. Three primary capsules have significantly different reactions between real and fake inputs. Although their weights are also different, there is strong agreement in the output capsules.

the output capsules. An example of the activation of the output capsules is illustrated in Figure 4.4.

Let us call the output vector of each primary capsule $\mathbf{u}^{(i)} \in \mathbb{R}^k$ and each output vector capsule $\mathbf{v}^{(j)} \in \mathbb{R}^l$. There are m primary capsules and n output capsules. $\mathbf{W}^{(i)} \in \mathbb{R}^{l\times k}$ is the matrix used to route an $\mathbf{u}^{(i)}$ to all $\mathbf{v}^{(j)}$, and r is the number of iterations. The dynamic routing algorithm is shown in Algorithm 3.

We slightly improved the algorithm of Sabour *et al.* [126] by introducing two regularizations: adding random noise to the routing matrix and adding a dropout operation. They are used **only during training** to reduce overfitting. Their effectiveness is discussed in the Evaluation section. Furthermore, a squash function (equation 4.1) is applied to $\mathbf{u}^{(i)}$

Algorithm 3 Dynamic routing between capsules.

```
procedure ROUTING($\mathbf{u}^{(i)}, \mathbf{W}^{(i)}, r$)
    $\widehat{\mathbf{W}}^{(i)} \leftarrow \mathbf{W}^{(i)} + \text{rand}(\text{size}(\mathbf{W}^{(i)}))$
    $\hat{\mathbf{u}}^{(i)} \leftarrow \widehat{\mathbf{W}}^{(i)}\text{squash}(\mathbf{u}^{(i)})$                ▷ $\hat{\mathbf{u}}^{(i)} \in \mathbb{R}^l$
    $\hat{\mathbf{u}}^{(i)} \leftarrow \text{dropout}(\hat{\mathbf{u}}^{(i)})$
    for all output capsules $j$ do
        $\mathbf{B}^{(j)} \leftarrow 0$                ▷ $\mathbf{B}^{(j)} \in \mathbb{R}^{l\times m}$
    for $r$ iterations do
        for all output capsules $j$ and all vector elements do
            $(c^{(j)}_{\_,1}, c^{(j)}_{\_,2}, \dots, c^{(j)}_{\_,m}) \leftarrow \text{softmax}(b^{(j)}_{\_,1}, b^{(j)}_{\_,2}, \dots, b^{(j)}_{\_,m})$
        for all output capsules $j$ do $\mathbf{s}^{(j)} \leftarrow \sum_i^m \mathbf{c}^{(j)}_{:,i} \odot \hat{\mathbf{u}}^{(i)}$
        for all output capsules $j$ do $\mathbf{v}^{(j)} \leftarrow \text{squash}(\mathbf{s}^{(j)})$
        for all input capsules $i$ and output capsules $j$ do
            $\mathbf{B}^{(j)} \leftarrow \mathbf{B}^{(j)} + \begin{bmatrix}\hat{\mathbf{u}}^{(1)} & \hat{\mathbf{u}}^{(2)} & \dots & \hat{\mathbf{u}}^{(m)}\end{bmatrix} \odot \mathbf{v}^{(j)}$
    return $\mathbf{v}^{(j)}$
```

before routing to normalize it, which helps stabilize the training process. The squash function is used to scale the vector magnitude to unit length.

$$squash(\mathbf{u}) = \frac{\|\mathbf{u}\|_2^2}{1 + \|\mathbf{u}\|_2^2} \frac{\mathbf{u}}{\|\mathbf{u}\|_2} \tag{4.1}$$

In practice, to stabilize the training process, the random noise should be sampled from a normal distribution ($\mathcal{N}(0, 0.01)$), the dropout ratio should not be greater than 0.05 (we used 0.05 in all experiments), and two iterations ($r = 2$) should be used in the dynamic routing algorithm. The two regularizations are used along with random weight initialization to increase the level of randomness, which helps the primary capsules learn with different parameters.

To calculate predicted label $\hat{y}$, we apply the softmax function to each dimension of the output capsule vectors to achieve stronger polarization rather than simply using the length of the output capsules [126]. The final results are the means of all softmax outputs:

$$\hat{\mathbf{y}} = \frac{1}{l} \sum_i^l \text{softmax}(v_i^{(1)}, v_i^{(2)}, \dots, v_i^{(n)}), \tag{4.2}$$

where $\hat{\mathbf{y}}$ is the predicted probabilities vector. Since there is no reconstruction in the Capsule-Forensics method, we simply use the cross-entropy loss function and the Adam optimizer [133] to optimize the network.

4.3.4 Visualization

To illustrate how Capsule-Forensics works, we used a Capsule-Forensics network with three primary capsules trained on the FaceForensics++ database [23]. For visualization, we applied and modified an open-source tool [134] implementing the guided back propagation algorithm [135]. To visualize each primary capsule in this way, we chose the latent features extracted before the statistical pooling layers since they still had the 2D structure.

The activations of each capsule and of the whole network are illustrated in Figure 4.5. The differences in activation among capsules and between each capsule and the whole network are also shown. The regions of interest mainly include the eyes, nose, mouth region, and facial contours. Some capsules missed some of these regions, and some failed to detect the manipulated input (*i.e.*., the third capsule in Figure 4.6). Nevertheless, the final results mostly focused on the important regions detected by all capsules due to agreement driven by the dynamic routing algorithm between the other two capsules. A CNN using only the third primary capsule would fail to detect the manipulated input.

The behavior of the Capsule-Forensics network for the deepfake detection problem differs from that of the original capsule network for the inverse graphics problem, in which the focus is on the spatial hierarchies between simple and complex objects [126–128]. In the deepfake detection problem, abnormal appearances are the key features, so each primary capsule is designed to capture them and communicate its findings to the other capsules. This behavior is similar to that of jurors during a trial, and the consensus judgment is the final detection result.

4.4 Evaluation

We conducted several experiments to test the detection performance of the Capsule-Forensics network. After describing the datasets and metrics we used (section 4.4.1 and 4.4.2), we discuss the effects of the size of the input image, number of primary capsules, and dropout in the dynamic routing algorithm (section 4.4.3). We then compare several candidate feature extractors (section 4.4.4) and evaluate the effectiveness of the

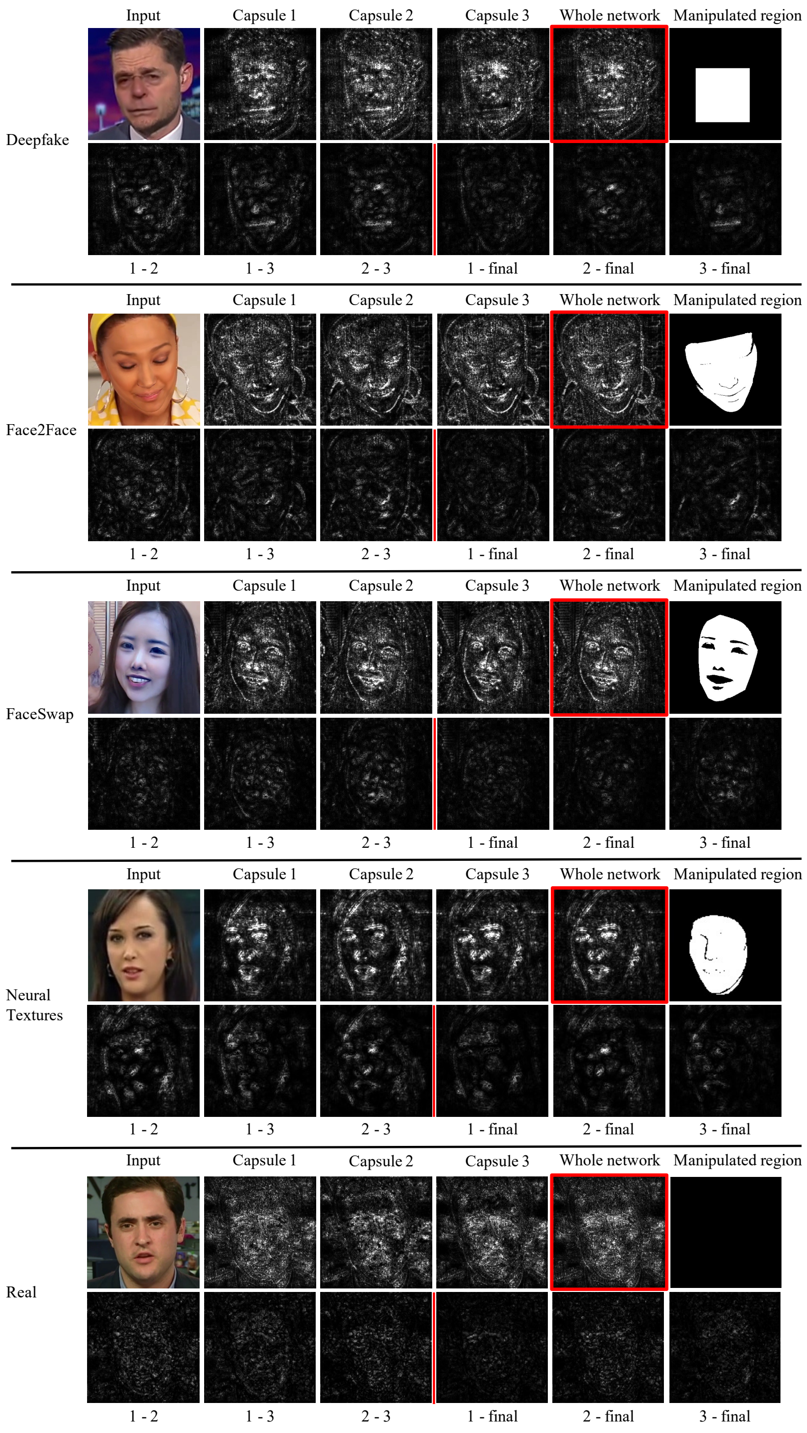

FIGURE 4.5: Activation of three capsules and entire Capsule-Forensics network (columns 2, 3, 4, and 5, respectively) on images created using deepfake [7] (row 1), Face2Face [17] (row 3), FaceSwap [23] (row 5), and Neural Textures [18] (row 7) methods and on a real image. Column 6 shows the manipulated regions corresponding to the manipulated images in column 1. The first three columns of rows 2, 4, 6, 8, and 10 show the differences between the activations of capsules 1 and 2, 1 and 3, and 2 and 3 on the corresponding row above, respectively. The three last columns in order show the differences between the activations of capsules 1, 2, and 3 and the activation of the whole network.

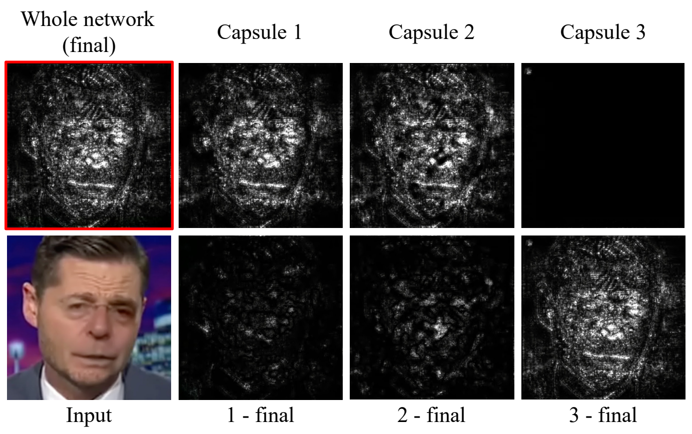

Figure 4.6: Example case in which one capsule did not work correctly. First row shows activation of whole network and of three capsules. Second row from left to right shows input image and differences between activation of each capsule and of whole network. Although capsule 3 failed to detect manipulated image, final result was correct due to agreement between other two capsules.

statistical pooling layer used in each primary capsule (section 4.4.5). Finally, we compare the detection performance of the Capsule-Forensics network with that of a CNN on both seen and unseen attacks (section 4.4.6 and 4.4.7, respectively). For the CNNs, we used the corresponding version of the Capsule-Forensics network using feature aggregation instead of the dynamic routing algorithm, the multi-task learning network [33], the XceptionNet version used in FaceForensics++ work [23], and the EfficientNet network [136]. Among them, the multi-task learning network is a generative classifier while the rest are discriminative classifiers. For the multi-task learning network, in addition to ground-truth labels, segmentation masks of the manipulated regions are needed for training. When testing, since segmenting manipulated regions is beyond the scope of this work, we used only its encoder part to perform binary classification. For XceptionNet, we modified its fully connected layer and trained it in two phases. For EfficientNet [136], which recently received a high score in the Deepfake Detection Challenge, we used the B4 version (denoted as EfficientNet-B4) which requires an input size of 380×380 pixels. The larger versions (B5, B6, and B7) require larger inputs and have more parameters, making it impossible to train them on a single-GPU machine.

For simplicity, we used only multi-class classification to compare the original setting in our previous work [125] with the new setting in this work. For the remaining experiments, we tested only binary classification. Except for the one discussed in section 4.4.7, all the evaluations were for performance on seen attacks.

4.4.1 Datasets

4.4.1.1 Face swapping datasets

We used videos from the FaceForensics++ dataset [23], supplemented with the Google DFD dataset [101] for all following experiments except for the GAN image detection experiment. We used all three levels of compression (none, moderate, and high) and mixed them together to make multiple compression datasets for our experiments. For training, we used version 1 of the FaceForensics++ dataset including original videos and three corresponding manipulated videos created by deepfake [7], Face2Face [17], and FaceSwap [23] methods. For testing, two scenarios were used: seen attacks and unseen attacks. For seen attacks, we used a test set from version 1 of the FaceForensics++ dataset. For unseen attacks, we used test videos created using Neural Textures [18] (unseen method), which was added in version 2 of the FaceForensics++ dataset, and the entire Google DFD dataset [101] (unseen data).

We took the first 100 frames of the input video for the training set and the first 10 frames for the validation and test sets. FaceForensics++ dataset version 1 (for seen attacks) was divided into a training set, a validation set, and a test set, as shown in Table 4.1. The test sets for unseen attacks are shown in Table 4.2.

TABLE 4.1: Configuration of training, validation, and test sets from FaceForensics++ dataset version 1 (for seen attacks) [23].

Type	Training set	Validation set	Test set
Real	720×3 vids $72,000 \times 3$ imgs	140×3 vids $1,400 \times 3$ imgs	140×3 vids $1,400 \times 3$ imgs
Deepfake	720×3 vids $72,000 \times 3$ imgs	140×3 vids $1,400 \times 3$ imgs	140×3 vids $1,400 \times 3$ imgs
Face2Face	720×3 vids $72,000 \times 3$ imgs	140×3 vids $1,400 \times 3$ imgs	140×3 vids $1,400 \times 3$ imgs
FaceSwap	720×3 vids $72,000 \times 3$ imgs	140×3 vids $1,400 \times 3$ imgs	140×3 vids $1,400 \times 3$ imgs

TABLE 4.2: Configuration of test sets for unseen attacks created using Neural Textures method [18] and Google DFD dataset [101].

Type	Neural Textures (unseen method)	Google DFD dataset (unseen data)
Real	0 vids 0 imgs	140×3 vids $1,400 \times 3$ imgs
Fakes	358×3 vids $3,580 \times 3$ imgs	$3,065 \times 3$ vids $30,650 \times 3$ imgs

Table 4.3: Configuration of GAN image dataset. Texts in red mean unseen GAN methods.

Type	Real	GAN
Training set	19,745 VoxCeleb	Low-quality
	20,000 FF++Orignial	• 9,745 StarGAN
		High-quality:
		• 10,000 ProGAN
		• 10,000 StarGAN2
		• 10,000 StyleGAN
	39,745 Total	**39,745 Total**
Validation set	100 VoxCeleb	Low-quality:
	100 FF++ Orignial	• 50 StarGAN
	50 VidTIMIT	• 50 RelGAN
		High-quality:
		• 50 ProGAN
		• 50 StarGAN2
		• 50 StyleGAN
	250 Total	**250 Total**
Test set	400 VoxCeleb	Low-quality:
	400 FF++ Orignial	• 200 StarGAN
	400 VidTIMIT	• 200 RelGAN
		High-quality:
		• 200 ProGAN
		• 200 StarGAN2
		• 200 StyleGAN
		• 200 StyleGAN2
	1,200 Total	**1,200 Total**

4.4.1.2 GAN image dataset

We designed a balanced dataset with both real and GAN images, divided into training, validation, and test sets, shown in Table 4.3. For real images, to ensure diversity, we borrowed video frames from the VoxCeleb dataset [137], the original part of the FaceForensics++ dataset [23], and the VidTIMIT dataset [138]. For GAN images, we used various GAN methods. In more detail, for low-quality GAN images, we used StarGAN [10] and RelGAN [139]. For high-quality GAN images, we used ProGAN [8], StyleGAN [9], StyleGAN2 [83], and StarGAN2 [11]. RelGAN and StarGAN2 were treated as unseen GAN methods.

4.4.2 Metrics

We used four metrics in our evaluation:

- Classification accuracy $= \frac{TP+\text{TN}}{\text{TP+TN+FP+FN}}$, where TP, TN, FP, and FN are true positive, true negative, false positive, and false negative, respectively.
- Equal error rate (EER): common value when false positive rate (FPR) equals false negative rate (FNR). FPR $= \frac{\text{FP}}{N}$ (number of false positives divided by number of negatives). FNR $= \frac{\text{FN}}{P}$ (number of false negatives divided by number of positives).
- Half total error rate (HTER): HTER $= \frac{\text{FPR+FNR}}{2}$.
- Attack presentation classification error rate (APCER): "proportion of attack presentations using the same PAI species incorrectly classified as bona fide presentations in a specific scenario."[1]

The thresholds used to determine whether the classification outputs were real or fake were selected on the basis of the EERs calculated for the development sets.

4.4.3 Effect of hyper-parameters

In the first experiment, we measured the effect of the size of the input image, number of primary capsules, and dropout in the dynamic routing algorithm. Since Capsule-Forensics is not limited to binary classification, we also evaluated its multi-class classification ability by changing the number of output capsules, from "Real" and "Fake" capsules to "Real," "Deepfake," "Face2Face," and "FaceSwap" capsules. This modification is obvious and did not require substantial changes to the network architecture.

[1]ISO/IEC 30107-3 definition. Accessed at `https://www.iso.org/obp/ui/#iso:std:iso-iec:19989:-1:ed-1:v1:en:term:3.1`

Table 4.4: Performance of Capsule-Forensics with various hyper-parameter values.

Input size	**No. of capsules**	**Random noise**	**Dropout**	**Binary classification accuracy (%)**	**Binary classification HTER (%)**	**Multi-class classification accuracy (%)**
128 × 128	3	No	No	87.45	15.41	85.89
128 × 128	3	Yes	No	88.57	15.35	87.12
300 × 300	3	No	No	89.88	11.28	87.51
300 × 300	3	Yes	No	90.86	11.29	87.54
300 × 300	10	No	No	91.61	11.52	88.51
300 × 300	10	Yes	No	91.32	12.07	89.98
300 × 300	3	No	Yes	91.33	12.37	89.19
300 × 300	3	Yes	Yes	91.19	11.93	88.44
300 × 300	10	No	Yes	92.17	10.70	90.51
300 × 300	**10**	**Yes**	**Yes**	**92.00**	**10.64**	**91.22**

As shown in Table 4.4, using larger images improved performance substantially as expected. Although the random noise did not result in improvement in all cases, it still played an important role in reducing the HTER when combined with dropout and increased the accuracy of multi-class classification. Increasing the number of primary capsules also helped improve performance. The combination of all three points achieved the best performance for both binary and multi-class classification.

4.4.4 Feature extractor comparison

The feature extractor is an important part of the Capsule-Forensics network (block **A** in Figure 4.2). Rather than training a simple CNN from scratch along with the other parts of the network, as is done in the traditional capsule network approach [126], we used part of a pretrained CNN (trained on the ImageNet dataset [50]). We selected three commonly used extractors as candidates:

- VGG-19 [51]: used from the beginning until the third max pooling layer.
- ResNet-50 [42]: used from the beginning until the end of the "conv3_x" layer.
- XceptionNet [91]: used from the beginning until the end of the first block of its "middle flow."

We name the Capsule-Forensics with three primary capsules and 128×128 input size "Capsule-Forensics light". The bigger version with ten capsules and 300×300 input size is named "Capsule-Forensics full". In addition to evaluating these two version with different feature extractor candidates, we evaluated a simple CNN with three convolutional layers as the feature extractor, like the ones used in conventional capsule networks. The CNN was trained along with the other parts of the Capsule-Forensics network. In addition, we also fine tuned the pretrained feature extractors (indicated by "FT" after their names) to check whether finetuning helps improve overall performance. The results are shown in Table 4.5.

All the extractors performed better with the Capsule-Forensics full. Finetuning did not help much for Capsule-Forensics full. Besides reducing memory usage and shortening training time, using pretrained feature extractors resulted in better performance than

TABLE 4.5: Performance (in %) of feature extractors with and without finetuning (FT).

Feature extractor	**Training accuracy**	**Test accuracy**	**Test HTER**	**No. of parameters**
Capsule-Forensics light:				
Simple CNN	98.97	83.36	25.42	371,712
VGG-19	99.81	88.57	15.35	2,325,568
VGG-19 FT	99.54	90.08	12.49	2,325,568
ResNet-50	99.60	88.21	16.09	225,344
ResNet-50 FT	99.69	87.45	13.60	225,344
XceptionNet	99.58	85.52	19.10	2,720,736
XceptionNet FT	99.45	85.41	18.91	2,720,736
Capsule-Forensics full:				
VGG-19	99.83	**92.00**	**10.64**	2,325,568
VGG-19 FT	99.63	90.98	13.40	2,325,568
ResNet-50	99.17	90.59	14.60	**225,344**
ResNet-50 FT	99.69	90.14	14.94	225,344
XceptionNet	99.79	90.42	13.35	2,720,736
XceptionNet FT	99.84	91.39	10.85	2,720,736

using a CNN extractor trained from scratch. These results support our hypothesis that using a pretrained feature extractor contributes to the superiority of our Capsule-Forensics network.

The ResNet-50 based feature extractor has the smallest number of parameters, making it about ten times smaller than the VGG-19 and XceptionNet ones. The VGG-19 extractor with the new setting achieved the highest classification accuracy and had the lowest HTER. For dealing with seen manipulations, if performance is more important than the number of parameters, VGG-19 is the best choice. Otherwise, ResNet-50 is more suitable.

4.4.5 Effect of statistical pooling layers

In another experiment, we compared the performance and size of two versions of the Capsule-Forensics network: one using and one not using a statistical pooling layer for each primary capsule (block **B** in Figure 4.2). Previous work [32, 132] suggested that using a statistical pooling layer is effective for detecting computer-generated images. For the version without statistical pooling layers, we replaced the 1D convolutional layers with 2D ones and added an adaptive average pooling layer at the end of each primary capsule. We hypothesized that the statistical pooling layer helps filter out unnecessary information, *i.e.*., information that is not relevant to deepfake detection. Therefore,

Table 4.6: Performance (in %) with and without statistical pooling (SP) layer in primary capsules with VGG-19 feature extractor. (Number of parameters does not include number for feature extractor.)

Settings	**Test accuracy**	**Test HTER**	**No. of parameters**
Capsule-Forensics light:			
With SP layer	88.57	15.35	1,571,070
Without SP layer	83.51	15.78	6,689,280
Capsule-Forensics full:			
With SP layer	**92.00**	**10.64**	**1,571,070**
Without SP layer	87.70	11.65	6,689,280

using a statistical pooling layer in each primary capsule helps reduce feature size and improve performance. Moreover, reducing the feature size results in a smaller routing matrix, which uses less memory and computation power. We used the VGG-19 feature extractor in this experiment. The results are shown in Table 4.6.

With both light and full version, using statistical pooling layers greatly improved classification accuracy and reduced the HTER for the seen test set. Moreover, using them reduced the number of parameters by 400%. These results support our hypothesis that using statistical pooling layers contributes to the superiority of our Capsule-Forensics network. An interesting observation from the results is that the number of parameters was independent of the input size (128×128 for light version and 300×300 for full version). This is because both the statistical and adaptive average pooling layers were designed to deal with variations in input size.

4.4.6 Capsule-Forensics network vs. CNNs: seen attacks

In a third experiment, we compared the performance of the dynamic routing algorithm used in the Capsule-Forensics network with that of traditional feature aggregation (block **C** in Figure 4.2). The VGG-19 feature extractor was used in both cases. We also evaluated the performance of the multi-task learning network [33], the XceptionNet network, and the EfficientNet-B4 network [136]. It is important to note that this version of XceptionNet differs from the one used in our feature extractor (section 4.4.4), which was pretrained on the ImageNet dataset [50], with only part of it used. Since the training dataset was imbalanced (the number of fake samples was three time the number of real samples), we additionally evaluated the effect of using a weighted softmax function during training. The experiment results are shown in Table 4.7.

Table 4.7: Performance (in %) of Capsule-Forensics using dynamic routing algorithm, its corresponding CNN using the traditional feature aggregation approach, and the other baselines on seen attacks. Number of parameters is for entire network, including feature extractor.

Settings	**Test accuracy**	**Test HTER**	**No. of parameters**
Capsule-Forensics light:			
Dynamic routing	88.57	15.35	2,796,889
Feature aggregation	86.26	15.15	2,798,059
Capsule-Forensics full:			
Dynamic routing	**92.00**	10.64	**3,896,638**
Feature aggregation	91.82	11.51	3,903,328
Multi-task learning [33]	73.08	26.30	148,200
XceptionNet [23]	90.73	9.91	20,811,050
EfficientNet-B4 [136]	**92.82**	**8.67**	17,552,202
Using weighted softmax:			
Dynamic routing	**92.21**	10.91	**3,896,638**
Feature aggregation	91.75	10.68	3,903,328
XceptionNet [23]	91.83	10.14	20,811,050
EfficientNet-B4 [136]	91.49	**8.64**	17,552,202

The effect of using a weighted softmax function is not clear. Since the dataset was not heavily imbalanced, this result is reasonable. Although having the smallest number of parameters, the multi-task learning network had the worst performance. The dynamic routing algorithm helped the Capsule-Forensics network achieve higher performance, especially with the full version. The numbers of parameters for the Capsule-Forensics network and the corresponding CNN using feature aggregation were almost the same, whereas the numbers for the EfficientNet-B4 and the XceptionNet networks were about 4.5 to 5.3 times larger. Moreover, the test accuracy of the Capsule-Forensics network and the Efficient-B4 network were almost the same. The large input size of the EfficientNet-B4 network (380×380 vs 300×300) might be the reason for its lower HTER.

