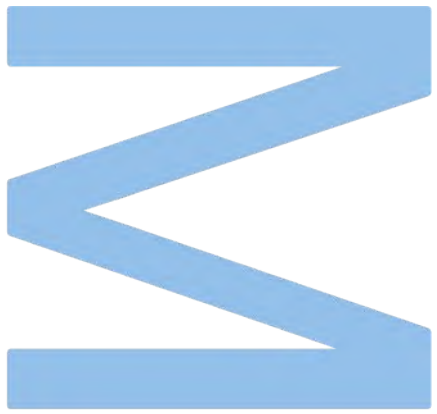
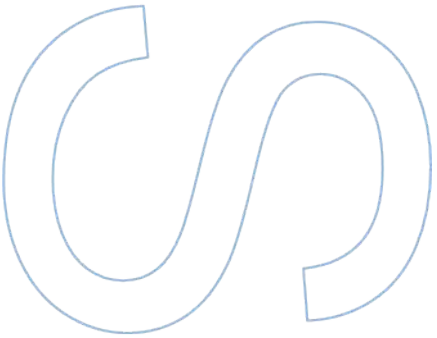
Deepfake Generation for use in Dictionary Attacks on Facial Recognition Systems

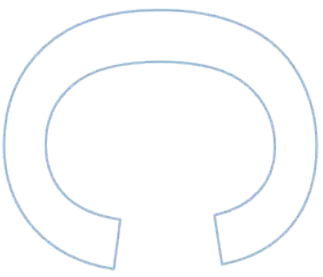


Vasco Mucha Barros
Mestrado em Engenharia de Redes e Sistemas Informáticos
Departamento de Ciência de Computadores
2023

Orientador
Mário João Gonçalves Antunes, Professor Coordenador, Escola Superior de Tecnologia e Gestão do Politécnico de Leiria

Coorientador
Manuel Eduardo Carvalho Duarte Correia, Professor Associado, Faculdade de Ciência da Universidade do Porto






U.PORTO
FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Abstract

Biometric authentication, although not new, is seeing an increase in popularity and usage in current times, particularly facial recognition. At the same time, image manipulation technology is also booming, with software becoming accessible to virtually anyone who has an above-average machine. Deep learning, though deepfakes, is already a cause of concern due to its usage in manipulated content such as fake news, in an era of digital information. As facial recognition becomes more prominent in people's daily lives, the chances for deepfakes to be used maliciously in that another field should not be ignored.

In order to analyze the extents of the vulnerabilities of these systems, we propose training a Generative Adversarial Network that generates a particular type of deepfake, and then using those fakes, testing whether or not various systems recognize them as real faces and furthermore whether they can or not breach the systems' security. The reasoning for the specific particularity of the fakes is discussed further ahead.

In addition to that, we discuss the concerning results produced by some of the previous works on this topic, as well as how those could still be taken even further if different attacking angles are combined into a single attack. To contrast, we also talk about anti-deepfake tools and how they can prove to be crucial to combat these new type of threats to facial authentication systems.

Resumo

Autenticação biométrica, embora não sendo algo novo, está a aumentar em termos de popularidade e de utilização nos tempos que correm, em particular o reconhecimento facial. Ao mesmo tempo, tecnologia de manipulação de imagem está também a crescer, com software a ficar acessível a praticamente qualquer indíviduo com uma máquina acima da média. *Deep Learning*, através dos *deepfakes*, já causa preocupação devido ao seu uso na criação de conteúdo manipulado como as *fake news*, nesta era de informação digital. À medida que o reconhecimento facial se torna mais proeminente na vida diária das pessoas, as chances de que os *deepfakes* venham a ser usados maliciosamente nesse campo não devem ser ignoradas.

De modo a analisar até que ponto estes sistemas estão vulneráveis, propomos o treino de uma Generative Adversarial Network que gere um tipo particular de *deepfake*, e que esses "falsos" sejam usados para testar se vários sistemas os reconhecem ou não como faces reais e até mesmo se são capazes ou não de penetrar a segurança dos sistemas. O motivo da particularidade destes *deepfakes* é discutido mais à frente.

Além disso, discutimos os resultados preocupantes obtidos por alguns trabalhos prévios neste tópico, além do facto de que estes podem ainda ser mais potentes caso diferentes ângulos de ataque sejam combinados num só ataque. Para contrastar, falamos também sobre ferramentas *anti-deepfake* e como estas se podem mostrar cruciais para combater este tipo de ameaças aos sistemas de autenticação facial.

Agradecimentos

Obrigado aos meus orientadores, prof. Mário Antunes e prof. Eduardo Correia, por me acompanharem ao longo deste período e por estarem sempre disponíveis.

Obrigado à minha família por me ter apoiado durante todo o meu percurso académico e por não ter deixado de me "chatear" para terminar esta etapa final.

Obrigado aos meus colegas, especialmente ao Afonso, ao Gustavo e à Maria por toda a ajuda e toda a companhia.

E obrigado à Carolina por me suportar incondicionalmente, suporte que me fez continuar quando o caminho ficava mais complicado.

Contents

List of Tables

List of Figures

Acronyms

AI Artificial Intelligence

ANN Artificial Neural Network

CNN Convolutional Neural Network

DFL DeepFakeLab

DL Deep Learning

FD Face Detection

FR Face Recognition

GAN Generative Adversarial Network

HOG Histogram of Oriented Gradients

ML Machine Learning

MTCNN Multi-tasking Cascaded Convolutional Neural Network

S3FD Single Shot Scale-invariant Face Detector

Chapter 1

Introduction

Since the inception of the concept of private property, humankind has looked for ways to protect its belongings from unauthorized access. Mechanical locks are everywhere in our society, from our houses to our cars. Additionally, most services rely on some form of authorization mechanism to confirm the identity claimed by the entity trying to access it. Be it passwords, ID cards, there are various ways to perform this verification.

The field of security has been constantly evolving. Password technology hasn't stayed the same, and neither did smart cards and other similar devices. However, along with the evolution of these more traditional means, recently a new "competitor" as been gaining traction in both physical and digital security. Biometrics is a field that is not new. Its appearance in the field of law enforcement dates back at least a century. The developments in the area are making it more widely accessible, and while their use creates an ongoing debate, the opinions tend to be more positive than negative [30].

Given these advances, and the fact that the COVID pandemic pushed businesses and services into going digital, which in turn caused a boom in cybercrime, it is normal to see biometrics, which is deemed to be more safe than the "traditional" means of authentication, being more widely adopted and deployed. For example, it has gained the attention of big industries like banks [23], and is even being used by governments, such as the Portuguese one, as a mean of access to official documents [35].

But not only are biometrics getting popular due to their strong security capabilities, another factor for this is their usability. We can see an example of this in smartphones, where although not fully replacing the PIN code functionality, almost all new models incorporate a fingerprint reader, which instantly unlocks the phone. Apple takes this one step further by using face recognition (FR) technology not only for unlocking the phone, but also for things such as authorizing payments [1].

Biometrics are expected to keep growing in the future, with projections pointing towards the revenue of companies in the field nearly duplicating in the near future (see Figure 1.1).

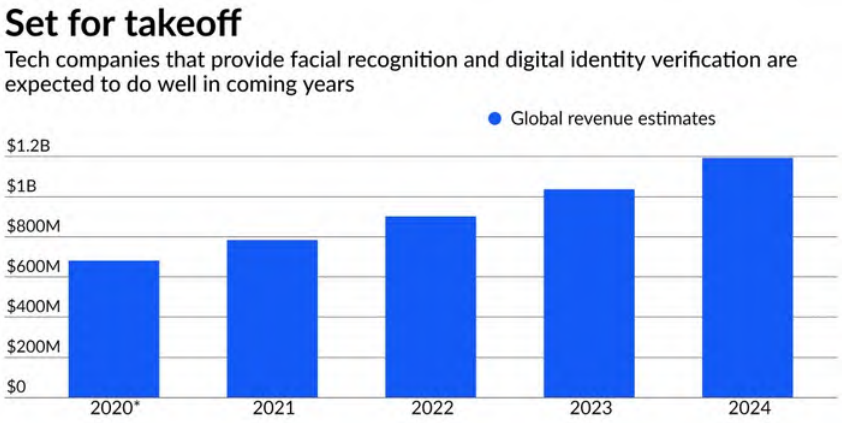


Figure 1.1: Estimated evolution of revenue for companies in the biometrics market [23]

However, in the natural fashion of security work, when the defenses advance, so do the attacks to match it.

Machine Learning (ML), more specifically Deep Learning (DL), has come so far to the point where it can create seemingly perfect images of human faces, as shown in Figure 1.2. The quality is so high that it is unfeasible for a person to tell that the person in the image is not a real one, but instead a creation from a Generative Adversarial Network (GAN).

Figure 1.2: Face generated by a GAN [2]

GANs are so powerful that not only are they capable of fooling humans, they have also been used to "fool" biometric authentication systems. Both facial authentication and fingerpint authentication have been shown to be able to be compromised using "master faces"/"master prints" [12][34]. These terms derive from the commonly known "master key", the key which opens all doors. In a similar fashion, master faces and master prints are capable of breaking the respective authentication systems and giving attackers access to things they shouldn't have.

In this work, we use GANs to produce fake images, and then proceed to use a face detector with FR capabilities to first check if our fakes pass as human faces, and then analyze whether or not they bypass the recognition step. We will take a different approach by training the GAN with modified, anatomically impossible faces. Given that all of the work already developed on this topic uses regular human faces, we hope to explore the response of facial authentication systems when presented with the elements of a regular face, but not in the normal location and/or number. This is especially pertinent given the fact that these systems don't see human faces in the same way that we do [20], which means this distorted faces that no sane person would identify as human may as well be recognized by the system as such.

By taking this route, we hoped to not only study the way in that FR deals with this type of images, but also to be able to analyze any possible abnormal behaviour that could end up showing up. Also, in the case of our fakes successfully tricking the system, we could observe patterns in the different faces and study their possible correlation to the phenomenon.

1.1 Main goals

The goals of the research work in this thesis can be summarized by the following:

- To train a system capable of automatically mass generating different faces, which while containing the elements of a human face, do not look like real faces.
- To evaluate whether or not face detection (FD) systems identify this type of faces as regular human faces.
- To use the faces that pass the FD check to analyze the behaviour of FR systems when presented with this type of faces.

1.2 Contributions

The main contributions of this work can be considered to be the following:

- A dataset containing manipulated versions of regular human faces.
- A generator capable of creating multiple deepfakes.
- A pipeline that can be used to test the deepfakes against various face detection and recognition systems.

1.3 Thesis layout

The layout for this thesis can be described in the following manner:

- **Chapter 1: Introduction**

  The initial chapter of the thesis, which covers the motivation for the work developed as well as topics such goals, contributions and the way the document is organized.

- **Chapter 2: Fundamentals**

  This section aims to cover relevant concepts and information about topics which are directly related with the work being developed. It will also contain information about relevant state-of-the-art applications in the area.

- **Chapter 3: Proposed architecture**

  In this section we intend to present the overall architecture of our work, as well as to address the steps of the development process.

- **Chapter 4: Results**

  In this section we will be presenting the results of our work, and consequentially discussing them.

- **Chapter 5: Conclusions**

  Finally, this last chapter is intended for discussion of the work done, how well the objectives were achieved and what went wrong for the ones who didn't. We also discuss possibilities for future work related to this thesis.

Chapter 2

Fundamentals

In this chapter we aim to provide the reader with some background on the topics covered and technologies used in our project, so that they can better understand it. In the following sections you can read about biometrics, image manipulation and deep learning.

2.1 Biometrics

Like the name - deriving from the ancient Greek *bios* (life) and *metron* (measure) - might imply, biometrics refer to biological characteristics of an individual. This means the collected and used information will be something the person in question *is*, in contrast to something the person knows or has. This differentiation intends to provide more reliability and robustness to the use cases [46].

Biometrics can be used either to perform one-to-one correspondence - authentication - or one-to-many correspondence - identification. Identification is of more use to organizations like police or security agencies, while authentication is more widely used, due to being used in regular services such as banking. We will be focusing on the latter.

While nowhere near the level that we have today, techniques using biometrics have been observed in the past; it became common for documents to be signed with ink applied with the skin of the signer after the invention of paper on the 1st century in China [10]. Another great example of this is the Bertillon method, developed by the french police officer Alphonse Bertillon in the 19th century, where different types of body measurements were used to identify criminals [18]. However, the problem with this method and the reason it was so short lived was that due to the type of measurements used, it didn't take long before it started resulting in a lot of false positive identifications, which brings us to the essential point that the metrics used must be sufficiently **distinctive** between individuals. Another important point is that the metric used remains roughly unchanged throughout the span of the user's life - or at least the span a user is expected to use a certain service. To this we call **permanence** [22].

