Department of Information Engineering and Computer Science

Bachelor's Degree in
Computer, Communication and Electronic Engineering

Final Dissertation

Towards efficient deepfakes detection with self supervised learning

Supervisor

Prof. Giulia Boato

Student

Lorenzo Marogna

Co-Supervisors

Andrea Montibeller, PhD

Francesco Paissan

Elisabetta Farella, PhD

Academic year 2023/2024

Acknowledgements

At the end of this work, I find it essential to dedicate a few lines to those who, without the slightest hesitation, contributed to the realization of this project. First and foremost, it is my duty to thank my supervisor, Giulia Boato, and my co-supervisor, Andrea Montibeller, for their patience and availability, and for having accompanied me throughout this important phase of my academic journey. A heartfelt thanks to the Bruno Kessler Foundation for giving me the opportunity to experience firsthand a large and important research reality, allowing me to develop the curiosity and interest that subsequently led me to the creation of this work. I also thank my advisors, Francesco Paissan and Dr. Elisabetta Farella, along with all the colleagues of the E3DA office, for their valuable advice and for having guided me within the FBK reality. A single thank you would not be enough to repay my family for all the affection and support received during my life's journey: thanks to Dad and Mom for allowing me to get this far and for having believed in me from the very beginning. Finally, I thank all the friends and classmates who, by living this adventure with me, have given added meaning to my academic path.

Contents

Summary

The concept of *Deepfakes*, which emerged just recently, commonly refers to fabricated digital content, created by either altering existing footage or generating entirely new visuals from scratch [31]. The term 'deep' signifies the use of deep learning algorithms, a powerful branch of Machine Learning (ML) with applications in computer vision, speech recognition, and various other fields [38]. Deep learning has achieved impressive results, sometimes even exceeding human experts. While deepfakes were once easier to detect, advancements in artificial intelligence (AI) and increased computing power have fueled the development of highly realistic image-generation techniques. This has led to the creation of user-friendly apps that can generate deepfakes in real time. The spread of these fabricated videos and images on social media raises significant concerns about misinformation and the trustworthiness of online information.

Spotting deepfakes is even more challenging in the face generation realm. Even trained human observers struggle, with studies like [4] showing human accuracy varies between 30% and 85% depending on the image, averaging around 62%. Moreover, the article [4] shows that for one image out of five the accuracy does not reach 50%, making a coin toss preferable to the human judgment. Results are similar in [31], where human performances have been tested against technologies from more and less recent years. Here, humans labeled synthetic images as real 68% of the time, while only correctly identifying real images 52% of the time. These findings highlight the urgent need for automated detection tools, especially in areas relying on authenticity, like user verification and combating the spread of misinformation.

To address this challenge, researchers created AI models that detect deepfakes by analyzing patterns and artifacts left behind during the generation process [20]. However, social media sharing often degrades these traces by compressing with private algorithms, specific to each platform [3]. This process makes detection difficult, hindering model performance depending on the platform the image has been compressed by.

To combat this issue in real-world social media settings, a dataset called 'TrueFace' [3] was created. It contains real and fake images from Facebook, Telegram, and Twitter, allowing researchers to train models on images subject to social media compression. Ongoing research focuses on developing techniques that enable models to generalize across different platforms, regardless of their sharing history.

Machine Learning on Tiny Devices The Internet of Things (IoT) is a vast network of interconnected smart devices. To effectively process the massive amount of data they produce, we need to shift intelligence from centralized cloud servers to the edge of the network, closer to devices and sensors. However, edge processing presents unique challenges due to resource constraints such as limited memory, communication bandwidth, and battery power, as shown in [1]. These limitations make it difficult to run traditional AI algorithms on peripheral devices like sensor nodes.

To overcome these challenges and bring AI capabilities to the edge, Tiny Machine Learning (TinyML) has emerged [24]. TinyML focuses on developing lightweight machine learning models and algorithms that can operate efficiently within the constraints of edge devices. This technology offers numerous advantages in processing sensor data on low-power embedded devices. Since no connectivity is required for inference, data can be processed in real-time directly on the device, avoiding the need to transfer raw data to third parties for processing. This eliminates delays due to waiting for unnecessary transmission times and significantly reduces latency.

To understand the impact of low response latency, consider the continuous monitoring of machines in industrial settings to predict problems and potential failures. This type of application can provide a timely response to damage, reducing maintenance costs, risks of failure, and downtime, while improving performance and efficiency.

Embedded devices that support TinyML algorithms require a very small amount of power (in the order of milliwatts), which allows them to operate for long periods without needing to be charged if equipped with a battery (see example in [40]), or with minimal energy consumption if powered. Research for reducing power requirements does not only aim at bringing AI to microcontrollers. The training of big Machine Learning Models from the most powerful companies in the world like Microsoft, Meta, or Google, requires a lot of power for their training and deployment. Currently, investors are readily backing research in this direction, and only major corporations can manage the financial burden. But as models demand ever-greater resources, the long-term sustainability of this approach remains unclear. Recent studies tried to benchmark the CO_2 emissions of big language models like BLOOM or GPT-3 [42, 34]. Additionally, the competition among big techs for the best model is aggravating the situation. For example, ChatGPT from OpenAI and Gemini from Google are both computationally expensive and potentially contribute to similar environmental impacts, despite fulfilling similar purposes. This redundancy could lead to increased energy consumption without significant advancements in functionality.

Moving deepfake detection to edge devices offers several benefits. First, it expands the capabilities of microcontrollers (MCUs) with a new, challenging task. Secondly, reducing the number of operations speeds up inference time (the time for the model to analyze and judge an image). Faster inference brings us closer to real-time applications. While frame-by-frame deepfake analysis for videos may not seem immediately necessary, it's certainly not a drawback.

A significant advantage is that lower deployment requirements allow devices to perform inference locally [40]. For instance, a small network on a smartphone could verify content authenticity without using bandwidth. More broadly, faster inference is crucial given the vast number of images uploaded online every second. Unofficial statistics suggest Instagram sees about a thousand image uploads per second, 34% of which involve human subjects. If the goal is to reality-check every image before posting, efficiency is paramount.

Methods and results This thesis addresses the critical need to adapt deepfake detection to real-world challenges. It focuses on two key strategies: enhancing model resilience against social media compression artifacts and enabling efficient deepfake detection on edge devices.

The first strategy leverages Self-Supervised Learning (SSL) to improve model robustness against distortions introduced by social media compression algorithms. By training models on unlabeled data to learn inherent structures and relationships, SSL enhances their ability to recognize deepfakes despite compression. Detailed exploration of SSL is provided in Section 1.1.1.

Experimental results demonstrate that SSL significantly mitigates performance degradation on compressed images, narrowing the accuracy gap between pristine and manipulated content. However, this increased robustness may come at the cost of overall detection accuracy, highlighting the need for careful consideration of trade-offs based on specific application requirements.

The second strategy investigates lightweight neural network architectures, MicroNet [33] and PhiNet [40], for on-device deepfake detection. These architectures prioritize a small parameter count and reduced computational complexity, essential for deployment on resource-constrained edge devices.

Research findings show that under controlled conditions, these compact models can achieve accuracy comparable to larger architectures like ResNet50 [22]. For instance, MicroNet's accuracy on pre-social media images is only marginally lower than ResNet50's, while utilizing a fraction of the parameters. This finding holds significant promise for enabling real-time deepfake analysis directly on user devices, thereby potentially mitigating the spread of misleading content before it reaches wider audiences.

Thesis Structure This thesis is organized into four chapters.

Chapter One provides background information on machine learning paradigms, deepfake generation techniques, state-of-the-art deepfake detection methods, and edge architectures, including PhiNet [40] and MicroNet [33].

Chapter Two presents the research methodology. It begins by establishing a baseline using a standard machine learning approach for comparison with SSL results. Subsequently, the SSL technique,

specifically Contrastive Learning using the SimCLR framework, is described in detail.

Chapters Three and Four provide the experimental setup, enabling reproducibility, and the research findings, respectively. The results are accompanied by an analysis to understand how this work can advance real-world deepfake detection.

1 Related Works

This chapter provides an overview of deepfakes, from creation to detection. We begin by establishing foundational knowledge of key machine learning paradigms (supervised, unsupervised, and self-supervised learning) and explore some popular techniques used in deepfake generation.

Next, we critically examine state-of-the-art deepfake detection methods, highlighting their strengths and weaknesses. The chapter then explores the world of edge architectures, focusing on the challenges of implementing TinyML models on edge devices. We conclude by introducing Phinet and Micronet, the two edge architectures central to this work.

1.1 Background

1.1.1 Learning Paradigms

This section will explore some of the main learning paradigms in the machine learning realm: Supervised Learning, Unsupervised Learning, and Self Supervised Learning.

Supervised learning is a learning paradigm where an algorithm is trained on a labeled dataset. This means that the dataset includes both input data (features) and the corresponding correct output labels. The goal is for the algorithm to learn the relationship between the input and output data to make accurate predictions on new, unseen data. This paradigm suits very well classification problems like Spam Filtering, Image Recognition, and Fraud Detection, where the correct class of the output is represented by the label. This clear dependence between the input and the expected output usually results in high, measurable performance for the model. However, labeled datasets are typically harder to find because they often need human labor or expertise to classify elements by hand. For example, labeling medical images or classifying complex financial transactions need specialists who can accurately interpret the data.