In addition to the results on the mixed compression test set shown in Table 4.7, we also broke it down into three compression levels, as shown in Table 4.8. There were no substantial differences between the performances of Capsule-Forensics, XceptionNet, and EfficientNet-B4. Their performances were degraded from no compression to moderate compression to high compression. With their average accuracy about 84%, detecting highly compressed deepfake videos was still challenging when most of the deepfake artifacts were erased by the compression algorithm. Capsule-Forensics and EfficientNet handled the moderately compressed deepfake videos quite well, with only about 3% degradation in accuracy compared with the uncompressed ones.

Table 4.8: Performance (in %) of Capsule-Forensics and other classifiers at three levels of compression on the FaceForensics++ dataset.

Detector	No compression		Moderate compression		High Compression	
	Accuracy	**HTER**	**Accuracy**	**HTER**	**Accuracy**	**HTER**
Capsule-Forensics	97.27	3.87	94.62	6.42	84.11	21.64
Multi-task learning [33]	81.12	17.80	69.23	25.94	68.86	35.19
XceptionNet [23]	96.12	4.80	92.82	7.60	83.25	17.33
EfficientNet-B4 [136]	98.37	2.50	95.50	4.88	84.96	18.62

Using the Capsule-Forensics network can save a large amount of memory and computation power compared with the amounts used by CNNs while maintaining high performance even for compressed videos. This is important for applications integrating a presentation attack detector into an Internet of things or a handheld device that does not have powerful hardware to prevent unauthorized facial authentication. The Capsule-Forensics network demonstrated it effectiveness against this kind of attack [125].

4.4.7 Capsule-Forensics network vs. CNNs: unseen attacks

Detecting unseen attacks is a difficult problem in deepfake detection, especially for machine-learning-based detectors. When the data distribution changes, the learned features and decision boundaries are usually no longer correct. Furthermore, large networks with a large number of parameters tend to memorize the training data, especially when the data amount is small. We expected that the Capsule-Forensics network can be better generalized than large networks thanks to the statistical pooling operation and dynamic routings of the primary capsules. To test this, we performed one last experiment in which we tested the detectors on a challenging unseen manipulation method, Neural Textures [18]. It is unlike any of the methods normally used to create seen datasets. We also tested the detectors on a different large deepfake dataset, the Google DFD dataset. We evaluated three full versions of the Capsule-Forensics network with different feature extractors (VGG-19, ResNet-50 (lightweight) and finetuned XceptionNet) and with two versions of a CNN using feature aggregation (with VGG-19 and ResNet-50 feature extractors), the multi-task learning network [33], the XceptionNet network [23], and the EfficientNet-B4 network [136].

As shown in Table 4.9, all the detectors performed poorly on the Neural Textures method, with APCERs greater than 50%. The three best detectors on seen attacks

TABLE 4.9: Performance (in %) of three versions of Capsule-Forensics network, two versions of the corresponding CNN, and other baselines on unseen attacks. Number of parameters is for **entire network**, including feature extractor.

Detectors	Neural Textures		Google DFD dataset		No. of
	Accuracy	APCER	Accuracy	HTER	parameters
Capsule-Forensics (VGG-19)	24.33	75.67	44.51	40.29	3,896,638
Capsule-Forensics (ResNet-50)	**37.93**	**62.07**	**64.98**	40.89	**1,796,414**
Capsule-Forensics (XceptionNet FT)	31.38	68.62	55.73	38.30	4,007,673
Feature aggregation (VGG-19)	28.81	71.19	58.09	38.70	3,903,328
Feature aggregation (ResNet-50)	24.00	76.00	62.48	37.70	1,803,104
Multi-task learning [33]	**44.69**	**55.31**	**78.74**	42.21	**148,200**
XceptionNet [23]	26.79	73.21	47.29	40.37	20,811,050
EfficientNet-B4 [136]	31.55	68.45	58.63	**34.23**	17,552,202

(Capsule-Forensics using VGG-19, XceptionNet, and EfficientNet-B4 - which are discriminative classifiers) had the worst performances on this method. The multi-task learning network (which is a generative classifier) achieved the best results, followed by the lightweight Capsule-Forensics network using the ResNet-50 feature extractor. The performances of all detectors were slightly better on the Google DFD dataset. The Capsule-Forensics network using ResNet-50 again had the second highest accuracy, below the multi-task learning network. Since the multi-task learning network was specially designed to deal with unseen attacks, it was able to beat all the other detectors. However, its drawback is poor performance on seen attacks, as seen in the previous section.

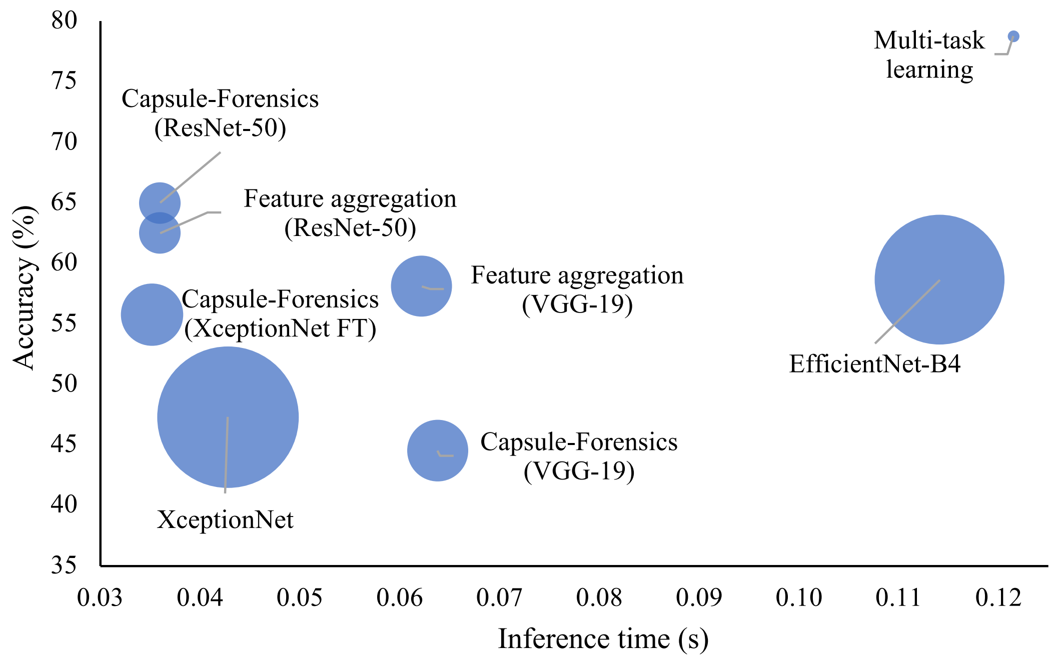

FIGURE 4.7: Comparison between several versions of Capsule-Forensics network and CNNs for classification accuracy, inference time, and model size on Google DFD dataset [101].

Figure 4.7 shows a comparison on the classification accuracy, inference time (for one image), and model size of all detectors on the Google DFD dataset [101]. All tests were done using a NVIDIA DGX Station machine. The Capsule-Forensics network using the ResNet-50 feature extractor and its corresponding CNN using feature aggregation had the second smallest sizes and were the second fastest detectors. They were a bit slower than the Capsule-Forensics network using the XceptionNet feature extractor. Due to the design of the VGG-19 network, detectors using it as the feature extractor have the longest inference times (about twice the shortest times). The XceptionNet-based detector had the largest size but had limited detection accuracy. The EfficientNet-B4-based detector and the multi-task learning detector were the two slowest ones. It is important to note that we measured only the inference time of the encoder part of the multi-task learning detector for the binary classification task. Although it has fewer parameters than the other detectors, some memory-related operations slowed it down.

Although having limited performance on unseen attacks, this experiment demonstrated that the Capsule-Forensics network is better able to detect deepfakes than CNNs. Between the two versions of the Capsule-Forensics network, if performance on seen attacks is more important, using VGG-19 as the feature extractor is the better choice. If performance on unseen attacks is more important, or a lightweight and fast network is needed, using ResNet-50 as the feature extractor is the better choice.

4.4.8 Detect GAN images

Different from the above experiments, the fake images in this experiment were synthesized entirely by GANs. We used the GAN image dataset described in Table 4.3 to train and evaluate the detectors. The results are shown in Table 4.10. All three detectors had approximately the same high performance ($> 99\%$ accuracy). Although there were images synthesized by an unseen GAN method (StyleGAN2) in the test set, the detectors easily detected them using their knowledge of GAN artifacts.

TABLE 4.10: Performance (in %) of Capsule-Forensics, XceptionNet, and EfficientNet-B4 networks on GAN image dataset.

Detectors	**Accuracy**	**EER**	**HTER**
Capsule-Forensics (VGG-19)	99.33	0.58	0.67
XceptionNet [23]	99.79	0.25	0.21
EfficientNet-B4 [136]	99.25	0.67	0.75

4.5 Conclusion

Our experiments demonstrated that the Capsule-Forensics network is better able to detect deepfakes than conventional CNNs. Its use of a pretrained feature extractor, statistical pooling layers, and a dynamic routing algorithm enable it to achieve better performance with fewer parameters than corresponding CNNs. Furthermore, it has better performance than other discriminative classifiers on unseen manipulations, although further improvement is needed. Visualization of the activation of each capsule enables the learned features to be analyzed. These promising results and the understanding gained from the analysis should lead to further research on and development of capsule networks, not only for digital forensics but also for many other applications.

Another important point is that facial images generated using current GAN methods can be effectively detected by the Capsule-Forensics network and other state-of-the-art classifiers. This means that we can use these detectors to detect master faces in digital attacks.

Future work includes enabling the Capsule-Forensics network to use temporal information to detect fake videos and improving its generalizability (in other words, reducing the gap between discriminative classifiers and generative classifiers). Moreover, deepfake datasets mostly contain images and videos containing only one or two people. In reality, deepfake methods can be applied to a crowd; therefore, deepfake detection in the wild is also an important research direction.

Chapter 5

Deepfake Segmentation

Another major concern in digital image forensics is locating manipulated regions. In this chapter, we introduce a novel method for segmenting manipulated regions in addition to detecting deepfake images. The proposed method can easily adapt to new deepfake methods even with a limited amount of fine-tuning data.

5.1 Introduction

Explainability of deepfake detection is important, especially in some critical applications like journalism or law enforcement. Most deepfake countermeasure methods only provide the label of the input or its probability of being manipulated. Moreover, most deepfake detectors are CNN-based, which are treated like black-boxes. Therefore, the detection results are inconvincible. One solution is to provide the segmentation of the manipulated regions. The shapes of the segmentation masks could also reveal hints about the type of manipulation used, as illustrated in Figure 5.1. Most existing forensic segmentation methods focus on traditional manipulations methods with three commonly used means of tampering: removal, copy-move, and splicing [95, 96, 140]. As in other image segmentation tasks, these methods need to process full-scale images. Rahmouni *et al.* [132] used a sliding window to deal with high-resolution images, as subsequently

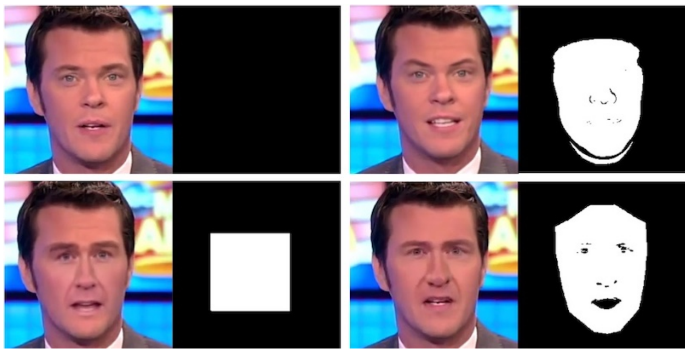

FIGURE 5.1: Original video frame (top left), video frame modified using Face2Face method [17] (top right, smooth mask almost completely covers the skin area), using Deepfakes method [7] (bottom left, rectangular mask), and using FaceSwap method [23] (bottom right, polygon-like mask).

used by us [32] and Rossler *et al.* [99]. This sliding window approach effectively segments manipulated regions in spoofed images [99] created using the Face2Face method [17]. However, these methods need to score many overlapped windows by using a spoofing detection method, which takes a lot of computation power. Another challenging problem is to locating manipulated regions of deepfake images and videos produced by unseen methods. Moreover, it is challenging to make the detectors work with new methods when the training data for them is limited.

We develop a multi-task learning approach for simultaneously performing classification and segmentation of manipulated facial images [33]. Our autoencoder comprises an encoder and a Y-shaped decoder and is trained in a semi-supervised manner. The activation of the encoded features is used for classification. The output of one branch of the decoder is used for segmentation, and the output of the other branch is used to reconstruct the input data. The information gained from these tasks (classification, segmentation, and reconstruction) is shared among them, thereby improving the overall performance of the network. This also helps the model to have better adaptability with unseen deepfake methods.

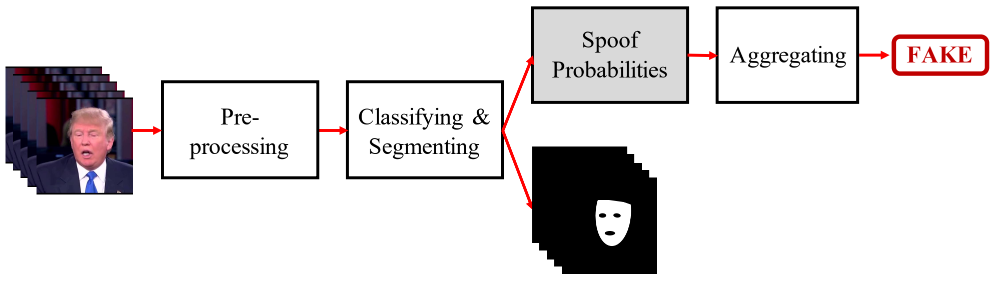

Figure 5.2: Overview of proposed network.

5.2 Proposed Method

5.2.1 Overview

Unlike other single-target methods [96, 125, 141], our proposed method outputs both the probability of an input being spoofed and segmentation maps of the manipulated regions in each frame of the input, as diagrammed in Figure 5.2. Video inputs are treated as a set of frames. We focused on facial images in this work, so the face areas are extracted in the pre-processing phase. In theory, the proposed method can deal with various sizes of input images. However, to maintain simplicity in training, we resize cropped images to 256×256 pixels before feeding them into the autoencoder. The autoencoder outputs the reconstructed version of the input image (which is used only in training), the probability of the input image having been spoofed, and the segmentation map corresponding to this input image. For video inputs, we average the probabilities of all frames before drawing a conclusion on the probability of the input being real or fake.

5.2.2 Y-shaped autoencoder

The partitioning of the latent features (motivated by Cozzolino *et al.*'s work [141]) and the Y-shaped design of the decoder enables the autoencoder to share valuable information between the classification, segmentation, and reconstruction tasks and thereby improve overall performance by reducing loss. There are three types of loss: activation loss $\mathcal{L}_{act}$, segmentation loss $\mathcal{L}_{seg}$, and reconstruction loss $\mathcal{L}_{rec}$.

Given label $y_i \in \{0, 1\}$, activation loss measures the accuracy of partitioning in the latent space on the basis of the activation of the two halves of the encoded features:

$$\mathcal{L}_{act} = \frac{1}{N} \sum_i |a_{i,1} - y_i| + |a_{i,0} - (1 - y_i)|, \tag{5.1}$$

where N is the number of samples, $a_{i,0}$ and $a_{i,1}$ are the activation values and defined as the L_1 norms of the corresponding halves of the latent features, $h_{i,0}$ and $h_{i,1}$ (given K is the number of features of $\{h_{i,0}|h_{i,1}\}$):

$$a_{i,c} = \frac{1}{2K} \|h_{i,c}\|_1, \quad c \in \{0, 1\}. \tag{5.2}$$

This ensures that, given an input x_i of class c, the corresponding half of the latent features $h_{i,c}$ is activated ($a_{i,c} > 0$). The other half, $h_{i,1-c}$, remains quiesced ($a_{i,1-c} = 0$). To force the two decoders, D_{seg} and D_{rec}, to learn the right decoding schemes, we set the off-class part to zero before feeding it to the decoders ($a_{i,1-c} := 0$).

We utilize cross-entropy loss as the segmentation loss to measure the agreement between the segmentation mask ($s_i = \mathcal{D}_{seg}(\{h_{i,0}|h_{i,1}\})$) and the ground-truth mask (m_i) corresponding to input x_i:

$$\mathcal{L}_{seg} = \frac{1}{N} \sum_i \|m_i \log(s_i) + (1 - m_i) \log(1 - s_i)\|_1. \tag{5.3}$$

The reconstruction loss uses the L_2 distance to measure the difference between the reconstructed image ($\hat{x} = \mathcal{D}_{rec}(\{h_{i,0}|h_{i,1}\})$) and the original one ($x_i$). For N samples, the reconstruction loss is

$$\mathcal{L}_{rec} = \frac{1}{N} \sum_i \|x_i - \hat{x}_i\|_2. \tag{5.4}$$

The total loss is the weighted sum of the three activation losses:

$$\mathcal{L} = \gamma_{act}\mathcal{L}_{act} + \gamma_{seg}\mathcal{L}_{seg} + \gamma_{rec}\mathcal{L}_{rec}. \tag{5.5}$$

Unlike Cozzolino *et al.* [141], we set the three weights equal to each other (equal to 1). This is because the classification task and the segmentation task are equally important, and the reconstruction task plays an important role in the segmentation task. We experimentally compared the effects of the different settings (described below).

5.2.3 Implementation

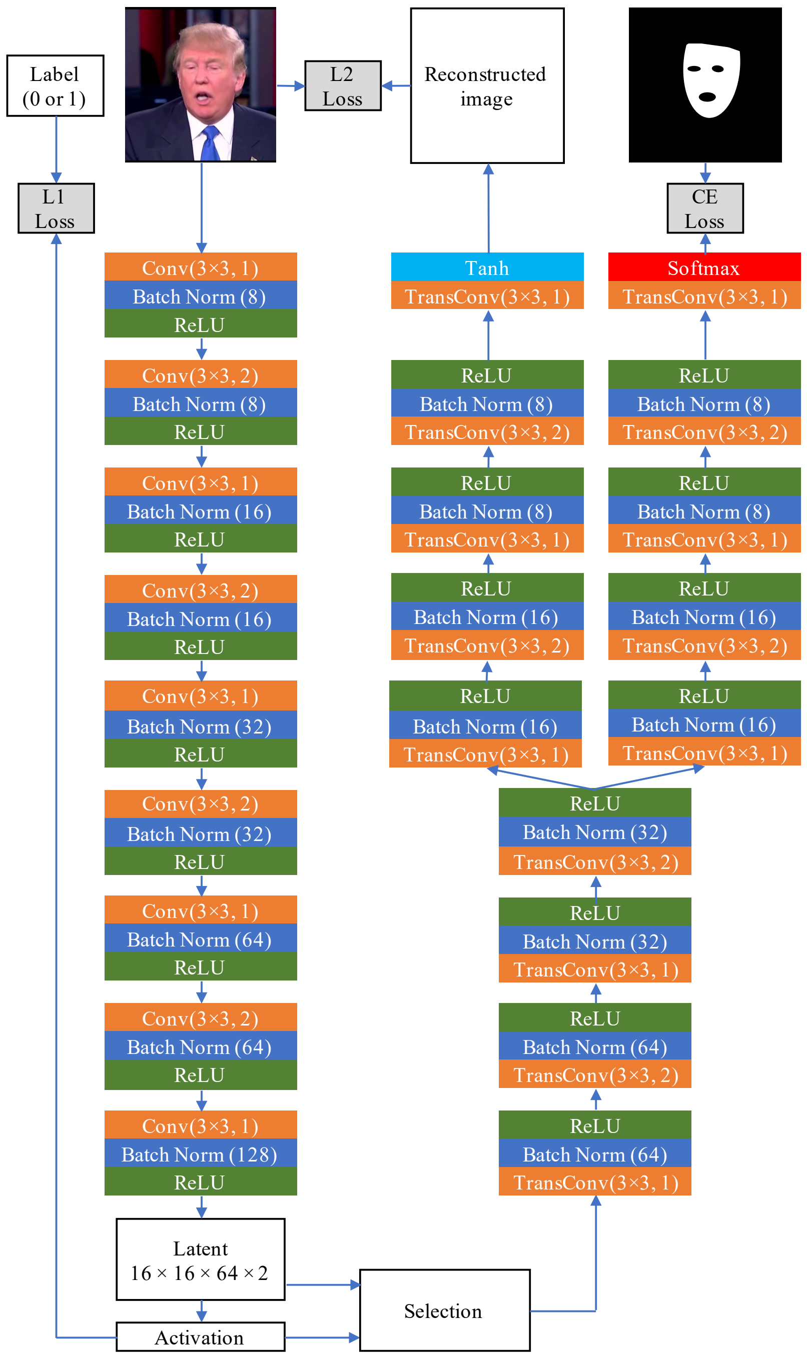

FIGURE 5.3: Proposed autoencoder with Y-shaped decoder for detecting and segmenting manipulated facial images.

The Y-shaped autoencoder was implemented as shown in Figure 5.3. It is a fully connected CNN using 3×3 convolutional windows (for the encoder) and 3×3 deconvolutional windows (for the decoder) with a stride of 1 interspersed with a stride of 2. Following each convolutional layer is a batch normalization layer [142] and a rectified linear unit

(ReLU) [143]. The selection block allows only the true half of the latent features (h_{i,y_i}) to pass by and zeros out the other half ($h_{i,1-y_i}$). Therefore, the decoders ($\mathcal{D}_{seg}, \mathcal{D}_{rec}$) are forced to decode only the true half of the latent features. The dimension of the embedding is 128, which has been shown to be optimal [141]. For the segmentation branch ($\mathcal{D}_{seg}$), a softmax activation function at the end is used to output segmentation maps. For the reconstruction branch ($\mathcal{D}_{rec}$), a hyperbolic tangent function (tanh) is used to shape the output into the range $[-1, 1]$. For simplicity, we directly feed normalized images into the autoencoder without converting them into residual images [141]. Further work will focus on investigating the benefits of using residual images in the classification and segmentation tasks.

Following Cozzolino *et al.*'s work [141], we trained the network using the ADAM optimizer [133] with a learning rate of 0.001, a batch size of 64, betas of 0.9 and 0.999, and epsilon equal to 10^{-8}.

5.3 Experiments

5.3.1 Databases

We evaluated our proposed network using two databases: FaceForensics [99] and FaceForensics++ [23]. The FaceForensics database contains 1004 real videos collected from YouTube and their corresponding manipulated versions, which are divided into two subdatasets:

- Source-to-Target Reenactment dataset containing 1004 fake videos created using the Face2Face method [17]; in each input pair for reenactment, the source video (the attacker) and the target video (the victim) are different.
- Self-Reenactment dataset containing another 1004 fake videos created again using the Face2Face method; in each input pair for reenactment, the source and target videos are the same. Although this dataset is not meaningful from the attacker's perspective, it does present a more challenging benchmark than does the Source-to-Target Reenactment dataset.

TABLE 5.1: Design of training and testing datasets.

Name	**Source dataset**	**Description**	**Manipulation Method**	**Number of Videos**
Training	FaceForensics **Source-to-Target**	Training set used for all tests	Face2Face	704 ×2
Test 1	FaceForensics **Source-to-Target**	Match condition for seen attack	Face2Face	150 ×2
Test 2	FaceForensics **Self-Reenactment**	Mismatch condition for seen attack	Face2Face	150 ×2
Test 3	FaceForensics++ **Deepfakes**	Unseen attack (deep-learning-based)	Deepfakes	140 ×2
Test 4	FaceForensics++ **FaceSwap**	Unseen attack (computer-graphics-based)	FaceSwap	140 ×2

Each dataset was split into 704 videos for training, 150 for validation, and 150 for testing. The database also provided segmentation masks corresponding to manipulated videos. Three levels of compression based on the H.264 codec[1] were used: no compression, light compression (quantization = 23), and strong compression (quantization = 40).

The FaceForensics++ database is an enhanced version of the FaceForensics database and includes the Face2Face dataset plus the FaceSwap[2] dataset (graphics-based manipulation) and the DeepFakes[3] dataset (deep-learning-based manipulation) [23]. It contains 1,000 real videos and 3,000 manipulated videos (1,000 in each dataset). Each dataset was split into 720 videos for training, 140 for validation, and 140 for testing. The same three levels of compression based on the H.264 codec were used with the same quantization values.

For simplicity, we used only videos with light compression (quantization = 23). Images were extracted from videos using Cozzolino *et al.*'s settings [141]: 200 frames of each training video were used for training, and 10 frames of each validation and testing video were used for validation and testing, respectively. There is no detailed description of the rules for frame selection, so we selected the first (200 or 10) frames of each video and cropped the facial areas. For all databases, we applied normalization with mean = $(0.485, 0.456, 0.406)$ and standard deviation = $(0.229, 0.224, 0.225)$; these values have been widely used in the ImageNet Large Scale Visual Recognition Challenge [50]. We did not apply any data augmentation to the trained datasets.

[1]`http://www.h264encoder.com/`
[2]`https://github.com/MarekKowalski/FaceSwap/`
[3]`https://github.com/deepfakes/faceswap/`

Table 5.2: Settings for autoencoder.

No.	Method	Depth	Seg. weight	Rec. weight	Rec. loss	Comments
1	*FT_Res*	Shallower	0.1	0.1	L1	Re-implementation of ForensicsTransfer [141] using residual images as input
2	*FT*	Shallower	0.1	0.1	L1	Re-implementation of ForensicsTransfer [141] using normal images as input
3	*Deeper_FT*	Deeper	0.1	0.1	L1	Proposed deeper version of *FT* (*Proposed_Old* method without segmentation branch)
4	*Proposed_Old*	Deeper	0.1	0.1	L1	Proposed method using ForensicsTransfer settings
5	*No_Recon*	Deeper	1	1	L2	Proposed method without reconstruction branch
6	***Proposed_New***	Deeper	1	1	L2	Complete proposed method with new settings

The training and testing datasets were designed as shown in Table 5.1. For the Training, Test 1, and Test 2 datasets, the Face2Face method [99] was used to create manipulated videos. Images in Test 2 were harder to detect than those in Test 1 since the source and target videos used for reenactment were the same, meaning that the reenacted video frames had better quality. Therefore, we call Test 1 and Test 2 the **match** and **mismatch** conditions for a seen attack. Test 3 used the Deepfake attack method while Test 4 used the FaceSwap attack method, presented in the FaceForensics++ database [23]. These both attack methods were not used to create the training set, therefore they were considered as unseen attacks. For the classification task, we calculated the accuracy and equal error rate (EER) of each method. For the segmentation task, we used pixel-wise accuracy between ground-truth masks and segmentation masks. The *FT_Res*, *FT*, and *Deeper_FT* method could not perform the segmentation task. All the results were at the image level.