Many individual characteristics fit the criteria above, but some end up being more common than others due to factors such as difficulty/cost of implementation and error rate. The most popular ones are fingerprint recognition and FR, and while we will focus on the latter due to the nature of this work, they both share the same working principal of matching a sample provided by the one attempting authentication with the data stored, relative to the person they are trying to authenticate as.

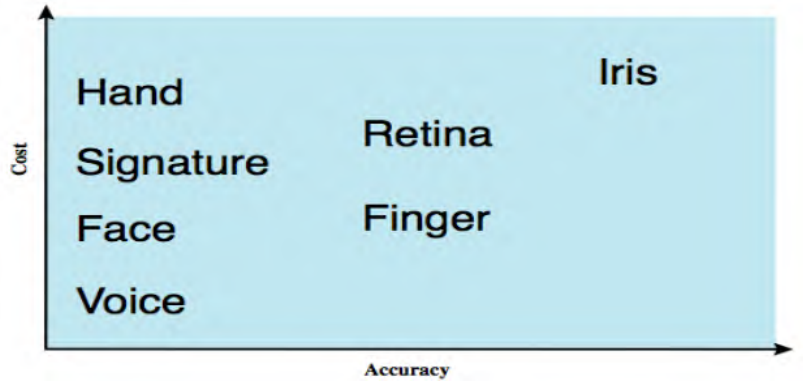


Figure 2.1: Cost versus accuracy for various biometric authentication methods [33]

2.1.1 Facial Recognition/Authentication

Contrary to some other forms of biometrics that end up using expensive equipment, facial authentication only needs a camera, something overly common in the days we live in as virtually everyone has access to one in their smartphone and/or computer. This, coupled with the fact that this is not an intrusive method and one that works in all sorts of environments with different conditions, can be seen as the factor for the system's popularity [36].

Generally, a FR system is divided into four phases [25]. Its flow can be observed in Figure 2.2, and consists of the following modules:

- **Face and Landmark Localization**

  In this step, the system extracts first the face from the image or video (in that case, over multiple frames) and then the relevant facial landmarks (such as eyes, mouth and nose location) from the face.

- **Face Normalization**

  In this step, faces are subjected to various possible transformations in order to ensure that it fits into a standard frame. This ensures that systems are capable of recognizing faces under a diversity of conditions.

- **Feature Extraction**

  In this step, the system extracts the information relevant for the next step from the normalized face. It's relevant to note that this information - features - is different from the

information extracted from our original face in the first step - landmarks - in the way that feature extraction returns values that are used in comparisons, while landmark extraction generally returns values that represent the location of specific attributes of a face, which is used in the normalization step.

- **Feature Matching**

  In this step, the face(s) extracted previously are compared to the ones present in the system's enrollment database using the features obtained in the previous step. There will either be 1-to-1 or 1-to-many checks, and 'yes' or 'no' will be returned in the first case while an identity (or lack of) will be returned in the latter.

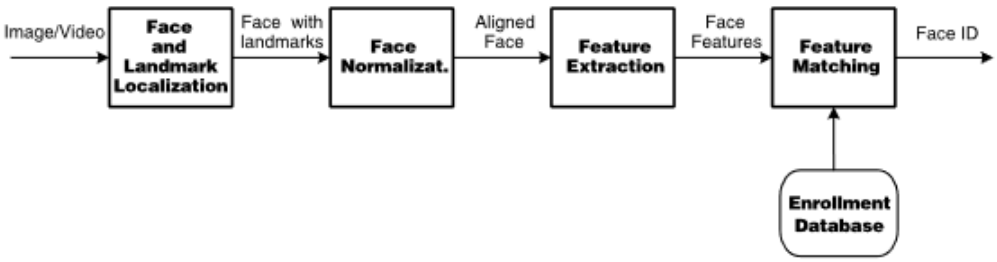


Figure 2.2: Face recognition processing flow [25]

Obviously, the pipeline described above is more of a generalized template than anything else. It provides no actual detail about concrete implementations of any of the modules. So, in the following section, we will be taking a closer look at some possible examples - which, when put together would very well work as a proper FR system.

2.1.1.1 Histogram of Oriented Gradients

Histogram of Oriented Gradients (HOG) is a method used in FD, and can be characterized by the following steps [17]:

1. Converting the picture to black and white.
2. For each pixel, analyzing the respective four surrounding pixels, and in accordance to that drawing an arrow in the direction of which the picture gets darker.
3. Dividing the picture in 16x16 segments, and in each segment counting the total number of arrows pointing in each direction, in order to then replace the 16x16 segment with an arrow pointing to the most represented direction.
4. Comparing the obtained pattern of arrows with a known pattern generated from training with a multitude of faces.

Step 1. and 2. are necessary due to the fact for the same person, two pictures with different brightness levels can have a very different representation if we consider pixel values as the metric. However, when we replace the pixels with gradients (the arrow showing the direction of the change in brightness), we get a similar representation for both pictures.

Step 3. is responsible for making sure having too much detail is avoided, as this can bring issues. By replacing the individual gradients with the predominant gradient in a determined area, we can get a simple representation of the image which still keeps the necessary detail for the next step.

Step 4. is the final step and is when we use what we got from the previous ones to finally check if we have a face or not. By comparing an existing HOG face pattern (which has been generated by applying the previous steps to various images which we know have faces present) to the HOG version of our image, we will be able to identify a specific region (the detection window) which will be very similar to the pattern - that's where the face will be.

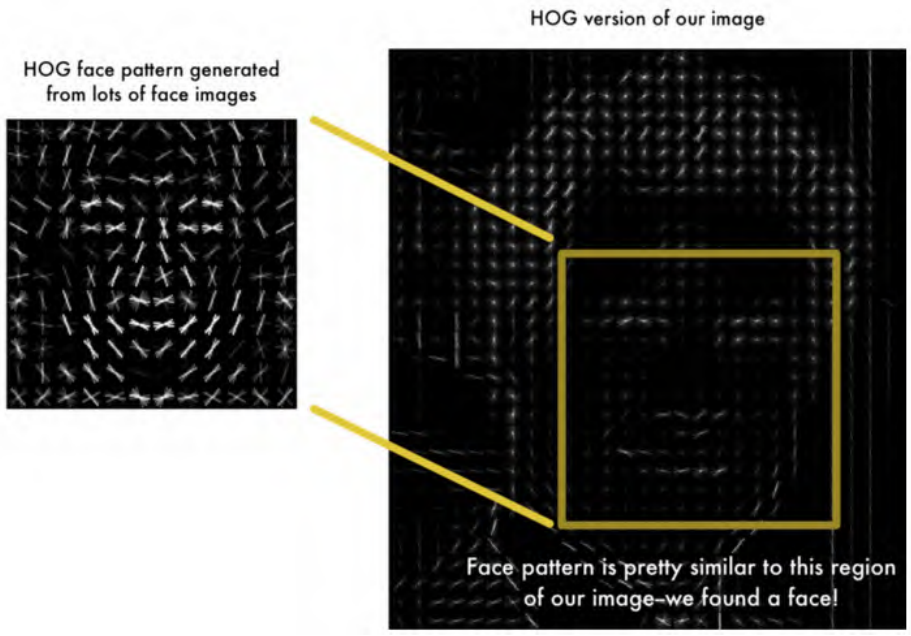


Figure 2.3: HOG methodology for finding faces in images [20]

2.1.1.2 Face Landmark Estimation and Face Alignment

The approach described here is a state-of-the-art method for face landmark estimation, and consequentially, face alignment/normalization. Since the transformations done to the face in this process are simple ones, such as rotating or scaling, the most crucial aspect for face alignment to be done correctly ends up being the proper determining of the landmarks, which are responsible for guiding the whole process.

The algorithm uses a cascade of regressors to detect 194 points in a face, which correspond

to the desired landmarks [28]. The regression functions will estimate these points based on an initial estimate and the intensities of a set of pixels relative to it, based on the optimization of a loss function via gradient boosting. There are two key elements incorporated into the learning, which are:

- Indexing of pixel intensities relative to the current estimate of the shape - to predict the shape, the algorithm needs accurate features of the face. However, to extract these accurate features, the estimation of the shape needs be reliable. To address this, the image is transformed according to the given estimate of the shape. The features are extracted from this new representation, and then used to update the shape estimation. This iteractive process is then repeated until the end result is achieved.
- Combating the difficult problem of inference - given the nature of the problem of estimating the shape of a face, it is necessary to avoid local optima solutions. The algorithm addresses this by upholding the assumption that the desired shape must exist in a linear subspace. With this assumption, the possible number of shapes to be considered during inference is reduced greatly, which in turns diminishes the chances of local optima results.

Figure 2.4: Example of landmarks detected in some faces [28]

2.1.1.3 FaceNet

FaceNet [40] is a system developed by researchers at Google that can be used for both feature extraction and face matching. The key part of this method is the production of Euclidean embeddings for each image, which then are used to calculate the Euclidean distance between different images, which in turn is used as the metric for assessing matches.

For the purpose of feature extraction, FaceNet trains a Deep Convolutional Neural Network, in a cycle that has two main steps:

- Initially, the network determines the measurements for each image in its training dataset. Its crucial that this batch has a great amount of pictures of different persons, as well as more than one picture of each individual identity.
- Following that, three images from among those used will be selected: the 'anchor', the 'positive', which corresponds to a different picture of the same identity as the anchor, and the 'negative', which is a picture of a different identity than that of the anchor. The model will then look at the embeddings it generated for each picture, and make changes in the network in order to maximize the Euclidean distance between the anchor and the negative while minimizing it between the anchor and the positive. This is referred to by the authors as "Triplet Loss".

By constantly iterating between both phases while continuously increasing the difficulty of the Triplet Loss, the final result will be a network capable of determining accurate embeddings for any image.

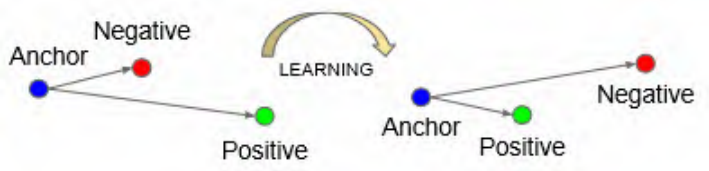


Figure 2.5: Visualization of the Triplet Loss process [40]

Face matching on the other hand is done in a much more straightforward way. The network provides the embeddings of both faces, and the distance is calculated, which is then compared to a designated threshold. If the value is bellow it, then the images correspond to the same person.

2.1.2 Biometric Authentication Attacks

Attacks on biometric authentication systems are classified according to the different part of the system exploited. Typically, there are eight vulnerable points, with each of the specified attacks referring to one of them in particular [26]. These are:

- **Type 1 - Sensor Module Attack**

  This type of attack involves the sensor of the system, which captures the biometric data, and is carried on by feeding synthetic attributes to it or by damaging it to the point it malfunctions and bloats the system with corrupted information. This kind of attack is also known as direct attack, as it requires no prior knowledge of the system and the sensor is in general the least protected part of it.

- **Type 2 - Man-in-the-Middle Attack - Sensor Module**

This attack involves stealing the data of a enrolled face to later use it by intercepting the communication channel between the sensor module and the feature extraction module and storing the data being transmitted.

- **Type 3 - Feature Extractor Attack**

  This attack consists in tampering with the feature extraction module, in order to make it produce values chosen by the attacker instead of the correct ones regarding the data obtained.

- **Type 4 - Man-in-the-Middle Attack - Feature Extractor**

  This attack is similar to the Type 2 attack in the way it's executed. It differs, however, at the moment of execution, with Type 4 targeting the communication channel between the feature extraction module and the matcher module.

- **Type 5 - Matcher Attack**

  This attack requires the attacker to tamper with the matcher module in order to have it produce false matching results for the given input, similarly to Type 3 attacks.

- **Type 6 - Database Attack**

  This attack is perpetrated by altering the enrollment database, adding new entries and/or removing or modifying existing ones, in order for the matcher to produce a positive result for the given input.

- **Type 7 - Entry Interception**

  This type of attack involves intercepting the communication channel between the enrollment database and the matcher module, and then tampering with the information being transmitted in order to benefit the attacker.

- **Type 8 - Result Interception**

  This attack targets the communication channel between the matcher module and the end application, with the attackers altering the real result given by the system.

2.1.2.1 Face Spoofing attacks

A face spoofing attack, or presentation attack, consists of feeding fake data to the system to wrongly bypass authentication [9]. This attack can be executed in three different ways:

- **Photo attack** - this is the simplest of the three, where the attacker presents a photo - generally, one of an enrolled subject - to the system, in an attempt of tricking it. A static picture can be used, but the attack is more effective when the eyes and the mouth are given some liveliness, either by animating it digitally or printing a photo and removing the eyes and the mouth to use it as a "mask" above the attacker's real face.