It follows that research also developed some techniques to leverage unlabeled data. With Unsupervised Learning, there are no pre-defined target outputs or labels associated with the input data. Instead, the algorithm aims to discover hidden structures, patterns, or relationships within the data autonomously. This paradigm can be used in multiple use cases. For example, identifying different customer groups of a shop based on their purchase behaviors is a clustering problem that often has a big unlabeled dataset. Another class of problem addressed by unsupervised learning can be anomaly detection, which consists of identifying outliers data points from a bigger set. Ultimately, generative adversarial networks (GANs) harness unlabeled datasets to learn how to generate images from random numbers. These networks have grown a lot in recent years, especially in the generation of synthetic faces. The TrueFace dataset presented in 3.1 collects fake images generated from these networks. The creation of deepfakes will be better explored in 1.1.2.

Alternatively, Self-supervised learning (SSL) is a machine learning technique where a model learns meaningful representations of data without the need for explicitly labeled examples. The core idea is to design tasks that the model can learn from the unlabeled data. These tasks are crafted to teach the model valuable features about the data indirectly. They don't directly correspond to the final goal (like deepfake detection in this case) but help the model develop useful skills. Image reconstruction can be an example, where the model sees a corrupted image and is tasked with reconstructing the original version. This helps the model learn to understand the underlying structure and relationships within the image. The knowledge gained from the pre-trained model (feature representations) can then be applied to new tasks, such as the deepfake detection problem. The model can be fine-tuned on the labeled TrueFace dataset to adapt its learned features to the specific classification task. Both self-supervised and unsupervised learning involve training machine learning models without explicit human-labeled data. However, they differ in how they utilize the available data because self-supervised learning can be seen as a form of unsupervised learning where the model generates its supervisory

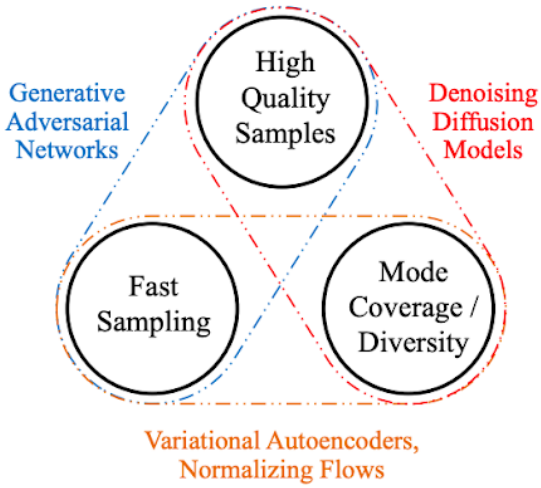


Figure 1.1: The Generative Learning Dilemma: currently, no technique achieves all the 3 requirements. GANs lack diversity and mode coverage, Diffusion Models require high generation time and Autoencoders do not reach a sufficient quality.

information, while the standard approach mainly aims at specific tasks like clustering, dimensionality reduction, or anomaly detection.

1.1.2 Deepfakes generation

While the term *deepfake* usually refers to media content generated from deep neural networks and representing a person whose face was swapped with someone else's likeness, in this work the term will refer to images of faces also generated from scratch. Deepfakes have evolved significantly and are now incredibly realistic and challenging to detect [31]. For wide adoption in real-world applications, generative models should ideally satisfy three key requirements:

- **High-quality sampling**: To interact directly with users, images must be detailed and of high quality to be indistinguishable from natural ones.
- **Mode coverage and sample diversity**: If the training data contains a vast amount of diversity, the model should be capable of capturing such diversity without sacrificing generation quality.
- **Fast and computationally inexpensive sampling**: Many interactive applications require low latency sampling, for example, real-time image editing.

These requirements currently constitute the *Generative learning dilemma* [49], since there is no generative model yet that can address all three points at the same time. However, some ongoing research in the generative models' realm is focused on tackling this problem [49]. While variational autoencoders shown in Figure 1.1 are not so popular for the generation of synthetic faces due to their lack of quality, two prominent techniques used for this scope are Generative Adversarial Networks (GANs) and Diffusion Models (DMs), which we will discuss in this section.

Generative Adversarial Networks

A Generative Adversarial Network (GAN) is a class of machine learning frameworks that employs two neural networks in a competitive setting to facilitate generative modeling. Introduced by Ian Goodfellow et al. in 2014 [19], GANs have since emerged as a prominent approach to generative AI. The GAN framework comprises two primary components: a *generator* and a *discriminator*. The generator is a network that synthesizes new data instances that resemble a training dataset. A generator aims at producing outputs that are statistically indistinguishable from real samples. On the other side, the discriminator is used as a binary classifier to detect whether a datapoint is the

Figure 1.2: from [29]. Instance normalization causes water droplet -like artifacts in StyleGAN images. These are not always obvious in the generated images, but it is a systemic problem that plagues all StyleGAN images.

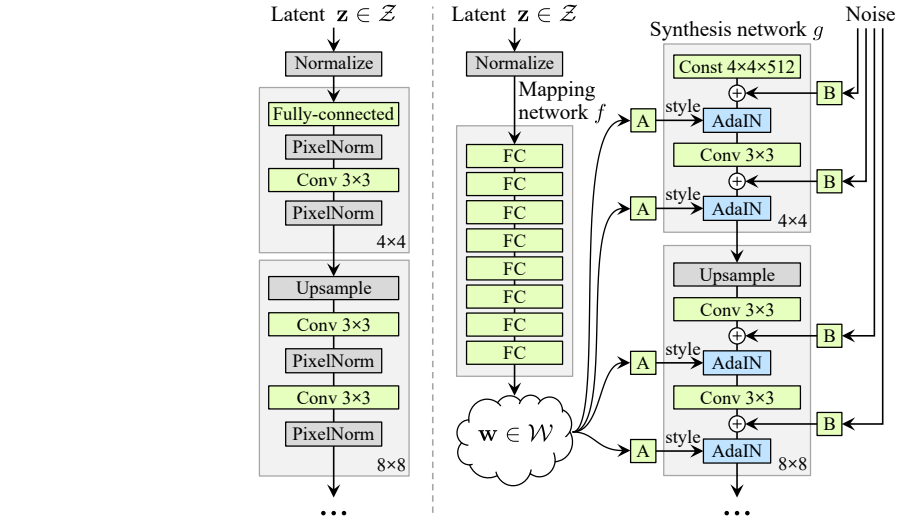
output of the generator or an instance of the dataset. It acts as a 'critic', providing feedback to the generator regarding the authenticity of its outputs. The training process involves an iterative game wherein the generator and discriminator engage in a constant feedback loop. The generator strives to produce increasingly realistic data to deceive the discriminator, while the discriminator continually refines its ability to distinguish between real and synthetic data. Through this competitive interplay, both networks progressively improve their performance, ultimately converging towards a state where the generator produces highly convincing outputs that the discriminator struggles to identify as fake. At this stage, the generator is used to produce new samples, while the discriminator is discharged. Among the many types of GANs available, StyleGAN [27] is an architecture developed by Nvidia researchers. First introduced in 2018, it has gained significant attention for its ability to generate high-quality, realistic images, particularly of faces. StyleGAN is the first GAN architecture capable of generating images with a resolution of 1024 pixels. Since its initial release in 2019, StyleGAN has undergone several improvements and iterations. StyleGAN2 [29] addressed some of the artifacts present in the original model and showed in Figure 1.2, while StyleGAN3 [26] further enhanced the model's ability to generate consistent and realistic images.

The particularity of the family of networks relies on their capability to control the style of generated images. The concept of style comes from a deep learning technique called style transfer, which combines the content of an image with the artistic style of another one, creating a result that blends both elements. In style transfer, this is obtained by utilizing two different CNNs, one for extracting the content representation and another one for style representation. The results of these networks can be then reassembled, interchanging the style of an image with the content of another one. Although this is not the technology used in StyleGANs, the concept of style is used similarly. These networks are leveraging a *style vector*, that is used as a way to gain control over style attributes like hair color, adding wrinkles, or modifying facial expressions. The style vector is the result of an initial projection of the classical random noise z into an intermediate latent space $\mathcal{W}$ with a mapping network f so that the style vector $w = f(z), z \in \mathcal{Z}, w \in \mathcal{W}$. The style vector can be passed to different layers of the traditional GAN network to implement a certain style characteristic. In this phase, a scaled and shifted version of the style vector is passed using an operation called adaptive instance normalization (AdaIN), resulting in fine-grained control over features. Moreover, StyleGANs introduce stochastic variation by adding random noise to the style vector at each layer. This randomness contributes to the diversity of the generated images and allows for the creation of unique variations of the same style. An example of the StyleGAN architecture compared to a standard GAN is presented in Figure 1.3.

Diffusion Models

Diffusion models (DMs) are another type of generative model in machine learning that recently became very popular for the creation of high-quality data such as images, videos, audio, and more. Thanks to their versatility, DMs have already been applied to a variety of generation tasks, such as image, speech, 3D shape, and graph synthesis. The operation of a diffusion model is based on two main phases: the forward diffusion process and the reverse diffusion process.

The forward diffusion process maps data to noise by gradually perturbing the input data. This is formally achieved by a simple stochastic process that starts from a data sample and iteratively



(a) Traditional (b) Style-based generator

Figure 1.3: from [28]. While a traditional generator [25] feeds the latent code through the input layer only, in [28] they first map the input to an intermediate latent space $\mathcal{W}$, which then controls the generator through adaptive instance normalization (AdaIN) at each convolution layer. Gaussian noise is added after each convolution, before evaluating the nonlinearity. Here "A" stands for a learned affine transform, and "B" applies learned per-channel scaling factors to the noise input. The mapping network f consists of 8 layers and the synthesis network g consists of 18 layers — two for each resolution ($4^2 - 1024^2$). The output of the last layer is converted to RGB using a separate 1×1 convolution, similar to Karras et al. [25].

Figure 1.4: Diffusion model processes moving to and from data and noise

generates noisier samples using a simple Gaussian diffusion kernel. That is to say, at each step of this process, Gaussian noise is incrementally added to the data, until the original image is completely transformed into pure noise.