5.3.2 Training Y-shaped autoencoder

To evaluate the contributions of each component in the Y-shaped autoencoder, we designed the settings as shown in Table 5.2. The *FT_Res* and *FT* methods are re-implementations of Cozzolino *et al.*'s method with and without using residual images [141]. They can also be understood as the Y-shaped autoencoder without the segmentation branch. The *Deeper_FT* method is a deeper version of *FT*, which has the same depth as the proposed method. The *Proposed_Old* method is the proposed method using weighting settings from Cozzolino *et al.*'s work [141], the *No_Recon* method is the version of the proposed method without the reconstruction branch, and the *Proposed_New* method is the complete proposed method with the Y-shaped autoencoder using equal losses for the three tasks and the mean squared error for reconstruction loss.

Since shallower networks take longer to converge than deeper ones, we trained the shallower ones with 100 epochs and the deeper ones with 50 epochs. For each method, the training stage with the highest accuracy for the classification task and a reasonable segmentation loss (if available) was used to perform all the tests described in this section.

5.3.3 Dealing with seen attacks

The results for the match and mismatch conditions for seen attacks are shown in Table 5.3. The deeper networks (the last four) had substantially better classification performance than the shallower ones (the first two) proposed by Cozzolino *et al.* [141]. Among the four deeper networks, there were no substantial differences in their performances on the classification task. For the segmentation task, the *No_Recon* and *Proposed_New* methods, which used the new weighting settings, had higher accuracy than the *Proposed_Old* method, which used the old weighting settings.

The performances of all methods was slightly degraded when dealing with the mismatch condition for seen attacks. The *FT_Res* and *Proposed_New* methods had the best adaptation ability, as indicated by the lower degradation in their scores. This indicates the importance of using residual images (for the *FT_Res* method) and of using the reconstruction branch (for the Y-shaped autoencoder with new weighting settings: *Proposed_New* method). The reconstruction branch also helped the *Proposed_New* method achieve the highest score on the segmentation task.

TABLE 5.3: Results for Test 1 and Test 2 (image level).

Method	**Test 1**			**Test 2**		
	Classification		**Segmentation**	**Classification**		**Segmentation**
	Acc(%)	**EER (%)**	**Acc(%)**	**Acc(%)**	**EER (%)**	**Acc(%)**
FT_Res	82.30	14.53	_	82.33	15.07	_
FT	88.43	11.60	_	87.33	12.03	_
Deeper_FT	**93.63**	7.20	_	92.70	**7.80**	_
Proposed_Old	92.60	7.40	81.40	91.83	8.53	81.40
No_Recon	93.40	**7.07**	89.21	**92.83**	8.29	89.10
Proposed_New	92.77	8.18	**90.27**	92.50	8.07	**90.20**

Table 5.4: Results for Test 3 and Test 4 without fine-tuning (image level).

Method	**Test 1**			**Test 2**		
	Classification		**Segmentation**	**Classification**		**Segmentation**
	Acc(%)	**EER (%)**	**Acc(%)**	**Acc(%)**	**EER (%)**	**Acc(%)**
FT_Res	**64.75**	**30.71**	_	53.50	43.10	_
FT	62.61	37.43	_	52.29	41.79	_
Deeper_FT	51.21	42.71	_	53.39	37.00	_
Proposed_Old	53.75	42.00	70.18	**56.82**	36.29	84.23
No_Recon	51.96	42.45	**70.43**	54.86	35.86	**84.86**
Proposed_New	52.32	42.24	70.37	54.07	**34.04**	84.67

5.3.4 Dealing with unseen attacks

5.3.4.1 Evaluation using pre-trained model

When encountering unseen attacks, all six methods had substantially lower accuracies and higher EERs, as shown in Table 5.4. In Test 3, the shallower methods had better adaptation ability, especially the *FT_Res* method, which uses residual images. The deeper methods, which had a greater chance of being over-fitted, had nearly random classification results. In Test 4, although all methods suffered from nearly random classification accuracies, their better EERs indicated that the decision thresholds had been moved.

A particularly interesting finding was in the segmentation results. Although degraded, the segmentation accuracies were still high, especially in Test 4, in which FaceSwap copied the facial area from the source faces to the target ones using a computer-graphics method. When dealing with unseen attacks, this segmentation information could thus be an important clue in addition to the classification results for judging the authenticity of the queried images and videos.

5.3.4.2 Fine-tuning using small amount of data

We used the validation set (a small set normally used for selecting hyper-parameters in training that differs from the test set) of the FaceForensics++ - FaceSwap dataset [23] for fine-tuning all the methods. To ensure that the amount of data was small, we used only ten frames for each video. We divided the dataset into two parts: 100 videos of each class for training and 40 of each class for evaluation. We trained them using 50

TABLE 5.5: Results for Test 4 after fine-tuning (image level).

Method	**Classification**		**Segmentation**
	Acc (%)	**EER (%)**	**Acc (%)**
FT_Res	80.04 (↑ 26.54)	17.57 **(↓ 25.53)**	-
FT	70.89 (↑ 18.60)	25.56 (↓ 16.23)	-
Deeper_FT	82.00 (↑ 28.61)	17.33 (↓ 19.67)	-
Proposed_Old	78.57 (↑ 21.75)	20.79 (↓ 15.50)	84.39 (↑ 0.16)
No_Recon	82.93 (↑ 28.07)	16.93 (↓ 18.93)	92.60 (↑ 7.74)
Proposed_New	**83.71 (↑ 29.64)**	**15.07** (↓ 18.97)	**93.01 (↑ 8.34)**

epochs and selected the best models on the basis of their performance on the evaluation set.

The results after fine-tuning for Test 4 are shown in Table 5.5. Their classification and segmentation accuracies increased around 25% and 8%, respectively, which are remarkable compared with the small amount of data used. The one exception was the *Proposed_Old* method – its segmentation accuracy did not improve. The *FT_Res* method had better adaptation than the *FT* one, which supports Cozzolino *et al.*'s claim [141]. The *Proposed_New* method had the highest transferability against unseen attacks as evidenced by the results in Table 5.5.

5.4 Conclusion

The proposed convolutional neural network with a Y-shaped autoencoder demonstrated its effectiveness for both classification and segmentation tasks without using a sliding window, as is commonly used by classifiers. Information sharing among the classification, segmentation, and reconstruction tasks improved the network's overall performance, especially for the mismatch condition for seen attacks. Moreover, the autoencoder can quickly adapt to deal with unseen attacks by using only a few samples for fine-tuning. Future work will mainly focus on investigating the effect of using residual images [141] on the autoencoder's performance, processing high-resolution images without resizing, improving its ability to deal with unseen attacks, and extending it to the audiovisual domain.

Chapter 6

Conclusion

6.1 Summary

Since faces are the ultimate way for people to recognize and communicate with each other, people are in no doubt confident about their natural ability to perceive and recognize faces. That is both good and bad. The good side is that people are way better than machines in avoiding master face attacks. The bad side is that they generally fail to “detect” high-quality computer-generated and manipulated facial images and videos, which makes them susceptible to the consequences of deepfakes. This thesis identifies critical problems in facial biometrics and suggests solutions to deal with them.

On the attacker side, we demonstrate that master face attacks pose a significant security threat against state-of-the-art face recognition systems. By using an improved version of latent variable evolution, we can generate high-quality master faces for use in performing white-box, gray-box, and even some black-box attacks on face recognition systems. The required materials for the attackers are simple and can be easily obtained from the Internet. By evaluating the results for various settings of the LVE algorithm parameters and analyzing the distributions of the face embedding (identity) spaces, we obtained insights into master faces and identified weakness in face recognition systems. We hypothesize that master faces originate from clusters in the identity spaces, which are not well-distributed. This finding provides hints for improving the robustness of those systems.

On the defender side, we address both deepfake detection and deepfake segmentation problems. For deepfake detection, we adopted an approach different from that of our predecessors: instead of increasing the detector network's depth, its width, and/or the number of inner connections, we redesigned its architecture using the novel concept of the capsule network. By doing so, we avoid the problem of computation and memory resource consumption while maintaining high detection performance. The proposed Capsule-Forensics network has a special design in which a feature extractor is used for preventing over-fitting and for leveraging prior knowledge, statistical pooling layers (which are effective for deepfake detection and greatly reduce the number of parameters) are used in the primary capsule, and a dynamic routing algorithm that produces better fusion than the traditional feature aggregation approach is used. We also provide a theoretical explanation of the Capsule-Forensics network and visualization of the activation of its components to improve its explainability.

Besides deepfake detection, we present a multi-task learning network that can locate manipulated regions. By locating them, we improve the trustworthiness of the detection results. If one of the two tasks fails, the other task can compensate for its failure or warn the user of the need for further consideration of the result. This network is a Y-shaped autoencoder with a shared encoder and a Y-shaped decoder, one branch for deepfake segmentation and one branch for input reconstruction. The network is trained in a semi-supervised manner. Thanks to its special design and training strategy, it can adapt well to new deepfake methods even with a small amount of fine-tuning data. Moreover, unlike the approach in which a sliding window with a detector is used for segmentation, segmentation runs only one time per input image in our approach, thus reducing the computation load.

6.2 Remaining Problems

On the attacker side, the performance of deepfake generation methods is not always stable, especially when end-to-end manipulation is applied automatically without human revision. The main problem lies face detection and segmentation, which is not 100%

accurate. This problem becomes worse when the input images or videos are of bad quality (*e.g.*, bad lighting conditions, noisy, blurry, low-resolution). Another problem is the blending of manipulated faces into the original images and videos. Although there are improvements in the blending algorithms, there are still artifacts at the border between the manipulated region and original region in many cases. Moreover, the mismatch between these two regions (*e.g.*, lighting condition, skin color, or noise) can degrade the quality of manipulated images and videos and make them easier to be detected. A third problem is temporal consistency in manipulated videos. It is easier to fool humans using still manipulated images than videos. Although frame-level quality is good, inconsistency between frames is a strong clue for identifying manipulation. In a broader scope, liveness properties (for example, biological signals such as blood circulation) have not been considered for most manipulation techniques. Although these properties may not be easily perceived by humans, machines can generally detect them.

On the defender side, the identity spaces of face recognition systems are not uniformly distributed, resulting in the existence of dense areas. This makes them vulnerable to master face attacks. Our hypothesis is that this problem may be due to an imbalance in the training data and/or the lack of a good objective function. Although master face attacks can be mitigated by using presentation attack detectors and deepfake detectors before the recognition phase, false acceptance still occurs. The ultimate goal is to improve the distributions of the identity spaces, which will help deal with master face attacks and improve recognition performance.

Detection results in the field have significantly improved for seen manipulations. Unfortunately, generalization is still a big problem, especially when deep learning (machine learning) is the core of almost state-of-the-art deepfake detectors. Although recent work, including ours, has achieved better results in unseen deepfake methods, performance is still low. On the other hand, it is challenging to work with low-quality images and videos. These two problems make deepfake detectors unreliable for real-world applications. Moreover, the lack effective explanations of how they operate makes them less convincing, especially in critical applications like journalism and law enforcement.

Regarding the evaluation of deepfake detection results, many deepfake datasets provide annotations of the face regions. In most work on deepfake detection, the assumption has been that this pre-processing is available. Therefore, only the cropped faces were used for benchmarking. The reported scores are only for deepfake detection. In reality, face detection does not always work correctly, especially in extreme conditions, which greatly affects overall performance.

Regarding public datasets, most of them are still small and monotonic compared with those in other areas like image classification and speech (*e.g.*, ImageNet [50] and VoxCeleb [137]). One of the largest datasets, the Deepfake Detection Challenge dataset [31], has only 128,154 videos representing less than 20 types of manipulation methods. Moreover, most datasets contain only one or two persons in an image or video, except for recently released datasets [5, 105], and they are easily perceived and detected (excepting those that are small or obscured). There is thus a gap between deepfake detection in the laboratory and in the field. The only dataset containing manipulated or synthesized speech is the Deepfake Detection Challenge dataset, and the audio manipulation methods represented are hard to consider as "fake." Many manipulation techniques have been introduced, but only a few have been used to create deepfake datasets.

6.3 Final Remarks

For deepfake detection, there is an upper bound on the generalizability of detectors, especially when deepfake techniques are rapidly advancing. Online learning is one promising way to deal with this limitation. Unfortunately, current datasets are not optimal for conducting online learning research. Therefore, focusing on creating better datasets suitable for online learning and then developing better online learning techniques are top priorities.

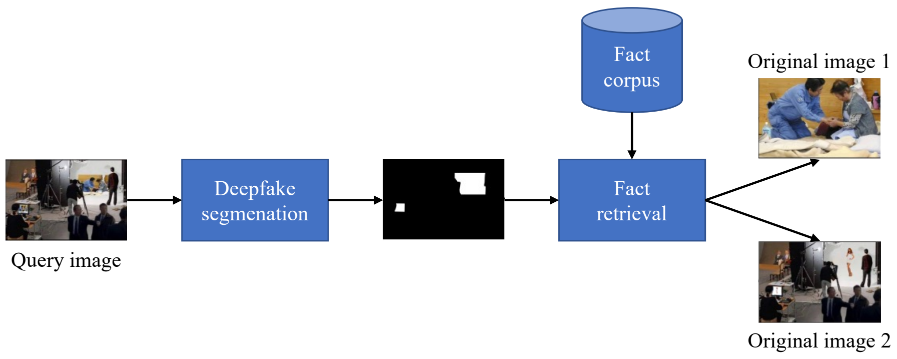

FIGURE 6.1: Overview of fact retrieval.

The explanations of deepfake detector models and their results are still limited to visualizing network activation and providing segmentation maps, heat maps, and/or the blending boundaries. This is not enough for making critical decisions, for example, in law enforcement. There is a need for providing the original version of a manipulated image or video. Therefore, fact retrieval for deepfakes is an important topic. That is, given a fact corpus and a query sample, the task is to check whether the query sample is fake and then, if it is, localize the manipulated region(s) and find the corresponding original version(s), as shown in Fig. 6.1. Similarity search can be used to find the original version of the manipulated region(s) and the remaining region. However, finding and verifying facts to create the fact corpus is a challenging undertaking.

Appendix A

Enhancing Computer-Generated Images to Avoid Being Detected

In this appendix, we introduce a method for use on the attacker side that enhances computer-generated (CG) images so that they cannot be detected by CG image detectors. It works independently of the generating method and thus can be applied to any CG image.

A.1 Introduction

A presentation attack is commonly used to bypass authentication systems using biometrics information (face, fingerprint, iris, and/or voice). Integration of a spoofing detector into the system before the authentication phase is one approach to preventing such attacks. A good candidate for this is liveness detection, which generally uses a challenge-response protocol in which the user is asked to perform an action such as blinking, smiling, or moving the lips. However, recent work has shown that it is possible to avoid liveness detection by, for example, using real-time face capture and reenactment [17]. It has thus become necessary to develop and implement natural–CG image/video discriminators.

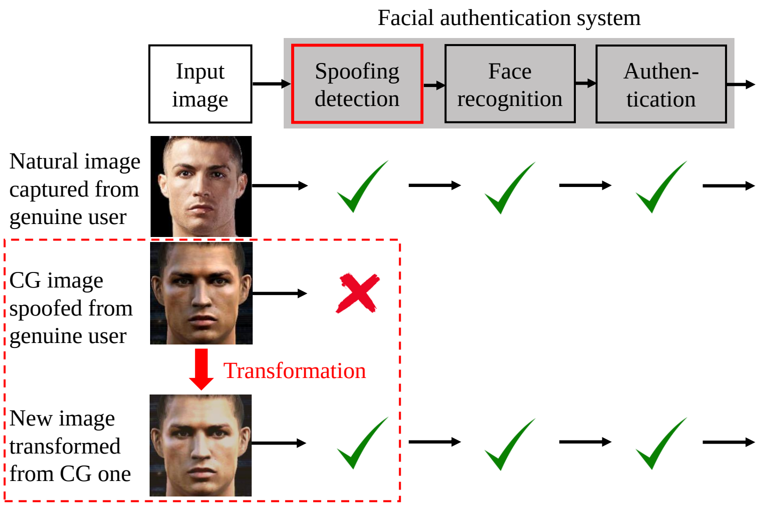
Facial authentication system
Input image
Spoofing detection
Face recognition
Authen-tication
Natural image captured from genuine user
CG image spoofed from genuine user
Transformation
New image transformed from CG one

FIGURE A.1: Scenario of proposed presentation attack method.

Forensic research on discriminating between CG and natural images has focused on both images in general and facial images. As an example of the former, Wu *et al.* extracted statistical features from histograms of differential images [144]. Although this approach was proposed several years ago, our evaluation demonstrated that it has fast feature extraction and good performance. Therefore, we used it as the basis of the discriminator used to train our CNN. In 2017, Peng *et al.* [145] reported a detector based on multi-fractal and regression analysis. As an example of focusing on facial images, our previous work [146] focused on facial smoothness as represented by edges and local entropy of the skin areas.

In the research reported here, we developed a method for avoiding detection by spoofing detectors like those natural–CG image discriminators mentioned above. It works by transforming CG images input to facial authentication systems to make them appear more natural. Our work is motivated by the idea of "adversarial machine learning" of Huang *et al.* for attacking machine learning based systems [147]. Although we did not target a specific system, we used a spoofing detection algorithm as the basis of the discriminator. The attack scenario is illustrated in Fig. A.1. Given two sets of data (a set of natural images and a set of CG ones which are not necessarily corresponding person-to-person or pose-to-pose), a system using the proposed method implemented as a CNN transforms the CG input images in an attempt to make them indistinguishable from the natural counterparts.

Unlike the original generative adversarial network (GAN) [35], the discriminator used to train our CNN was pre-trained, kept fixed during training, and could not perform back propagation. Moreover, the generated images retained important information from the input images, such as the person’ s identity, the facial expression, or the lips’ shapes, which are hard to control when using the original GAN. When designing our CNN we focused on minimizing the number of parameters so that it can work without consuming a large amount of GPU memory. We also considered the “training with small dataset problem,” which is often faced by attackers when collecting data.

Our approach is different from those of other computer graphic engines, which mainly focus on the rendering phase. It also differs from the style transfer problem, which mainly focuses on applying the “look” or textures of one image (the style) to another one (the content). In our case, the style is not just a picture or an artistic style, so it is hard to define.

Our contributions here are threefold:

- Presentation of a CNN comprising two autoencoders and a transformer net that increases the difficulty of detecting computer-generated facial images.
- Suggestion of a method for dealing with back propagation when training using an external black-box discriminator that does not have gradient information.
- Raising of an alarm about the robustness of facial authentication systems, which are being implemented in many mobile devices and have become a tempting target for attacker.

A.2 Model Architecture

A.2.1 Overview

It is very hard to explicitly point out what makes natural images look “natural” and CG ones look “unnatural.” We humans have been “trained” by our encounters with many natural things and scenes since we were born, so we can intuitively distinguish which images were produced by computer. Some forensic researchers have tried describing these intuitive feelings in terms of specific properties [145, 146]. Others, such as Wu *et al.* [144], have tried using statistical methods to distinguish natural images from CG

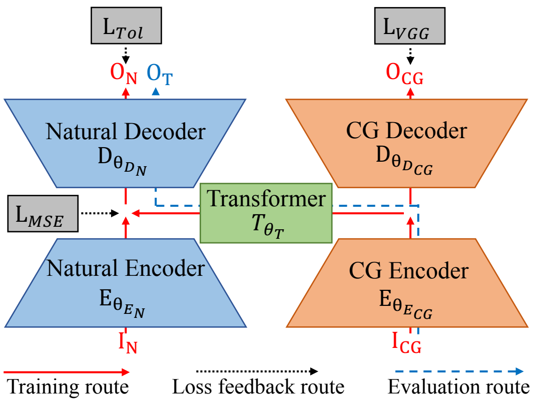
L_{Tol}
L_{VGG}
O_N O_T
O_{CG}
Natural Decoder $D_{\theta_{D_N}}$
CG Decoder $D_{\theta_{D_{CG}}}$
L_{MSE}
Transformer T_{θ_T}
Natural Encoder $E_{\theta_{E_N}}$
CG Encoder $E_{\theta_{E_{CG}}}$
I_N
I_{CG}
Training route
Loss feedback route
Evaluation route

FIGURE A.2: H-Net architecture.

images. However, these approaches are problematic, require lengthy experiments and do not sufficiently describe the essence of the two kinds of images.

To tackle this feature-finding problem, we use two autoencoders [148] with the same design but different weights ($\{E_{\theta_{E_N}}, D_{\theta_{D_N}}\}$ and $\{E_{\theta_{E_{CG}}}, D_{\theta_{D_{CG}}}\}$) to automatically extract latent features from natural image I_N and CG image I_{CG}. They are trained to learn representations of the inputs and to regenerate them with the same properties. To transform the latent feature space of CG images into those of natural ones, we use a transformer net T_{θ_T} with five bottleneck residual blocks [42]. This block design reduces the number of parameters for the network. Old information given by the skip connection combines with new one created by the layers in a residual block are suitable for transformation. Since our network is shaped like an "H," as illustrated in Fig. A.2, we call it "H-Net." In the evaluation phase, the CG encoder $E_{\theta_{E_{CG}}}$, the transformer T_{θ_T}, and the natural decoder $D_{\theta_{E_N}}$ transform CG input I_{CG} into a natural-looking image O_T that is similar to natural image I_N from the perspective of a natural–CG image discriminator.

The encoder has four convolutional layers with 3×3 kernels and a stride of 1. Batch normalization [142] is used to deal with high learning rates and the less careful initialization problem. After batch normalization layers are exponential linear unit (ELU) layers, which give better performance than traditional rectified linear units (ReLUs), leaky ReLUs (LReLUs), and parametrized ReLUs (PReLUs) [149]. To reduce the spatial size of the representation, we use 2×2 average pooling layers, which have recently come to be used instead of max pooling ones. In the decoder, the convolutional layers are replaced with their counterparts: transposed convolutional ones and two sub-pixel

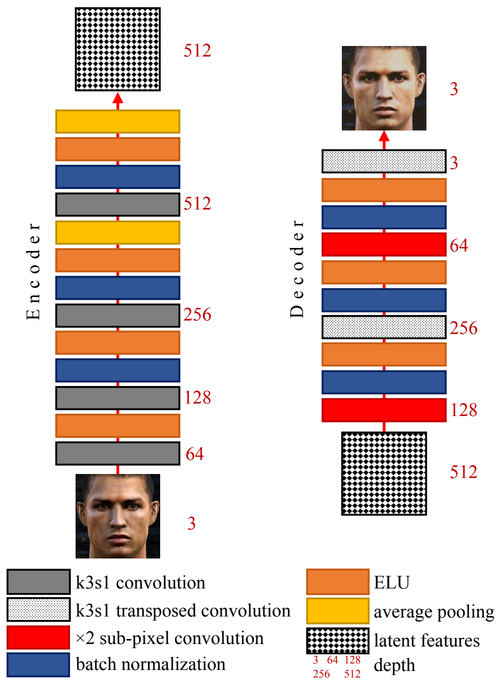

FIGURE A.3: Design of encoder net (left) and decoder net (right).

convolutional ones, which have been reported to give better upscaling results than previous approaches[150]. The detailed designs of the encoder and decoder are shown in Fig A.3.

A.2.2 Loss functions

Loss functions are used to optimize the model in the training phase. Pixel-wise loss functions such as mean squared error (MSE) ones are widely used but have limited capabilities for images when measuring perceptual quality [151]. To overcome this limitation, Dosovitskiy and Brox [152] used Euclidean-based loss in feature space in combination with adversarial loss. Another approach is to use a perceptual loss function based on the Euclidean distance between feature maps extracted from a VGG network proposed by the Visual Geometry Group at the University of Oxford [51]. Ledig *et al.* combined both VGG loss and adversarial loss in their super-resolution generative adversarial network [153].

We use four loss functions:

1. MSE loss L_{MSE} is only used to calculate the loss between two feature maps, including high-level presentations encoded by the encoders and others extracted from the VGG-19 network.

2. VGG loss L_{VGG}, or perceptual loss, is the MSE loss of features maps (of I_1 and I_2) extracted using the VGG-19 network: $L_{VGG} = L_{MSE}(VGG(I_1), VGG(I_2))$. VGG loss represents the perceptual quality of generated images.

3. Adversarial loss L_{Adv} is the binary cross entropy loss between two labels: one is from the discriminator and the other is the destination label. Details of the discriminator are discussed in section A.2.3.

4. Total loss L_{Tol} is the combination of L_{VGG} and L_{Adv}, formulated as equation A.1. It represents the quality of images generated by natural decoder $D_{\theta_{E_N}}$. In our experiments, we set the hyper-parameter α to 5×10^{-3}. Gradient back propagation is discussed in section A.2.3.

$$L_{Tol} = (1 - \alpha)L_{VGG} + \alpha L_{Adv} \tag{A.1}$$

A.2.3 Discriminator

Unlike the original GAN proposed by Goodfellow *et al.* [35], we use a pre-trained discriminator for training our H-Net. It can be any black-box spoofing detector. Since it is unlikely for attackers in the wild to have full access to the spoofing detectors in authentication systems, it is more realistic to assume that the spoofing detector used as the discriminator used for training is a "black-box" rather than a differentiable white-box discriminator. In our experiments, the use of a "traditional" discriminator in the GAN, such as the one used by Ledig *et al.* [153], results in very poor performance. This is because performing only convolution is not enough for extracting the features needed for distinguishing natural images from CG images. Hence, we borrow the detector of Wu *et al.*, which focuses on statistical features from histograms of differential images [144]. We use a neural net instead of Fisher's linear discriminant analysis (FLDA) as Wu *et al.* did because we can easily run the discriminator in parallel with H-Net in the training phase without using an additional framework.

We need to compute the gradient of total loss L_{Tol} in equation A.1 with respect to the parameters in the generator. However, since the discriminator is a black-box, the adversarial loss L_{Adv} is not differentiable. One possible approach is to approximate it on the basis of the gradient of the VGG loss L_{VGG} only. However, the adversarial loss makes no contribution to the gradient. Therefore, we approximate the total gradient as follows:

$$grad_{L_{Tol}} = \frac{(1-\alpha)L_{VGG} + \alpha L_{Adv}}{L_{VGG}} grad_{L_{VGG}} \tag{A.2}$$

A.2.4 Training and evaluation

H-Net is trained using two sets of images: natural images and CG ones. The two sets do not have to be pairwise correlated, which means that they may contains images of different people with different poses or facial expressions. This loose condition comes from reality; *i.e.*, it is very hard for an attacker to find a large number of natural–CG image pairs with similar perceptual content.