- **Video attack** - this can be considered an advanced version of the photo attack, since it is reproduced in the same manner but the use of video instead of a photo increases its effectiveness since there is actually movement involved.
- **Mask attack** - this attack is the most complex of the three, which in turn means it is the least common but most effective. To execute it, the attacker creates a mask based on a face on the system and then uses it to authenticate himself. By using a physical mask, the attacker is provided not only with live movement but also with depth on the face, and that additional factor is what provides the high effectiveness.

Photo and video attacks are classified as 2D spoofing, while the mask attack is considered to be 3D spoofing.

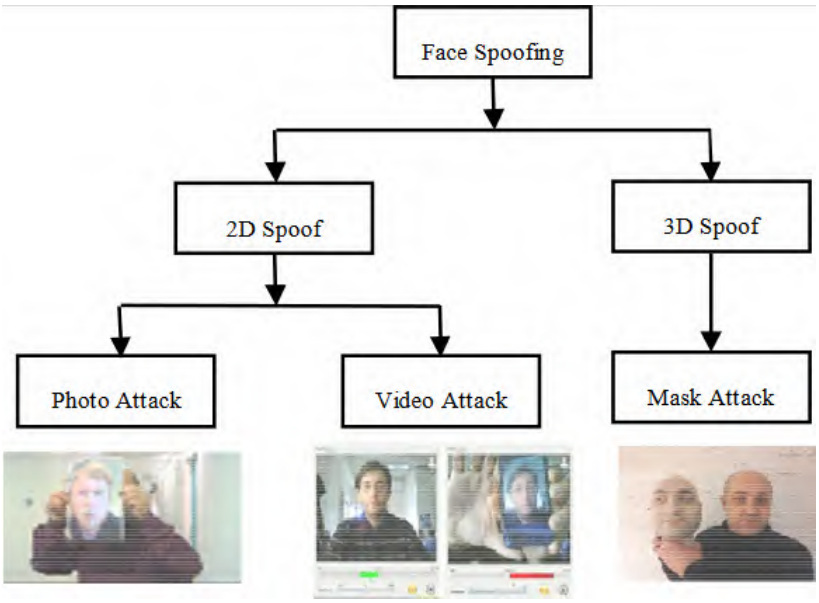


Figure 2.6: Types of face spoofing attacks [31]

Given the increase in popularity of face authentication systems, especially in contexts with no external supervision where attackers are free to get as creative as their attacks, and the fact that it is fairly easy to find pictures and videos of people online nowadays thanks to factors such as social media, spoofing attacks became an important issue to address and various techniques to detect and stop them have become the focus of many researchers, as it can be observed in Figure 2.7.

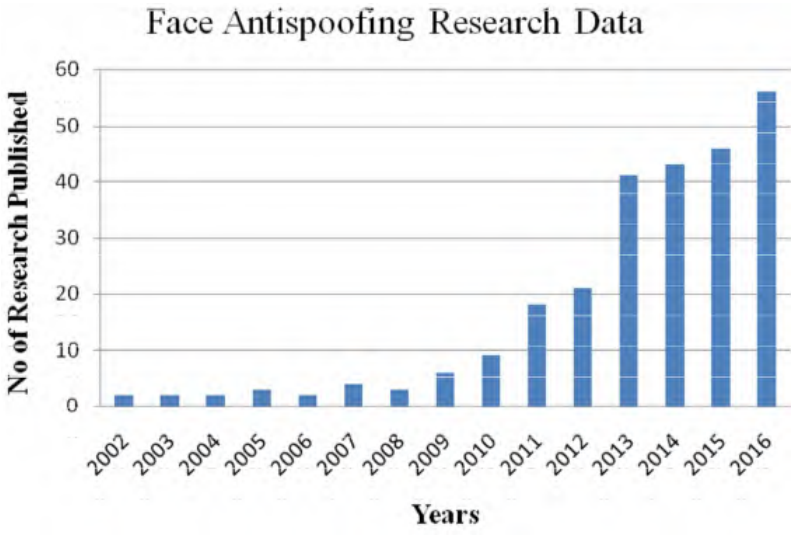


Figure 2.7: Increase in antispoofing research over the last few years [31]

2.2 Image Manipulation

Manipulation of images consists in the use of different techniques and tools to tamper with a picture. It can be done in ways that only slightly change the image but it can also give it a complete different meaning. Similarly, it can be done with harmless intentions, such as to create pieces of artwork, but it can also be used in malicious ways, such as tricking crime authorities [42].

Given the increase in publicly available tools, as well as advances at the technical levels of said tools, image manipulation has become a real issue. To add on to this problem, said advances are such that it has become extremely difficult for human subjects to identify said manipulations. This conjecture of factors lead to an increase in research of image manipulation techniques, as it can be observed in Figure 2.8, as ways to accurately detect the tampering of media content were deemed necessary to stop the potential problems associated with this.

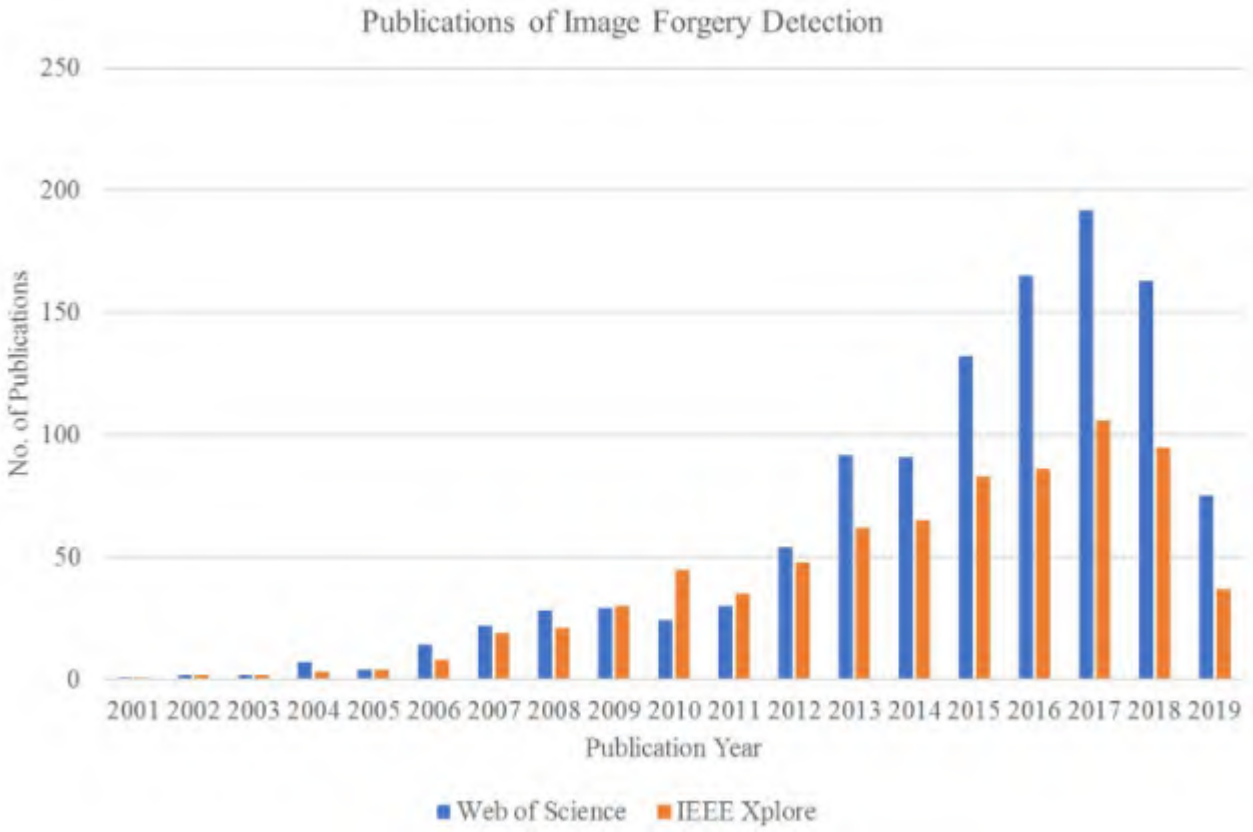


Figure 2.8: Increase in Image Forgery Detection over the last few years [42]

There are multiple techniques used for image manipulation, but these are the most common ones:

- **copy-move**: this technique consists in altering an image by copying one or more of its elements and pasting them in other locations of the picture. Given that the copied contents belong to the same picture they are being moved into makes this technique very appealing since making the alteration will be easier and it will be harder to tell. It is also worth noting that many image manipulation programs have pre-built functions for performing copy-move, which lowers even more the required skill level for the task.

Figure 2.9: Example of the copy-move technique. Original image on the left, tampered on the right [5]

- **splicing**: this technique also involves altering an image by copying elements into it, although unlike copy-move the elements come from a different picture than the original. This is an harder technique to properly execute, since the integration of components from a foreign source into an image brings problems such as irregularities and different contrasts in the corners, edges, and smooth regions of the external element. Given this fact, in order to achieve results similar to copy-move, splicing requires more post-processing and knowledge than the other method.

Figure 2.10: Example of the splicing technique. Original images above, tampered image below [8]

- **removal**: this technique involves removing part of the image, and after that applying a post-processing technique such as impainting to normalize the picture. Removal and copy-move can have similar goals, since by using copy-move to copy regions of the background of an image, for example, a similar result can be achieved by covering up elements of the picture instead of removing them directly.

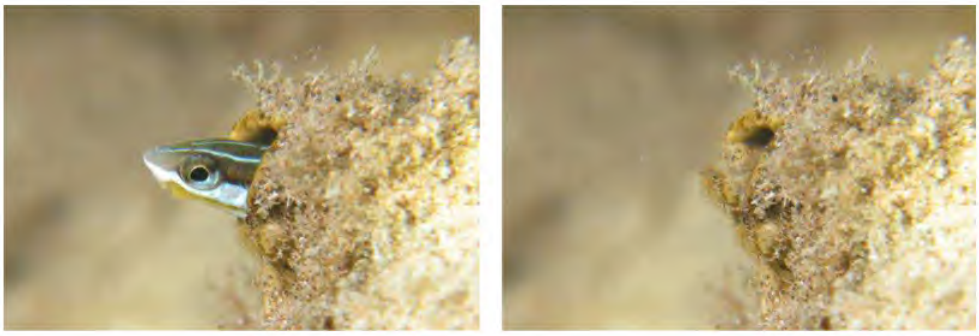


Figure 2.11: Example of the removal technique. Original image on the left, tampered on the right [48]

- **resampling**: this technique involves altering the image, or parts of it, using various types of geometric transformations. In contrast with the techniques discussed above, resampling is not usually used as to produce a final product, but is instead part of a bigger process.

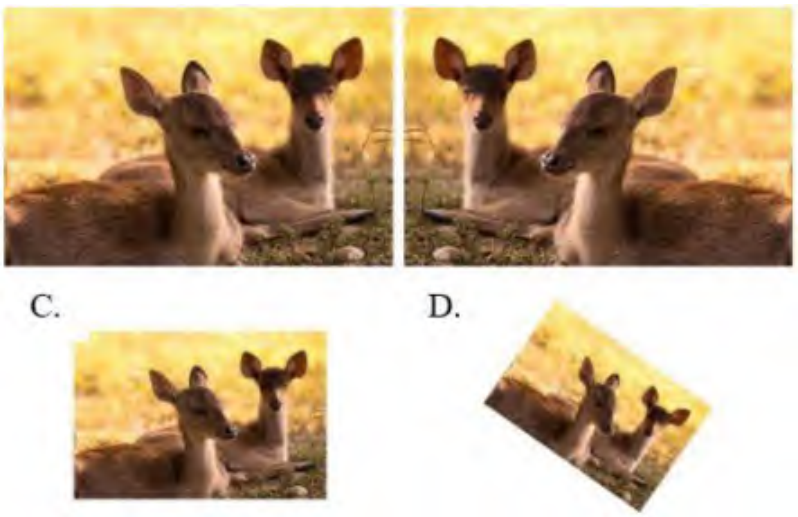


Figure 2.12: Example of the resampling technique. Original image top left, the rest are tampered images [6]

- **retouching**: this technique involves editing the attributes of an image, such as colors or saturation, in order to obtain an "higher quality" version of it. Retouching is possibly the most known technique in this list, since it is widely used in publications such as magazines where editors resort to it to make corrections to or embellish the published pictures. Comparing to the others already covered, this manipulation by itself does not alter the image to the point where it can become misleading. However, retouching techniques are used in manipulations like copy-move and splicing in order to correct inconsistencies that arise from the tampering.

Figure 2.13: Example of the retouching technique. Original on the left, tampered on the right [37]

2.3 Deep Learning

DL is a type of ML approach. ML is a field of Artificial Intelligence (AI) that consists on teaching models by training algorithms with data, which will then become capable of taking decisions and making predictions by themselves, when presented with data they have not been fed before [11]. This way, these models are capable of solving a huge range of problems in various fields. Some very common problems are classification and regression problems. In a classification problem, the model learns a set of possible classes (categories) to which an observation can belong to, and is then tasked with classifying future observations. For example, the previously mentioned face identifier algorithm is a classification problem, where the model is tasked with classifying the image it receives as "contains a face" or "does not contain a face" (which are our classes in this case). Regression problems on the other hand explore correlations between one or more predictors and an outcome. For example, someone who wants so study the correlation between hours spent studying and grades in the final of a semester - this would be one case of a regression problem.