The second process is a parametrized reverse process that undoes the forward diffusion and performs iterative denoising. This process represents data synthesis and is trained to generate data by converting random noise into realistic data. It is also formally defined as a stochastic process, which iteratively denoises input images using trainable deep neural networks.

Forward and reverse processes often use thousands of steps for gradual noise injection and during generation for denoising.

Although Diffusion Models are not famous for the generation of synthetic faces in particular, they are generally capable of outperforming older GANs. However, Diffusion Models are expensive in terms of time for generating a new sample, due to the repetitive denoising steps.

One of the most well-known diffusion models is DALL-E 2, developed by OpenAI, which can generate images from textual descriptions. This model has demonstrated an extraordinary ability to understand natural language and translate words into realistic and creative images.

1.2 Deepfake detection

The detection of synthetic images, especially those generated by GANs and DMs, relies heavily on identifying subtle traces or artifacts embedded in the image during the generation process. These 'forensic traces', whether visible anomalies or latent patterns in the frequency domain, serve as fingerprints that distinguish synthetic images from those captured by real-world devices. This section surveys state-of-the-art methods that leverage these traces, exploring both spatial and frequency domain analyses, and delving into the challenges of generalization and robustness in the face of evolving generation architectures and image manipulations.

Especially when the generative models were less sophisticated, some works focused mainly on the inability of such generators to perfectly reproduce the high-level semantic features of natural images [21]. They can produce visible artifacts, such as chromatic anomalies (Figure 1.2) or lack of natural symmetries. For example, generated faces may have differently colored left and right eyes, unconvincing eye reflexes, or irregular pupil shapes [21]. For DMs [12] in particular, it was noted that the lack of explicit 3D modeling of objects and surfaces causes asymmetries in shadows and reflected images. Furthermore, global semantic inconsistency can be observed, to some extent, in lighting. However, new DM or GAN-based methods can overcome these limitations [48] and generate images that satisfy all necessary semantic constraints, be it lighting, perspective, or any other aspect.

For this reason, most state-of-the-art detectors of fake images rely on traces that are invisible to the human eye [20]. Visually perfect images, with no evident semantic inconsistency, can be distinguished from real images based on the traces that are inherent in the generation process. Any method used to create synthetic visual data embeds some peculiar traces [8]in their output images, that are related to the actions taken in the generation process. Moreover, each generative architecture is characterized by its peculiar traces, thereby allowing also for source attribution. The presence and distinctiveness of such traces have been proven by extracting a sort of spatial-domain artificial fingerprints, but also through frequency-domain analyses showing that the upsampling operation performed in most GAN architectures gives rise to clear spectral peaks. The complex processing pipeline needed to generate synthetic images inevitably leads to the introduction of digital artifacts that characterize the specific architecture and significantly differ from those typical of traditional acquisition devices.

When trying to exploit artifacts in the spatial domain, different techniques were tested. For example, [37] leverages the fact that GANs produce only a limited range of intensity values, and do not generate saturated and/or under-exposed regions. Likewise, [32] exploits GANs failure to accurately preserve the natural correlation among color bands. Consequently, to extract discriminative features for detection, the chrominance components are high-pass filtered and summarized by their co-occurrence matrices. A co-occurrence matrix is a statistical tool used for analyzing the texture of an image. It quantifies how often pairs of pixels with specific values and in a specific spatial relationship occur in an image. It is worth noting that co-occurrences of high-pass filtered versions of the image are popular tools in image forensics since invisible artifacts are often present in the high-frequency signal components [47]. Co-occurrence matrices extracted from the RGB channels are also used in [16] as input of a CNN.

Alternatively, the frequency domain provides useful information for detection, since GAN images display clear traces of their synthetic origin in the Fourier domain. The detector proposed in [51] exploits the presence of spectral peaks caused by the upsampling operations routinely performed in most GAN architectures. Frequency-domain analysis is carried out also in [17] to study the presence of artifacts across different network architectures (Figure 1.5), datasets, and resolutions. Again, these artifacts are used to tell apart generated images from real ones. In particular, a CNN-based classifier is trained with Fourier spectra taken from both real images and their synthetic versions obtained through an adversarial autoencoder. Likewise, in [14] it is shown that GAN images do not faithfully mimic the spectral distributions of natural images. A simple detector is proposed that takes the energy spectral distribution as an input feature. The authors also propose a spectral loss to use during GAN training to limit the appearance of spectral artifacts.

However, the fully supervised approaches described above are all very effective when the GAN images under test come from a model that is also present in training, but they fail to generalize to data generated by new unseen models. Therefore, some methods have been proposed recently

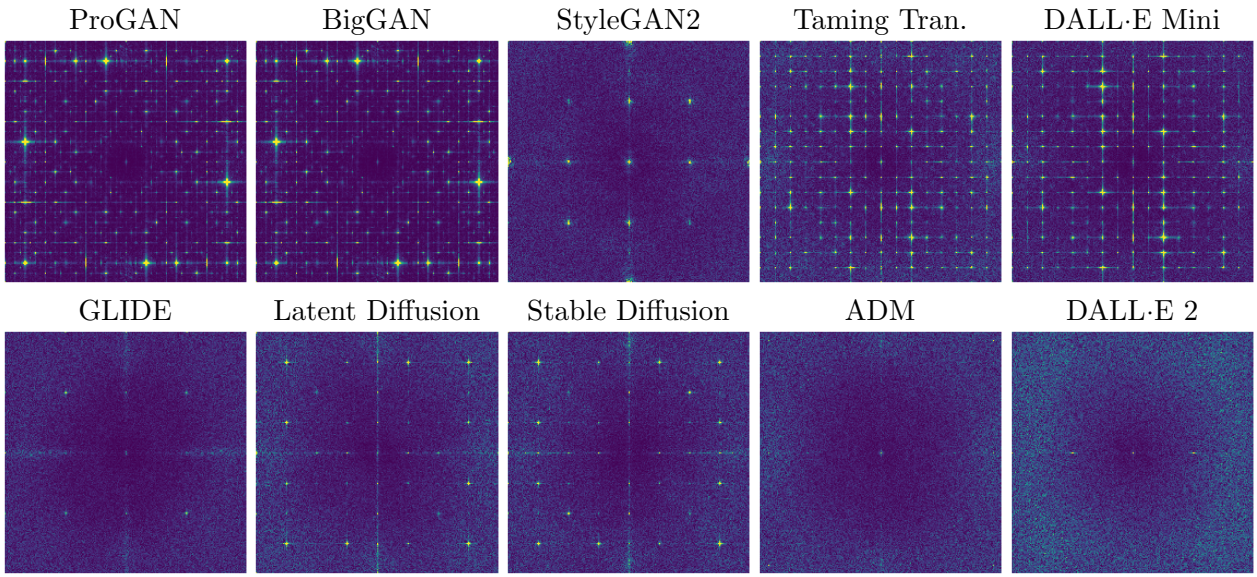


Figure 1.5: from [8]. Fourier transform (amplitude) of the artificial fingerprint estimated from 1000 image residuals. Top row: from left to right ProGAN [25], BigGan [5], StyleGAN2 [30], Taming Transformers [15], DALL·E Mini [11]. Bottom row: GLIDE [39], Latent Diffusion [44], Stable Diffusion [45], ADM [12], DALL·E 2 [43]

.

to address this problem. In [9, 13] few-shot learning strategies are proposed, with an autoencoder-based architecture, to adapt to new manipulations with just a few examples. In [35], instead, an approach based on incremental learning is used. Despite the improved generalization, these methods still need some examples of the new GAN architecture, which is not always realistic. A different solution is proposed in [50]. The idea is to carry out augmentation by Gaussian blurring to force the discriminator to learn more general features. A similar approach is followed in [48] where a pre-trained ResNet50 is further trained with a strong augmentation based on compression and blurring. Experiments show that, even by training on a single GAN architecture, the learned features generalize well to unseen architectures, datasets, and training methods. A different perspective is taken in [6] where a fully convolutional patch-based classifier is proposed. The authors show that by focusing on local patches rather than global structures, they can achieve better performance.

Based on these concepts, several promising CNN-based detectors of GAN images have been developed. However, the problem is far from being solved. On one hand, new and more sophisticated generation architectures are proposed by the day, some of them, like StyleGAN3 [26], aiming explicitly at minimizing the presence of these undesired traces. On the other hand, detectors suffer a significant performance drop when image quality is impaired [3], as always happens on social networks, which routinely apply some resizing and compression operations. This is because forensic traces are delicate and can be easily removed even by non-malicious processing steps such as compression, resizing, or shearing.

The reproducibility of these techniques on resource-constrained platforms remains an open question and a central focus of this work. To address this, the subsequent section delves into the architecture and functionality of two prominent families of networks designed for such platforms: PhiNets [40] and MicroNet [33].

1.3 Edge architectures

The Internet of Things (IoT) is a world of interconnected smart devices, both mobile and fixed. To handle the vast amount of data they generate, we need to move intelligence away from centralized clouds and closer to the devices themselves, at the network's "edge." However, this edge processing presents a challenge. Resource limitations like low memory, restricted communication capabilities, and limited battery budget make running traditional AI algorithms on these peripheral devices, like

sensor network nodes, quite difficult. This necessitates innovative methods to bring AI to the edge. In this section, we will analyze the main limitations in terms of resource-constrained architectures, analyzing the two architectures that will be used in this work: an MCU-friendly, scalable backbone called PhiNet[40] and a famous architecture for TinyML called Micronet [33].

1.3.1 Limitations of MCU

When working on microcontrollers, resources are particularly precious. For example, an STM32H743 MCU has $2MB$ of internal Flash and $1MB$ of RAM, and the input voltage is usually around a few Volts. These requirements are important and depend on the microcontroller. When developing an edge architecture that needs to be deployed on an MCU, these are the main factors that will be used.