In training, four optimization processes are performed sequentially:

Step 1: Training the natural autoencoder $\{E_{\theta_{E_N}}, D_{\theta_{D_N}}\}$ with natural image I_N by minimizing equation A.3.

$$\underset{\theta_{E_N},\theta_{D_N}}{\operatorname{argmin}} L_{Tol}[D_{\theta_{D_N}}(E_{\theta_{E_N}}(I_N)), I_N] \tag{A.3}$$

The output image $O_N = D_{\theta_{D_N}}(E_{\theta_{E_N}}(I_N))$ must satisfy two conditions: (1) have the same perceptual content as input I_N and (2) be classified as a natural image by the discriminator.

Step 2: Training the CG autoencoder $\{E_{\theta_{E_{CG}}}, D_{\theta_{D_{CG}}}\}$ with CG image I_{CG} by minimizing equation A.4.

$$\underset{\theta_{E_{CG}},\theta_{D_{CG}}}{\operatorname{argmin}} L_{VGG}[D_{\theta_{D_{CG}}}(E_{\theta_{E_{CG}}}(I_{CG})), I_{CG}] \tag{A.4}$$

In this step, the discriminator loss is not necessary. The only requirement is that output image O_{CG} be perceptually similar to I_{CG}.

Step 3: Training the transformer net T_{θ_T} to convert latent features encoded by CG encoder $E_{\theta_{E_{CG}}}$ into ones that have the same distribution of features as those extracted by the natural encoder $E_{\theta_{E_N}}$. As mentioned above, due to the lack of pairwise correlated I_N and I_{CG} pairs, it is very hard to minimize the loss between $E_{\theta_{E_N}}(I_N)$ and $T_{\theta_T}(E_{\theta_{E_{CG}}}(I_{CG}))$. An acceptable solution is feeding I_N into both networks and minimizing the features they encode, formulated in equation A.5.

$$\underset{\theta_T}{\operatorname{argmin}}\, L_{MSE}[E_{\theta_{E_N}}(I_N), T_{\theta_T}(E_{\theta_{E_{CG}}}(I_N))] \tag{A.5}$$

Step 4: Training the CG transformation path, which includes CG encoder $E_{\theta_{E_{CG}}}$, transformer T_{θ_T}, and natural decoder $D_{\theta_{D_N}}$. As in the first step, total loss must be used to ensure that transformed output $O_T = D_{\theta_{D_N}}(T_{\theta_T}(E_{\theta_{E_{CG}}}(I_{CG})))$ is not classified as a CG image. The lack of pairwise correlated I_N and I_{CG} pairs in Step 3 appears here as well, so we need to replace I_N with I_{CG} in equation A.6.

$$\underset{\theta_{E_{CG}},\theta_T,\theta_{D_N}}{\operatorname{argmin}} \quad L_{Tol}[D_{\theta_{D_N}}(T_{\theta_T}(E_{\theta_{E_{CG}}}(I_{CG}))), I_{CG}] \tag{A.6}$$

Doing this could also bring back some "balance" from the effect of I_N on CG encoder $E_{\theta_{E_{CG}}}$ in step 3.

When we compute the transformed output from the CG input in the evaluation phase, we use a step similar to step 4 in the training phase:

$$O_T = D_{\theta_{D_N}}(T_{\theta_T}(E_{\theta_{E_{CG}}}(I_{CG}))) \tag{A.7}$$

We do not use the *tanh* function to scale the output images because of the resulting low contrast. To avoid extreme values with resulting abnormal color spots, the pixel intensities are restricted to the range $[-1.8, 1.8]$ before being denormalized to $[0, 255]$.

A.3 Evaluation

In facial image forensic research, there was no standardized dataset used for mutual comparison. Each research group tended to have its own datasets. Therefore, we had to combine pieces from five different sources to create three datasets used for evaluation

Table A.1: Three datasets used in evaluation.

No.	Components	Size	Description
1	Dang-Nguyen *et al.* [154]	CG: 240 Nat: 240	40 very realistic CG images collected from Web plus 200 good quality CG images extracted from PES 2012 soccer game 240 natural images retrieved from Internet
2	Basel (CG) [155] Caltech99 (Nat) [156]	CG: 270 Nat: 270	3D face scans and rendered images from Basel Face Model Natural images from Caltech Faces 1999 dataset
3	MIT (CG - Grayscale) [157] MS-Celeb-1M (Nat) [46]	CG: 3236 Nat: 3236	CG images extracted from MIT CBCL dataset Natural images selected from MS-Celeb-1M cropped version

(see Table A.1). All images were resized to 256 × 256 pixel resolution. We tested H-Net on two scenarios: (1) the attacker knows the dataset used for training the spoofing detector, and (2) the attacker has no knowledge of the training dataset. Three spoofing detectors were used: Wu *et al.*'s [144], Peng *et al.*'s [145], and ours [146]. Note that only Wu *et al.*'s spoofing detector was used as a discriminator for training H-Net; the other spoofing detectors were unseen by the attacker. Moreover, in Wu *et al.*'s one, we used FLDA for classification as in the authors' report. We compared both the accuracies and detection rates of the three spoofing detectors. Let n_{TP} and n_{TN} be respectively the number of images correctly classified as CG or natural, n_{FN} be the number of CG or transformed images misclassified as natural ones, and N be the total number of images. Accuracy is defined as sum of n_{TP} and n_{TN} over N: $\frac{n_{TP}+n_{TN}}{N}$. The detection rate represents the ability of the spoofing detector to detect positive items: $\frac{n_{TP}}{n_{TP}+n_{FN}}$.

Table A.2: Scenario 1 - Accuracies (%) and detection rates (%) of three spoofing detectors on three datasets before and after performing transformation on CG parts.

Spoofing detectors	Dataset 1				Dataset 2				Dataset 3			
	Accuracy		Detection rate		Accuracy		Detection rate		Accuracy		Detection rate	
	Before	After↓	Before	After↓	Before	After↓	Before	After↓	Before	After↓	Before	After↓
Wu *et al.* [144]	92.71	**48.33**	93.33	**0.00**	83.89	**55.37**	93.33	**36.30**	64.65	**56.83**	35.51	**19.90**
Peng *et al.* [145]	90.63	**52.71**	92.08	**16.25**	59.26	**19.26**	97.41	**17.41**	50.20	**13.58**	100.00	**26.79**
Our previous detector [146]	83.54	**64.79**	87.08	**48.33**	**17.78**	32.78	**0.00**	30.00	**32.31**	63.23	**0.49**	62.11

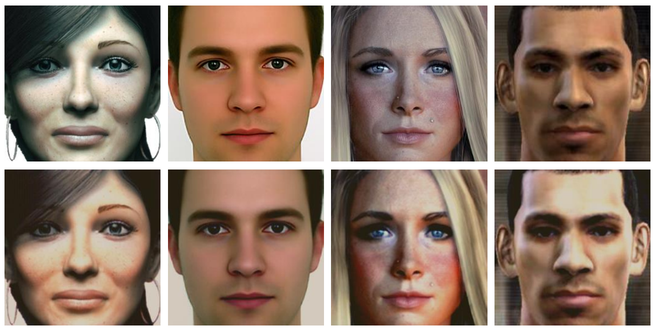

Figure A.4: Original images (top) and transformed ones (bottom) from dataset 1. Color and brightness of images in second row were normalized by H-Net as learned from training data. First two images demonstrate good transformation in perception. Contrast of third image was improved. In last image, skin color was whitened a bit undesirably.

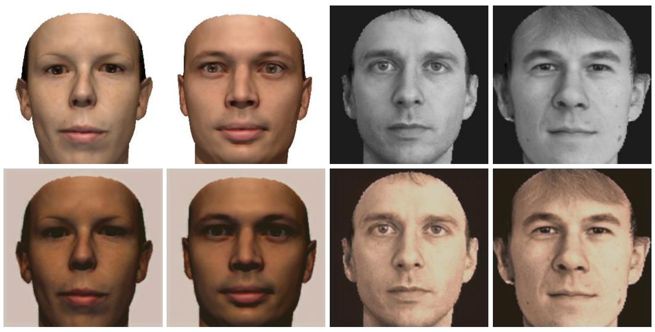

Figure A.5: Original images (top) and transformed ones (bottom) from dataset 2 (left) and dataset 3 (right). Brightness of first two image was unexpectedly reduced due to bright background. Although grayscale images were given skin-like color, they can easily be converted back into grayscale.
©Copyright University of Basel. ©Copyright 2003-2005 Massachusetts Institute of Technology. All Right Reserved.

A.3.1 Scenario 1: Attacker knows training dataset of spoofing detector

We trained both spoofing detectors and H-Net on dataset 1. We then evaluated them on all datasets to see if the images transformed by H-Net could avoid detection by these pre-trained spoofing detectors. Comparisons of sample images before and after transformation are shown in Figs. A.4 and A.5. As shown in Table A.2, the detection rates significantly decreased when the CG images were transformed, especially for Wu *et al.*' s and Peng *et al.*'s detectors.

Although our previous detector had the lowest performance on dataset 1, its detection rate after transformation was the highest (nearly 50%). On datasets 2 and 3, its performance before transformation was very poor; it was better after transformation. Our analysis shows that this detector was over-fitted for dataset 1 which had high-quality CG images. It had a tendency to classify fine-texture images as CG ones. On datasets 2 and 3, which did not have fine-texture CG images, it classified almost of the images as natural ones. Because of the spoofing detector's preset threshold, some transformed images had good enough texture to be classified as CG ones, which increased the detection rate. Therefore, if an attacker tries to avoid detection by this spoofing detector, he or she may be successful the first time without the help of H-Net.

A.3.2 Scenario 2: Attacker does not know training dataset of spoofing detector

Unlike in the first scenario, we trained the H-Net and the spoofing detectors on difference datasets (switched between dataset 1 and 2), and evaluated them on dataset 3.

A.3.2.1 Scenario 2.1: H-Net was trained on dataset 1, spoofing detectors were trained on dataset 2

The evaluation results on dataset 3 are shown in Table A.3. The transformed images again significantly reduced the detection rates of all spoofing detectors, especially those of Wu *et al.* and Peng *et al.*, which were nearly 0%. Our previous detector had a detection rate of around 50%, down from nearly 100%, meaning that the attacker had a 50–50 chance of avoiding detection by this spoofing detector. In this case, this spoofing detector learned that low-texture images had a high probability of being CG ones, which was opposite to its knowledge in scenario 1. Therefore, after transformation on dataset 3, the textures of the CG images were improved so that the images would likely be classified as natural ones. This also clarifies the contrasting changes in the detection rate in the first scenario on dataset 1 vs. datasets 2 and 3.

A.3.2.2 Scenario 2.2: H-Net was trained on dataset 2, spoofing detectors were trained on dataset 1

The evaluation results on dataset 3 are shown in Table A.4. Before transformation, Peng *et al.*'s detector seemed to classify all input as CG. After transformation, its decision was changed that all transformed CG images were classified as natural ones, therefore both the accuracy and the detection rate were around 0. Our previous detector had the same behavior as in the first scenario as expected. The performance of Wu *et al.*'s detector was increased after transformation, which was different from the two above scenarios. A possible explanation for this phenomenon is that the training dataset used for H-Net is small and monotonous. As the result, H-Net did not have enough knowledge about other kinds of CG images.

Table A.3: Scenario 2.1 - Evaluation results on dataset 3.

Spoofing detectors	**Accuracy**		**Detection rate**	
	Before	**After**	**Before**	**After**
Wu *et al.* [144]	56.38	**6.46**	100.00	**0.19**
Peng *et al.* [145]	92.32	**42.57**	100.00	**0.49**
Our previous detector [146]	96.72	**71.54**	99.20	**48.89**

Table A.4: Scenario 2.2 - Evaluation results on dataset 3.

Spoofing detectors	**Accuracy**		**Detection rate**	
	Before	**After**	**Before**	**After**
Wu *et al.* [144]	**64.65**	82.73	**35.51**	71.66
Peng *et al.* [145]	50.20	**0.20**	100.00	**0.00**
Our previous detector [146]	**32.31**	41.27	**0.49**	18.17

A.4 Conclusion and Future Work

The performances of both H-Net and detectors are depended on the quality of training datasets. However, in most cases, over 50% of the CG images transformed using our H-Net avoided detection by three state-of-the-art spoofing detectors. Since the facial features were preserved, facial recognition was unaffected. This means that the network can be trained using a black-box discriminator that cannot perform back propagation. However, H-Net has some limitations, especially when up to 50% transformed images are still separable from natural ones. Future work will mainly focus on this limitation as well as finding better datasets. We will also evaluate the local substitute method to perform black-box attacks [158]. Other issues to be addressed are solving the black skin-color problem, dealing with larger images, and reducing network size to enable it to work smoothly with video frames in real time.

Appendix
B

Distinguishing between Computer Graphics Images and Photographic Images

In this appendix, we introduce a modular classifier for distinguishing computer graphics images (images generated using a computer graphics model) and photographic images. The use of a probabilistic patch aggregation strategy reduces computational expense. In addition, the use of a sliding window enables segmentation as well as classification to be performed.

B.1 Introduction

Statistical properties obtained from transformed images (*e.g.*, from wavelet transform or differential operators) have been widely used to distinguish computer graphics images (CGIs) from photographic images (PIs) [144, 159–163] and were recently demonstrated to be the best features for discrimination by Rahmouni *et al.* [132]. They also demonstrated that applying automatic feature extraction using a convolutional neural network (CNN) can substantially improve classification compared with using handcrafted features.

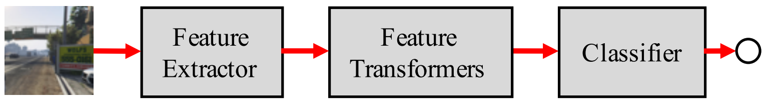

Figure B.1: Overview of modular CNN discriminator.

In addition, the pre-trained VGG networks [51] (VGG-16 and VGG-19) have been widely used in areas outside their originally intended scope as image classification networks, such as for perceptual loss in the style transfer problem and for the super-resolution problem [153, 164]. Furthermore, these VGG networks were trained using a large-scale dataset [50], which maximizes the generalization ability of a CNN.

In the research reported here, we leveraged the generalization ability of the VGG-19 network, combined with statistical properties applicable to CNNs, to build a modular CGI–PI classifier. To deal with high-resolution images while minimizing computational cost, we use a probabilistic patch aggregation strategy that reduces V-RAM usage and shortens classification time.

B.2 Network Architecture

B.2.1 Overview

Our modular CNN for discriminating between CGIs and PIs includes three modules, as illustrated in Figure B.1. Unlike recent work [132], we do not train the whole network end-to-end. The biggest problem with CNNs is the need to use a large-scale and diverse-content training dataset in order to achieve the best generalization. The dataset used by Rahmouni *et al.* [132] is relatively large but is less diverse in content than the ILSVRC15 dataset [50]. Unfortunately, the ILSVRC15 dataset was designed for visual recognition, not digital image forensics research. However, CNNs have the ability to transfer learning, so the knowledge gained from solving one problem can be used to solve a different but related problem. Therefore, we used one of the winners of the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) – the pre-trained VGG-19 network, as the feature extractor module. It is important to note that we did not fine tune the feature extractor in the training process.

Although recent work [132, 144] has shown that statistical properties obtained from transformed images are the best features for CGI–PI discrimination, the features extracted from the pre-trained VGG-19 network were designed for visual recognition. Therefore, we constructed feature transformer modules to transform the output extracted by the feature extractor into statistical features. The number of convolutional layers in the transformers must be limited to prevent them from extracting semantic information, but there must be a sufficient number of such layers to be able to extract good statistical information.

The final module is a classifier. For this module, we selected the machine learning algorithm among state-of-the-art ones that has the best classification results.

B.2.2 Feature extractor

Johnson *et al.* [164] suggested that the results obtained from some activation layers of the pre-trained VGG-16 network can be used to calculate the feature reconstruction loss and the style reconstruction loss, which are used for both the style transfer problem and the image super-resolution problem. Ledig *et al.* [153] argued that, in the case of feature reconstruction loss, using output from a deeper activation layer of the pre-trained VGG-19 network results in better perceptual quality than that with Johnson *et al.*'s approach. Therefore, there is no standard guideline for the utilization of the VGG network family. In the case of digital forensics, we hypothesized that features in lower layers have more discriminating power than ones from higher levels, which mostly contain semantic information. Moreover, instead of using the output of the rectified linear units (ReLUs) [143], for which negative values are omitted, we extracted output immediately after the convolutional layers.

To verify this hypothesis, we performed an experiment using the patches dataset proposed by Rahmouni *et al.* [132] and the pre-trained VGG-19 network. We extracted the outputs after five convolutional layers located immediately before the max-pooling layers as shown in Figure B.2. For the five settings given in Table B.1, the combination of layers 1, 2, and 3 gave the highest classification accuracy. These results indicate that using only one layer does not produce the highest accuracy. However, if semantic layers were included, the classification performance would be affected by this irrelevant

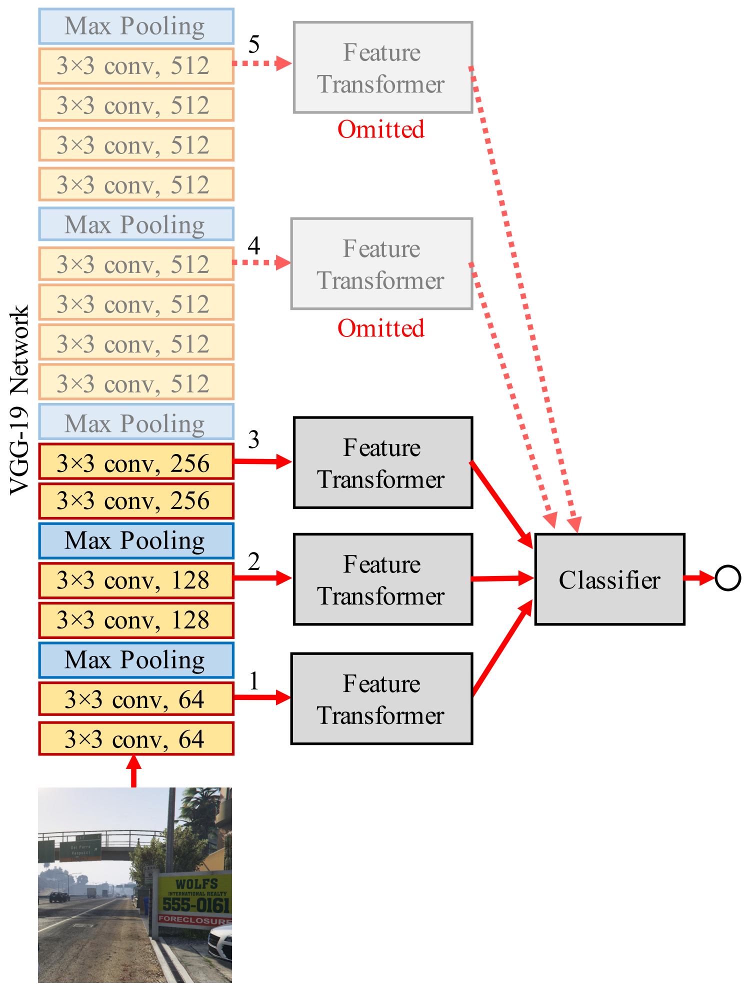

FIGURE B.2: Detailed design of feature extractor and its connections with feature transformers and classifier.

TABLE B.1: Accuracies for training using patches dataset for five settings.

Setting	Accuracy (%)
1	95.40
1 + 2	97.60
1 + 2 + 3	**97.70**
1 + 2 + 3 + 4	96.50
1 + 2 + 3 + 4 + 5	96.10

information. Therefore, we chose outputs from layers 1, 2, and 3 in Figure B.2 (conv1_2, conv2_2, and conv3_4, respectively) as features to be extracted by the feature extractor.

B.2.3 Feature transformers

The role of the feature transformers is to transform features encoded by the pre-trained VGG-19 network into statistical properties that can be used to distinguish CGIs from

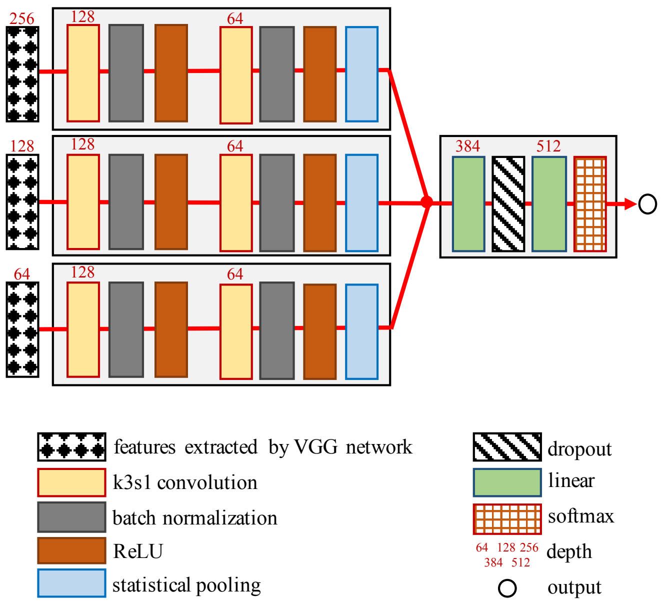

FIGURE B.3: Detailed settings of feature transformers and classifier.

PIs. Because there are three feature transformer modules, it is necessary to minimize their depths. Moreover, a deep feature transformer may produce unnecessary semantic information, which could negatively affect the network. However, a shallow network has a limited ability to transform the features. Therefore, we used two convolutional layers with 3×3 kernels and a stride of 1. We integrated batch normalization layers [142] into the transformers to regularize their training processes. Following the batch normalization layers are the ReLU activation layers. We attached a statistical pooling layer at the end of the modules to extract the statistical properties. The three feature transformers share the same architecture, as illustrated in Figure B.3.

We built the statistical pooling layer following Rahmouni *et al.*'s approach [132]. However, we assumed that finding the maximum and minimum of each filter was not necessary and that these actions would consume computational power, especially when performing back propagation in the training phase. Therefore, we calculate only the mean and variance of each filter, which are important in statistics and also are differentiable.

- Mean:

$$\mu_k = \frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} I_{kij}$$

.

- Variance:
$$\sigma_k^2 = \frac{1}{H \times W - 1} \sum_{i=1}^{H} \sum_{j=1}^{W} (I_{kij} - \mu_k)^2$$
.

The k represents the layer index, H and W are respectively the height and width of the filter, and I is a two-dimensional filter array.

B.2.4 Classifier

Feed-forward multilayer networks, or multilayer perceptrons (MLPs), [165] are widely used to build classifiers in CNNs because of their differentiable property. However, there are other strong classification algorithms that have been widely used such as Fisher's linear discriminant analysis (LDA) algorithm [166] and the support vector machine (SVM) algorithm [167]. Therefore, we first use an MLP to build the classifier to train the feature transformers (as well as to train the classifier itself). After the training, the feature transformers are kept fixed, and the classifier is trained using the LDA and SVM classification algorithms. The learning curves of these algorithm are plotted in Figure B.4. The proposed network converged very quickly in the few first epochs. The MLP algorithm had high accuracy but was less stable than the LDA and SVM algorithms. Since the LDA algorithm usually has higher accuracy than the SVM one, we evaluated only MLP and LDA classifiers, as described in section B.4.

In more detail, two properties are extracted by each statistical pooling filter: the mean μ_i and the variance σ_i. Each pooling layer has 64 filters. Since there are three feature extractor modules, the classifier receives a 384-dimension vector. For the MLP algorithm, we used two hidden layers and one dropout layer [168] in between (with a dropout rate of one-third to avoid over-fitting). A classifier using the MLP algorithm is illustrated in Figure B.3. For the LDA and SVM classifiers, we used the LinearDiscriminantAnalysis and SVC module of the scikit-learn library. [1]

To choose the best weights for the feature transformers and the classifier, we begin from epoch 20 and use the one with the highest score in the validation set. Although the proposed network converged very quickly, it is better to use a longer training time to optimize its weights before harvesting.

[1]http://scikit-learn.org/

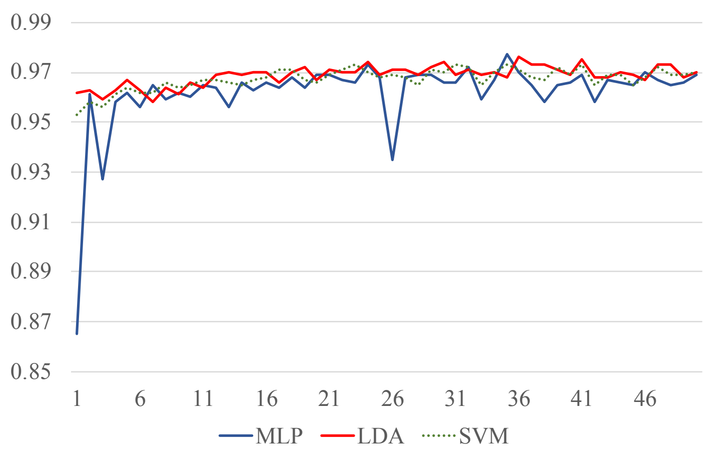

FIGURE B.4: Learning curves of MLP, LDA, and SVM classifiers on Patch-100-Full validation set described in section B.4.1
.

B.3 Patch Aggregation

Using a CNN with large-scale input requires a large amount of GPU memory. One possible solution is to split the input into patches, perform classification, and aggregate the results [132]. Although this approach can also detect local CGI inlay in large PI images (or vice-versa), it has high computational cost, especially when dealing with very large images. For instance, an image 4900×3200 pixels in size would require 1568 patches if the patch size was 100×100 pixels. This would result in 1568 classification calculations.

To reduce the number of calculations, we devised an approach using a probability sampling method that randomly selects a portion of the patches, performs classification using the selected patches, calculates the average of the predicted probabilities, and uses it as the final decision. Two patch selection strategies are illustrated in Figure B.5. For some fixed number of patches (*e.g.*, 10, 25, or 50), we could integrate them into one batch and feed that batch into the network instead of feeding each patch separately into the network, thereby shortening the computation time.

Let

- y_{pred} be the predicted label of input image I, which is either 0 (PI) or 1 (CGI).
- W be the set of patches w_i extracted from the full-size image I, $|W| = N$ (patches).