There are mainly three types of learning methods:

- **Supervised Learning**

  Supervised learning is perhaps the most common one, due to its simpler nature. In this approach, the data fed to the model has already been completely labeled - returning to the example of the face detection algorithm, at the learning phase, it already knew which images contained faces and which didn't. Naturally, problems such as classification and regression will fall in this category, due to the requirement of having the necessary labels in the data in order for the model to work correctly.

- **Unsupervised Learning**

On the other hand, unsupervised learning works with unlabeled data. This approach is often used with the intent of finding correlations in data that we as humans are unable to naturally find, or in the fields of statistics.

- **Reinforcement Learning**

  Finally, reinforcement learning ends up approaching a different type of problem altogether. In this case, the agent (which is our model) will be presented with an environment (which is related to the problem we want to solve). This agent, through a possible set of actions, will explore the environment and at the same time receive positive or negative feedback (depending on the actions it takes). This way, the agent will start to learn, and will then be able to exploit that knowledge to make better decisions in the exploration phase. This cycle of exploring and exploiting goes on, progressively giving more relevance to exploit rather than exploit as time goes on, and that's how we achieve our end goal.

DL involves using a particular class of ML models which are referred to as Artificial Neural Networks (ANN) to solve problems. ANNs get their name from the fact that their architecture, along with the way they work, is heavily inspired by the human brain. These networks are constituted by a number of nodes, known as "neurons", that are all connected, and combined form the various layers of the network [47].

An ANN has an input layer, an output layer and then multiple hidden layers. The input and output layers receive the initial data and determine the result, respectively, while in the hidden layers the neurons receive data, perform all the computation required and then pass it to the next one. The number of layers is directly related to the complexity of what a network can learn, althought too many layers can bring problems such as overfitting.

DL models are generally taught in a supervised learning environment and through a process called backpropagation. The way this works is by using input data with a corresponding output and training the network so that the weights in the neurons - the parameter which influences its computations - are adjusted in a way that minimizes the error between the output of the network and the true output for each respective input [39].

Since DL models require a vast quantity of data to train, as well as the fact that these networks are a lot more intensive in terms of computer resources when compared to traditional ML models, it is often used in areas such as image classification and natural language processing [41].

2.3.1 Generative Adversarial Networks

A GAN is a type of DL model that consists of two competing networks, paired together: network G, standing for generator, and network D, standing for discriminator. The idea behind this architecture is that by paring these two together, we will be gradually able to obtain improved results, since the generator will constantly improve the quality of its fakes due to the need of

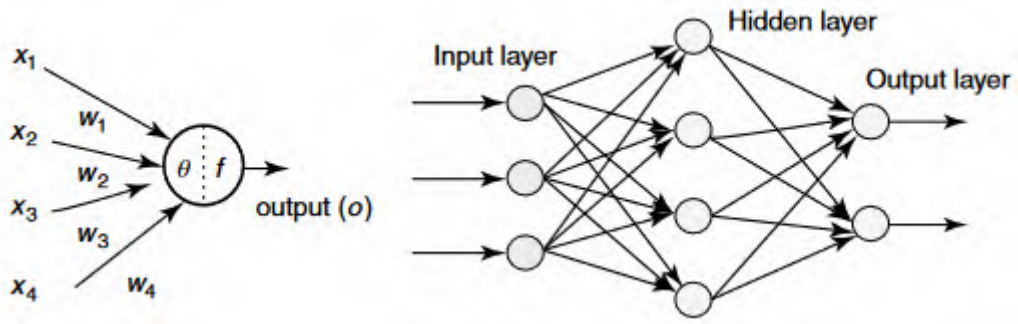


Figure 2.14: Representation of a neuron and of an ANN [3]

tricking the discriminator.

The generator can be considered as a mapping from a representative space (called the latent space) into image space. In a similar fashion, the discriminator acts as a function that maps the image space into a probability that the image hails from a real data distribution (0 for fakes created by the generator and 1 for real images).

To train a GAN, the user provides a set of images that will make up the dataset used for training. However, contrary to what one might initially assume, these images will not be accessible to the generator for it to base itself off for its creations. Instead, they will only be available to the discriminator. They will be used in the initial discriminatory part of the process, where the discriminator is tasked with learning about the specific training domain. When we reach a certain level of optimization for the discriminator, then the generator starts generating fakes using random vectors as input. This fakes will then be presented to the discriminator for it to classify them as real or not. If it correctly classifies them, then the generator will be updated. On the contrary, if the generator successfully tricks it, then it's the discriminator turn to be updated. This means that the initial goal is to find parameters that maximize the classification accuracy of the discriminator, and then the ones that maximize the confusion the generator causes on the discriminator. Finally, when the model reaches a state where the discriminator can't differentiate between real and fake images with more than 50% of accuracy, the training is considered complete.

2.3.1.1 StyleGAN2

This GAN architecture was developed by researchers at NVIDIA[1], who also developed the original StyleGAN[2], which was a state-of-the-art method for generating images of high resolution. However, it still had visible issues, mainly the existence of blob-like artifacts in the pictures (droplet artifacts) and highly localized features which would stay pinned in certain positions (phase artifacts). StyleGAN was also able to mix the styles of two different images to create a third one, but this functionality had artifact related problems of its own. StyleGAN2 also fixed

[1] https://www.nvidia.com/en-us/research/
[2] https://github.com/NVlabs/stylegan

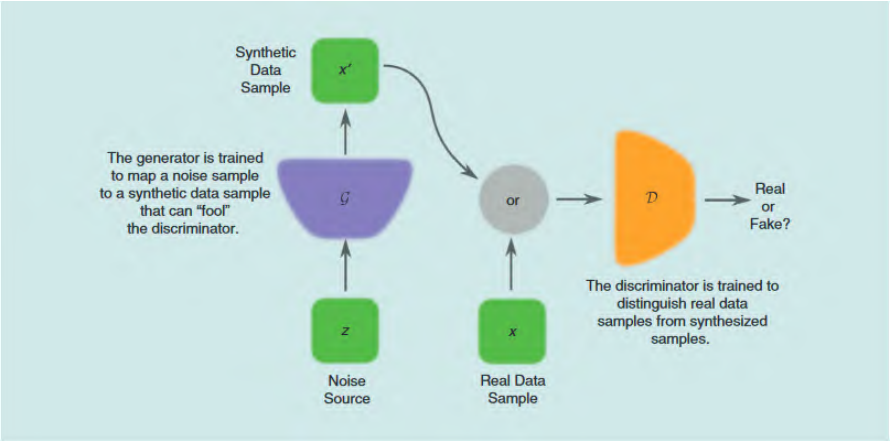


Figure 2.15: Overview of the training of a GAN [16]

these issues [27].

Bellow are some hand-picked images by the NVIDIA researchers themselves, which demonstrate the astounding capabilities of this architecture.

Figure 2.16: Faces generated by StytleGAN2, handpicked by NVIDIA

2.3.2 Convolutional Neural Network

Convolutional Neural Networks (CNNs) are an extremely popular architecture for ANN models. They are prevalent in the area of image classification problems, given that they allow for the processing of large volumes of data - such as the dimensionality of an image - while maintaining the quality of the information contained; as such, it makes sense to see them used in the context of face recognition [14].

The principle of CNNs is the mathematical operation convolution. This consists on producing a new function from two existing functions, where the new one represents how the first is modified by the second (or vice versa). In this context, the first function is an image - which is nothing more than a N x N matrix of values in the case of greyscale images, or N x N x N in the case of RGB, while the second is a filter - which is typically 3 x 3 [24]; generally, the filter is odd-sized due to ease of implementation, and 3x3 provides an optimal correlation between accuracy and computational costs. Through convolution, the dimensions of the original image will be reduced.

As we can see in the example shown in Figure 2.17, by applying the filter to an area of the image, calculating what is denominated as dot product - a product between the values of the pixels and those of the filter, and then repeating this process by shifting the filter to other areas in order to cover all the picture. The end result is a new matrix containing all the dot products, and this is often referred to as a feature map.

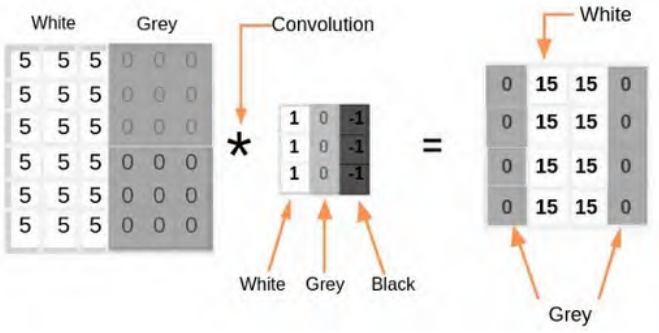


Figure 2.17: Convolution on a greyscale image [43]

CNNs consist of an input layer, an output layer, and then a combination of three possible types of hidden layers:

- **convolutional layer**: as the name might imply, these are the most important layers in the network, as they are responsible for the convolution operations. A CNN will have an hierarchy of convolutional layers, each one extracting more detail and more complex information than the one prior.
- **pooling layer**: in this layer, CNN performs dimensionality reduction. Also refered to as downsampling, it is a process that creates a version of the original input which while maintaining the relevant features, has a lower resolution. Similarly to the convolutional layer, a filter will be applied to different segments of a data matrix, but in this case it will not have any values; its purpose is to define which values the layer will pool from. Pooling can be done in two different ways: max pooling, where the maximum value will be selected, or average pooling, where the average of the values is calculate instead.

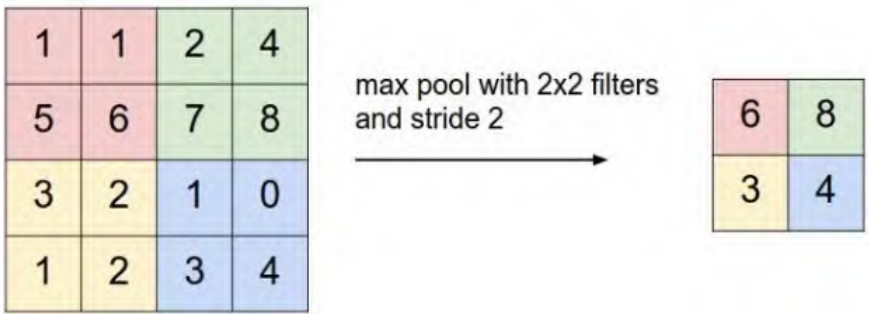


Figure 2.18: Visual example of max pooling [43]

- **fully-connected layer**: these are the final layers of a CNN, and they get their name

from the fact that every neuron of a previous layer is connected to every node of the following. They receive the information from the convolutional and pooling layers and learn to associate features with classes to perform classification, using either softmax for multi-classification problems or sigmoid for binary classification.

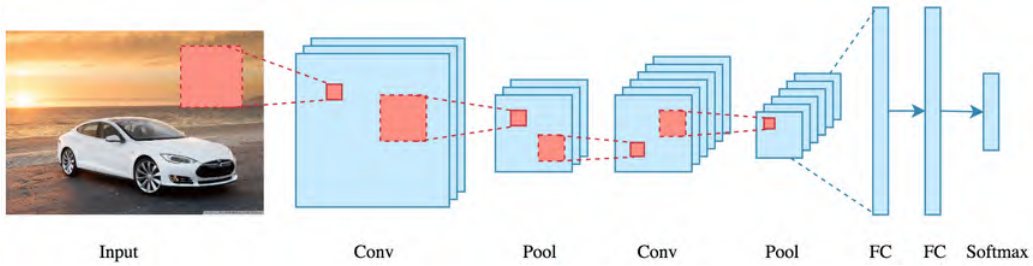


Figure 2.19: CNN architecture example [32]

2.3.3 Deepfake

Deepfakes - combination of deep (from DL) and fake - are the product of a phenomenon that has emerged along with the advances in DL. Using ML, people can manipulate real images and videos in an extraordinarily convincing way. The end result is often a video or picture, in which a person will appear doing or saying something which they very possibly haven't said or done. They came into the attention of mainstream media due to their use on the creation of celebrity pornographic content, and later due to their potential of use in information manipulation (the commonly known 'fake news'), especially when paired with text-to-speech technology. However, they also present a great threat to facial authentication systems, as it has already been tried and tested that this technology can be used to bypass them [44].

Deepfakes use the same GAN technology that we've already covered in Section 2.3.1. This means that the images generated by GANs can also be classified as deepfakes, and throughout this document, that's what we will be referring to, unless stated otherwise.