- **MAC** stands for Multiply Accumulate and it is indeed the amount of operations required by the model to perform the inference step. It is one of the default operations that a processing unit is capable of doing and is taken as a reference because it constitutes the bulk of the computational workload in the case of convolutional neural networks, they are used to multiply the weights of a kernel with activation values and accumulating the results.
- **RAM usage** is a major constraint when deploying neural networks on edge devices. It directly relates to the size of the largest tensors required during the network's inference step. The chosen inference engine also plays a role. However, relative improvements in RAM usage for a fixed engine remain significant, making comparisons between operators valid and generalizable across different engines.
- **Parameter Count** imposes a hard constraint on which platforms can host a specific neural network. This is because parameter count directly correlates to the amount of FLASH memory used.
- **Arithmetic intensity** measures the average number of arithmetic operations performed per byte loaded into a CPU registry. For a fixed amount of computation, higher arithmetic intensity translates to lower memory access costs. This makes the operation less memory-bound and, ultimately, more efficient.

1.3.2 PhiNets

Phinets' architecture consists of a class of scalable networks optimized for deep-learning-based image processing on resource-constrained platforms [41]. Particularly for the various tasks of image classification, real-time object detection, and tracking, the convolutional networks of this family are distinguished by the structure of the last layer (*classifier* or *detection head*), while maintaining a common *backbone*, which is the set of convolutional layers used for feature extraction. To make these models scalable, it is essential to optimize the network's backbone as it is responsible for most of the resource consumption during model inference. The techniques used for this purpose mainly involve the exploitation of efficient convolutional blocks and the use of an architecture that can be modified through appropriate scaling factors.

PhiNets architectures are particularly optimized in terms of computational requirements thanks to the use of a specific type of convolution in each layer. In fact, *inverted residual blocks* have proven to be particularly effective for this purpose since their introduction in MobileNetV2 [46]. This kind of block is a different version of a famous convolutional block called *residual block*. These residual blocks are named after the *skip connection* operation that links the input and output of the convolutional operation, which propagates residuals of the input representation. This is done to allow the reuse of input features in the following layers and address the problem of the *vanishing gradients*. It was noted that the gradients get smaller when the network is deeper, making it difficult for the model to learn effectively. The skip connection allows gradients to flow directly through the block, making training deeper networks easier. The basic operation for the residual block involves reducing the number of channels in the entering feature map through an inexpensive convolutional operation, followed by the expansion of the number of channels (returned to the initial value), and the application of the skip connection, which leaves residues of the input representation in the output feature map. As an

alternative, *inverted residual blocks* were introduced. The operating procedure is very similar to that of traditional residual blocks, with the difference that the number of channels in the input representation is initially expanded to extract more detailed information, and then compressed at the end of the block to obtain an output representation with the same dimensionality as the input. Inverted residual blocks are particularly suited for edge architectures thanks to the use of special convolutions called point-wise and depth-wise convolution. Given a tensor with C channels that needs to be mapped to a tensor with C' channels, a standard convolution uses a kernel of size k^2C and the whole mapping operation needs k^2CC' parameters. This is the kind of convolution that is performed, for example, in the residual blocks of a ResNet50. On edge architectures, the standard convolution is 'factorized' into a point-wise and a depth-wise convolution. The point-wise convolution addresses inter-channel communication on a single pixel at the time and allows the model to expand or compress the number of channels. On the other side, the depth-wise convolution only focuses on a single channel of the input tensor at the time, but it lives in the spatial domain. With this method, the kernel has a size of k^2 (k^2C operations) for the depth-wise convolution and C (CC' operations) for the point-wise convolution. The reduction in complexity is the reason why inverted residual blocks are so much better than residual blocks in terms of number of operations and parameters. The downside of this approach lies in the fact that a 2D convolution can operate on spatial resolution on different channels simultaneously, which usually leads to higher performances. To this basic structure of the inverted residual block, the PhiNets architecture adds an operation called *squeeze and excitation* before the final compression of the number of channels. This function helps to emphasize the information present in the channels of the input tensor before performing the compression.

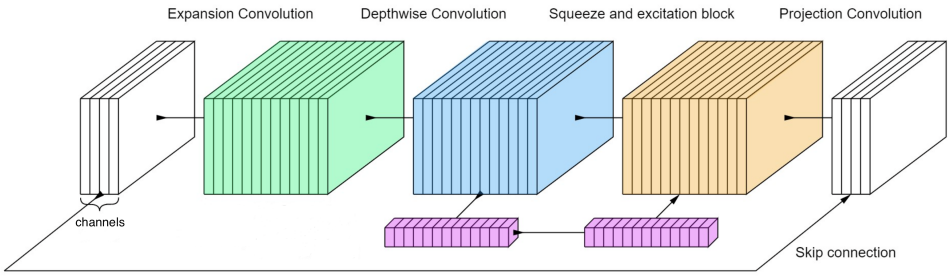


Figure 1.6: An overview of the PhiNets convolutional block structure. First, the number of channels is increased with a point-wise convolution, followed by a depth-wise convolution and SE block. Finally, a second point-wise convolution connects to the low-dimensionality bottleneck block.

PhiNets' architectures adopt the use of certain scaling factors to adapt the structure and resource consumption of the network to the specific platform used. Specifically, PhiNets use 4 different hyper-parameters to modify the model structure:

- α (*width factor*): Adjusts the network's width by changing the number of output channels from each layer.
- β (*shape factor*): Change the spatial resolution of the images in each convolutional layer.
- N: is the total amount of layers, which determines the depth of the network.
- t_0 (*base expansion factor*): Controls the number of channels in the expanded representations during the execution of each inverted residual block. Specifically, the *expansion factor* t varies linearly with the depth of the network with the formula $t = t_0(\frac{(\beta-1)n+N}{N})$, where n in the index of the convolutional block we are referring to.

By leveraging these parameters, it is possible to have full control of the model's structure and resource usage (Figure 1.7). In particular, the number of MAC depends on the resolution of the image $w \times h$, on the depth of the network (N), and from its width, which scales quadratically with α. The number of parameters is determined from the dimension of the kernels in the convolutional blocks, which increases linearly with N and β.

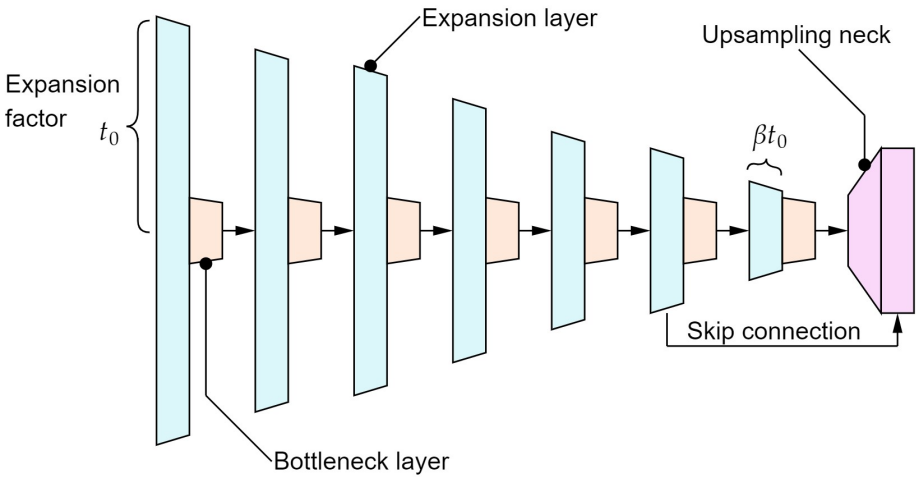


Figure 1.7: PhiNets body structure. Each hyperparameter α, β, t_0, and N influences the general structure.

1.3.3 MicroNet

Micronet is another family of networks that aims at addressing the problem of substantial performance degradation when minuscule computational cost is required [33].

This is achieved with two technical features. Firstly, micro-factorized convolution is a method that factorizes a convolution matrix into low-rank matrices, allowing the system to perform fewer operations. This happens both for point-wise and depth-wise convolutions, by then combining the two techniques. In both cases, the general formula is describing the initial kernel $\boldsymbol{W}$ as a product of three smaller matrices:

$$\boldsymbol{W} = \boldsymbol{P}\boldsymbol{\Phi}\boldsymbol{Q}^T \tag{1.1}$$

For example, the $k \times k$ depth-wise convolution kernel can be factorized into a $k \times 1$ and a $1 \times k$ kernel. This follows Eq. 1.1, with per channel $k \times k$ kernel matrix $\boldsymbol{W}$, $k \times 1$ vector $\boldsymbol{P}$, $1 \times k$ vector $\boldsymbol{Q}^T$ and $\boldsymbol{\Phi}$ a scalar of value 1. This low-rank approximation reduces the computational complexity from $\mathcal{O}(k^2C)$ to $\mathcal{O}(kC)$.

Additionally, a novel activation function called *Dynamic Shift Max* is utilized. The name *dynamic* refers to the fact that the function utilizes weights that depend on the input, contrary to static networks. This function is designed to enhance non-linearity through the following steps:

- **Circular shifting channels**: DSM takes an input feature map and creates multiple shifted versions of it by circularly shifting groups of channels.
- **Dynamic fusion**: These shifted feature maps are then combined with the original input feature map in a way that's learned by the network during training and depending on the input. It consists of a sequence of average pooling, two fully connected layers, and a sigmoid layer, as in the Squeeze-and-Excitation block.
- **Maxing Out**: Finally, DSM applies the maximum operation across all the fused feature maps, selecting the most salient features from each.