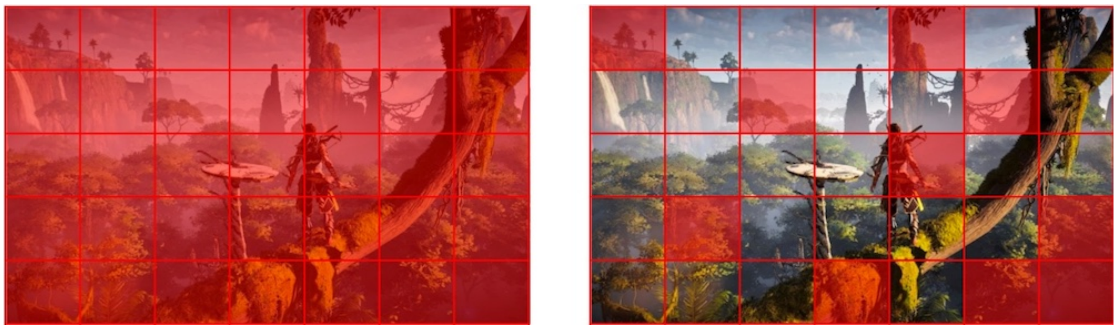

FIGURE B.5: Patch selection strategies: Selecting all patches (left) vs. random sampling (right).

- $p(w_i) = \mathcal{D}(w_i)$ be the probability of patch w_i being classified by the proposed network $\mathcal{D}$ as CGI.

The probability of I being classified as CGI is calculated using

$$p(I) = \frac{1}{N}\sum_{i=1}^{N} p(w_i). \tag{B.1}$$

Hence, the predicted label of I is

$$y_{pred} = \begin{cases} 1, & \text{if } p(I) > 0.5 \\ 0, & \text{otherwise.} \end{cases} \tag{B.2}$$

B.4 Evaluation

B.4.1 Datasets

For the image datasets, we began with the one recently constructed by Rahmouni *et al.* [132]. Its CGI part contains 1800 high-resolution (around 1920×1080 pixels) screenshots in JPEG format from five photo-realistic video games. The PI part includes 1800 very high-resolution JPEG images (around 4900×3200) directly converted from RAW format. Both parts cover many kinds of indoor and outdoor environments. Sample images from this dataset are shown in Figure B.6.

We made one major change to this dataset. We contend that the reduced-size images created by cropping high-resolution images to 650×650 are not appropriate for our

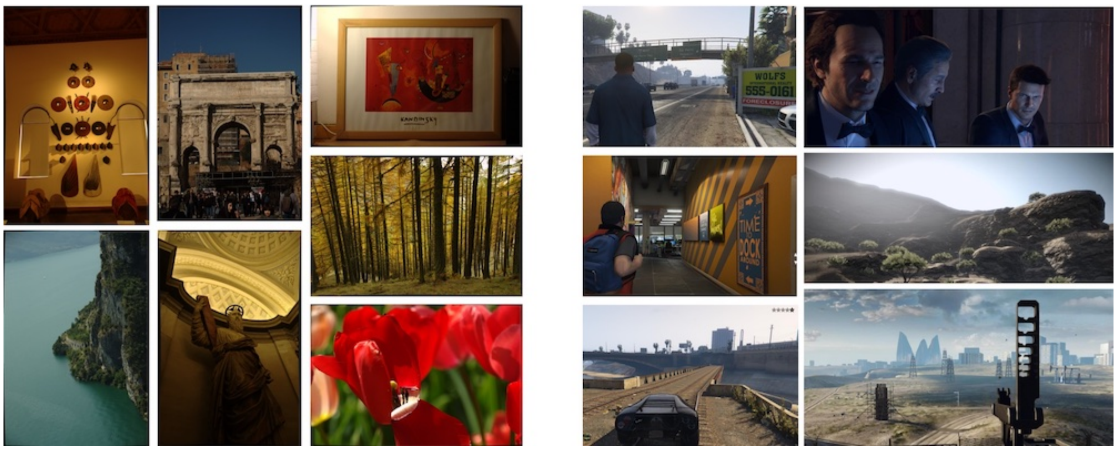

FIGURE B.6: Sample images from dataset constructed by Rahmouni *et al.* [132]. Images on the left are PIs and those on the right are CGIs.

TABLE B.2: Datasets used for evaluation.

Name	No. for training	No. for valid.	No. for testing	Image size
Full-Size	2,520	360	720	High-resolution
Patch-100-Full	40,000	1,000	2,000	100 × 100
Patch-256-Full	40,000	1,000	2,000	256 × 256
Reduced-Size	2,520	360	720	360p
Patch-100-Reduced	40,000	1,000	2,000	100 × 100

purposes because their quality is still good. In reality, many images and videos have low quality, and a malicious person could additionally apply transformation to the CGIs, for example, scaling them to produce lower quality, to disguise the attack. Therefore, instead of cropping, we resized each high-resolution image to $360p$ resolution using a bilinear interpolation algorithm. This increased the diversity in quality of images used for evaluation.

In addition to using a patch size of 100×100, we also used a patch size of 256×256 for the high-resolution images to reduce the number of patches. This larger patch size could be used with large-memory GPUs. Moreover, a larger patch size should contain more valuable information, and with the size is the power of 2, we could reduce the effect of JPEG artifacts. In addition, we also extracted 100×100 patches from the reduced-size images. The datasets derived from the original one are summarized in Table B.2.

We trained each discriminator on the training sets of the patch datasets. The valid. sets were used to validate the training process. After training, the discriminators were tested on the testing sets of both patch datasets and their corresponding Full-Size or Reduced-Size ones. Moreover, as described in section B.4.3, we also tested the discriminators

Table B.3: Accuracy for several patch aggregation strategies on Full-Size dataset. The random sampling strategy was evaluated three times.

Classifier		**MLP**				**LDA**			
Patch size	**No. of patches**	**1**	**2**	**3**	**Avg.**	**1**	**2**	**3**	**Avg.**
100 × 100	10	99.31	99.72	99.86	99.63	99.86	99.31	99.72	99.63
	50	99.86	99.86	99.86	**99.86**	99.86	99.86	99.86	**99.86**
	100	99.86	99.86	99.86	**99.86**	99.86	99.86	99.86	**99.86**
	All				**99.86**				**99.86**
256 × 256	5	99.72	99.44	99.72	99.63	99.44	99.03	99.58	99.35
	10	100.00	99.72	100.00	**99.91**	99.86	99.58	99.72	99.72
	25	99.86	99.86	99.86	**99.86**	99.72	99.72	99.72	99.72
	All				**99.86**				99.72

which were trained using the Patch-100-Full dataset on the Reduced-Size dataset to check whether this training strategy is capable of generalization.

B.4.2 Testing on high-resolution images

For testing on high-resolution images, we trained our proposed method and Rahmouni *et al.*'s one [132] on the Patch-100-Full and the Patch-256-Full datasets. We then evaluated them on both the corresponding patch dataset and the Full-Size one. The proposed method was also tested for several patch aggregation strategies, as presented in Table B.3. For the 100 × 100 patch size, it was sufficient to sample only 50 patches to obtain performance equivalent to that of evaluating all patches on the Full-Size dataset. When the sampling process avoided some confused areas in the images, sampling only 10 256 × 256 patches outperformed sampling 25 patches or evaluating all patches, achieving an accuracy of 100%. Otherwise, the accuracy was slightly lower (*e.g.*, 99.72%).

Our proposed method substantially outperformed Rahmouni *et al.*'s method [132] on both the Patch-100-Full and Patch-256-Full datasets. It also had the highest results on the Full-Size dataset, reaching 100%. A comparison of accuracy between Rahmouni *et al.*'s method [132] and the proposed method is shown in Table B.4. Comparing the original 100 × 100 patch size with the 256 × 256 one shows that increasing the patch size improves the accuracy of Rahmouni *et al.*'s method. Moreover, use of the MLP classifier rather than the LDA one in the proposed method resulted in higher accuracy for both the Reduced- and Full-Size datasets. The ROC curves for the Patch-100-Full and Full-Size dataset discriminators are plotted in Figure B.7.

TABLE B.4: Comparison of accuracy between Rahmouni *et al.*'s method [132] and proposed method.

Method	Patch-100-Full	Patch-256-Full	Full-Size
Rahmouni *et al.* - 100 [132]	86.10	×	96.94
Rahmouni *et al.* - 256 [132]	×	93.95	98.75
Proposed method - MLP - 100	**96.55**	×	**99.86**
Proposed method - LDA - 100	96.40	×	99.86
Proposed method - MLP - 256	×	**98.70**	**99.72 - 100.00**
Proposed method - LDA - 256	×	98.70	99.58 - 99.86

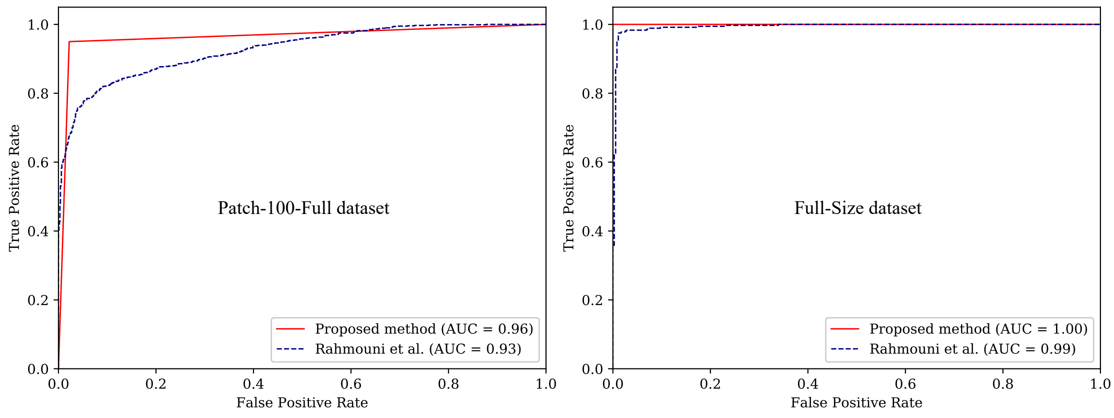

FIGURE B.7: ROC curves of discriminators tested on Patch-100-Full dataset (left) and Full-Size dataset (right). Proposed method used MLP classifier.

B.4.3 Dealing with low-resolution images

In reality, many videos on social networks such as YouTube, Facebook, and Vimeo have 360p quality. Attackers can take advantage of this to produce low-resolution videos (and images) that are more difficult to detect. The results shown in Table B.5 highlight this problem for discriminators trained on the Patch-100-Full dataset. Their performance substantially decreased to the random-selection level. To solve this problem, we mixed the Patch-100-Full and the Patch-100-Reduced datasets to form the Patch-100-Mixed dataset. We then retrained the discriminators on this new dataset and evaluated them on the Patch-100-Reduced & Reduced-Size datasets and Patch-100-Full & Full-Size datasets.

The results in Table B.5 show that both discriminators had better performance on the Patch-100-Reduced and the Reduced-Size datasets. However, their performance on the Patch-100-Full and the Full-Size datasets was slightly lower than with the previous scheme for high-resolution datasets. The difference in performance between the proposed method and Rahmouni *et al.*'s was also substantially greater. The results also

TABLE B.5: Accuracy of classifiers trained on Patch-100-Full dataset (Old) and on Patch-100-Mixed dataset (New). For simplicity, proposed method used all-patch strategy.

Method	Patch-100-Reduced	Reduced-Size	Patch-100-Full	Full-Size
Rahmouni *et al.* (old) [132]	51.50	50.97	86.10	96.94
Proposed method - MLP (old)	52.55	51.81	**96.55**	**99.86**
Proposed method - LDA (old)	52.35	51.53	96.40	99.86
Rahmouni *et al.* (new) [132]	60.45	79.72	81.20	95.00
Proposed method - MLP (new)	88.60	96.67	93.40	97.64
Proposed method - LDA (new)	**89.95**	**97.92**	94.80	98.89

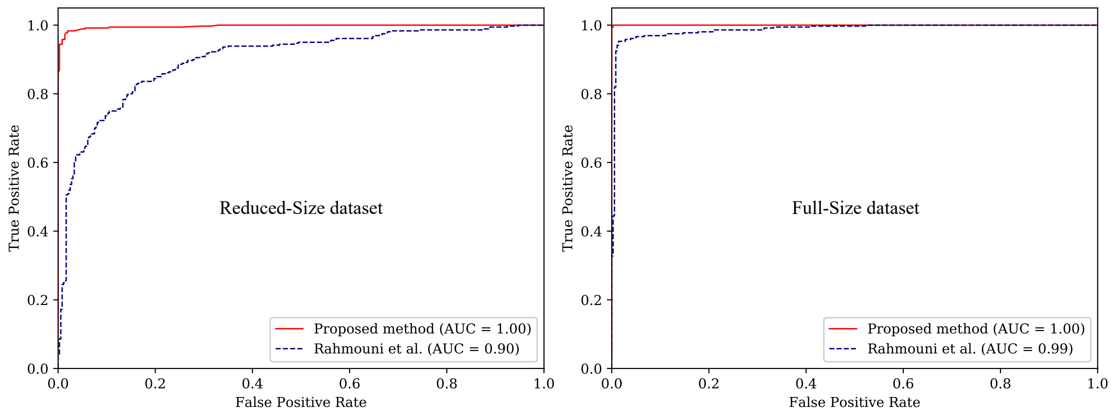

FIGURE B.8: ROC curves of retrained discriminators tested on Reduced-Size dataset (left) and Full-Size dataset (right). Proposed method used LDA classifier.

demonstrated the advantage of choosing among state-of-the-art classifiers to find the best one; *i.e.*, use of the LDA classifier resulted in higher accuracy when the Patch-100-Mixed dataset was used. The ROC curves for the Reduced-Size and Full-Size dataset discriminators after being retrained are shown in Figure B.8.

B.4.4 Detecting image splicing

In an experiment, we used the discriminators to detect image splicing. Along with the normal way of dividing the test input into 100×100 patches, we also used an overlapping patch strategy. The probability of splicing for each area is the average of the probabilities of all patches to which the area belongs. Although this strategy has a higher calculation cost, it produces smoother output than the non-overlapping one. Example images are shown in Figure B.9; the input sizes were 1800×1200 and 1200×800 pixels. Our proposed method (both overlapped and non-overlapped patches) outperformed Rahmouni *et al.*'s one [132]. Although our method did not flawlessly separate all the splices and had a few minor false positives, it could detect their relative positions. Rahmouni *et al.*'s one,

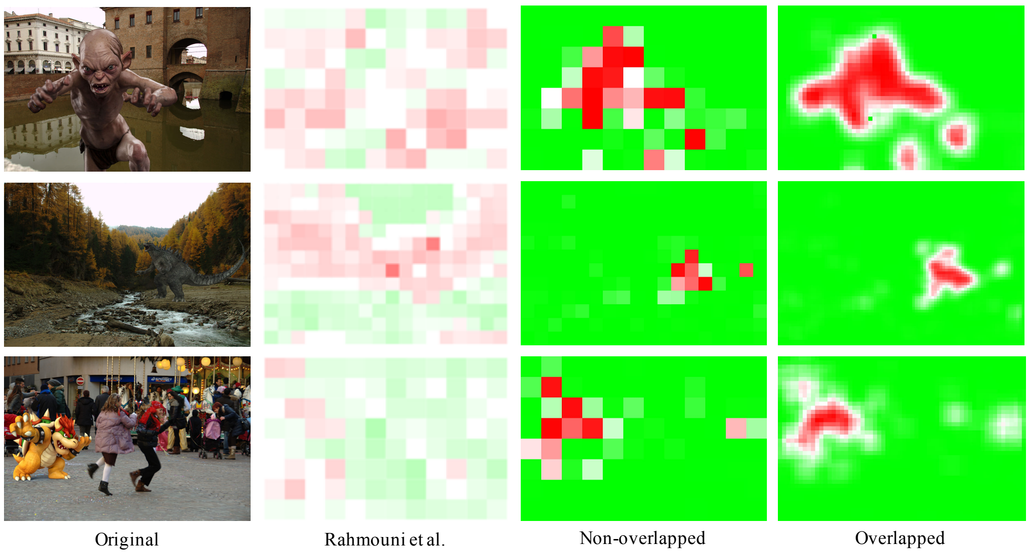

FIGURE B.9: Three examples of splice detection. Patches detected as CGI are in red; those detected as PI are in blue. The color intensities illustrate the probabilities of classes.

on the other hand, failed to detect the splice in the first image and was confused in the second image.

B.5 Summary and Future Work

The proposed modular CGI–PI discriminator uses the VGG-19 network as the feature extractor, statistical convolutional neural networks as the feature transformers, and the machine learning algorithm among state-of-the-art ones that has the best classification results as a discriminator. It outperformed a state-of-the-art CGI–PI discriminator. The proposed random sampling strategy used for patch aggregation was demonstrated to be effective for large images. Testing showed that using only high-resolution images for training is not sufficient to counter real-world attacks.

Our top priority now is to use ensemble adversarial training [37] to counter adversarial machine learning attacks [147]. This kind of attack is becoming more common and is very effective against machine-learning-based discriminators. A promising candidate to replace patch aggregation for dealing with high-resolution images is the attention-based approach [169]. We also plan to adapt the proposed discriminator to enable it to work with videos, not simply extracting data frame-by-frame and performing classification to reduce computational time.

Appendix C

Detecting and Correcting Adversarial Images

Adversarial attacks can target any deep neural network by adding certain perturbations (such as noises and/or patterns) to the network inputs and thereby alter their outputs. DNN-based deepfake detectors are not an exception [170]. Therefore, it is vital to detect and correct adversarial examples to ensure the robustness of deep neural networks. In this appendix, we introduce novel methods for detecting and correcting adversarial images crafted using classical adversarial attacks.

C.1 Introduction

Despite the success of deep learning in both academia and industry, deep neural networks (DNNs) are vulnerable to adversarial attacks [36], and this has attracted much attention and effort. Besides traditional logical attacks in which adversarial noise is added to image or audio files, attackers can now create physical adversarial examples [171–175]. When autonomous systems have become mainstream, physical adversarial attacks may threaten their safety and reliability. Besides white-box attacks, in which attackers have full knowledge of the inner configuration of the target models, attackers will also be able to perform black-box attacks, which are more likely since attackers need acquire only the models' outputs [176]. Moreover, attackers are able to create universal adversarial perturbations that are applicable to multiple inputs [177] and to create adversarial examples that can be used to attack multiple DNNs [178]. Such adversarial examples could be used to directly attack DNN-based systems or to poison the training data of DNNs to corrupt their models (a "data poisoning attack") [179].

Approaches to counter adversarial attacks can be classified into four groups: adversarial example detection, adversarial training, input pre-processing, and randomization or private keying. Some approaches were designed for multiple domains while others were simply designed for a single domain like the image one. For adversarial example detection, a statistical-based approach is commonly used [180, 181]. Another approach is to build a detector that takes raw images [182] or features from intermediate layers of the targeted DNN [183, 184] as input. Ma *et al.* used local intrinsic dimensionality to characterize the adversarial subspace [185]. Xu *et al.* presented a feature squeezing method in which the differences in the DNN's outputs between the normal and squeezed images are used for detection [186]. Liang *et al.* subsequently proposed an adaptive noise reduction method [187]. For adversarial training, there are several approaches including distillation [188], obfuscating gradients [189], optimizing the saddle point formulation [190], and applying the reverse cross-entropy loss function [191]. For input pre-processing, Guo *et al.* trained DNNs on transformed images (to which cropping, total variance minimization, and/or quilting operations had been applied) so as to mitigate adversarial noise [192]. Prakash *et al.* proposed using a DNN-based adaptive JPEG encoder to pre-process the input [193]. For randomization or private keying, Taran *et al.* proposed a key-based diversified aggregation mechanism to defend against gray- and black-box adversarial attacks [194]. Several adversarial databases have been independently created for evaluation, but guidelines for creating them have not been reported in detail [180, 186, 192].

In this appendix, we present methods for detecting and correcting adversarial examples in the digital image domain. By limiting our focus to the digital image domain, we can use more domain knowledge and can use many potential efficient and cheap image processing operations for our framework. We target classical adversarial attacks that do not consider optimizing the robustness of adversarial perturbations against transformations. These attacks generally require less computation than robust ones, so attackers

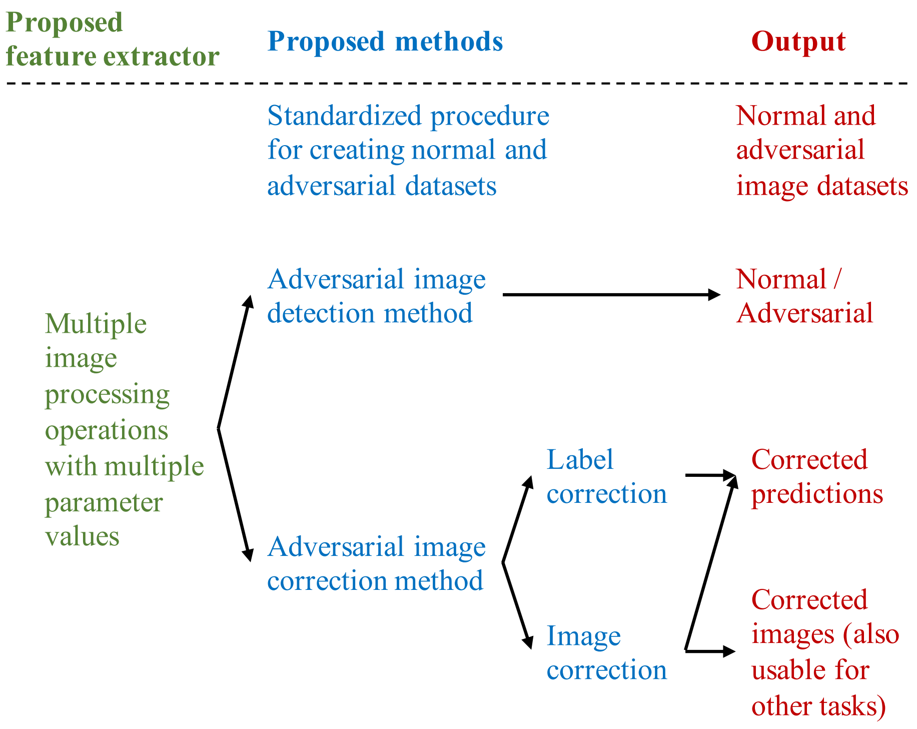
Proposed feature extractor
Proposed methods
Output
Standardized procedure for creating normal and adversarial datasets
Normal and adversarial image datasets
Multiple image processing operations with multiple parameter values
Adversarial image detection method
Normal / Adversarial
Adversarial image correction method
Label correction
Image correction
Corrected predictions
Corrected images (also usable for other tasks)

FIGURE C.1: Visualization of our contributions.

can easily create a massive number of adversarial examples in a short period of time. One practical use is for data poisoning. We target both black-box and white-box attack scenarios.

Our approach is to use multiple image processing operations with multiple parameters for detection and correction. These parameters include the quality factor (QF) of JPEG compression, the scaling factor, the size of the Gaussian blur kernel, and the rotation angle. We hypothesized and then verified that the classification labels of the adversarial images change when the values of the parameters change while the classification labels of normal images remain mostly unchanged. Our approach to detection and correction does not require any modification or re-training of the target DNNs. In summary, our contribution is three-fold (visualized in Fig. C.1):

- We introduce a standardized procedure for creating a normal dataset and an adversarial dataset. The latter includes several state-of-the-art targeted and non-targeted adversarial attacks [176]. Targeted attacks are attacks that try to change the output label of a DNN to a predefined label while non-targeted attacks are attacks that aim to make the output label of a DNN different from the original one. Both datasets are derived from the ImageNet validation set [50]. The adversarial dataset created using this procedure has protocols for evaluating seen and unseen attacks to measure the models' generalizability. This standardized procedure is expected to promote fair comparisons and reproducible research in adversarial machine learning (ML).

- We present a method that uses operation-oriented characteristics for detecting adversarial images. It is unrealistic to use an exhaustive search to find the optimal combination of multiple image processing operations with multiple parameter values. We thus observed the changes in DNN output with an increase in the strength of image processing operations, which work as a kind of noise removal filter. Using simulation, we quantitatively evaluated and identified the best choice of operation-oriented characteristics. This method is more advanced than the feature squeezing method [186], which uses only three image processing operations with manually set parameter values.
- We present a method that uses for the first time two levels of adversarial image correction: label correction (to restore the correct labels of the adversarial images for the current task) and image correction (to mitigate the adversarial noise in the adversarial images so that the targeted DNN can correctly work on them and making these images usable in other tasks). In label correction, only the outputs of the DNNs are of interest while in image correction, the quality and usability of the corrected images as well as the labels are of interest. Our proposed method is heuristic and is based on using multiple image processing operations with multiple parameter values. Unlike Guo *et al.*'s method [192], our method does not require re-training of the DNNs and is applicable to their pre-trained models. Unlike Prakash *et al.*'s method [193], our method uses multiple standardized image processing operations rather than a customized DNN-based JPEG encoder, which is not robust to adversarial attacks.

The rest of this appendix is organized as follows: In section C.2, we introduce our standardized procedure for creating the normal and adversarial datasets used for the experiments described in this appendix. Next, in section C.3, we discuss the effects of using multiple image processing operations with multiple parameter values on both normal and adversarial images. The observed distinctive effects are used as the backbone for the proposed adversarial image detection method described in section C.4 and the image correction method described in section C.5. We summarize the key points and discuss future work in section C.6.

C.2 Dataset Creation

C.2.1 Overview

Although several normal and adversarial datasets have been independently created, detailed guidelines for their creation are lacking [180, 186, 192]. We thus created our own datasets as shown in Fig. C.2 for use in our experiments. We applied several policies when designing them:

- They must be large enough to be used to train both handcrafted and convolutional neural network (CNN) based methods.
- The adversarial dataset must contain images for various types of adversarial attacks including both targeted and non-targeted ones [176]. This is crucial to building protocols for seen and unseen attacks.
- Since the datasets were derived from the ImageNet database, we followed its protocol by utilizing the top-5 accuracy as the metric for all experiments. A correct classification is defined to be when one of the top-5 predicted labels is the true label.
- All images in the normal dataset must be correctly classified by the targeted CNNs (100% accuracy). All images in the adversarial dataset must be misclassified by the targeted CNNs (0% accuracy). To make the attacks realistic, the images in the adversarial dataset are saved as JPEG files since some adversarial noise is mitigated when saving.
- Since some labels fall into the same category (for example, different breeds of dogs), the adversarial attacks should produce adversarial images with a label belonging to a different category than the original one.

In the next sections, we describe in detail the dataset construction and the adversarial attacks used. Finally, we describe the analysis we performed on the newly created datasets to determine their quality and to gain some insight about them.