There are various ways for people to create them, even without any knowledge of DL or even what a GAN is, as there are various free tools that allow their creation, only needing as much as a source and a destination media.

2.3.3.1 DeepFaceLab

DeepFaceLab[3] (DFL) is a state-of-the-art tool which (they claim) is used in the creation of more than 95% of deepfake videos. While this tool requires only one piece for source and one piece for destination, the process itself is more complex and requires a bit of time when compared to other existing deepfake tools. However, the results themselves are of much greater quality,

[3] https://github.com/iperov/DeepFaceLab

Figure 2.20: A face-swapping deepfake [38]

providing reliable face-swapping even in scenarios such as pictures with bad illumination and faces presented sideways [38].

The DFL pipeline can be divided in three main steps:

- face extraction
- face alignment
- face segmentation

In the initial step, DFL uses a face detection algorithm to extract both the faces of the source and of the destination data. DFL uses Single Shot Scale-invariant Face Detector[4] (S^3FD), but DFL's paper[5] mentions the algorithm in this step can be easily replaced by another one.

After that, to align the faces, DFL extracts facial landmarks in order to grant stability to this process. It uses two different algorithms depending on whether the face is in a standard pose or not, and after extracting the landmarks, it also offers an optional step where they can be smoothed in consecutive shots to further provide more stability. Finally, it uses them to create a similarity transformation matrix which will be used for the alignment.

The final step consists on creating a segmentation mask for the images. This mask can be automatically generated by a network, or the user can manually create, since even state-of-the-art algorithms sometimes fail in creating a fine-tuned mask for particular shots.

[4] https://github.com/yxlijun/S3FD.pytorch
[5] https://arxiv.org/abs/2005.05535

Chapter 3

Design and development

In this chapter, we present the proposed architecture of our workflow and discuss the development process. First, we start by identifying the necessary steps to build the envisioned system, and following that we cover the development and implementation processs.

The strength of this architecture is its modularity; throughout the various steps of the process, the software used can be replaced by a preferred option without with ease and without compromising the performance of the project as a whole, which grants possible users a good degree of customization.

3.1 Proposed Architecture

The proposed architecture is depicted in Figure 3.1

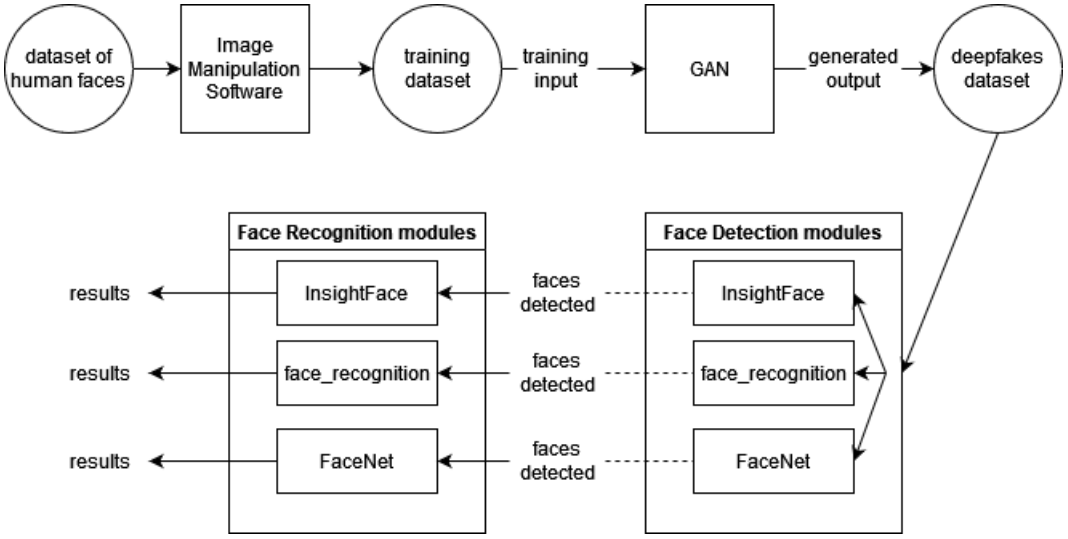


Figure 3.1: Project architecture

We chose to divide our workflow into these different steps, as detailed above:

1. Creating the initial dataset of manipulated faces, which will then be used for training.
2. Training the GAN, so that it becomes capable of generating a dataset of deepfakes.
3. Testing how FD and FR systems react to this type of faces.

As previously discussed in Section 2.3.1, the discriminator learns from the input data, and it is necessary that this data is representative of a specific domain. If this is not the case, we will end up with a "confused discriminator", which in turn will result in a suboptimal generator since it is not being "challenged" by the discriminator. As such, it is crucial to use a dataset which is specifically curated to the task at hand. In our case, we needed a dataset of manipulated human faces.

To ensure optimal conditions for the GAN training, along with an appropriate dataset, it is also necessary to have an environment that is up to the task. DL is extremely resource intensive, and GANs are not an exception. Given that they go through multiple back-and-forth iterations between both discriminator and generator, it is either needed to have an extremely potent machine or to run the training algorithm for a great duration of time.

Contrasting with the training, generating fakes is a rather quick task for the GAN. The reason for this is that it is a crucial part of the training itself; at any given time, the fakes we get as a product of the GAN are no different than the ones it will use in the next training iteration. This means that once a point where the generator is producing "quality" fakes is reached, creating a dataset is trivial since it will provide countless images for it.

The final segment of the pipeline are the FD and FR modules. First and foremost, it's important to note that we opted for FR instead of face authentication. The reason for this choice is that for testing, recognition is much more practical since instead of trying to individually match each different fakes with each real face, we can instead check once for each fake if there is any match amongst the reals. Since the matching criteria is the same in the two situations, the results we get with recognition will also be applicable to the authentication context.

As for the modules itself, we chose them based on the following criteria:

- They should be able to perform both facial detection and facial authentication, separately.
- They should have a high level of accuracy.
- They should be easily integrable into a script, and scalable for a great number of pictures.

After identifying all the components and tasks necessary to the deployment of our system, we then started working towards that end goal.

3.2 Development

Since the core of our work was the deepfake generator, our first concern was finding a tool that would be able to serve in this task. We quickly came across StyleGAN2 thanks to This Person Does Not Exist[1], a website that whenever accessed presents a fake image of a person, generated by the network. The site had multiple resources where it was possible to learn about the technology, as well as the github with the code for training our own, so after consulting said resources we were satisfied with its possiblies and decided to use it.

After that, we had to obtain the training dataset. Initially, we searched for an existing one, in hopes of not having to create one ourselves, since training DL models requires large amounts of data. Unfortunately, we were unable to find one, and thus we had to work on creating our own. In order to do that, we needed to find a way to generate a great number of manipulated images, in the way that we wanted. So, we researched for ways to create new faces by editing existing ones.

Our first big hit seemed to be SEAN[2], a project on Github which allowed the user to edit a face by using a style image in conjuction with a segmentation mask [49]. This seemed to be exactly what we wanted, since the mask was divided into various segments that correspond to relevant face landmarks such as left and right eye, nose, ears, etc., which were exactly the characteristics we were trying to change. Unfortunately, when we tried using SEAN, we were unable to getting it to work properly. When we tried running the interface which enables the user to edit the face, it would inevitably crash before it was possible to do anything. After attempting to troubleshoot and fix the issue, we resorted reaching out and contacting the developers, but sadly there was no reply which in turn meant that we had to drop SEAN.

After that, we came across DeepFaceEditing[3], another Github project on face editing. This time, it provided a way to generate a new face using an existing one and the geometry of another face or that of a sketch [15]. We then decided to make a script that would generate sketches of faces, with the option to insert the coordinates of the relevant landmarks in the image to allow us to have control over the generation of our custom faces. However, when testing the program, we weren't able to reproduce the results that were shown in the paper, even when using the base image and the geometry of the sketch that authors showed as an example of the capabilities of their tool. Once again we tried contacting the developers with hopes of fixing any possible issues, but in similar fashion to the previous project there was no answer. We also checked other existing "sketch-editing" projects, but there were also problems with those, namely the editing itself being too "rigid" - no matter the sketch, the created faces would all have a normal human structure. Given that those faces would not be fit for our dataset, we had to consider other options.

Finally, we opted to use a image editing software, Photopea[4], to manually manipulate and

[1] https://thispersondoesnotexist.com/
[2] https://github.com/ZPdesu/SEAN
[3] https://github.com/IGLICT/DeepFaceEditing-Jittor
[4] https://www.photopea.com/

Figure 3.2: Example of manipulation of an image for our dataset. In this case, the normal amount of landmarks are still present, but their location has been altered as to create a new face which does not resemble a real human

create images. This obviously was not optimal, given the dimension of the dataset we had to generate. Nevertheless, we used This Person Does Not Exist to collect the faces for editing, keeping in mind to have a good amount of diversity in regards to age, skin color, hair color, and such. This was due directly connected to the fact that manually editing each face took a fair amount of time, so we tried to maximize the small differences in them in hopes of later minimizing the risks of overfitting. This made us realize that StyleGAN2 is noticeably biased towards generating images of middle-aged caucasian people, as images of other races were considerably less common and had more imperfections. In any case, we were able to collect the images we wanted. Then, using the software, we applied manipulation techniques, mainly copy-move followed by removal and then retouching, to drastically change the faces. An example can be observed in Figure 2.9.

The final version of the training dataset did not come into fruition until there had already been some results from the GAN, so we will also address that step now.

As previously mentioned, we initially chose to use the StyleGAN2 architecture. However, we later discovered StyleGAN2-ADA[5], a variation of the original that uses an adaptive discriminator augmentation mechanism in order to minimize overfitting when training with smaller quantities of data. Since we were manually generating our dataset, this seemed to be the optimal option, even more considering that StyleGAN2-ADA had similar results to StyleGAN2. As such, we got the necessary contents from the project's Github page and set up the environment in the most potent machine we had available, making sure that it had an appropriate GPU. With that, we could start the training. To do that, its necessary to run the script and provide the location of

[5] https://github.com/NVlabs/stylegan2-ada

the dataset as well as the number of cycles the GAN should run before saving a "checkpoint". Checkpoints are instances at which the training is paused and all the progress is saved, and represent the state of the network after a certain amount of training time. It is important to balance the creation of checkpoints since the process of their creation takes a considerable amount of time, so it is not optimal to constantly create a new one, but at the same time it is important to regularly save the progress in order to avoid problems in case of errors or other situations that could influence the normal functioning of the training process. With that in mind, we initially chose to create a new checkpoint each twenty five cycles, but after experimenting for some time and learning that it took our machine fifteen minutes for cycle, we opted for higher values, according to the time we had at hand at the moment of each training session. It was also in our interest to have more checkpoints in the initial phases of the process as it allowed us to keep better track of the results produced, and more quickly identify when changes had to be made. As for the dataset, it started off small and then we gradually incremented it. It was our goal to observe how much it was possible to push this training, but we had to manage the balance between the duration of the process and the contents of the training set. As such, this meant that the process of altering the dataset, running the script and waiting for a checkpoint to evaluate had to be repeated multiple times, until we eventually reached a point where the level of quality of the deepfakes was acceptable.

At this time in development, we considered our generator to be ready, and so we started looking into evaluating its results. For that, we created scripts which implemented our chosen libraries: `face_recognition`, `insightface` and `FaceNet`. Although different, they all do the same thing: validate the pictures we generated by looking for faces in them, and then trying to match those faces against others "known" to the system. For this matter, we chose to use the CelebA[6] and the FFHQ-small[7] datasets. Both contain a diverse amount of faces, with the first providing useful annotations and the latter being of the same image quality as our produced fakes.

The script using `face_recognition`, observed in algorithm 1, loops through all images one by one, loading them and identifying the presence of one (or more) faces in them using the library's functions. Then by checking if the length of the returned array is greater than zero - meaning, at least a face was found, it is determined if the image is used in the next phase of the process. In this phase, first the script loops through the dataset of known pictures, storing their encoding in a list. After that, it loops once again through the pictures, comparing their encoding with those of the other pictures. If there is at least one match, a counter is incremented to reflect that, and at the end we can observe how many fake faces were wrongly matched with real ones.