By leveraging these two techniques it is possible to create *Micro-Blocks*, which are a combination of depthwise convolutions, pointwise convolutions, and the Dynamic Shift Max activation function. Micro-Block A, B, and C are then created, and they constitute the building block for MicroNet. More details can be found on the official paper [33].

Given the potential of PhiNet and MicroNet, this work explores their capability to perform comparably to ResNet50 [22] in the task of deepfake detection. Addressing this would allow for faster inference time and lower power usage, enabling real-time image validity checks. Additionally, a self-supervised learning technique aims to enhance the performance of these models on compressed images, where standard models often struggle.

2 Methods

In this study, we explored two distinct approaches for deepfake detection and image analysis. The first approach, detailed in section 2.1, adopts a traditional Supervised Learning methodology. This involves training models on a dataset of Pre-Social images, followed by a crucial fine-tuning step on Post-Social media data. This methodology, while well-established, will be critically evaluated to assess its strengths, weaknesses, and suitability for deployment in real-world, uncontrolled scenarios. Sequentially, section 2.2 introduces a more innovative approach. Here, we explore the potential of Self-Supervised Learning, specifically utilizing the SimCLR framework proposed by Chen et all [7]. This approach aims to circumvent the need for Post-Social fine-tuning by leveraging the inherent structure of Pre-Social data during the training process. We will investigate the effectiveness of this self-supervised approach in comparison to the traditional supervised method.

To ensure a comprehensive evaluation, we will implement both approaches across a variety of model architectures. This will include the two cutting-edge architectures presented in Section 1.3, as well as a standard ResNet50 model, whose architecture is depicted in Figure 2.1. This comparative analysis will provide valuable insights into the performance and generalizability of each approach across different model complexities and architectures.

By exploring these diverse methodologies and model architectures, we aim to contribute to the ongoing research in deepfake detection. We believe that the findings of this study will be of interest to both researchers and practitioners working in this field.

2.1 Pre-Social Supervised Learning and Post-Social Fine Tuning

This section explores a two-stage approach utilizing supervised learning on pre-social content and fine-tuning for analyzing post-social images. The pipeline involves leveraging pre-trained models and adapting them to the specific task of distinguishing real from fake images on social media platforms.

The first stage begins with utilizing deep learning architectures, such as ResNet, PhiNet, or XiNet, pre-trained on a large-scale image dataset like ImageNet. ImageNet contains millions of images labeled by humans, allowing the model to learn generic image representations, that can be re-used for image processing tasks [2, 23]. During pre-training, the model develops a backbone that encodes an image into a *feature vector*, followed by a head that classifies the image into one of the thousands of categories within the dataset. The pre-training phase focuses on building a strong foundation for understanding visual content. After this stage, the head is discarded, and the backbone's parameters are frozen. Freezing prevents the backbone's knowledge from being overwritten during subsequent training stages, which is an issue known as catastrophic forgetting [18, 36].

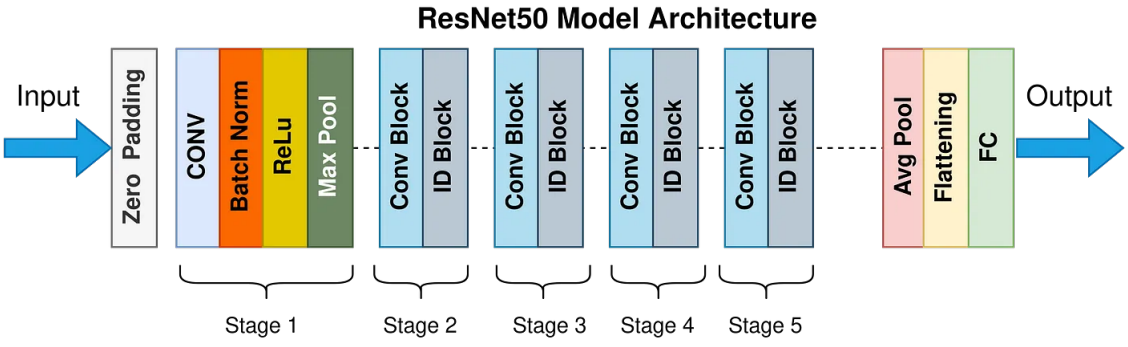


Figure 2.1: The architecture of a standard ResNet50. The fully connected (FC) is removed and a projection head or a new layer is used instead, depending on the method we are testing.

In traditional machine learning, a model is usually trained from scratch for each new task. Sometimes, *Transfer Learning* [52] can be used to reduce training time and achieve higher performances. This technique takes advantage of the fact that many tasks share common features or patterns, and it consists of harnessing a model that has been trained on one task to re-purpose it for use on a second, related task. In this approach, models were pre-trained on ImageNet for image classification and their encoding skills are transferred for the deepfake detection task. The second stage involves using the frozen backbone as a foundation for a new head specifically designed for the task of classifying real and fake images from TrueFace. Since the backbone remains frozen, this stage concentrates solely on training the new head for binary classification based on the feature vectors extracted by the backbone. Here, the training is performed on pre-social images, which have not been shared on any social media platform.

Once the head is trained, the model is evaluated on images shared on platforms like Twitter, Facebook, and Telegram. This evaluation step assesses how the image compression processes employed by these platforms might impact the model's performance. This step is crucial for understanding the potential limitations of the model when applied to real-world social media scenarios.

The potential lack of generalization across all social media platforms is acknowledged as a limitation. To address this, a final step involves creating copies of the trained head and fine-tuning them on post-social image datasets specific to each platform (Twitter, Facebook, Telegram). This fine-tuning process aims to bridge any performance gaps identified during the evaluation phase and enhance the model's ability to handle the nuances of images from each social media platform. This way, we create an upper bound for performance in the three different settings, which can be used as a point of comparison for different techniques.

Loss function Since deepfake detection is a binary classification problem (real or fake), the appropriate loss function is Binary Cross Entropy (BCE) loss. It measures the difference between the model's predicted probabilities for the real class and the actual labels (0 for fake, 1 for real) in the training data. Given a data point, let y be the true binary label, and let $\hat{y} = f(x)$ be the predicted probability for the true class y outputted by the model (between 0 and 1) for a given input x. The BCE Loss for this data point is:

$$\mathcal{L}(\hat{y}, y) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y}) \tag{2.1}$$

The term $\log(\hat{y})$ represents the contribution to the loss for the true class. A higher predicted probability leads to a lower loss for this term in the case of a positive label. Symmetrically, the term $\log(1 - \hat{y})$ represents the contribution to the loss for the false class. Although the BCE loss function expects the model's predicted probability to fall within the range $[0, 1]$, the model's output is initially in the form of a *logit*, a real number on the infinite number line. To bridge this gap, the sigmoid function is introduced. This function acts as a translator, transforming the logit values into probabilities suitable for the BCE loss. The sigmoid function is defined as follows:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.2}$$

For example, a logit value of 0 gets mapped to 0.5 by the sigmoid function. This represents a state of complete uncertainty for the model, where it can't confidently say if the image is real or fake. As the logit value increases towards positive infinity, the sigmoid function approaches 1. This signifies the model becoming increasingly confident that the image is real.

2.2 Self Supervised Learning with Supervised Fine Tuning

The second approach tackles deepfake detection on social media by prioritizing robustness against image compression from the outset, potentially eliminating the need for social media-specific fine-tuning later.

After harnessing transfer learning similarly to the last approach, the second stage introduces self-supervised training using the pre-trained backbone. It does so using a pretext task called *contrastive*

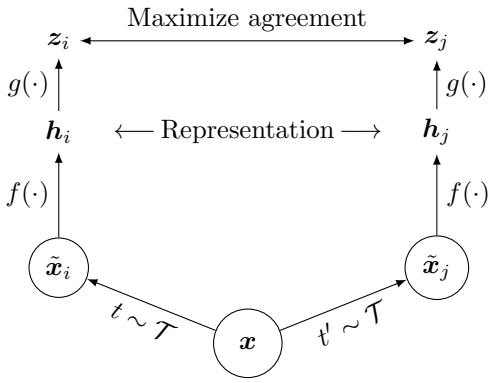


Figure 2.2: From [7]. A simple framework for contrastive learning of visual representations. Two separate data augmentation operators are sampled from the same family of augmentations ($t \sim \mathcal{T}$ and $t' \sim \mathcal{T}$) and applied to each data example to obtain two correlated views. A base encoder network $f(\cdot)$ and a projection head $g(\cdot)$ are trained to maximize agreement using a contrastive loss. After training is completed, we throw away the projection head $g(\cdot)$ and use encoder $f(\cdot)$ and representation $\boldsymbol{h}$ for downstream tasks.

learning (CL), which consists of training a model to pull together representations of similar data points (positive pairs) while pushing apart representations of dissimilar data points (negative pairs). While this is a general approach to self supervised learning and is adaptable to audio, images, text, and various types of data, this stage employs an implementation of CL called SimCLR, a contrastive learning framework (further explored in subsection 2.2.1) to train the backbone further. This training focuses on a task that is agnostic to deepfake detection but encourages the model to learn feature representations that remain consistent even under various image alterations, including compression typically encountered on social media platforms. Here, a projection head is used for flexibility during training but ultimately discarded. The goal is for the backbone to generate feature vectors that are resistant to such image manipulations.

Similar to the supervised approach, a new head is attached to the frozen backbone in the third stage. This head is then trained for binary classification on the TrueFace dataset to distinguish real from fake images. The key difference lies in the pre-trained backbone's enhanced robustness, potentially reducing the need for further fine-tuning specific to each social media platform.

Following the fine-tuning stage, the model is evaluated on images shared on platforms like Twitter, Facebook, and Telegram. The success of the self-supervised training is measured by the model's ability to perform well without additional fine-tuning for each platform. Ideally, the feature vectors learned during the self-supervised stage should be generalizable enough to handle image compression variations across different social media.