C.2.2 Dataset construction

We used the images from the validation set of the ImageNet database [50] (which has ground-truth labels) to generate the adversarial images used in our experiments. There were 1,000 labels in total. For the object-recognition CNNs, we used the VGG-16 and VGG-19 networks [51] and the ResNet-18 and ResNet-50 networks [42] pre-trained on the ImageNet database, implemented using the PyTorch framework [195]. We used the Pillow library[1] for image processing and the FoolBox library (version 1.8.0) [196] for adversarial image generation. Twelve commonly used methods (Table C.1) were used to perform targeted and non-targeted adversarial attacks.

As shown in Fig. C.2, we used each CNN to classify five million images from the ImageNet 2012 validation set. Classification was considered correct when the ground-truth label was one of the predicted top-5 labels. We then randomly selected 1,000 images per CNN from the correctly classified images; therefore, there were 4,000 normal images in total. We limited the number because the time required to craft adversarial images from them is quite long. These 4,000 images were added to the normal dataset. We then performed the 12 adversarial attacks listed in Table C.1 on the 4,000 images as described in section C.2.3. The Pillow library was used to save them as JPEG files with a QF

[1]https://pillow.readthedocs.io/en/stable

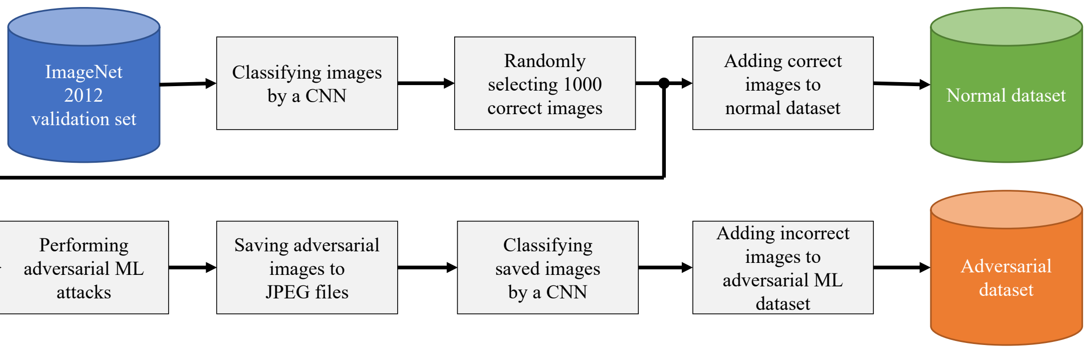
ImageNet 2012 validation set
Classifying images by a CNN
Randomly selecting 1000 correct images
Adding correct images to normal dataset
Normal dataset
Performing adversarial ML attacks
Saving adversarial images to JPEG files
Classifying saved images by a CNN
Adding incorrect images to adversarial ML dataset
Adversarial dataset

FIGURE C.2: Overview of dataset creation procedure using a CNN. The same procedure was used with VGG-16, VGG-19, ResNet-18, and ResNet-50 networks to together create a full normal dataset (for normal images) and a full adversarial dataset (for adversarial images).

of 100 to preserve most of the adversarial noise. We then loaded the saved images and used the same CNN to classify them again to ensure that they were still misclassified (the predicted top-5 results did not contain the ground-truth label). We obtained 22,326 misclassified adversarial images in total and added them to the adversarial dataset.

We distributed the obtained normal and adversarial images into training (train), development (dev), and evaluation (eval) sets as detailed in Table C.2. The train set was used for training, the dev set was used to select the model, and the eval set was used to test the detectors. We ensured that the normal images and their adversarial versions in the train, dev, and eval sets did not overlap so that the detectors would not remember the training images. There were three attack settings: targeted, non-targeted, and combination. The targeted and non-targeted attack settings were used to test the generalization of the adversarial image detectors while the combination setting was used mainly for hyper-parameter selection for the proposed detection method. This combination attack setting was also used for all experiments on adversarial image correction.

C.2.3 Crafting adversarial images

To craft the adversarial images used for the targeted attacks, we used the limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) method proposed by Szegedy *et al.* [36], the basic iterative method (BIM) using L-infinity, the projected gradient descent

Table C.1: Number of successful adversarial images created from 1,000 input images using FoolBox library [196] on VGG-16, VGG-19, ResNet-18, and ResNet-50 networks.

Method	**VGG-16**	**VGG-19**	**ResNet-18**	**ResNet-50**
Targeted attacks:				
L-BFGS [36]	618	623	348	229
BIM [171]	693	711	531	431
PGD [171]	651	678	484	348
L1-iter [196]	661	591	680	513
L2-iter [196]	820	746	722	583
Non-targeted attacks:				
Gradient [196]	654	548	590	517
FGSM [197]	509	450	451	379
Deep Fool [198]	707	671	658	604
Newton [199]	571	505	558	425
ADef [200]	1	4	1	1
JSMA [201]	102	86	114	71
Carlini-Wagner [202]	427	361	401	299

TABLE C.2: Details of normal and adversarial image datasets, which were divided into training, development, and evaluation sets for three attack settings.

Attack Setting/Datasets	Normal	Adversarial	Total	Ratio
Targeted attacks:				
Train	2,800	8,392	11,192	1:3.00
Dev	600	1,623	2,223	1:2.71
Eval	600	1,646	2,246	1:2.74
Non-targeted attacks:				
Train	2,800	7,718	10,518	1:2.75
Dev	600	1,469	2,069	1:2.45
Eval	600	1,478	2,078	1:2.46
Combination attacks:				
Train	2,800	16,110	18,910	1:5.75
Dev	600	3,092	3,692	1:5.15
Eval	600	3,124	3,724	1:5.21

(PGD) method described by Kurakin *et al.* [171], and the L1- and L2- versions of BIM (L1-iter and L2-iter) implemented in FoolBox [196]. For the indexes of the image labels of the ImageNet database, nearby labels often fall into the same group. For example, 239 is "Bernese mountain dog," 245 is "French bulldog," and 250 is "Siberian husky." All of them are dog breeds. To maximize the effect of the adversarial attacks, the target label should be in a different category than the original label. The target label for the adversarial attack image was thus shifted 100 steps right from the predicted top-1 label for the normal image. It is important to note that we used the top-1 predicted label, not the ground-truth label annotated in the database to perform adversarial attacks in order to make them more realistic (which is in accordance with the viewpoint of an attacker attacking a large-scale system without any knowledge of the ground-truth labels). Since there were 1,000 labels in total, modular operation was used to ensure the shifted label index was in the range $[0, 1000)$. The attack objective was to achieve a target class probability of 99%.

For non-targeted attacks, we used the basic gradient attack method implemented in FoolBox [196], the fast gradient signed method (FGSM) proposed by Goodfellow *et al.* [197], the Deep Fool method proposed by Moosavi-Dezfooli *et al.* [198], the Newton method proposed by Jang *et al.* [199], the ADef method proposed by Alaifari *et al.* [200], the Jacobian-based saliency map attack (JSMA) method proposed by Papernot *et al.* [201], and the method proposed by Carlini and Wagner [202]. Since these methods are non-targeted attacks, the attack objective was to change the predicted top-5 labels so that they differed from the original predicted top-1 labels.

C.2.4 Dataset analysis

As shown by the results in Table C.1, the VGG networks were generally more vulnerable to targeted attacks than the ResNet networks, resulting in more misclassified images. One possible explanation is that the skip connections used by ResNet networks make them more robust than the VGG networks. The modified BIM attack using the L2 distance (L2-iter) was the most effective attack overall. The non-targeted attacks were more difficult to carry out since they needed to change the top-5 labels so that they did not include the current top-1 labels. Among the attack methods, Deep Fool was the most successful while the ADef attack was the least successful overall, producing only one or four adversarial images for each network. The JSMA method also had limited success, with around 100 adversarial images for each network.

Beside the success rate of attacks, we also evaluated the quality of the obtained adversarial images by calculating the peak-signal-to-noise ratio (PSNR) and the structural similarity index measure (SSIM) [203] between them and their corresponding original images. The results are shown in Fig. C.3. Although there were some extreme cases, the average PSNR was about 40 dB while the average SSIM was greater than 0.9. This means that the quality of the adversarial images was high and that the perceived quality of the original images was preserved. The L-BFGS, Deep Fool, and Carlini-Wagner attacks produced adversarial images with the highest quality whereas the Gradient and FGSM attacks produced ones with the lowest quality. The L-BFGS, PGD, Deep Fool, ADef, JSMA, and Carlini-Wanger attacks produced adversarial images with stable and high SSIM.

C.3 Effects of Image Processing Operations on Normal and Adversarial Images

Some image processing operations like JPEG compression, spatial smoothing, and scalar quantization have recently been found to have noticeable effects on adversarial noise [186, 187, 192]. We expand on this and hypothesize that using multiple common image processing operations (already implemented in many image processing libraries) and multiple parameter values will provide much more useful information than using operations

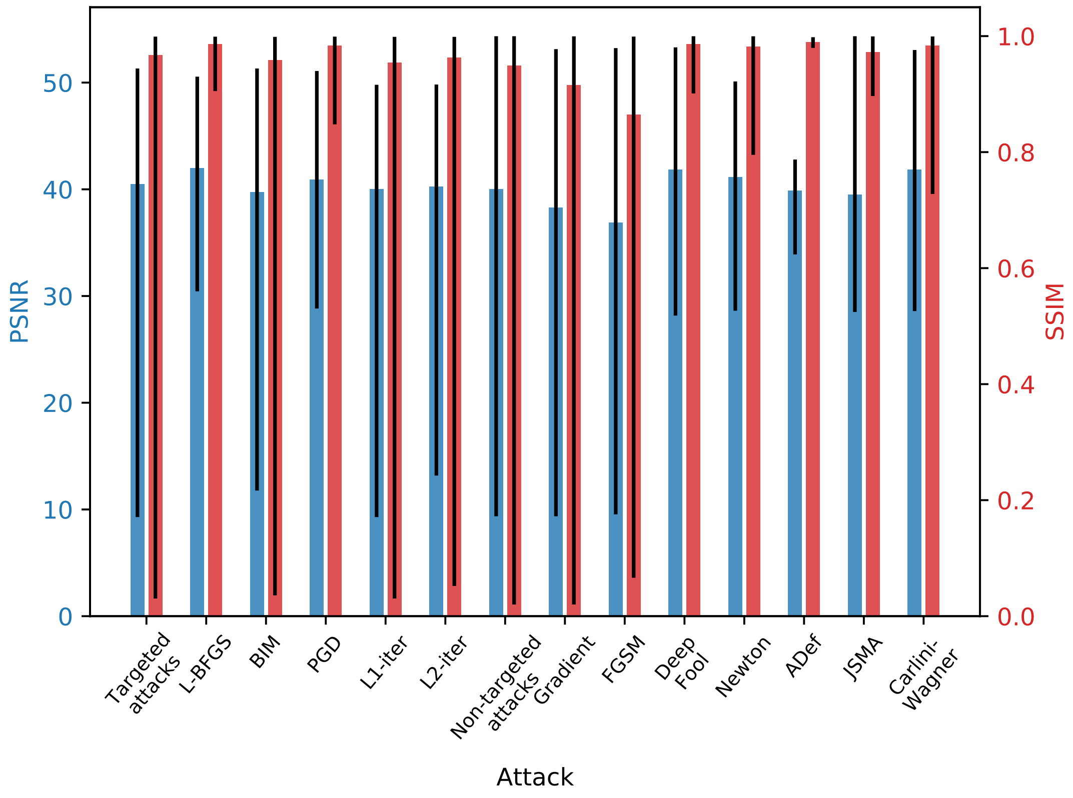

FIGURE C.3: Average PSNR (blue) and SSIM (red) between adversarial images used for targeted and non-targeted attacks and corresponding original images. Error bars indicate min and max values.

with fixed values as in previous work. Furthermore, the effects of multiple parameter values on adversarial images differ from those on natural images depending on the differences in the values. Although common image processing operations like JPEG compression, Gaussian blurring, rotation, and scaling do not substantially affect the usability of the processed images, they have certain non-linear effects on adversarial noise. JPEG compression reduces the number of bits needed for storage. Gaussian blurring removes high-frequency components and thus acts as a low-pass filter. Rotation, which requires the use of an interpolation algorithm to adjust the pixels, removes noise[2]. Scaling up also removes noise while scaling down, which reduces the entropy of an image (the degree of disorder, which is used to characterize the texture of an image), reduces the amount of noise. These effects on adversarial noise can be used to distinguish adversarial images from normal images as well as to correct adversarial images. It should thus be possible to identify operation-dependent characteristics from the outputs of an object-recognition CNN as the strength of an operation is gradually increased

[2]We used nearest-neighbor interpolation in our experiments.

To confirm this hypothesis, we applied the four image processing operations to images from our two datasets and classified the resulting images using the VGG-16, VGG-19, ResNet-18, and ResNet-50 networks. The parameter values were as follows:

- JPEG compression with QF $\in$ {100, 95, 90, 85, 80, 75, 70, 65, 60, 55, 50, 45, 40, 35, 30, 25}.
- Gaussian blurring with kernel size $\in$ {2, 3, 4, 5}.
- Clockwise image rotation with angle $\in$ {1°, 2°, 3°, 4°, 5°, 6°, 7°, 8°} and without reversing back. Nearest-neighbor interpolation was used.
- Image scaling with scale $\in$ {0.75, 0.8, 0.85, 0.9, 0.95, 1.05, 1.1, 1.15, 1.2, 1.25} and without reversing back. Nearest-neighbor interpolation was used.

The image operations were done using Pillow version 6.1.0. Some of the results for ResNet-50 are shown in Table C.3. Similar effects were also observed for VGG-16, VGG-19, and ResNet-18. The combined result of all networks is visualized in Fig. C.4. JPEG compression with a QF of 100 changed the top-5 labels for both the normal and adversarial images, and the effect on adversarial images was clearer. Reducing image quality by increasing the compression ratio greatly reduced the number of misclassified adversarial images and slightly increased that of misclassified normal images. A large increase in the ratio increased the misclassification rate for normal images. The results for scaling and rotation were similar to those for compression while those for Gaussian blurring differed slightly. JPEG compression is thus the best candidate to eliminate adversarial noise while scaling is the best for preserving the correctness of normal images. One result in particular should be noted: the number of misclassified normal images after applying a 3×3 Gaussian blur kernel was higher than after applying the other operations, which is not good for our objectives, *i.e.*, detection and correction of adversarial images. In summary, these results support our hypothesis that the classification labels of the adversarial images change when the values of the parameters change while the classification labels of normal images remain mostly unchanged. The next two sections describe the methods we devised for detecting and correcting adversarial examples by exploiting variations in the outputs of a DNN.

TABLE C.3: Number of top-5 **misclassified** images for ResNet-50 network from both normal and adversarial datasets before and after applying image processing operations.

Attack Setting/Datasets	**Original**	**JPEG Compression**					**Scaling**						**Gaussian Blurring**	**Rotation**	
		100	**80**	**60**	**40**	**20**	**0.75**	**0.85**	**0.95**	**1.05**	**1.15**	**1.25**	**3 × 3**	**2°**	**5°**
Normal images	0	1	18	33	46	92	43	23	18	13	26	29	259	32	57
Targeted attack:															
L-BFGS [36]	229	221	27	31	41	67	45	27	26	25	38	36	43	40	58
BIM [171]	431	423	47	49	59	88	51	38	40	42	57	50	65	60	79
PGD [171]	348	339	28	40	47	82	44	29	28	27	42	34	58	48	69
L1-iter [196]	513	507	51	51	56	81	55	48	51	55	67	59	73	79	84
L2-iter [196]	583	575	44	46	55	88	56	42	46	46	64	52	67	74	89
Non-targeted attack:															
Gradient [196]	517	515	112	89	93	114	94	77	91	107	124	115	108	126	155
FGSM [197]	379	373	128	129	134	139	100	71	92	120	155	127	95	138	158
Deep Fool [198]	604	594	32	21	27	55	33	24	34	37	34	30	59	59	50
Newton [199]	425	411	60	38	36	61	34	27	41	49	47	38	82	76	68
ADef [200]	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
JSMA [201]	71	72	29	18	29	38	35	23	23	20	25	20	33	35	44
Carlini-Wagner [202]	299	289	28	24	31	60	29	20	25	18	33	27	38	43	49

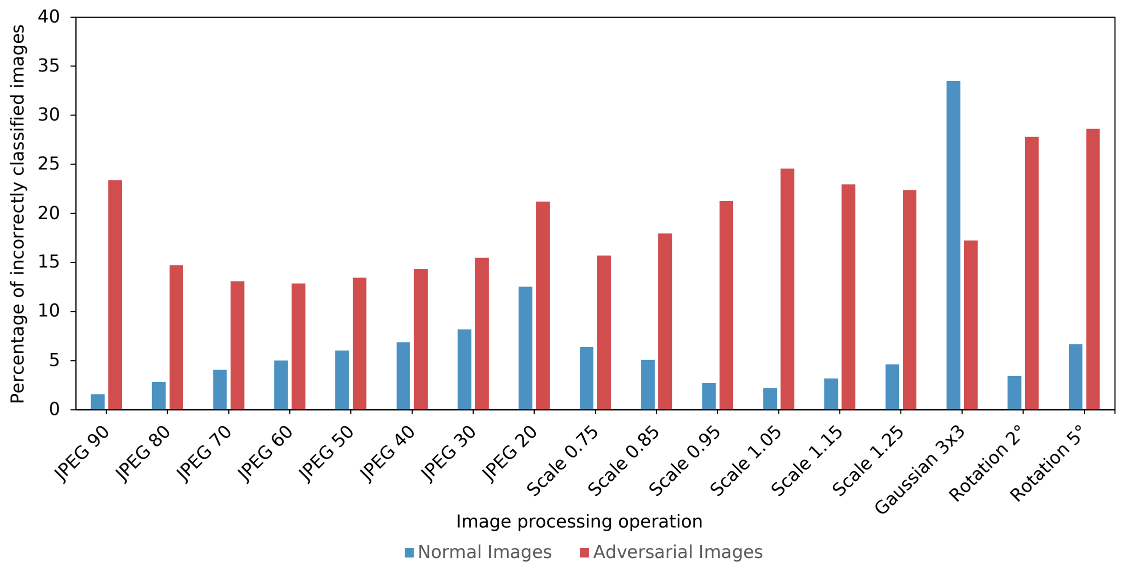

FIGURE C.4: Percentages of incorrectly classified normal and adversarial images after applying four image processing operations. Note that before applying these operations, the accuracy on normal images was 100% and on adversarial images was 0%. (Due to our dataset design, unqualified images were eliminated.)

C.4 Detecting Adversarial Images

As demonstrated in the previous section, the four image processing operations had different effects on normal and adversarial images. Moreover, changing the parameter values changed the DNN outputs. We utilize these effects and two statistical-based features, a counting feature (described in section C.4.1) and a differences feature (described in section C.4.2), to detect adversarial images. With each of these two features, we use four traditional machine learning classifiers (**traditional classifiers**, described in section C.4.3), and a CNN-based classifier (**stats-CNN classifier**, described in section C.4.4). For comparison, we used a feature squeezing method [186] as a baseline since it and our detection method are conceptually similar. To make our results more convincing, we also used as baselines two CNN-based classifiers that take raw images as input (**raw-image CNN classifiers**) (described in section C.4.4).

As statistical-based features of our detection method, we define three variables:

- $L = (a, b, c, d, e)$: top-5 label for image I (normal or adversarial) predicted using a CNN **before** applying image processing operation i (*e.g.*, JPEG compression with a QF of 80 or 5°clockwise rotation).

- $L_i = (a_i, b_i, c_i, d_i, e_i)$: top-5 label for an image **after** applying image processing operation i.
- n: total number of image processing operations (38 in our experiments).

C.4.1 Counting feature

We define $C(a)$ as the number of occurrences of label $a \in L = (a, b, c, d, e)$ at the first position in an ordered top-5 label set $\{L_i | i = 1, \ldots, n\}$.

$C(a)$ is defined as

$$C(a) = \sum_{i=1}^{n} \delta(a, a_i), \tag{C.1}$$

with

$$\delta(a, a_i) = \begin{cases} 1, & \text{if } a = a_i \\ 0, & \text{if } a \neq a_i. \end{cases} \tag{C.2}$$

The same equation is used for b, c, d, and e at the second, third, fourth, and fifth positions, respectively. Therefore, the features of each image are $\{C(a), C(b), C(c), C(d), C(e)\}$.

The following is a simple example of the counting feature using three image processing operations. Given $L = (100, 105, 198, 213, 479)$, we have

- $L_1 = (100, 97, 198, 220, 221)$
- $L_2 = (101, 80, 201, 119, 212)$
- $L_3 = (100, 89, 213, 117, 304)$

after applying the image processing operations. The resulting counting features are $\{2, 0, 1, 0, 0\}$.

C.4.2 Differences feature

We define $\Delta(a, a_i)$ as the binary differential function used to measure the difference between two labels a and a_i:

$$\Delta(a, a_i) = 1 - \delta(a, a_i). \tag{C.3}$$

The differences feature derived from input image I can be expressed as the set

$$\{(\Delta(a, a_i), \Delta(b, b_i), \Delta(c, c_i), \Delta(d, d_i), \Delta(e, e_i))|i = 1, \dots, n\}.$$

The differences features obtained using this example are $\{0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1\}$.

C.4.3 Feature selection and hyper-parameter tuning

Using the two statistical-based features introduced above (counting and differences), we evaluated the performance of detectors using one of the four traditional machine learning classifiers: the C-support vector classification (SVC) version of the support vector machine (SVM) classifier [167], the random forest classifier (RF) [204] with 100 trees and a maximum depth of 2, the linear discriminant analysis (LDA) classifier [166], and the multiple layer perception (MLP) classifier [165]. All of them were implemented in the scikit-learn library version 0.21.3[3]. For simplicity, we call them **traditional classifiers**. It is important to note that we use the terms “detector” and “classifier” interchangeably. In this feature selection and hyper-parameter tuning step, we trained them on only the dataset for the combination attack setting.

As shown in Table C.4, the differences feature and the counting one generally produced similar accuracies. Among the individual image processing operations, the scaling one achieved the highest accuracy for all detectors. The combination of JPEG compression and scaling and the combination of all operations resulted in higher accuracy than using any of them individually. However, these combinations also increased the feature size, and more classification operations were required for the object detection CNNs (VGG-Nets and ResNets) to produce those features. For the counting feature, using the MLP-based detector on the features from JPEG compression and the scaling operation resulted in the highest accuracy (93.56%) while for the differences feature, using the SVM-based detector on all features from all image processing operations resulted in the highest accuracy (94.20%).

[3]https://scikit-learn.org/stable

Table C.4: Accuracy (in %) of each classifier using counting feature (count) and differences feature (diff.) on eval set.

Classifier	JPEG Compression		Scaling		Gaussian Blurring		Rotation		JPEG + Scaling		All	
	Count	**Diff.**	**Count**	**Diff.**	**Count**	**Diff.**	**Count**	**Diff.**	**Count**	**Diff.**	**Count**	**Diff.**
SVM [167]	90.98	91.92	92.56	92.32	87.35	87.35	89.90	89.39	92.86	**94.04**	93.39	**94.20**
RF [204]	90.57	89.82	91.08	91.27	86.87	86.04	88.24	88.59	92.16	91.94	92.35	92.35
LDA [166]	89.98	90.36	91.97	91.62	85.77	87.35	89.15	89.29	92.86	92.67	92.21	92.86
MLP [165]	91.25	90.41	92.32	92.35	86.95	87.38	89.66	89.02	**93.56**	92.19	**93.37**	92.29

C.4.4 CNN-based classifiers

In addition to the traditional machine learning algorithms described in section C.4.3, we used two simple feed-forward CNNs as classifiers, one that takes the counting feature as input and one that takes the differences feature. The two CNNs used all the features extracted using the four image processing algorithms: JPEG compression, scaling, Gaussian blurring, and rotation. Each CNN had five layers of 1-D convolution (each convolution layer was followed by a batch normalization layer and a rectified linear unit), and two fully connected layers at the end with a dropout rate of 50%. For simplicity, we call these two networks **stats-CNN classifiers**.

We also used two common CNNs that take raw images as input: ResNet-50 [42] and XceptionNet [91]. The XceptionNet one is commonly used for forgery detection [23]. We modified their last fully connected layer so that the output was binary. These two CNN-based classifiers take images as input and output the probabilities that those images are adversarial. Since there was limited training data, beside training them from scratch, we also fine-tuned their ImageNet pre-trained versions. For simplicity, we call these two networks **raw-image CNN classifiers**.

For both the stats- and raw-image CNN classifiers, we used a learning rate of 5×10^{-4} for all cases. Each network was trained for 150 epochs, and the checkpoint with the highest accuracy on the dev set was selected for evaluation.

C.4.5 Evaluation

We selected the two best traditional classifiers, the SVM (SVC) one using all features and the MLP one using the JPEG compression and scaling features, to compare with the stats-CNN classifiers, the raw-image CNN classifiers, and the feature squeezing method [186]. We tested them on two scenarios: (1) seen attacks in which all classifiers were trained and tested on the dataset for the combination setting and (2) unseen attacks in which all classifiers were trained on the dataset for the targeted attack setting and were tested on the dataset for the non-targeted attack setting and vice versa. The dataset details are listed in Table C.2. Since the feature squeezing method only requires training on normal images, we used the “Best Joint Detection (5-bit, 2x2, 11-3-4)” setting [186] (designed for the ImageNet database) with a threshold of 1.2128 pre-trained for all scenarios. Since the output of the feature squeezing method is binary, EERs could not be calculated. For the other methods, the accuracies were calculated using a threshold of 0.5. The experiment results are shown in Table C.5.

Overall, the raw-image CNN classifiers outperformed the statistical-based ones (including the traditional classifiers and the stats-CNN classifier) and the feature squeezing method in both the seen and unseen scenarios. Between the two raw-image CNN classifiers, surprisingly, the XceptionNet one trained from scratch achieved better performance and outperformed its fine-tuned version with accuracies greater than 99% and EERs less than 1%. On the other hand, the ResNet classifier with fine-tuning outperformed its trained version. The performances of the traditional classifiers and the stats-CNN classifiers were similar, indicating that their scores are the upper limits for the statistical features. Although having limited results and limited generalizability compared with the raw-image CNN classifiers, our statistical-based classifiers nevertheless have reasonable discriminative ability and overall outperformed the feature squeezing method. These results support our hypothesis that using various parameter values is better than using fixed ones. In addition, it is important to note that CNN-based classifiers, especially raw-image ones, are potentially vulnerable to second-level adversarial attacks. That is, attackers can disguise their adversarial images by adding secondary adversarial noise to alter the output of the adversarial image detector from adversarial to normal. In this case, our statistical-based detection method is more robust.