The script using `insightface`, observed in algorithm 2 functions in a fairly similar way to the previous one when it comes to face detection. It also loops through all of our fakes looking by one or more faces in the image. The difference between the two libraries is that while `face_recognition` calls a function that allows us to choose which model to use (and other

[6] https://www.kaggle.com/datasets/jessicali9530/celeba-dataset
[7] https://www.kaggle.com/datasets/tommykamaz/faces-dataset-small

Algorithm 1 face_recognition

```
1: i ← 0
2: f ← 0
3: deepfakes encodings ← [ ]
4: known faces encodings ← [ ]
5: define deepfake images folder path
6: create list of image paths
7: for path in list do
8:     load image from path
9:     use module to look for faces in the image
10:    if one or more faces are identified then
11:        increment i by 1
12:        calculate image encodings
13:        append encodings to the deepfakes encodings list
14:    end if
15: end for
16: define known images folder path
17: create list of image paths
18: for path in list do
19:    load image from path
20:    use module to look for faces in the image
21:    if one or more faces are identified then
22:        calculate image encodings
23:        append encodings to the known faces encodings list
24:    end if
25: end for
26: for entry in deepfakes encodings list do
27:    match entry with all entries on known faces encodings list
28:    if embedding matches any in the real embeddings list then
29:        increment f by 1
30:    end if
31: end for
```

sorts of options) every iteration of the loop, `insightface` has us set up an instance of the "application" initially, with all the necessary configuration, and after that we only need to call it when we want to do our evaluations. Apart from this, the scripts work pretty much in the same way, as we also check the length of a returned array to determine whether or not faces have been detected. However, this script does not contain a segment for FR. The reason for this is that we wrote the script in two parts, and since none of our fakes were detected as faces by this library, as we'll be seeing in Chapter 4, we deemed unnecessary to implement that component.

Finally, our script which implemented `FaceNet`, observed in algorithm 3, was the most

Algorithm 2 insightface

```
    i ← 0
 2: define parameters for 'app'
    instantiate 'app'
 4: define image folder path
    create list of image paths
 6: for path in list do
        load image from path
 8:     use 'app' to look for faces in the image
        if one or more faces are identified then
10:         increment i by 1
        end if
12: end for
```

different of the three. We start by defining an instance of a MTCNN with the necessary parameters, as well as loading the chosen model into an instance of a Residual Neural Network. These will be responsible for the FD and FR segments, respectively. After that, we loop through the real images, but instead of giving us a 'yes' or a 'no', `FaceNet` gives us a probability of the image having a face. We consider every value greater or equal than 0.9 to correspond to a 'yes'. For each positive result, we then store the embeddings on a list. After that, we loop through our fakes, again following the same logic as we did with the reals. If a face is identified in the fake, we calculate its embeddings and then calculate the Euclidean distance between it and each one of the real face's embeddings. We store this calculations on a list, and once they are done, we check the smallest value on the list. If this value is lower than a given threshold - in this case, 1.1, as it is demonstrated in the `FaceNet paper` - we increment our fake image counter.

Algorithm 3 FaceNet

```
f ← 0
define type of device to be used
define parameters for MTCNN
load ResNet model into device
real images embeddings ← [ ]
load real images into dataset
for image in dataset do
    calculate probability of image having a face using the MTCNN
    if probability ≥ 0.9 then
        calculate image embeddings using the ResNet
        append embeddings to the real images embeddings list
    end if
end for
save list as PyTorch file
define image folder path
create list of image paths
for path in list do
    load image from path
    calculate probability of image having a face using the MTCNN
    if probability ≥ 0.9 then
        calculate image embeddings using the ResNet
        load list of real embeddings from PyTorch file
        distance ← []
        for embedding in list do
            calculate the Euclidean distance between both embeddings
            store the distance on the distance list
        end for
        if lowest value in distance list < 1.1 then increment f by 1
        end if
    end if
end for
```

Chapter 4

Results and analysis

In this chapter, we present the results obtained in our work and discuss them. We will start by analyzing the GAN and its fakes, and following that we will be checking the behaviour of FD and FR modules against them.

4.1 Generated Fakes

As previously addressed in Chapter 3, the process of developing our tool was one of constant evolution. In this section, we present some of the initial products, along with some of the final results, in order to demonstrate the improvements in our fakes. During training, a set of images that represented the state of the network at that specific point of the process was generated, and using these we can better visualise this progress.

As aforementioned in Section 2.3.1, the starting point for the generating process is the noise. A representation of this noise can be seen in Figure 4.1. No matter what training data was used, this pattern of colors stayed consistent in our every attempt.

Initially, and in order to do a sort of benchmark test, we started with only three fakes in the training set. After some training time, we got to the point were our fakes were all the same, as it can be seen in figure 4.2. Apart from the obvious overfit, this images particularly resembles one of the three from the training data, which was relatively curious.

With our training setup seemingly working as intended, we then repeated the process above with different iterations of our set, each of these containing a gradually greater number of manual fakes. As the number of images fed increased, so did the quality of the generated fakes, and so we kept creating more, but still there were issues with the images that we were not quite able to avoid.

However, it was not until our last iteration that we realised something crucial. To illustrate the following point, we have figures 4.3 to 4.7. These images show representations of the GAN at different timestamps of the training process, corresponding to our designated checkpoints;

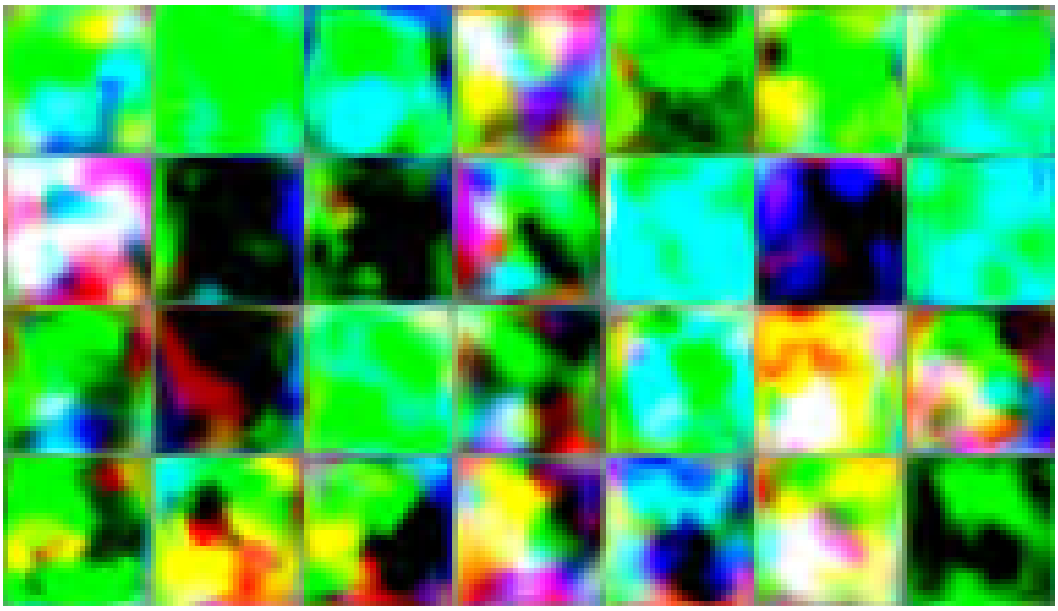


Figure 4.1: GAN's initial noise

Figure 4.2: Overfitted GAN training result

we start by seeing the semblance of a face forming in the first one, then something even closer in the next; up to this point, everything seems normal. However, by the next checkpoint we noticed that in a somewhat similar fashion to the first test we ran, the fakes were starting to closely resemble the fed data. After some more time, we realised the overall color pallet of the images was considerably changing and there was even some rotations of the orientation (not depicted in the figures). Eventually, we wounded up with the fakes we see in figure 4.7, which are undesirable. At the same time, when looking at the logs for our training, we noticed that when resuming from a previous checkpoint after a pause, one parameter - augmentation - began to increase at four times the amount it used to grow when the training was done without stopping. After we looked into it, we discovered this augmentation was the use of modified versions of

Figure 4.3: Final dataset - state of the GAN after about 3h20

our original images (different hue, different rotation, etc. - as we saw) in order to provide the model with more training data and improve its performance [7]. Additionally, we also learned about "augmentation leak": a problem in GANs when the augmented images "leak" into the generator and it starts to learn the input used, therefore "cheating" and making the process even more prone to overfitting [4]. This was certainly the situation we were being faced with, very possibly because of the abnormal values that got exponentially out of control. Some online research showed that this was a probable bug that also happened to other people; we tried fixing it, however with no success.

We then decided to train a new network with the same dataset, but this time continuously in order to avoid the leaking of the augmentation. Unfortunately, this limited the amount of time we could spend, but nevertheless this gave us the best results we got, as we can observe in figure 4.8, since we chose not to proceed further with this part of our work.

In the end, we can certainly say that the process of training a network was much more complicated than what was initially envisioned. Not only did we have issues obtaining a satisfactory number of images to use in the process, but we were also limited by the sheer need of time and computing resources this task entailed. In both aspects, we came up short, as we could not simply upgrade our machine and manually creating the type of images we wanted until we had the amount needed was unfeasible. Ultimately, this meant that as we can see, our fakes were extremely similar to one another, due to the probable over fitting, and while the image quality was high, the faces themselves were clearly computer-generated.

Even so, one of the initial premises of our work was that we wanted to see how facial recognition systems reacted to faces which were not necessarily of a human. And so, lets proceed

Figure 4.4: Final dataset - state of the GAN after about 6h40

Figure 4.5: Final dataset - state of the GAN after about 10h50

Figure 4.6: Final dataset - images with different hue

Figure 4.7: Final dataset - state of the GAN after many training sessions

Figure 4.8: Final dataset - state of the GAN after continuous 11h

to the next step to see them in action.

4.2 Face Detection

As described above, evaluating whether or not our fakes were identified as real faces involved running the initial part of each of our scripts. Once we achieved satisfactory results with the generator, we fed the same 500 fakes to each detector. In addition to that, we also used another type of input, this time with the intent of using it as an additional benchmark the chosen libraries. These consisted of around 20 pictures of faces of monkeys and other 20 faces of real people.

For the tests done in this section, we will use the following metrics:

- True Positives (TP), in this context corresponding to the number of images containing human faces correctly identified as such.
- False Positives (FP), in this context corresponding to the number of images not containing human faces incorrectly identified as containing human faces.
- True Negatives (TN), in this context corresponding to the number of images not containing human faces correctly identified as such.
- False Negatives (FN), in this context corresponding to the number of images containing human faces incorrectly identified as not containing human faces.
- Accuracy (Acc.), corresponding to the number of positive results divided by the total number of images.

For this initial benchmarking, we got the results below:

Benchmarking test							
module	Monkeys	People	TP	FP	TN	FN	Acc.
face_detection	20	20	20	3	17	0	0,925
insightface	20	20	0	3	17	0	0,925
FaceNet	20	20	20	2	18	0	0,95

Table 4.1: Face detection benchmarking tests with images of monkeys and real people

Looking at these results, we outright get an indicator of possible success for our fakes, since even in a small sample, some images that were not supposed to be identified as faces incorrectly were. Also, we can somewhat assess that these libraries are all at the same level of "quality" - the results were pretty much similar between all three.

Once these tests were completed, we then moved to the aforementioned ones with our fakes. The results for those are presented in the following tables:

Test: our fakes						
module	Input size	TP	FP	TN	FN	Acc.
face_detection	500	0	500	0	0	0
insightface	500	0	0	500	0	1
FaceNet	500	0	500	0	0	0

Table 4.2: Faced detection tests with our fakes

The results were, to say the least, polarizing. Either all the fakes were recognized as faces, or none were. An initial thought was that the `insightface` module was possibly more accurate/strict, and so our fakes were unable to "trick" it. This thought, however, was contradicted by the results of the benchmarking tests, as `insightface` had an identical number of false positives to the other modules. As such, what is more likely is that given that each module works in a different way from the other, the face recognition module implemented by `insightface` - SCFRD [21] - is simply looking at the fake data in a way that invalidates our attack vector.

As for the other two modules, comparing the results of both sets of tests, we see that, even at a much larger scale, our fakes were capable of obtaining better results than the faces of monkeys, which honestly had already proven to be able to trick the systems to some sort of degree. However, this fact - that not even one of the images failed, coupled with how we previously saw that the generated faces were extremely similar, arose the question about whether or not it was possible that all the different images were identified as the system as having the same one face.

However, to confirm if this is indeed the case, we simply need to check the results of the next phase of testing. Also, given the outcome, we dropped `insightface`, leaving us with only the other two modules for the following phase of tests.

On one final note, additionally to that large scale test, we also conducted one where we used some the manually generated fakes used in the network training. The results were as follows:

Test: our original fakes						
module	Input size	TP	FP	TN	FN	Acc.
face_detection	39	0	39	0	0	0
insightface	39	0	0	39	0	1
FaceNet	39	0	39	0	0	0

Table 4.3: Faced detection tests with our manually created fakes

Again we see the same behaviour from the modules: `insightface` doesn't detect any of the images as faces while the other two modules detect all.

4.3 Face Recognition

For the FR segment, we used the datasets mentioned in Section 3.2, and created three different testing batches from them: one containing 200 faces from the CelebA dataset, another containing 500 faces from the FFHQ-small dataset and the final one containing 1000 faces from the FFHQ-small dataset. These will be used for the input of the recognition module, while for the set that will act as the system's identity database we used 500 of our deepfakes in each of the test cases.