2.2.1 SimCLR: A Contrastive Learning Framework

SimCLR learns representations by maximizing agreement between differently augmented views of the same data example via a contrastive loss in the latent space. As illustrated in figure 2.2, this framework comprises the following four major components:

- A module that performs stochastic *data augmentation* on an input image. It creates two modified versions, $\tilde{x}_i$ and $\tilde{x}_j$, of the same original sample. These modified versions are considered a 'positive pair' because they are related to each other while still containing variations. It is paramount to decide carefully the set and intensity of augmentation operations since this makes it harder for the model to recognize positive pairs back. An example of possible augmentations can be seen in Figure 2.3. Following examples from [10], no resize was applied in any experiment, as this might compromise some of the artifacts left by models like GANs in the generative process.

- A neural network *base encoder* (denoted by $f(\cdot)$) to process the augmented data. This encoder can be any type of neural network architecture such as the popular ResNet50 architecture [22]. The encoder takes an augmented data point ($\tilde{x}_i$) and transforms it into a numerical represen-

Figure 2.3: Examples of the effects of data augmentation operators. Each augmentation can transform data stochastically with some internal parameters (e.g. rotation degree, noise level). A batch size of 4 images is doubled and different augmentation is performed on each image. Some examples of augmentations are *horizontal flipping*, *color distortion*, *Gray scale conversion*, *Gaussian noise*, *Gaussian blurring*, *Cut Out* and *Image Compression*.

tation vector $\tilde{\boldsymbol{h}}_i = f(\tilde{x}_i) = \text{ResNet}(\tilde{x}_i)$. During the training phase, the component tries to learn embeddings of images that are as robust as possible to various forms of augmentation. As this component is equivalent to the backbone of models in step one of the supervised method, different architectures will be tested.

- An MLP module acts as the projection head. This head maps the representations extracted by the base encoder into a new latent space where the contrastive loss is applied.
- A *contrastive loss function*, that plays a crucial role in the contrastive prediction task. Given a set $\{\tilde{x}_k\}$ that includes a positive pair of samples $\tilde{x}_i$ and $\tilde{x}_j$, the models aims to identify $\tilde{x}_j$ within the set $\{\tilde{x}_k\}_{k\neq i}$ for a given $\tilde{x}_i$. Essentially, the contrastive loss encourages the model to learn representations where similar data points (like the augmented versions of the same image) are closer together in the latent space.

Algorithm 1 summarizes the proposed method.

The Normalized Temperature Scaled Cross Entropy Loss

To achieve the desired outcome of having similar augmented versions of the same image grouped in the latent space, while distancing representations from different images, a specific loss function is employed. This function essentially guides the model's learning process. A group (mini-batch) of images is chosen from the dataset. Each image is then augmented, creating two views of each original image. This doubles the number of images in the batch. Within the augmented batch, each image acts as an 'anchor'. Its corresponding augmented version is considered a positive example (similar view from the same source). All the remaining images (after removing the anchor and its positive pair) are treated as negative examples (different views from different sources). The loss function for a positive pair of examples (i, j) is defined as

$$\ell_{i,j} = -\log \frac{\exp(\text{sim}(\boldsymbol{z}_i, \boldsymbol{z}_j)/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{k\neq i} \exp(\text{sim}(\boldsymbol{z}_i, \boldsymbol{z}_k)/\tau)} \tag{2.3}$$

where

- $\mathbb{1}_{k\neq i} \in \{0, 1\}$ is an indicator function evaluating to 1 if $k \neq i$

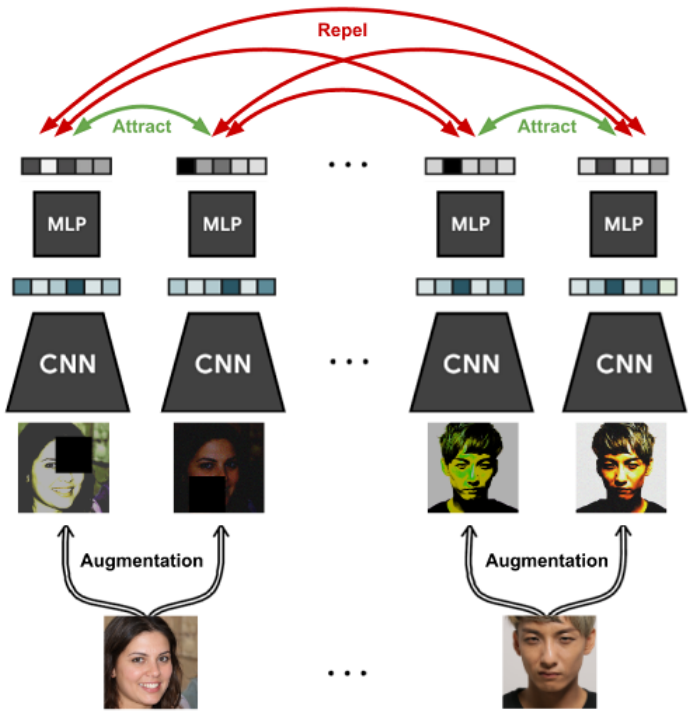


Figure 2.4: A representation of the contrastive learning process. Augmented views are passed to a base encoder and a projection head. Thanks to the NTXent Loss function, the model tries to group these similar versions, while keeping representations of completely different images far apart.

- $\text{sim}(\boldsymbol{z}_i, \boldsymbol{z}_j)$ is the cosine similarity of the projection of i and j
- τ denotes a temperature parameter, which can be passed as a hyperparameter.

To analyze the functioning of this formula deeply, it is possible to investigate each of its components separately:

- Positive Pairs: We focus on pairs of augmented images from the same source (let's call them the "anchor" image, i, and its corresponding view, j). The model aims to learn representations where these positive pairs are close together in a latent space.
- Cosine Similarity: It is the cosine of the angle between the vectors; that is, it is the dot product of the vectors divided by the product of their lengths.

$$\text{sim}(\boldsymbol{z}_i, \boldsymbol{z}_j) = \cos\theta = \boldsymbol{u}^\top \boldsymbol{v} / \|\boldsymbol{u}\| \|\boldsymbol{v}\| \tag{2.4}$$

  It follows that the cosine similarity does not depend on the magnitudes of the vectors, but only on their angle. The cosine similarity always belongs to the interval $[-1, 1]$. For example, two proportional vectors have a cosine similarity of 1, two orthogonal vectors have a similarity of 0, and two opposite vectors have a similarity of -1. It is important to notice that $\mathbb{1}_{k \neq i}$ is used in the formula because the similarity score of an image and itself will always be $\text{sim}(\boldsymbol{z}_i, \boldsymbol{z}_i) = 1$.

- Softmax and Temperature: The formula incorporates a softmax function with a temperature parameter (τ). Softmax transforms the similarity scores into a probability distribution, indicating how likely the model is to identify the correct target image (positive pair) compared to all other options in the batch.

$$\sigma(\boldsymbol{z})_j = \frac{\exp(z_j)}{\sum_{k=1}^{K} \exp(z_k)}, \forall j = 1, \cdots, K \tag{2.5}$$

  The temperature controls how 'peaked' this probability distribution is. A low temperature leads

Algorithm 1 SimCLR's main learning algorithm.

input: batch size N, constant τ, structure of f, g, $\mathcal{T}$.
for sampled minibatch $\{\boldsymbol{x}_k\}_{k=1}^{N}$ **do**
  for all $k \in \{1, \ldots, N\}$ **do**
    draw two augmentation functions $t \sim \mathcal{T}$, $t' \sim \mathcal{T}$
    # the first augmentation
    $\tilde{\boldsymbol{x}}_{2k-1} = t(\boldsymbol{x}_k)$
    $\boldsymbol{h}_{2k-1} = f(\tilde{\boldsymbol{x}}_{2k-1})$ # representation
    $\boldsymbol{z}_{2k-1} = g(\boldsymbol{h}_{2k-1})$ # projection
    # the second augmentation
    $\tilde{\boldsymbol{x}}_{2k} = t'(\boldsymbol{x}_k)$
    $\boldsymbol{h}_{2k} = f(\tilde{\boldsymbol{x}}_{2k})$ # representation
    $\boldsymbol{z}_{2k} = g(\boldsymbol{h}_{2k})$ # projection
  end for
  for all $i \in \{1, \ldots, 2N\}$ and $j \in \{1, \ldots, 2N\}$ **do**
    $s_{i,j} = \boldsymbol{z}_i^{\top} \boldsymbol{z}_j / (\|\boldsymbol{z}_i\| \|\boldsymbol{z}_j\|)$ # pairwise similarity
  end for
  define $\ell(i, j)$ **as** $\ell(i, j) = -\log \frac{\exp(s_{i,j}/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{k \neq i} \exp(s_{i,k}/\tau)}$
  $\mathcal{L} = \frac{1}{2N} \sum_{k=1}^{N} [\ell(2k-1, 2k) + \ell(2k, 2k-1)]$
  update networks f and g to minimize $\mathcal{L}$
end for
return encoder network $f(\cdot)$, and throw away $g(\cdot)$

to more confident predictions, while a higher temperature encourages softer predictions with less certainty.

As the denominator of Eq. 2.3 depends on the number of elements in the batch, it is interesting to observe that the batch size has a strong influence on the loss value. This can be proven by studying what would be the loss value for a random model. Specifically, supposing a batch size of N, the model will have to find the anchor image among $2N - 1$ images. Supposing the model is perfectly balanced and assigns a probability of $1/(2N - 1)$ to each image then the loss would be $-\log(1/(2N - 1)) = \log(2N - 1)$. Examples can be seen in table 2.5.