Another interesting result is that the detectors trained on the dataset for the non-targeted attack setting had better generalizability than those trained on the dataset for the targeted attack one. This can be interpreted to mean that the adversarial noise created by non-targeted attacks is more general than that created by targeted attacks. Therefore, when designing an adversarial machine learning dataset, it is crucial to include a sufficient amount of data for non-targeted attacks.

Table C.5: Accuracy and EER (in %) of each detector on eval sets. First two detectors were best statistical-based classifiers selected on basis of results presented in previous section. Next four were CNN-based detectors used as baselines.

Attack Setting/Detector	Accuracy	EER
Seen attacks (Combination):		
MLP - counting feature	93.56	10.00
SVM - differences feature	94.20	9.67
CNN - counting feature	93.29	10.21
CNN - differences feature	94.09	9.50
Feature squeezing [186]	83.97	-
ResNet-50 (training)	83.89	49.09
ResNet-50 (fine-tuning)	99.49	0.66
XceptionNet (training)	**99.95**	**0.15**
XceptionNet (fine-tuning)	99.60	0.49
Targeted attacks → Non-targeted attacks:		
MLP - counting feature	68.62	18.81
SVM - differences feature	71.03	15.16
CNN - counting feature	60.39	21.31
CNN - differences feature	66.79	17.73
Feature squeezing [186]	72.71	-
ResNet-50 (training)	71.13	50.50
ResNet-50 (fine-tuning)	95.72	3.17
XceptionNet (training)	**99.42**	**0.60**
XceptionNet (fine-tuning)	83.69	12.95
Non-targeted attacks → Targeted attacks:		
MLP - counting feature	93.63	3.77
SVM - differences feature	93.99	3.76
CNN - counting feature	93.32	3.04
CNN - differences feature	93.32	4.02
Feature squeezing [186]	84.42	-
ResNet-50 (training)	99.42	0.61
ResNet-50 (fine-tuning)	99.02	0.97
XceptionNet (training)	**100.00**	**0.00**
XceptionNet (fine-tuning)	99.78	0.52

C.5 Correcting Adversarial Images

Given that the image processing operations substantially reduced the number of misclassified adversarial images while only slightly affecting the normal images, we utilized them to correct adversarial images. We developed a correction method with two levels of correction:

- **Label correction**: Restore the original labels of adversarial images. At this level, only the output labels are of interest; for example, only the true label of a manipulated stop sign is of interest to a self-driving car.

- **Image correction**: Mitigate adversarial noise in adversarial images to restore their original labels and make them usable for other tasks. At this level, the quality and usability of the corrected images as well as the labels are of interest. Image correction is therefore more complicated than label correction.

Our correction method can be used independently or in combination with an adversarial image detector. Since it is not always clear whether an image is normal or adversarial, a correction method should work well on both normal and adversarial images. Details of the label correction and image correction algorithms are described in the next two sections.

C.5.1 Label correction algorithm

We define $\mathbf{S_L} = \{L_i | i = 1, \ldots, n\}$ as the set of top-5 labels acquired by applying n image processing operations to image I. The frequencies of every label in $\mathbf{S_L}$ are calculated, and the five labels with the highest frequencies are identified as the corrected top-5 ones.

C.5.2 Image correction algorithm

In the context of image classification, an image correction method must satisfy the following conditions:

- Adversarial noise must be mitigated so that the corrected images can be correctly classified by the DNN.
- The usability of the adversarial images must be recovered and that of the normal images must be preserved.
- The quality of both the normal and adversarial images must be preserved as much as possible.

Since JPEG compression has good performance on both normal and adversarial images, we use it as the core image processing operation to eliminate adversarial noise. There are two ways of using JPEG compression for image correction:

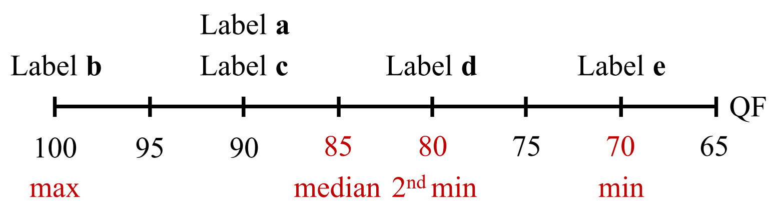

FIGURE C.5: Visualization of corrected top-5 labels calculated in step 1 of heuristic algorithm and their corresponding QFs calculated in step 2. The *max*, *median*, $2^{nd}min$, and *min* of S_{QF} labels on the axis represent the candidate QFs to be selected in step 3.

1. *Baseline:* Use a **fixed QF** for JPEG compression. The downside with this approach is that there are trade-offs between performance on normal and adversarial images and between performance and the quality of the corrected images.

2. *Proposed:* Use a **heuristic algorithm** to determine the optimal QF for JPEG compression. With this approach, the corrected labels calculated with the label correction algorithm are used to calculate the best QFs for producing those labels. This is discussed in detail below.

We define f_{label} as the corrector function described in section C.5.1 and I as the image to be corrected. The proposed heuristic algorithm has three steps:

- ***Step 1***: Calculate the corrected top-5 labels: $L_{corr} = f_{label}(I)$.
- ***Step 2***: For each label l in L_{corr}, select the highest QF from the sorted list of QFs $\{100, 95, 90, \ldots, 40\}$ that produces l and put it into S_{QF}.
- ***Step 3***: Select a QF from S_{QF} and use it to compress I to obtain the corrected image.

The strategies that can be used for selecting a QF from S_{QF} in step 3 are visualized in Fig. C.5. The candidates are *max*, *median*, $2^{nd}min$, and *min* of S_{QF}. The higher the QF, the better the quality of the corrected image. If the input image is natural, $max(S_{QF})$ is obviously the best choice. However, if the input image is adversarial, $min(S_{QF})$ seem to be the safest choice. The performances achieved with these choices are presented and discussed in the next section.

C.5.3 Evaluation

Since there is no learning required for the correction algorithms, we tested them on the entire normal dataset and adversarial image dataset. We tested label correction first and then image correction.

C.5.3.1 Label correction

We used JPEG compression, scaling, and their combination for label correction. As shown in Fig. C.6, JPEG compression had better performance than scaling for the adversarial images, and their combination produced the best overall performance. Only 1.88% of the normal images was misclassified after "correction" while 89.91% of the adversarial images were corrected. If this correction was performed after detection of adversarial images, about 98.12% of the false positive inputs (normal images misclassified as adversarial ones) would also be corrected, therefore boosting the overall performance.

C.5.3.2 Image correction

We tested the fixed QF baseline approach with several QF values from 40 to 90 and the proposed heuristic algorithm with S_{QF} values of *max*, *median*, $2^{nd}min$, and *min*. In addition, we built a convolutional denoising autoencoder [205] to check whether it is useful for adversarial noise removal. To avoid data overlapping, we trained it on the test set of the ImageNet database using Gaussian noise with $\sigma = 0.1$.

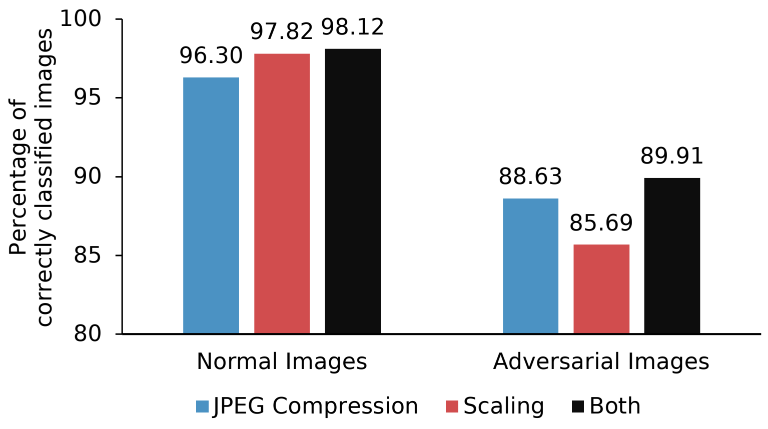

FIGURE C.6: Percentage of correct classification results after applying label correction method to both normal and adversarial images.

TABLE C.6: Percentage of correct classification of corrected images (both normal and adversarial).

Correction Method	**Normal Images**	**Adversarial Images**
QF 90	98.40	77.38
QF 80	**97.17**	**86.24**
QF 70	**95.93**	**88.07**
QF 60	**94.97**	**88.46**
QF 50	93.97	88.02
QF 40	93.12	87.28
Heuristic - *max*	99.65	25.81
Heuristic - *median*	99.55	54.03
Heuristic - $2^{nd}min$	98.98	72.96
Heuristic - min	**97.52**	**88.98**
Denoising autoencoder	57.77	49.81

As shown in Table C.6, the denoising autoencoder had poor performances on both normal and adversarial images. One explanation is that denoised images have different distributions than normal images; therefore, the DNNs could not correctly classify most of them. For the fixed QF baseline approach, there was a trade-off between performance on normal and adversarial images. The QFs between 60 and 80 are safe choices for achieving balanced performance between the two types of images. With the heuristic algorithm, using $max(S_{QF})$ preserved most of the normal images, but it was the worst at mitigating adversarial noise. Although it sits at the mid-point of the S_{QF} values, using *median* resulted in poor correction of the adversarial images. Using the two remaining values resulted in substantial improvement in mitigating adversarial noise. Using *min* achieved the best balance in performance between normal and adversarial images: 97.52% correct classification for normal images and 88.98% for adversarial images.

Examples of normal images, their corresponding adversarial images, and their corrected versions are shown in Fig. C.7. There were no perceived substantial differences between the different versions, meaning that adversarial noise is difficult to notice and that the correction algorithm did not substantially degrade the quality of both the normal and adversarial images. The average PSNR and SSIM between the corrected adversarial images and their original versions in the entire datasets are shown in Fig. C.8. The heuristic - *min* approach had higher PSNR and SSIM than the fixed QF approach. The explanation for this is illustrated in Fig. C.9, which is a histogram of the QFs selected using the heuristic - *min* approach for both normal and adversarial images. For the normal images, in most cases, a QF of 100 was selected, so their quality was almost

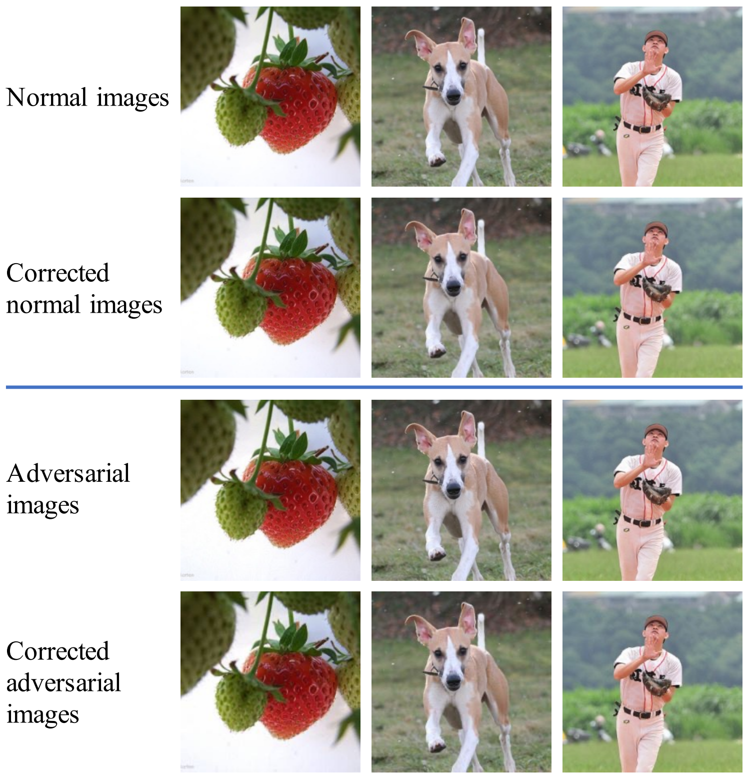

FIGURE C.7: From top to bottom: Examples of normal images, their corrected versions, their adversarial versions, and the corrected versions of the adversarial images (corrected using heuristic - *min* algorithm).

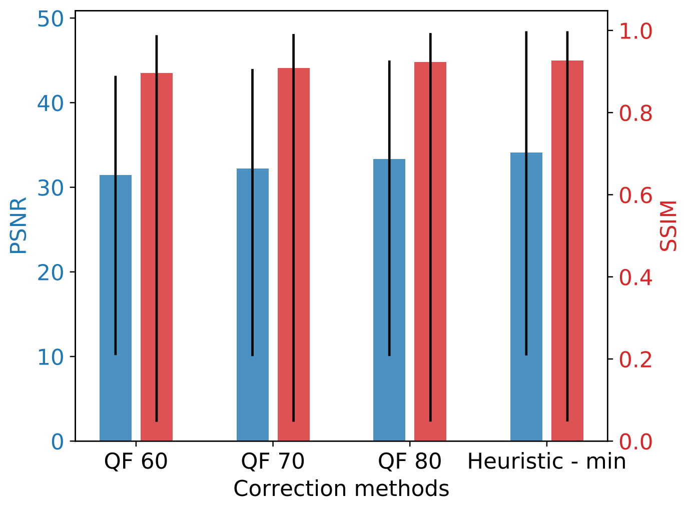

FIGURE C.8: Average PSNR (blue) and SSIM (red) between corrected adversarial images and corresponding original images. Error bars indicate min and max values.

completely preserved. For the adversarial images, more than two-thirds of the selected QFs were from 75 to 95, so their quality was also good. Since there are some extreme cases in the adversarial dataset, the correction algorithm could not fully recover their quality, resulting in low values of PSNR and SSIM.

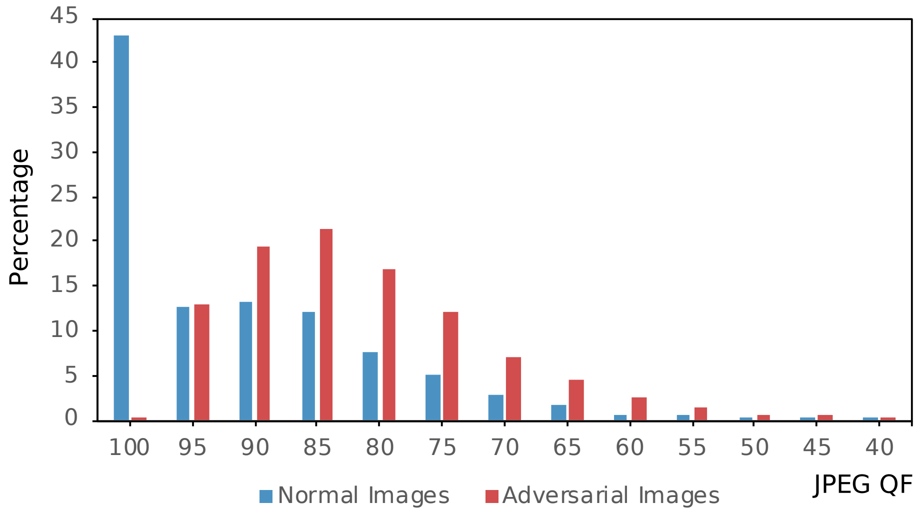

Figure C.9: Histogram of JPEG QF calculated using heuristic - *min* algorithm on both normal and adversarial images.

C.5.4 Considerations

Our experiments demonstrated the effectiveness of using image processing operations for both label correction and image correction. For label correction, using multiple operations with multiple parameter values helped maximize the chance of restoring the original labels. It also provides label information for image correction. The proposed heuristic - *min* algorithm effectively determines which QF is the best for image correction so that image quality is preserved as much as possible, which is better than using JPEG compression with a fixed QF. The correction algorithms can be used independently, for instance, in data cleansing, or in combination with an adversarial image detector to provide multiple outputs: (1) whether the input image is adversarial and (2) if yes, what its true label is.

C.6 Discussion and Future Work

Our results demonstrated that image processing operations like JPEG compression, scaling, Gaussian blurring, and rotation have different effects on normal and adversarial images depending on the parameter values. Although these operations have negligible effects on normal images, they are effective in mitigating adversarial noise, which is useful for both detection and correction of adversarial images. Different from adversarial training, the proposed detection and correction methods can be performed as a pre-filter to process the data before being processed by a DNN. They are thus applicable to all pre-trained DNNs without the need for re-training. One disadvantage of using multiple image processing operations with multiple parameter values is the computation cost when several processed images need to be classified .

Our results also demonstrated that using statistical features based on image processing operations and using feature squeezing are not as effective as using features automatically extracted by a CNN from raw images, especially the XceptionNet one, which can be trained with a small amount of data. However, the traditional classifiers using handcrafted features are more robust than the CNN-based ones when the attackers add adversarial noise to fool the adversarial image detector (due to the fact that most of the currently implemented image processing operations are non-differentiable). We also found that the adversarial noise created by non-targeted attacks is more general than that created by targeted attacks; therefore, it is important to include non-targeted adversarial images in the training data.

Future work includes testing using more adversarial attacks with multiple noise strengths on larger and more diverse databases. It also includes evaluating additional image processing operations and reducing the computational expense of detection and correction. Another important task is dealing with robust adversarial examples, which is a challenging problem.

Bibliography

[1] Goode Intelligence. The Goode intelligence biometric survey 2021, April 2021. URL https://www.goodeintelligence.com/report/the-goode-intelligence-biometric-survey-2021/.

[2] Sushil Bhattacharjee, Amir Mohammadi, André Anjos, and Sébastien Marcel. Recent advances in face presentation attack detection. In *Handbook of Biometric Anti-Spoofing*, pages 207–228. Springer, 2019.

[3] Trong-Le Do, Mai-Khiem Tran, Huy H Nguyen, and Minh-Triet Tran. Potential threat of face swapping to ekyc with face registration and augmented solution with deepfake detection. In *International Conference on Future Data and Security Engineering (FDSE)*, 2021.

[4] Pavel Korshunov and Sébastien Marcel. Subjective and objective evaluation of deepfake videos. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2510–2514. IEEE, 2021.

[5] Trung-Nghia Le, Huy H Nguyen, Junichi Yamagishi, and Isao Echizen. Openforensics: Large-scale challenging dataset for multi-face forgery detection and segmentation in-the-wild. In *International Conference on Computer Vision (ICCV)*, pages 10117–10127, 2021.

[6] Nicolas M Müller, Karla Markert, and Konstantin Böttinger. Human perception of audio deepfakes. *arXiv preprint arXiv:2107.09667*, 2021.

[7] Terrifying high-tech porn: Creepy 'deepfake' videos are on the rise. https://www.foxnews.com/tech/terrifying-high-tech-porn-creepy-deepfake-videos-are-on-the-rise. Accessed: 2018-02-17.

[8] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of GANs for improved quality, stability, and variation. In *ICLR*, 2018.

[9] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4401–4410, 2019.

[10] Yunjey Choi, Minje Choi, Munyoung Kim, Jung-Woo Ha, Sunghun Kim, and Jaegul Choo. StarGAN: Unified generative adversarial networks for multi-domain image-to-image translation. In *Conference on computer vision and pattern recognition (CVPR)*, pages 8789–8797, 2018.

[11] Yunjey Choi, Youngjung Uh, Jaejun Yoo, and Jung-Woo Ha. StarGAN v2: Diverse image synthesis for multiple domains. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8188–8197, 2020.

[12] Supasorn Suwajanakorn, Steven M Seitz, and Ira Kemelmacher-Shlizerman. Synthesizing obama: learning lip sync from audio. *ACM Transactions on Graphics*, 2017.

[13] Hadar Averbuch-Elor, Daniel Cohen-Or, Johannes Kopf, and Michael F Cohen. Bringing portraits to life. *ACM Transactions on Graphics*, 2017.

[14] Egor Zakharov, Aliaksandra Shysheya, Egor Burkov, and Victor Lempitsky. Few-shot adversarial learning of realistic neural talking head models. *arXiv preprint arXiv:1905.08233*, 2019.

[15] Oleg Alexander, Mike Rogers, William Lambeth, Jen-Yuan Chiang, Wan-Chun Ma, Chuan-Chang Wang, and Paul Debevec. The digital emily project: Achieving a photorealistic digital actor. *IEEE Computer Graphics and Applications*, 30(4): 20–31, 2010.

[16] Dexter studio. http://dexterstudios.com/en/. Accessed: 2019-09-01.

[17] Justus Thies, Michael Zollhofer, Marc Stamminger, Christian Theobalt, and Matthias Nießner. Face2Face: Real-time face capture and reenactment of RGB videos. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2016.

[18] Justus Thies, Michael Zollhöfer, and Matthias Nießner. Deferred neural rendering: Image synthesis using neural textures. In *Computer Graphics and Interactive Techniques (SIGGRAPH)*. ACM, 2019.

[19] Timo Ojala, Matti Pietikainen, and Topi Maenpaa. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Transactions on pattern analysis and machine intelligence*, 24(7):971–987, 2002.

[20] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Conference on computer vision and pattern recognition (CVPR)*, volume 1, pages 886–893. Ieee, 2005.

[21] Jessica Fridrich and Jan Kodovsky. Rich models for steganalysis of digital images. *IEEE Transactions on Information Forensics and Security*, 2012.

[22] Darius Afchar, Vincent Nozick, Junichi Yamagishi, and Isao Echizen. MesoNet: a compact facial video forgery detection network. In *International Workshop on Information Forensics and Security (WIFS)*. IEEE, 2018.

[23] Andreas Rössler, Davide Cozzolino, Luisa Verdoliva, Christian Riess, Justus Thies, and Matthias Nießner. Faceforensics++: Learning to detect manipulated facial images. In *International Conference on Computer Vision (ICCV)*, 2019.

[24] Hao Dang, Feng Liu, Joel Stehouwer, Xiaoming Liu, and Anil K Jain. On the detection of digital face manipulation. In *Conference on Computer Vision and Pattern recognition CVPR*, pages 5781–5790, 2020.

[25] Yuyang Qian, Guojun Yin, Lu Sheng, Zixuan Chen, and Jing Shao. Thinking in frequency: Face forgery detection by mining frequency-aware clues. In *European Conference on Computer Vision (ECCV)*, pages 86–103. Springer, 2020.

[26] Yuezun Li, Ming-Ching Chang, Hany Farid, and Siwei Lyu. In ictu oculi: Exposing AI generated fake face videos by detecting eye blinking. *arXiv preprint arXiv:1806.02877*, 2018.

[27] Shruti Agarwal, Hany Farid, Yuming Gu, Mingming He, Koki Nagano, and Hao Li. Protecting world leaders against deep fakes. In *Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 38–45, 2019.

[28] Ekraam Sabir, Jiaxin Cheng, Ayush Jaiswal, Wael AbdAlmageed, Iacopo Masi, and Prem Natarajan. Recurrent convolutional strategies for face manipulation detection in videos. In *Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 80–87, 2019.

[29] Pavel Korshunov and Sébastien Marcel. Speaker inconsistency detection in tampered video. In *European Signal Processing Conference (EUSIPCO)*, pages 2375–2379. IEEE, 2018.

[30] Peng Zhou, Xintong Han, Vlad I Morariu, and Larry S Davis. Two-stream neural networks for tampered face detection. In *Conference on Computer Vision and Pattern Recognition Workshop (CVPRW)*. IEEE, 2017.

[31] Brian Dolhansky, Joanna Bitton, Ben Pflaum, Jikuo Lu, Russ Howes, Menglin Wang, and Cristian Canton Ferrer. The deepfake detection challenge dataset. *arXiv preprint arXiv:2006.07397*, 2020.

[32] Huy H Nguyen, Ngoc-Dung T Tieu, Hoang-Quoc Nguyen-Son, Vincent Nozick, Junichi Yamagishi, and Isao Echizen. Modular convolutional neural network for discriminating between computer-generated images and photographic images. In *International Conference on Availability, Reliability and Security (ARES)*. ACM, 2018.

[33] Huy H Nguyen, Fuming Fang, Junichi Yamagishi, and Isao Echizen. Multi-task learning for detecting and segmenting manipulated facial images and videos. In *International Conference on Biometrics: Theory, Applications and Systems (BTAS)*. IEEE, 2019.

[34] Minha Kim, Shahroz Tariq, and Simon S Woo. Fretal: Generalizing deepfake detection using knowledge distillation and representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1001–1012, 2021.

[35] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NIPS*, pages 2672–2680, 2014.

[36] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[37] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Dan Boneh, and Patrick McDaniel. Ensemble adversarial training: Attacks and defenses. In *International Conference on Learning Representations (ICLR)*, 2018.

[38] Eric Wong, Leslie Rice, and J Zico Kolter. Fast is better than free: Revisiting adversarial training. In *International Conference on Learning Representations (ICLR)*, 2019.

[39] Ulrich Scherhag, Christian Rathgeb, Johannes Merkle, Ralph Breithaupt, and Christoph Busch. Face recognition systems under morphing attacks: A survey. *IEEE Access*, 7:23012–23026, 2019.

[40] Masashi Une, Akira Otsuka, and Hideki Imai. Wolf attack probability: A new security measure in biometric authentication systems. In *ICB*, pages 396–406. Springer, 2007.

[41] Philip Bontrager, Aditi Roy, Julian Togelius, Nasir Memon, and Arun Ross. Deepmasterprints: Generating masterprints for dictionary attacks via latent variable evolution. In *BTAS*, pages 1–9. IEEE, 2018.

[42] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[43] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *British Machine Vision Conference (BMVC)*. BMVA, 2016.

[44] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4700–4708, 2017.

[45] Dong Yi, Zhen Lei, Shengcai Liao, and Stan Z Li. Learning face representation from scratch. *arXiv preprint arXiv:1411.7923*, 2014.

[46] 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*, pages 87–102. Springer, 2016.

[47] Tiago de Freitas Pereira, André Anjos, and Sébastien Marcel. Heterogeneous face recognition using domain specific units. *IEEE Transactions on Information Forensics and Security*, 14(7):1803–1816, 2018.

[48] Florian Schroff, Dmitry Kalenichenko, and James Philbin. FaceNet: A unified embedding for face recognition and clustering. In *CVPR*, pages 815–823. IEEE, 2015.

[49] Jiankang Deng, Jia Guo, Niannan Xue, and Stefanos Zafeiriou. ArcFace: Additive angular margin loss for deep face recognition. In *CVPR*, pages 4690–4699, 2019.

[50] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 2015.

[51] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations (ICLR)*, 2015.

[52] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, 2017.

[53] Omkar M. Parkhi, Andrea Vedaldi, and Andrew Zisserman. Deep face recognition. In *BMVC*, pages 41.1–41.12. British Machine Vision Association, 2015.

[54] Xiang Wu, Ran He, Zhenan Sun, and Tieniu Tan. A light CNN for deep face representation with noisy labels. *IEEE Transactions on Information Forensics and Security*, 13(11):2884–2896, 2018.