We won't be using the metrics we used in the FD tests for these ones; instead, we will have the number of 'faces detected' - faces from the attacking dataset ('Matching Dataset' in the tables) that matched one or more in the 'enrollment database' (figure 2.2, 'Recognition Dataset' in the tables) - and the percentage.

For the `face_recognition` module, the results can be observed in the table below.

Recognition module: face_recognition			
Recognition Dataset	Matching Dataset	Faces Detected	% Detected
CelebA (200)	Fakes (500)	59	11,8
FFHQ-small (500)	Fakes (500)	465	93,0
FFHQ-small (1000)	Fakes (500)	477	95,4

Table 4.4: Face recognition tests with the `face_recognition` module, matching our fakes with reals

Immediately from this results we can see that the hypothesis we considered in the previous section, about all the faces being seen as the same one, is not true; if it was, all faces would've matched or none would, similarly to what happened before. However, the near 100% percentages in the tests involving the FFHQ-small subset made us realise a flaw in this method of testing. As we are counting whether or not there was at least one match, and not the specific number of matches, even if only one of the recognition's dataset faces is matching our fakes, as long as it matching a grand number of fakes incorrectly we will have inflated results.

In order to try to avoid this, and to get another perspective, we reversed the roles and used

our fakes as the recognition dataset while using the other sets as the matching ones. From this tests, the following results where produced, as observed in the following table.

Recognition module: face_recognition			
Recognition Dataset	Matching Dataset	Faces Detected	% Detected
Fakes (500)	CelebA (200)	8	4
Fakes (500)	FFHQ-small (500)	56	11,2
Fakes (500)	FFHQ-small (1000)	127	12,7

Table 4.5: Face recognition tests with the `face_recognition` module, matching reals with our fakes

As we can see, this batch of results is considerably different from the previous one. For CelebA, we see that only 8 faces matched the fakes, as opposed to 59 when the roles were reversed. Looking at percentages, we see 4% compared to roughly 12%, which translates to three times less. However, if this difference gives us some more insight into how realistic it is for this faces to be matched in pratice, where this can be really observed is in the difference between the matches with the FFHQ-small subsets.

The initial tests showed 465 and 477 faces to have matched, respectively representing 93% and 95% of the total matching sets. When we switched the sets, this percentages crashed to 11% and 13%, which represents a reduction of more than 80% - approximately 82% - in both cases. Also, we can observe that the percentages in both sets are very similar, which allows us to theorize that the FFHQ-small dataset is more susceptible to this type of attacks then CelebA one.

Now, for the same tests, but with a different module - `FaceNet`, we get the following results:

Recognition module: FaceNet			
Recognition Dataset	Matching Dataset	Faces Detected	% Detected
CelebA (200)	Fakes (500)	500	100
FFHQ-small (500)	Fakes (500)	500	100
FFHQ-small (1000)	Fakes (500)	500	100

Table 4.6: Face recognition tests with the `FaceNet` module, matching our fakes with reals

When we were faced with these results, we were obviously taken aback. Surely, there must have been some type of error in our method. Once again, we switch the roles of the datasets, and these were the results we got.

Contrary to what happened with the initial `face_recognition` module, in this case, the switch did not provide us with more realistic results. Faced with this apparently faulty module, we then decided to run a test in a smaller scale, in order to be able to better dissect the obtained outcome.

What we did was use a script that matched two sets against each other, and then constructed a matrix with the values obtained for each individual calculation between members of them. As

Recognition module: FaceNet			
Recognition Dataset	Matching Dataset	Faces Detected	% Detected
Fakes (500)	CelebA (200)	200	100
Fakes (500)	FFHQ-small (500)	500	100
Fakes (500)	FFHQ-small (1000)	1000	100

Table 4.7: Face recognition tests with the `FaceNet` module, matching reals with our fakes

previously mentioned, `FaceNet` calculates the euclidean distance between two faces and then we can assess whether or not those two are a match by comparing the obtained value with a defined threshold - both in this and in our previous tests we used the suggested value of 1.1.

For the two sets, we actually used the same one twice, which contained 11 pictures of myself (taken from various angles) and 11 of our fakes. We did this so that the matrix would simultaneously show the matches between my face and the fakes, my face with my face and the fakes with the fakes. Each picture was labeled individually in order to better to facilitate the analysis.

Surely enough, the results of this test were very different from the other ones. In this case, all matches were correct - the ones between images of my face had scores below 1.1, and so did the ones between images of fakes, while the ones between an image of my face and a fake had scores above 1.1.

This left us with a control sample in which there is no abnormal behaviour, and the test cases which present us with results that are simply too far fetched, but given the outcome of the control test, not incorrect. This led to many considerations, but eventually we decided that even though our method was correct, we could not take these values into consideration for realistic analysis.

Ultimately, this meant that for the FR segment, we were left with only the results of the `face_recognition` module for result analysis.

Motivated by what we saw in the `FaceNet`, we decided to run yet another set of tests. This time, we opted to match our testing datasets with one another - specifically, the 200 CelebA and 500 FFHQ-small subsets. The results can be observed in the table below:

Recognition module: face_recognition			
Recognition Dataset	Matching Dataset	Faces Detected	% Detected
FFHQ-small (500)	CelebA (200)	69	34,5
CelebA (200)	FFHQ-small (500)	133	26,6

Table 4.8: Face recognition tests with the `face_recognition` module, matching different sets of reals with one another

Once we had the outcome, we proceeded to compare it with the results presented in tables 4.4 and 4.5. Specifically, we looked at the performance of our fakes versus the performance of the

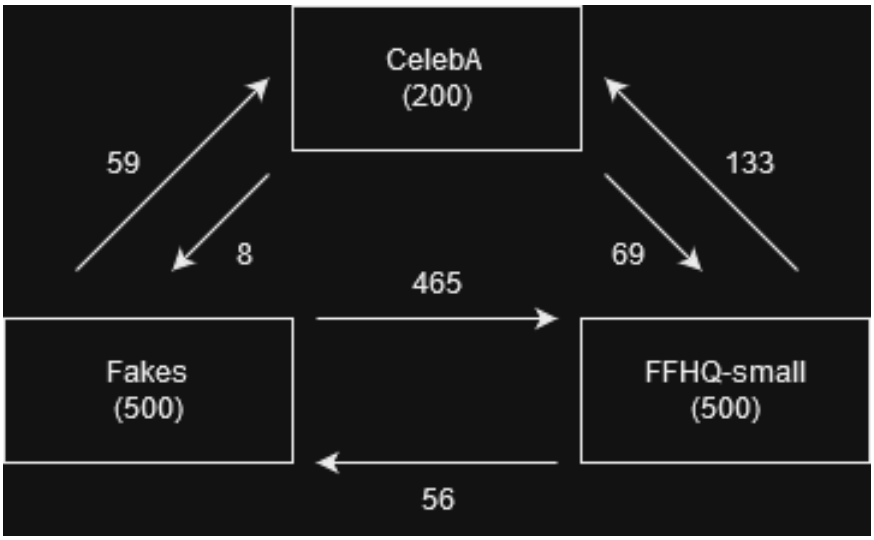


Figure 4.9: Diagram portraying the results of various test cases between three of our image sets

reals for the cases in which CelebA(200) and FFHQ-small(500) were assigned as the recognition sets, and also at the performance of both real sets when matching against the fakes. So, when considering all of those results together, we can observe the following:

- For CelebA, of our 500 fakes only 59 were macthed, compared to 133 of the 500 reals in the FFHQ-small matching subset.
- For FFHQ-small, 465 of the 500 fakes were matched, while for 200 CelebA faces 69 were false positives.
- For our Fakes, 56 of the 500 FFHQ-small faces were false positives, compared to 8 of the 200 CelebA faces.

Using this values, we constructed a diagram, in order to put into visual perspective how the three sets interacted between one another. We can see these relations in figure 4.9.

After analyzing it, we were able to properly outline the following points of relevance:

- For CelebA, the real images were more effective at "attacking" than our fakes.
- For FFHq-small, it's the contrary - our fakes proved better than the reals.
- Even though the number of faces from FFHQ-small that matched with the CelebA set was greater than the number of faces from FFHQ-small that matched with the fakes, the opposite is not true.

As such, from this three main points, we have obtain a somewhat contradicting set of conclusions. For one set of images, our fakes proved no better than regular, real images. For the other one, however, it was indeed better than the reals. Additionally, earlier in our testing we

formulated the hypothesis of the actual number of matches being the smaller number between the results of two tests in which the matching set and the recognition set switch roles. However, the fact that FFHQ-small had less matches with our fakes than with CelebA but the fakes had far more matches (both absolute and percentage wise) with FFHQ-small than CelebA did goes directly against this line of thought.

As for the performance, there is a fact that could possibly be related to having success in one case and failing in another, and it is that the images in the FFHQ-small dataset are of a much higher resolution (1024x1024 pixels) than the CelebA ones (256x256 pixels). We have previously covered how FR algorithms extract and analyze facial features and landmarks in a way different from the usual human method of pattern recognition. As such, it is not far fetched to hypothesize that the landmarks captured in a face of high resolution will be different from ones. Moreover, in section 2.1.1.1, we covered HOG - which is the specific method used by the `face_recognition` module. Recalling the process, we know that each pixel is initially looked at individually, together with the four surrounding pixels; following that, each 16x16 segment becomes a singular part of the bigger representation of the face. As such, considering our scenario - one where we have one set of images with sixteen times the number of pixels - and consequentially segments - than the ones in the other set, the final "product" being analyzed by the recognition system will surely be different.

Additionally, we looked at research on the topic of the effect of image quality on face recognition tasks. Initially, from what we could see, while this issue was indeed present, thresholds like 32x32 [13] and 64x48 [45] seemed to be sufficient to avoid problems. However, more recent findings show that in state-of-the-art networks, performance is indeed correlated with the quality of the images, decreasing as the quality does [29]. With this information, we felt more confident in our judgment.

As such, in the end, for each respective module:

- **insightface** was the most robust one to our fakes, as they didn't even get past the face detection step.
- **face_recognition** identified all our fakes as real faces, and in the recognition step, gave us the most realistic results. We were able to observe that even in medium-to-small scale tests like the ones we performed, systems like this one are able to be breached by brute forcing. Also, some faces may be more prone to this type of attack than others. We noted that our fakes did not necessarily perform better than existing, real faces, but we were able to identify that the fact that they were of a higher image-quality made them more effective when matched against other faces of the same high quality.
- **FaceNet** also identified all our fakes as real faces, but when it came to the recognition step, despite our best efforts, the results we produced were impossible to consider realistic.

Chapter 5

Conclusion

When it came to image generation, our GAN obviously does not come nearly close to state-of-the-art ones, but that was to be expected. Honestly, the main negative point about ours is the fact that our training set ended up being so limited. This is why it was so important for us to be able to find a way of mass generating the images of abnormal faces. This element that would end up being the differentiating factor about our network wound up not really being present in the final result. It is understandable, however, that there is no way of generating the training data. After all, any generational method would follow the same baseline as a GAN, and would need images for training. This is the reason for the rigidity we noticed in some of the tools described briefly in chapter 3; they had been trained with only human faces and as such were not able to create anything that differed too much from that.

As for the tests with FR systems, the results could also be described as lackluster. While it is true that for two of the systems all of our fakes passed the FD step, for another one not even one of the faces was correctly identified as a face, while even some of the monkey images managed to trick it. And for the modules in which detection was effectively bypassed, as we saw in the actual recognition tests, our fakes were not better than regular pictures of faces. If we add the fact that our testing was done under conditions that in practice will not be this vulnerable, we can't really evaluate the outcome as something positive.

Overall, the main shortcoming was the fact that we could not achieve the goal we set initially of a GAN capable of mass generating abnormal faces. The failure in this step spread to the rest of the work, as even if the results of tests with those type of fakes were similar to the ones we ended up obtaining, they would be much more meaningful due to the uniqueness of our experiment.

However, this is not to say that all the work developed was for nothing. Despite all the negatives, the premises in which we based our work, as well as some of the results we obtained, are worthy of discussion and carry with them value for future works.

Maybe the biggest positive amongst all of what was done, the tests with our manually made abnormal faces showed us that, at least for some of the algorithms used in these systems, this approach works because of the way that the images are "looked at" by the computer. And as we've

seen, for FR to happen various landmarks are extracted based on their specific location on the face. As such, FD is not only a "preemptive check" for the FR step but rather a complementary part of the process.

If we take this into consideration, then we can see how being able to successfully create fakes that clear this step is already big progress towards the goal of breaking the authentication system. In our case, we were apparently able to break two detection models; how realistic this is when applied to real life authentication systems, which have many more nuance to them than simply two types of checks, is another question, but nevertheless - these were state-of-the-art open-source modules, and they were breached.