We have seen that the approach involves using two augmented views of the same image and leveraging contrastive learning to encode these views similarly. It is interesting to observe the consequences of using 3 or more views while trying to apply the same principle. The difference from the standard

Batch size	Random Guessing Loss
2	1.09
4	1.94
8	2.70
16	3.43
32	4.14

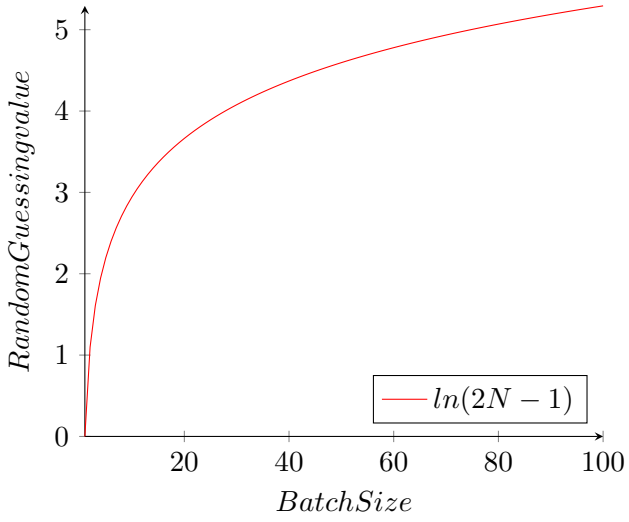


Figure 2.5: Samples and values for effects of batch size on random guessing Loss

approach with two views would be substantial in terms of problem formulation. For each image, there would be more than one corresponding image, transforming the problem from single-labeled to multi-labeled. Consequently, the loss function would require further adaptation.

3 Experimental Setup

In this chapter, we detail the experimental setup employed to evaluate the efficacy of the proposed deepfake detection approaches and models. This includes a description of the datasets used for training and testing, the specific architecture of the AI models, the evaluation metrics selected, and the chosen hyper-parameters for reproducing the results.

3.1 Dataset

A central portion of this work emphasises the effort to address the problem of bringing deepfake detection in a wild scenario, where images are uploaded on different social media and the compression operation compromises artifacts that are useful for the models. For this reason, *TrueFace* was chosen as the dataset. In fact, not only it contains real images of faces from FFHQ and fake images from StyleGAN and StyleGAN2, but it also provides a post-social section, where copies of these images have been uploaded and downloaded from Twitter, Telegram and Facebook. This allows experiments and comparisons analyzing the effects on media compression on the detection skills of a model. The TrueFace dataset is the first publicly available database for synthetic-vs-real image classification with data also shared via social networks.

The *pre-social* dataset contains a total of 150K images of faces, 70K of which are real and 80K are synthetic. The synthetic ones were generated by two popular generator models, namely StyleGAN [27] and StyleGAN2 [30], 40K each. Moreover, images for each model were equally divided according to the StyleGAN fantasy parameter (20K per parameter). The fantasy parameter ψ determines how far the generative network should deviate from the data average. By adjusting this parameter, the variety/quality trade-off of the output data can be determined.

Real images were obtained from the FFHQ Dataset [27], originally proposed as a benchmark for GANs. Both real and synthetic images are RGB and have a resolution of 1024×1024 pixels. The following table summarizes the content of the pre-social dataset.

Real	**StyleGAN**		**StyleGAN2**	
	$\psi = 0.5$	$\psi = 0.7$	$\psi = 0.5$	$\psi = 1$
70K	20K	20K	20K	20K

Table 3.1: Pre-social dataset.

The post-social dataset contains in total 60K images, half real and half fake, uploaded and downloaded on three of the most popular social media platforms: Facebook (FB), Telegram (TG), Twitter (TW). Only a fraction of 20K images in the post-social dataset for each social (10K real, 5K from StyleGAN, and 5K from StyleGAN2) come from the pre-social dataset. However, images in the three sections are the same, with the only difference being the compression operation.

Platform	**Real**	**StyleGAN**	**StyleGAN2**
Facebook (FB)	10k	5k	5k
Telegram (TL)	10k	5k	5k
Twitter (TW)	10k	5k	5k

Table 3.2: Post-social dataset

In order to present comparable results, the test set is composed by the same pictures in both pre and post media compression scenarios. For example, the test set on the Twitter section is the same

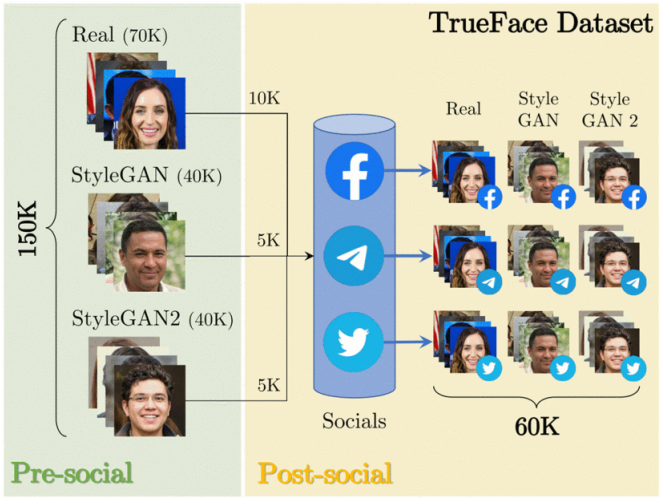


Figure 3.1: The trueface dataset. The pre-social collection includes 70k real faces and 80k GAN-generated images, half from StyleGAN [27] and half from StyleGAN2 [30]. The post-social collection includes the shared version of a portion of the pre-social part, where images have been uploaded and downloaded from Facebook, Telegram, and Twitter, for a total of 60k images. The dataset is available at https://bit.Ly/3baeh75.

as the test set on pre social images, once it is uploaded on the social media. This is done to obtain comparable results among different approaches and tests.

3.2 Architectures

This work analyzes the approach on three different architecture. Firstly, a standard ResNet50 [22] is used. The fully connected layer at the end is removed and the forward method relies on the feature extraction and the average pooling layer. The pretrained weights on ImageNet can be found on the invidia web page [1].

PhiNets [40] architecture was used as well as a backbone as shown in Figure 1.7. Following the implementation [2] and the weights on the Hugging Face website[3], this are the building parameters for the network: $\alpha = 1.1$, $\beta = 0.75$, $t_0 = 5$, $N = 8$.

MicroNet [33] implementation and pretrained weights were taken from the official repository [4]. Among the different available options, the MicroNet-M1 version with 1.8 milions of parameters was choosen, as this is the version with a comparable amount of parameters as the PhiNet's architecture.

3.3 Supervised Learning

In the supervised approach, every architecture uses its frozen backbone and leverages a head for classifying the encoding of the feature vector. While the ResNet50 comes with a feature vector on 2048 dimensions, Phinet and MicroNet come with a smaller vector of around 200 elements. For this reason, while the ResNet uses a single linear layer of shape $[2048, 1]$, the head of the other architectures also utilizes a hidden layers of size 512 to increase the expressiveness of the feature vector. The training was conduced for a total amount of 10 epochs. The learning rate is set to $2e-5$ and the batch size is 4.

3.4 Self Supervised Learning

For the pretext task, the choice of augmentation operations to perform is crucial. In particular, it is important to avoid shearing and resizing, as this compromises the artifacts that the model needs

[1] ResNet50 pre trained on ImageNet: https://catalog.ngc.nvidia.com/orgs/nvidia/models/resnet50_pyt_amp
[2] Micromind repository with PhiNet code: https://github.com/micromind-toolkit/micromind
[3] Hugging Face tutorial for PhiNet: https://huggingface.co/micromind/ImageNet
[4] MicroNet implementation and weights: https://github.com/liyunsheng13/micronet/tree/main

to distinguish [10]. Following the examples of [10, 7], the augmentation module is a composition of *Horizontal Flip*, *Color distortion (jitter)*, *Gaussian Blur*, *Gaussian Noise*, *Gray Scale Conversion*, *Cut Out*, and *JPEG Image Compression*. Sequentially, images are also normalized with standard values for mean and standard variation (mean $= [0.485, 0.456, 0.406]$, std $= [0.229, 0.224, 0.225]$) . Figure 2.3 shows some results. The batch size is two, which means that at each step , given an anchor, the model aims at finding the positive pair among three other images (one is correct and two are not). The temperature value is set at 0.07, which is also the default value. This is considered a low temperature that emphasizes small preferences of the model for a picture compared to another one. The projection head is composed of a fully connected MLP with a hidden layer of size 2048, a ReLU activation function and another layer with an output size of 128, which is the number of dimensions of the latent space. The training is conduced with a fixed learning rate of $2e-5$ for all models. For the supervised phase, only normalization is performed in the preprocessing phase. The learning rate is kept constant, and the batch size is four. After the contrastive learning phase, the backbone of the model is frozen and following supervised phase only consists in training a new head with the same structure as before (with a hidden layer for edge architectures).

3.5 Metrics

In this section, we outline the evaluation metrics employed to assess the performance of the models. Given the nature of our task, we have selected a combination of metrics that provide a comprehensive view of the model's effectiveness. These metrics include accuracy, Area Under the Receiver Operating Characteristic Curve (AUC), and F1 score. We discuss the rationale behind choosing these metrics and their significance in the context of our specific objectives.

A fundamental tool for understanding classification model performance is the confusion matrix, which summarizes the counts of true positive (TP) , true negative (TN) , false positive(FP), and false negative (FP) predictions. For the sake of this work, positive is intended as Real and negative as Fake, so that false positives signifies the amount of Fake images that the model evaluated as Real.