[55] David Sandberg. FaceNet: face recognition using tensorflow. https://github.com/davidsandberg/facenet, 2017.

[56] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. DeepFace: Closing the gap to human-level performance in face verification. In *CVPR*, pages 1701–1708, 2014.

[57] Luan Tran, Xi Yin, and Xiaoming Liu. Disentangled representation learning GAN for pose-invariant face recognition. In *CVPR*, pages 1415–1424. IEEE, 2017.

[58] Yueqi Duan, Jiwen Lu, and Jie Zhou. Uniformface: Learning deep equidistributed representation for face recognition. In *CVPR*, pages 3415–3424, 2019.

[59] Nalini K. Ratha, Jonathan H. Connell, and Ruud M. Bolle. Enhancing security and privacy in biometrics-based authentication systems. *IBM systems Journal*, 40 (3):614–634, 2001.

[60] Javier Hernandez-Ortega, Julian Fierrez, Aythami Morales, and Javier Galbally. Introduction to face presentation attack detection. In *Handbook of Biometric Anti-Spoofing*, pages 187–206. Springer, 2019.

[61] Jukka Määttä, Abdenour Hadid, and Matti Pietikäinen. Face spoofing detection from single images using micro-texture analysis. In *International joint conference on Biometrics (IJCB)*, pages 1–7. IEEE, 2011.

[62] Jukka Komulainen, Abdenour Hadid, and Matti Pietikäinen. Context based face anti-spoofing. In *International Conference on Biometrics: Theory, Applications and Systems (BTAS)*, pages 1–8. IEEE, 2013.

[63] Zinelabidine Boulkenafet, Jukka Komulainen, and Abdenour Hadid. Face anti-spoofing based on color texture analysis. In *International conference on image processing (ICIP)*, pages 2636–2640. IEEE, 2015.

[64] Keyurkumar Patel, Hu Han, and Anil K Jain. Secure face unlock: Spoof detection on smartphones. *IEEE transactions on information forensics and security*, 11(10): 2268–2283, 2016.

[65] David G Lowe. Object recognition from local scale-invariant features. In *International conference on computer vision (ICCV)*, volume 2, pages 1150–1157. Ieee, 1999.

[66] Anjith George and Sébastien Marcel. Deep pixel-wise binary supervision for face presentation attack detection. In *International Conference on Biometrics (ICB)*, 2019.

[67] Usman Muhammad and Abdenour Hadid. Face anti-spoofing using hybrid residual learning framework. In *International Conference on Biometrics (ICB)*, 2019.

[68] Gang Pan, Lin Sun, Zhaohui Wu, and Shihong Lao. Eyeblink-based anti-spoofing in face recognition from a generic webcamera. In *International conference on computer vision (ICCV)*, pages 1–8. IEEE, 2007.

[69] Olegs Nikisins, Anjith George, and Sébastien Marcel. Domain adaptation in multi-channel autoencoder based features for robust face anti-spoofing. In *International Conference on Biometrics (ICB)*, 2019.

[70] Guoqing Wang, Hu Han, Shiguang Shan, and Xilin Chen. Unsupervised adversarial domain adaptation for cross-domain face presentation attack detection. *IEEE Transactions on Information Forensics and Security*, 16:56–69, 2020.

[71] Soroush Fatemifar, Muhammad Awais, Shervin Rahimzadeh Arashloo, and Josef Kittler. Combining multiple one-class classifiers for anomaly based face spoofing attack detection. In *International Conference on Biometrics (ICB)*, 2019.

[72] Amir Mohammadi, Sushil Bhattacharjee, and Sébastien Marcel. Improving cross-dataset performance of face presentation attack detection systems using face recognition datasets. In *ICASSP*. IEEE, 2020.

[73] Manabu Inuma, Akira Otsuka, and Hideki Imai. Theoretical framework for constructing matching algorithms in biometric authentication systems. In *ICB*, pages 806–815. Springer, 2009.

[74] Huy H Nguyen, Junichi Yamagishi, Isao Echizen, and Sébastien Marcel. Generating master faces for use in performing wolf attacks on face recognition systems. In *IJCB*, 2020.

[75] Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001.

[76] Philip Bontrager, Wending Lin, Julian Togelius, and Sebastian Risi. Deep interactive evolution. In *International Conference on Computational Intelligence in Music, Sound, Art and Design*, pages 267–282. Springer, 2018.

[77] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. In *ICLR*, 2014.

[78] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *ICML*, pages 214–223, 2017.

[79] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of Wasserstein GANs. In *NIPS*, pages 5767–5777, 2017.

[80] Huaibo Huang, Ran He, Zhenan Sun, Tieniu Tan, et al. Introvae: Introspective variational autoencoders for photographic image synthesis. In *NIPS*, pages 52–63, 2018.

[81] Ali Razavi, Aaron van den Oord, and Oriol Vinyals. Generating diverse high-fidelity images with VQ-VAE-2. In *Advances in Neural Information Processing Systems*, pages 14866–14876, 2019.

[82] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale GAN training for high fidelity natural image synthesis. In *ICLR*, 2018.

[83] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of StyleGAN. In *CVPR*, pages 8110–8119, 2020.

[84] Hyeongwoo Kim, Pablo Garrido, Ayush Tewari, Weipeng Xu, Justus Thies, Matthias Nießner, Patrick Pérez, Christian Richardt, Michael Zollhöfer, and Christian Theobalt. Deep video portraits. In *International Conference and Exhibition on Computer Graphics and Interactive Techniques (SIGGRAPH)*. ACM, 2018.

[85] Soumya Tripathy, Juho Kannala, and Esa Rahtu. Icface: Interpretable and controllable face reenactment using gans. *arXiv preprint arXiv:1904.01909*, 2019.

[86] Yuval Nirkin, Yosi Keller, and Tal Hassner. Fsgan: Subject agnostic face swapping and reenactment. In *International Conference on Computer Vision (ICCV)*. IEEE, 2019.

[87] Taihong Xiao, Jiapeng Hong, and Jinwen Ma. Elegant: Exchanging latent encodings with gan for transferring multiple face attributes. In *European conference on computer vision (ECCV)*, pages 168–184, 2018.

[88] Ohad Fried, Ayush Tewari, Michael Zollhöfer, Adam Finkelstein, Eli Shechtman, Dan B Goldman, Kyle Genova, Zeyu Jin, Christian Theobalt, and Maneesh Agrawala. Text-based editing of talking-head video. In *International Conference and Exhibition on Computer Graphics and Interactive Techniques (SIGGRAPH)*. ACM, 2019.

[89] Konstantinos Vougioukas, Samsung AI Center, Stavros Petridis, and Maja Pantic. End-to-end speech-driven realistic facial animation with temporal gans. In *Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 37–40, 2019.

[90] Davide Cozzolino, Giovanni Poggi, and Luisa Verdoliva. Recasting residual-based local descriptors as convolutional neural networks: an application to image forgery detection. In *Workshop on Information Hiding and Multimedia Security (IH&MMSEC)*. ACM, 2017.

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

[92] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015.

[93] Sheng-Yu Wang, Oliver Wang, Andrew Owens, Richard Zhang, and Alexei A Efros. Detecting photoshopped faces by scripting photoshop. In *International Conference on Computer Vision (ICCV)*. IEEE, 2019.

[94] Belhassen Bayar and Matthew C Stamm. A deep learning approach to universal image manipulation detection using a new convolutional layer. In *Workshop on Information Hiding and Multimedia Security (IH&MMSEC)*. ACM, 2016.

[95] Peng Zhou, Xintong Han, Vlad I Morariu, and Larry S Davis. Learning rich features for image manipulation detection. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1053–1061, 2018.

[96] Jawadul H Bappy, Cody Simons, Lakshmanan Nataraj, BS Manjunath, and Amit K Roy-Chowdhury. Hybrid lstm and encoder-decoder architecture for detection of image forgeries. *IEEE Transactions on Image Processing*, 2019.

[97] Lingzhi Li, Jianmin Bao, Ting Zhang, Hao Yang, Dong Chen, Fang Wen, and Baining Guo. Face x-ray for more general face forgery detection. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5001–5010, 2020.

[98] Pavel Korshunov and Sébastien Marcel. Deepfakes: a new threat to face recognition? assessment and detection. Idiap-RR Idiap-RR-18-2018, Idiap, 2018.

[99] Andreas Rössler, Davide Cozzolino, Luisa Verdoliva, Christian Riess, Justus Thies, and Matthias Nießner. FaceForensics: A large-scale video dataset for forgery detection in human faces. *arXiv preprint arXiv:1803.09179*, 2018.

[100] Lingzhi Li, Jianmin Bao, Hao Yang, Dong Chen, and Fang Wen. Faceshifter: Towards high fidelity and occlusion aware face swapping. *arXiv preprint arXiv:1912.13457*, 2019.

[101] Contributing data to deepfake detection research. https://ai.googleblog.com/2019/09/contributing-data-to-deepfake-detection.html. Accessed: 2019-09-24.

[102] Yuezun Li, Xin Yang, Pu Sun, Honggang Qi, and Siwei Lyu. Celeb-df: A large-scale challenging dataset for deepfake forensics. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3207–3216, 2020.

[103] 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. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[104] Bojia Zi, Minghao Chang, Jingjing Chen, Xingjun Ma, and Yu-Gang Jiang. Wilddeepfake: A challenging real-world dataset for deepfake detection. In *ACM International Conference on Multimedia*, pages 2382–2390, 2020.

[105] Tianfei Zhou, Wenguan Wang, Zhiyuan Liang, and Jianbing Shen. Face forensics in the wild. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5778–5788, 2021.

[106] Ivan Perov, Daiheng Gao, Nikolay Chervoniy, Kunlin Liu, Sugasa Marangonda, Chris Umé, Mr Dpfks, Carl Shift Facenheim, Luis RP, Jian Jiang, et al. Deepfacelab: A simple, flexible and extensible face swapping framework. *arXiv preprint arXiv:2005.05535*, 2020.

[107] Faceswap-gan. https://github.com/shaoanlu/faceswap-GAN. Accessed: 2021-11-15.

[108] Stanislav Pidhorskyi, Donald A Adjeroh, and Gianfranco Doretto. Adversarial latent autoencoders. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14104–14113, 2020.

[109] Yujun Shen, Ceyuan Yang, Xiaoou Tang, and Bolei Zhou. Interfacegan: Interpreting the disentangled face representation learned by gans. *IEEE Transactions on pattern analysis and machine intelligence*, 2020.

[110] Ajian Liu, Jun Wan, Sergio Escalera, Hugo Jair Escalante, Zichang Tan, Qi Yuan, Kai Wang, Chi Lin, Guodong Guo, Isabelle Guyon, et al. Multi-modal face anti-spoofing attack detection challenge at cvpr2019. In *Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 0–0, 2019.

[111] Leland McInnes, John Healy, Nathaniel Saul, and Lukas Großberger. UMAP: Uniform manifold approximation and projection. *Journal of Open Source Software*, 3(29):861, 2018.

[112] Ralph Gross, Iain Matthews, Jeffrey Cohn, Takeo Kanade, and Simon Baker. Multi-PIE. *Image and Vision Computing*, 28(5):807–813, 2010.

[113] A. Anjos, M. Günther, T. de Freitas Pereira, P. Korshunov, A. Mohammadi, and S. Marcel. Continuously reproducing toolchains in pattern recognition and machine learning experiments. In *ICML*, August 2017.

[114] Gary B. Huang Erik Learned-Miller. Labeled faces in the wild: Updates and new reporting procedures. Technical Report UM-CS-2014-003, University of Massachusetts, Amherst, May 2014.

[115] Gary B. Huang, Vidit Jain, and Erik Learned-Miller. Unsupervised joint alignment of complex images. In *ICCV*. IEEE, 2007.

[116] Christopher McCool, Sebastien Marcel, Abdenour Hadid, Matti Pietikäinen, Pavel Matejka, Jan Cernockỳ, Norman Poh, Josef Kittler, Anthony Larcher, Christophe Levy, et al. Bi-modal person recognition on a mobile phone: using mobile phone data. In *ICMEW*, pages 635–640. IEEE, 2012.

[117] Brendan F Klare, Ben Klein, Emma Taborsky, Austin Blanton, Jordan Cheney, Kristen Allen, Patrick Grother, Alan Mah, and Anil K Jain. Pushing the frontiers of unconstrained face detection and recognition: IARPA Janus benchmark a. In *CVPR*, pages 1931–1939, 2015.

[118] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[119] Patrick Grother, Mei Ngan, and Kayee Hanaoka. *Face Recognition Vendor Test (FVRT): Part 3, Demographic Effects.* National Institute of Standards and Technology, 2019.

[120] Francesco Marra, Cristiano Saltori, Giulia Boato, and Luisa Verdoliva. Incremental learning for the detection and classification of GAN-generated images. In *WIFS*, pages 1–6. IEEE, 2019.

[121] Ning Yu, Larry S Davis, and Mario Fritz. Attributing fake images to GANs: Learning and analyzing GAN fingerprints. In *ICCV*, pages 7556–7566, 2019.

[122] Nils Hulzebosch, Sarah Ibrahimi, and Marcel Worring. Detecting CNN-generated facial images in real-world scenarios. In *CVPR Workshops*, pages 642–643, 2020.

[123] Ruben Tolosana, Ruben Vera-Rodriguez, Julian Fierrez, Aythami Morales, and Javier Ortega-Garcia. Deepfakes and beyond: A survey of face manipulation and fake detection. *Information Fusion*, 64:131–148, 2020.

[124] Pavel Korshunov and Sébastien Marcel. Vulnerability assessment and detection of deepfake videos. In *International Conference on Biometrics (ICB)*, 2019.

[125] Huy H Nguyen, Junichi Yamagishi, and Isao Echizen. Capsule-forensics: Using capsule networks to detect forged images and videos. In *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2307–2311. IEEE, 2019.

[126] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic routing between capsules. In *Conference on Neural Information Processing Systems (NIPS)*, 2017.

[127] Geoffrey E Hinton, Alex Krizhevsky, and Sida D Wang. Transforming auto-encoders. In *International Conference on Artificial Neural Networks (ICANN)*. Springer, 2011.

[128] Geoffrey E Hinton, Sara Sabour, and Nicholas Frosst. Matrix capsules with EM routing. In *International Conference on Learning Representations Workshop (ICLRW)*, 2018.

[129] Edgar Xi, Selina Bing, and Yang Jin. Capsule network performance on complex data. *arXiv preprint arXiv:1712.03480*, 2017.

[130] Canqun Xiang, Lu Zhang, Yi Tang, Wenbin Zou, and Chen Xu. Ms-capsnet: A novel multi-scale capsule network. *IEEE Signal Processing Letters*, 25(12):1850–1854, 2018.

[131] Mohammad Taha Bahadori. Spectral capsule networks. In *International Conference on Learning Representations (ICLR)*, 2018.

[132] Nicolas Rahmouni, Vincent Nozick, Junichi Yamagishi, and Isao Echizen. Distinguishing computer graphics from natural images using convolution neural networks. In *International Workshop on Information Forensics and Security (WIFS)*. IEEE, 2017.

[133] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

[134] Utku Ozbulak. Pytorch cnn visualizations. https://github.com/utkuozbulak/pytorch-cnn-visualizations, 2019.

[135] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *International Conference on Computer Vision (ICCV)*, pages 618–626. IEEE, 2017.

[136] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning (ICML)*, pages 6105–6114, 2019.

[137] Arsha Nagrani, Joon Son Chung, Weidi Xie, and Andrew Zisserman. Voxceleb: Large-scale speaker verification in the wild. *Computer Speech & Language*, 60: 101027, 2020.

[138] Conrad Sanderson. The vidtimit database. Technical report, IDIAP, 2002.

[139] Weili Nie, Nina Narodytska, and Ankit Patel. RelGAN: Relational generative adversarial networks for text generation. In *International conference on learning representations*, 2018.

[140] Jawadul H Bappy, Amit K Roy-Chowdhury, Jason Bunk, Lakshmanan Nataraj, and BS Manjunath. Exploiting spatial structure for localizing manipulated image regions. In *International Conference on Computer Vision (ICCV)*, pages 4970–4979, 2017.

[141] Davide Cozzolino, Justus Thies, Andreas Rössler, Christian Riess, Matthias Nießner, and Luisa Verdoliva. Forensictransfer: Weakly-supervised domain adaptation for forgery detection. *arXiv preprint arXiv:1812.02510*, 2018.

[142] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, pages 448–456, 2015.

[143] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, pages 807–814, 2010.

[144] Ruoyu Wu, Xiaolong Li, and Bin Yang. Identifying computer generated graphics via histogram features. In *ICIP*, pages 1933–1936. IEEE, 2011.

[145] Fei Peng, Die-lan Zhou, Min Long, and Xing-ming Sun. Discrimination of natural images and computer generated graphics based on multi-fractal and regression analysis. *AEU-International Journal of Electronics and Communications*, 71:72–81, 2017.

[146] Huy H Nguyen, Hoang-Quoc Nguyen-Son, Thuc D Nguyen, and Isao Echizen. Discriminating between computer-generated facial images and natural ones using smoothness property and local entropy. In *International Workshop on Digital Watermarking*, pages 39–50. Springer, 2015.

[147] Ling Huang, Anthony D Joseph, Blaine Nelson, Benjamin IP Rubinstein, and J Doug Tygar. Adversarial machine learning. In *Workshop on Security and Artificial Intelligence*, pages 43–58. ACM, 2011.

[148] Yoshua Bengio et al. Learning deep architectures for AI. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.

[149] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[150] Wenzhe Shi, Jose Caballero, Ferenc Huszár, Johannes Totz, Andrew P Aitken, Rob Bishop, Daniel Rueckert, and Zehan Wang. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *CVPR*, pages 1874–1883, 2016.

[151] Michael Mathieu, Camille Couprie, and Yann LeCun. Deep multi-scale video prediction beyond mean square error. *arXiv preprint arXiv:1511.05440*, 2015.

[152] Alexey Dosovitskiy and Thomas Brox. Generating images with perceptual similarity metrics based on deep networks. *Advances in neural information processing systems (NIPS)*, 29:658–666, 2016.

[153] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. *Computing Research Repository (CoRR)*, 2016.

[154] Duc-Tien Dang-Nguyen, Giulia Boato, and Francesco GB De Natale. Discrimination between computer generated and natural human faces based on asymmetry information. In *European Signal Processing Conference (EUSIPCO)*, pages 1234–1238. IEEE, 2012.

[155] Pascal Paysan, Reinhard Knothe, Brian Amberg, Sami Romdhani, and Thomas Vetter. A 3D face model for pose and illumination invariant face recognition. In *Advanced Video and Signal Based Surveillance*, pages 296–301. IEEE, 2009.

[156] M Weber. Caltech frontal face database. *California Institute of Technology*, 1999.

[157] Benjamin Weyrauch, Bernd Heisele, Jennifer Huang, and Volker Blanz. Component-based face recognition with 3D morphable models. In *CVPRW*, pages 85–85. IEEE, 2004.

[158] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *AsiaCCS*, pages 506–519. ACM, 2017.

[159] Siwei Lyu and Hany Farid. How realistic is photorealistic? *IEEE Transactions on Signal Processing*, 53(2):845–850, 2005.

[160] Wen Chen, Yun Q Shi, Guorong Xuan, and Wei Su. Computer graphics identification using genetic algorithm. In *International Conference on Pattern Recognition*, pages 1–4. IEEE, 2008.

[161] Wenxiang Li, Tao Zhang, Ergong Zheng, and Xijian Ping. Identifying photorealistic computer graphics using second-order difference statistics. In *International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, volume 5, pages 2316–2319. IEEE, 2010.

[162] Rong Zhang, Rang-Ding Wang, and Tian-Tsong Ng. Distinguishing photographic images and photorealistic computer graphics using visual vocabulary on local image edges. In *International Workshop on Digital Forensics and Watermarking (IWDW)*, pages 292–305. Springer, 2011.

[163] Zhenwei Chen and Yongzhen Ke. A novel photographic and computer graphic composites detection method. In *National Conference on Information Technology and Computer Science*. Atlantis Press, 2012.

[164] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European Conference on Computer Vision*, pages 694–711. Springer, 2016.

[165] Dennis W Ruck, Steven K Rogers, Matthew Kabrisky, Mark E Oxley, and Bruce W Suter. The multilayer perceptron as an approximation to a bayes optimal discriminant function. *IEEE Transactions on Neural Networks*, 1(4):296–298, 1990.

[166] Richard O Duda, Peter E Hart, David G Stork, et al. *Pattern classification*, volume 2. Wiley New York, 1973.

[167] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[168] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[169] Jimmy Ba, Volodymyr Mnih, and Koray Kavukcuoglu. Multiple object recognition with visual attention. In *International Conference on Learning Representations (ICLR)*, 2015.

[170] Rong Huang, Fuming Fang, Huy H Nguyen, Junichi Yamagishi, and Isao Echizen. Security of facial forensics models against adversarial attacks. In *International Conference on Image Processing (ICIP)*, pages 2236–2240. IEEE, 2020.

[171] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. In *ILCR-W*, 2017.

[172] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Florian Tramer, Atul Prakash, Tadayoshi Kohno, and Dawn Song. Physical adversarial examples for object detectors. In *WOOT*, 2018.

[173] Chawin Sitawarin, Arjun Nitin Bhagoji, Arsalan Mosenia, Mung Chiang, and Prateek Mittal. Darts: Deceiving autonomous cars with toxic signs. *arXiv preprint arXiv:1802.06430*, 2018.

[174] Adith Boloor, Xin He, Christopher Gill, Yevgeniy Vorobeychik, and Xuan Zhang. Simple physical adversarial examples against end-to-end autonomous driving models. *arXiv preprint arXiv:1903.05157*, 2019.

[175] Lea Schönherr, Katharina Kohls, Steffen Zeiler, Thorsten Holz, and Dorothea Kolossa. Adversarial attacks against automatic speech recognition systems via psychoacoustic hiding. In *NDSS*, 2019.

[176] Han Xu, Yao Ma, Haochen Liu, Debayan Deb, Hui Liu, Jiliang Tang, and Anil Jain. Adversarial attacks and defenses in images, graphs and text: A review. *arXiv preprint arXiv:1909.08072*, 2019.

[177] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. In *CVPR*, pages 1765–1773, 2017.

[178] Cihang Xie, Zhishuai Zhang, Yuyin Zhou, Song Bai, Jianyu Wang, Zhou Ren, and Alan L Yuille. Improving transferability of adversarial examples with input diversity. In *CVPR*, pages 2730–2739, 2019.

[179] Hengtong Zhang, Tianhang Zheng, Jing Gao, Chenglin Miao, Lu Su, Yaliang Li, and Kui Ren. Data poisoning attack against knowledge graph embedding. In *IJCAI*, 2019.

[180] Kathrin Grosse, Praveen Manoharan, Nicolas Papernot, Michael Backes, and Patrick McDaniel. On the (statistical) detection of adversarial examples. *arXiv preprint arXiv:1702.06280*, 2017.

[181] Xin Li and Fuxin Li. Adversarial examples detection in deep networks with convolutional filter statistics. In *CVPR*, pages 5764–5772, 2017.

[182] Zhitao Gong, Wenlu Wang, and Wei-Shinn Ku. Adversarial and clean data are not twins. *arXiv preprint arXiv:1704.04960*, 2017.

[183] Jan Hendrik Metzen, Tim Genewein, Volker Fischer, and Bastian Bischoff. On detecting adversarial perturbations. In *ICLR*, 2017.

[184] Kimin Lee, Kibok Lee, Honglak Lee, and Jinwoo Shin. A simple unified framework for detecting out-of-distribution samples and adversarial attacks. In *NIPS*, pages 7167–7177, 2018.

[185] Xingjun Ma, Bo Li, Yisen Wang, Sarah M Erfani, Sudanthi Wijewickrema, Grant Schoenebeck, Dawn Song, Michael E Houle, and James Bailey. Characterizing adversarial subspaces using local intrinsic dimensionality. In *ICLR*, 2018.

[186] Weilin Xu, David Evans, and Yanjun Qi. Feature squeezing: Detecting adversarial examples in deep neural networks. In *NDSS*, 2018.

[187] Bin Liang, Hongcheng Li, Miaoqiang Su, Xirong Li, Wenchang Shi, and XiaoFeng Wang. Detecting adversarial image examples in deep neural networks with adaptive noise reduction. *IEEE Transactions on Dependable and Secure Computing*, 2018.

[188] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *SP*, pages 582–597. IEEE, 2016.

[189] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *ICLR*, pages 274–283, 2018.

[190] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *ICLR*, 2018.

[191] Tianyu Pang, Chao Du, Yinpeng Dong, and Jun Zhu. Towards robust detection of adversarial examples. In *NeurIPS*, pages 4579–4589, 2018.

[192] Chuan Guo, Mayank Rana, Moustapha Cisse, and Laurens Van Der Maaten. Countering adversarial images using input transformations. In *ICLR*, 2018.

[193] Aaditya Prakash, Nick Moran, Solomon Garber, Antonella DiLillo, and James Storer. Protecting jpeg images against adversarial attacks. In *Data Compression Conference*, pages 137–146. IEEE, 2018.

[194] Olga Taran, Shideh Rezaeifar, Taras Holotyak, and Slava Voloshynovskiy. Machine learning through cryptographic glasses: combating adversarial attacks by key-based diversified aggregation. *EURASIP journal on information security*, 2020: 1–18, 2020.

[195] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

[196] Jonas Rauber, Wieland Brendel, and Matthias Bethge. Foolbox: A python toolbox to benchmark the robustness of machine learning models. *arXiv preprint arXiv:1707.04131*, 2017. URL `http://arxiv.org/abs/1707.04131`.

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

[198] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *CVPR*, pages 2574–2582, 2016.

[199] Uyeong Jang, Xi Wu, and Somesh Jha. Objective metrics and gradient descent algorithms for adversarial examples in machine learning. In *ACSAC*, pages 262–277. ACM, 2017.

[200] Rima Alaifari, Giovanni S Alberti, and Tandri Gauksson. ADef: an iterative algorithm to construct adversarial deformations. In *ILCR*, 2019.

[201] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *EuroS&P*, pages 372–387. IEEE, 2016.

[202] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *SP*, pages 39–57. IEEE, 2017.

[203] Alain Hore and Djemel Ziou. Image quality metrics: Psnr vs. ssim. In *ICPR*, pages 2366–2369. IEEE, 2010.

[204] Tin Kam Ho. Random decision forests. In *ICDAR*, pages 278–282. IEEE, 1995.

[205] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning.* MIT press, Cambridge, Massachusetts, 2016.