Analyzing the direct impact of the abnormal characteristics of our fakes in the feature extraction process was not feasible, as an image will have a great number of different ones. This is why we ran the tests we did, but unfortunately, since our fakes ended up not being that abnormal themselves, we can't really how different the results would be had we achieved our goal.

Another thing we would like to discuss is the fact that in a "real" setting, normally an actual camera would be used for image capture and as such an attack like this would not be possible. However, if we considered the already discussed trend of facial authentication being made available in people's personal devices and take another look this article [44], we can see how in reality there will be many opportunities to employ this attack in an actual setting. The author described how he used a modified smartphone to feed the authentication system a video he created, instead of the actual imagery captured by the camera. Coupled with the fact that he had unlimited attempts to match, the setup he had going on was extremely similar to ours.

In his case, the author had the advantage of attacking the account of someone he knew and thus he had access to imagery of the person in question to produce the attacking deepfakes. An attack like this is not necessarily what we did here and what we had initially in mind for the motivation of our work, but still, there's a lot to be taken from this attempt by this individual. The technology he used for generating his deepfakes - DeepfakeLab - has clear value for attacks on this type of systems, given that, as we've previously seen, the majority of them have developed anti-spoofing techniques as consequence of initial attacks consisting of using stand-still images; a live deepfake gets around this. He mentions how there is probably a human at the end making a "final" verification, in his case, and while this would obviously spell demise for attacks using abnormal faces, the mass scale use which is being envisioned for facial authentication systems is incompatible with manual verification. Possible, people will audit randomly selected attempts; but for the majority, computer systems are expected to do all of the work.

I believe this type of approach could find real success in non-specific attacks by utilizing these techniques in conjunction with master faces. As we have seen previously, master faces are designed to be able to match various faces in a FR database. However, in a practical case, a master face will never be able to break a system due to their static nature and anti-spoofing measures. On the other hand, an attack with a live deepfake will be able to bypass the anti-spoofing mechanisms, but requires the face of the target to be known in order to succeed. As such, these two approaches

complement each other perfectly given that the strengths of each one negate the weaknesses of the other.

But, as we discuss attacking, we should also equally discuss defending. A manual check by a human, as previously mentioned, could work in theory, but is faulty given the fact that humans are naturally prone to error and the quality of deepfakes just keeps on increasing to points where the naked eye has trouble identifying them. We can look at the live deepfake attack article as an example, as the author was capable of successfully bypassing the authentication system even when there were manual checks taking place. Additionaly, since the end goal is to automatize all of the processes, this type of defense mechanism does not fit the criteria. Modifications could be done to already existing anti-spoofing mechanism to introduce interactivity and randomness in order to combat pre-made live deepfakes, but this method could be circumvented with the development of a new approach to this type of attacks.

Popularized initially by the messaging app 'Snapchat', facial filters have become a standard in other social media applications such as 'Instagram' or 'TikTok'. The base for this filters is FD, and the process is somewhat similar to DFL - create a model of the face, and then apply another model - the filter - to it based on the landmarks detected. All of this, in real time. As such, if there was a way to alter a device so that this kind of filter would always be applied when the user used their camera, and some type of deepfake filter was utilized, a real-time deepfake would in theory be possible.

Realistically, DFL and the likings of that type of technology would have to evolve considerably, as in the state they are at the production of the fake takes a considerable amount of time. Nevertheless, that is not the main reason for why the modifications towards more real-time interaction in authentication systems is not the most pertinent defense mechanism. If we consider the whole of the attack vectors previously mentioned in this section, we can identify that they all relied on deepfake technology to succeed. This, in turn, means that a mechanism that focused strictly on detecting this fake, tampered content would prove effective at stopping various angles of attack that rely on it. By implementing an additional module on the FR pipeline, preferably at the very beggining of it, systems would be able do promptly identify fake content fed to them and thus shut down the login attempt immediately. Models like this already exist, although directed more at the field of digital forensics [19]. The concern for the impact of this additional check on the overall performance of the system is a valid one, but it is a tradeoff that is worthy of serious consideration, as well as the fact that it is to be expected that if this type of algorithms were to become more relevant and necessary, effort towards their development would result in optimization and better performance.

To finalize, looking at the objectives initially set on Section 1.1:

- While arriving at a state of a GAN that could mass generate the type of faces desired, in the end I would say the goal was not achieved given that the faces were not sufficiently different from regular, human faces and also not sufficiently different between themselves.

- Our fakes were indeed capable of fooling two of the three chosen FD modules; as such, this goal can be considered achieved.
- Finally, the behaviour of the FR systems used was nothing out of the ordinary when we compare the results of our fakes with results obtained using only dataset of real people. Nevertheless, the inference of this conclusion is enough to consider the initial goal achieved.

5.1 Future work

Overall, I believe this topic to be an extremely relevant one given the paradigm of the evolution of authentication systems, and the plans for their integration into unsupervised systems. By working in this project, I was able to learn about different subjects, which end up coming together to form this "ecosystem" we ended up analyzing. If I were to continue working on this topic, while still curious about how much of an impact the abnormal faces could have, I would change my focus away from those and into the type of attack we described above of using live deepfakes in conjuncture with master faces. When you consider the time and effort needed for creating just the fakes to feed to a GAN, it just is not justifiable when you could allocate those same resources into developing something that has more backbone to it than just an hypothesis. Additionally, from our tests we could tell that some face authentication systems can be naturally robust to that method - which will not be the case with the live deepfake method.

Bibliography

[1] About face id advanced technology. *Apple Support*.

[2] This person does not exist. https://thispersondoesnotexist.com/. Accessed: 2023-09-15.

[3] Ajith Abraham. Artificial neural networks. *Handbook of measuring system design*, 2005.

[4] Mayank Agarwal. Training gans with limited data, 2020.

[5] Osamah M Al-Qershi and Bee Ee Khoo. Passive detection of copy-move forgery in digital images: State-of-the-art. *Forensic science international*, 231(1-3):284–295, 2013.

[6] Loai Alamro and Nooraini Yusoff. Copy-move forgery detection using integrated dwt and surf. *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)*, 9(1-2): 67–71, 2017.

[7] Panagiotis Antoniadis. Using gans for data augmentation, 2021.

[8] Khurshid Asghar, Zulfiqar Habib, and Muhammad Hussain. Copy-move and splicing image forgery detection and localization techniques: a review. *Australian Journal of Forensic Sciences*, 49(3):281–307, 2017.

[9] Manpreet Bagga and Baljit Singh. Spoofing detection in face recognition: A review. In *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, pages 2037–2042. IEEE, 2016.

[10] Jeffery G Barnes et al. Fingerprint sourcebook-chapter 1. *History*, 2011.

[11] Qifang Bi, Katherine E Goodman, Joshua Kaminsky, and Justin Lessler. What is machine learning? a primer for the epidemiologist. *American journal of epidemiology*, 188(12): 2222–2239, 2019.

[12] Philip Bontrager, Aditi Roy, Julian Togelius, Nasir Memon, and Arun Ross. Deepmasterprints: Generating masterprints for dictionary attacks via latent variable evolution. In *2018 IEEE 9th International Conference on Biometrics Theory, Applications and Systems (BTAS)*, pages 1–9. IEEE, 2018.

[13] Bastiaan J Boom, GM Beumer, Luuk J Spreeuwers, and Raymond NJ Veldhuis. The effect of image resolution on the performance of a face recognition system. In *2006 9Th international conference on control, automation, robotics and vision*, pages 1–6. IEEE, 2006.

[14] Rahul Chauhan, Kamal Kumar Ghanshala, and RC Joshi. Convolutional neural network (cnn) for image detection and recognition. In *2018 first international conference on secure cyber computing and communication (ICSCCC)*, pages 278–282. IEEE, 2018.

[15] Shu-Yu Chen, Feng-Lin Liu, Yu-Kun Lai, Paul L. Rosin, Chunpeng Li, Hongbo Fu, and Lin Gao. DeepFaceEditing: Deep face generation and editing with disentangled geometry and appearance control. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2021)*, 40(4):90:1–90:15, 2021.

[16] Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A Bharath. Generative adversarial networks: An overview. *IEEE Signal Processing Magazine*, 35(1):53–65, 2018.

[17] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, volume 1, pages 886–893. Ieee, 2005.

[18] Richard Farebrother and Julian Champkin. Alphonse bertillon and the measure of man: More expert than sherlock holmes. *Significance*, 11(2):36–39, 2014.

[19] Sara Ferreira, Mário Antunes, and Manuel E. Correia. Exposing manipulated photos and videos in digital forensics analysis. *Journal of Imaging*, 7(7), 2021. ISSN: 2313-433X. doi:10.3390/jimaging7070102.

[20] Adam Geitgey. Machine learning is fun! part 4: Modern face recognition with deep learning, 2016.

[21] Jia Guo, Jiankang Deng, Alexandros Lattas, and Stefanos Zafeiriou. Sample and computation redistribution for efficient face detection. *arXiv preprint arXiv:2105.04714*, 2021.

[22] John Harvey, John Campbell, Stephen Elliott, Michael Brockly, and Andy Adler. Biometric permanence: Definition and robust calculation. In *2017 Annual IEEE International Systems Conference (SysCon)*, pages 1–7. IEEE, 2017.

[23] David Heun. Facial recognition tech is catching on with banks. *American Banker*.

[24] IBM. What are convolutional neural networks?, 2022.

[25] Anil K Jain and Stan Z Li. *Handbook of face recognition*, volume 1. Springer, 2011.

[26] Rubal Jain and Chander Kant. Attacks on biometric systems: an overview. *International Journal of Advances in Scientific Research*, 1(07):283–288, 2015.

[27] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8110–8119, 2020.

[28] Vahid Kazemi and Josephine Sullivan. One millisecond face alignment with an ensemble of regression trees. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1867–1874, 2014.

[29] Martin Knoche, Stefan Hörmann, and Gerhard Rigoll. Susceptibility to image resolution in face recognition and training strategies to enhance robustness. *Leibniz Transactions on Embedded Systems*, 8(1):01–1, 2022.

[30] Genia Kostka, Léa Steinacker, and Miriam Meckel. Between security and convenience: Facial recognition technology in the eyes of citizens in china, germany, the united kingdom, and the united states. *Public Understanding of Science*, 30(6):671–690, 2021.

[31] Sandeep Kumar, Sukhwinder Singh, and Jagdish Kumar. A comparative study on face spoofing attacks. In *2017 International Conference on Computing, Communication and Automation (ICCCA)*, pages 1104–1108. IEEE, 2017.

[32] Manav Mandal. Introduction to convolutional neural networks, 2022.

[33] Rolando Martins. Segurança de sistemas de dados. 2022.

[34] Huy H Nguyen, Junichi Yamagishi, Isao Echizen, and Sébastien Marcel. Generating master faces for use in performing wolf attacks on face recognition systems. In *2020 IEEE International Joint Conference on Biometrics (IJCB)*, pages 1–10. IEEE, 2020.

[35] Observador. Presidente da república promulga diploma que aprova novas formas de adesão à chave móvel digital, 2022.

[36] Ilesanmi Olade, Hai-ning Liang, and Charles Fleming. A review of multimodal facial biometric authentication methods in mobile devices and their application in head mounted displays. *2018 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, pages 1997–2004, 2018.

[37] Patrick Pérez, Michel Gangnet, and Andrew Blake. Poisson image editing. In *ACM SIGGRAPH 2003 Papers*, pages 313–318. 2003.

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

[39] David E Rumelhart, Richard Durbin, Richard Golden, and Yves Chauvin. Backpropagation: The basic theory. *Backpropagation: Theory, architectures and applications*, pages 1–34, 1995.

[40] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015.

[41] Pramila P Shinde and Seema Shah. A review of machine learning and deep learning applications. In *2018 Fourth international conference on computing communication control and automation (ICCUBEA)*, pages 1–6. IEEE, 2018.

[42] Rahul Thakur and Rajesh Rohilla. Recent advances in digital image manipulation detection techniques: A brief review. *Forensic science international*, 312:110311, 2020.

[43] TowardsDataScience. Convolutional neural network, 2019.

[44] Payment Village. How i used deepfakes to bypass security verifications in a bank, 2022.

[45] Jingdong Wang, Changshui Zhang, and Heung-Yeung Shum. Face image resolution versus face recognition performance based on two global methods. In *Proceedings of Asia Conference on Computer Vision*, volume 47, pages 48–49. Citeseer, 2004.

[46] Alfred C Weaver. Biometric authentication. *Computer*, 39(2):96–97, 2006.

[47] Xiaojun Yang. Artificial neural networks. In *Handbook of research on geoinformatics*, pages 122–128. IGI Global, 2009.

[48] Peng Zhou, Xintong Han, Vlad I Morariu, and Larry S Davis. Learning rich features for image manipulation detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1053–1061, 2018.

[49] Peihao Zhu, Rameen Abdal, Yipeng Qin, and Peter Wonka. Sean: Image synthesis with semantic region-adaptive normalization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5104–5113, 2020.