Accuracy Accuracy represents the proportion of correct predictions made by the model over all instances. Mathematically, it is calculated as

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN} \tag{3.1}$$

While accuracy is a straightforward and intuitive metric, it can be misleading in situations with imbalanced class distributions, where one class is significantly more prevalent than the other. In such cases, a high accuracy might not necessarily reflect good performance on the minority class. Although in our case the dataset is only slightly imbalanced (46% Real, 54% Fake), we complement accuracy with additional metrics like AUC and F1 score, which provide a more nuanced understanding of model performance across different classes.

F1 Score The F1 score is a valuable metric that combines precision and recall into a single value, providing a balanced assessment of a model's performance. Precision measures the proportion of true positive predictions out of all positive predictions made by the model. In other words, it answers the question: "Of all the instances predicted as positive (Real), how many were actually positive?". It is described as:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{3.2}$$

Recall (also known as sensitivity) measures the proportion of true positive predictions out of all actual positive instances. It answers the question: "Of all the actual positive (Real) instances, how many did the model correctly identify?" Recall is computed as:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{3.3}$$

The F1 score is the harmonic mean of precision and recall, giving equal weight to both. It is

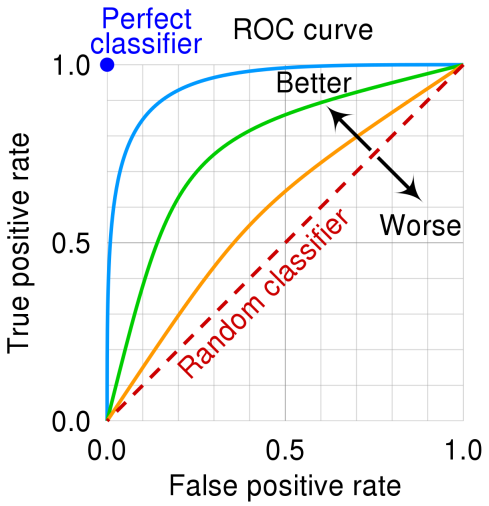


Figure 3.2: The ROC space for a "better" and "worse" classifier.

calculated as:

$$\text{F1 Score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3.4}$$

A higher F1 score indicates better model performance, with a perfect F1 score of 1 achieved when both precision and recall are perfect.

AUC AUC stands for Area Under Curve. Said curve is the Receiver Operating Characteristic (ROC) curve, and it is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied. It plots the Recall, also known as True Positive Rate (TPR), against the False Positive Rate (FPR). The FPR is defined as

$$\text{FPR} = \frac{FP}{FP + TN} \tag{3.5}$$

and it's the probability that a Fake image passes through the filter and gets evaluated as Real.

Figure 3.2 shows example of how ROC curves are plotted and what's their meaning.

The AUC is the area under this ROC curve. A model with perfect discrimination between positive and negative classes will have an AUC of 1, while a model with no discrimination ability will have an AUC of 0.5 (equivalent to random guessing).

4 Results

In this chapter, we will analyze the results obtained from the previously employed approaches across the three distinct architectures, and we will validate the efficacy of utilizing contrastive Learning for this specific task.

Supervised approach The first tested method is the supervised approach, where the model is trained on pre-social images first and sequentially fine-tuned on compressed images. Tables 4.1, 4.2 and 4.3 show the great ability of the networks to leverage the feature vector produced by the backbone to classify images correctly. It is important to remember that the feature vector is the result of the pre-training of the model on the ImageNet dataset, which gives the network skills in terms of general-purpose interpretation of an image. Models can not perform as well on media-shared images. According to the results, there is a heavy performance drop without fine-tuning, although no clear pattern emerges across different social media. As it was anticipated in sections 1.1.2 and 1.2, the invisible artifacts that are hidden in the image are very delicate and get disrupted with the uploading process, leading to the evident effects. However, it is possible to make the gap up by fine-tuning the model on a specific social. The drawback is that now a different head is produced when the model is fine-tuned on each social, so that we have a head for Telegram, one for Facebook, and one for Twitter. In the TrueFace paper [3] it is shown that a head for Twitter also has some skills for Telegram and Facebook as well and vice-versa, but with lower performances. Moreover, it is important to notice how performances vary across different architectures. Although differences in sizes among these networks are going to be discussed later, it is worth noticing that the ResNet50 is the one with the highest results on average but PhiNet and MicroNet are not doing so badly either. MicroNet is almost at the same level as ResNet50 on pre-Social Images, while the PhiNet sets for lower performances. However, the latter seems more robust on post-social without fine-tuning. After training on post-social images, the ResNet50 is the one that gets closer to pre-social results, followed by MicroNet and PhiNet.

Accuracy	**PRE SOCIAL**	**POST SOCIAL**					
		No Fine Tuning			**Fine Tuning**		
		Facebook	**Telegram**	**Twitter**	**Facebook**	**Telegram**	**Twitter**
ResNet50	99.9%	54.6%	59.5%	78.3%	93.1%	97.0%	94.6%
PhiNet	93.1%	66.8%	74.2%	57.2%	85.0%	83.6%	91.5%
MicroNet	99.8%	46.6%	46.6%	46.6%	86.6%	93.6%	86.6%

Table 4.1: Accuracy results with Supervised approach

F1 Score	**PRE SOCIAL**	**POST SOCIAL**					
		No Fine Tuning (F1 Score)			**Fine Tuning (F1 Score)**		
		Facebook	**Telegram**	**Twitter**	**Facebook**	**Telegram**	**Twitter**
ResNet50	92.5%	63.1%	72.3%	65.4%	85.3%	88.6%	87.3%
PhiNet	85.4%	67.2%	72.3%	63.4%	73.0%	79.6%	84.3%
MicroNet	91.5%	59.4%	59.1%	59.5%	77.2%	86.1%	80.6%

Table 4.2: F1 Score results with Supervised approach

AUC	PRE SOCIAL	POST SOCIAL					
		No Fine Tuning (AUC)			Fine Tuning (AUC)		
		Facebook	Telegram	Twitter	Facebook	Telegram	Twitter
ResNet50	87.6%	64.8%	83.9%	61.7%	85.9%	86.0%	85.8%
PhiNet	85.3%	68.5%	78.0%	54.6%	81.1%	83.8%	85.3%
MicroNet	86.8%	46.3%	72.4%	54.5%	79.5%	86.1%	86.9%

Table 4.3: Area Under Curve results with Supervised approach

Self supervised approach The second approach leverages self-supervised learning, with a pretext task based on contrastive learning to gain robustness on media compression operations and a supervised phase for Deepfake detection. Therefore, the following results present the performances of the models after they proceeded through both the pretext task and the supervised phase. Pre-social results show that performances see a gap when compared to the standard approach. This signifies that after the pretext task, the backbone of the model learns features that are not good enough to effectively distinguish between real and fake images as before. On the other side, the effects of Contrastive Learning are appreciable. The drop between performance on pre and post-social images is almost completely negligible, and in some cases, post-social results are even better (Table 4.4). This is particularly interesting, considering that the model was trained on pre-social images only. This consistency between pre and post-social results might suggest that, by now raising performances in pre-social, post-social improvement will follow. Unfortunately, we tried to do so and this is not the case. Apart from varying hyperparameters and augmentation intensity, the closest attempt was to avoid freezing the backbone after the pretext task. The hope is to take the stable representations after SimCLR and finetune the backbone to make it produce specific features for Deepfake detection. Indeed, doing this achieves high performances in the supervised task, similar to pre-social results in table 4.1, but it follows the same pattern when tested on post-social images. Unfreezing the backbone allows the model to 'forget' the robustness achieved previously and present the same drop of accuracy measured before. These results are somewhat reminiscent of the Stability / Plasticity trade-off in Continual Learning. To learn a new task, that is, making ImageNet features robust to augmentation, the model partially forgets the old features and can't perform equally well when trained on the Real vs Fake task. Similarly, as before, edge architectures are not performing poorly compared to the ResNet50, even though there is a gap between the latter and the PhiNet. On the other hand, numbers show that MicroNet is the one with the highest results in terms of accuracy, f1 score, and area under the curve, indicating that a reduction in power requirements and number of parameters is not compromising performances at all.

Accuracy	PRE SOCIAL	POST SOCIAL		
		Facebook	Telegram	Twitter
ResNet50	67.1%	66.8%	67.5%	67.1%
PhiNet	63.8%	61.2%	64.8%	63.3%
MicroNet	70.5%	63.8%	70.1%	69.7%

Table 4.4: Accuracy results with Self-Supervised approach

AUC	PRE SOCIAL	POST SOCIAL		
		Facebook	Telegram	Twitter
ResNet50	72.8%	61.3%	65.6%	63.6%
PhiNet	68.3%	58.4%	61.9%	59.3%
MicroNet	63.8%	57.8%	65.3%	63.9%

Table 4.5: Area Under Curve results with Self-Supervised approach

F1 Score	PRE SOCIAL	POST SOCIAL		
		Facebook	**Telegram**	**Twitter**
ResNet50	59.3%	51.5%	54.3%	54.4%
PhiNet	62.0%	46.1%	56.3%	54.6%
MicroNet	61.6%	44.1%	62.4%	61.0%

Table 4.6: F1 Score results with Self-Supervised approach

Model Size As previously noted, PhiNet and MicroNet performed competitively with ResNet50 in both approaches. However, to fully appreciate their performance, it's crucial to consider the significant differences in model size. Table 4.7 reveals that ResNet50 has 23 million parameters, while PhiNet and MicroNet have approximately a twentieth of that size. The disparity is even more striking when considering the number of operations. Analyzing a 1024 x 1024 image requires 85 billion operations for ResNet50, but only 4 billion for PhiNet and a mere 7.6 million (<1% of ResNet) for MicroNet. This substantial difference between PhiNet and MicroNet arises because MicroNet is specifically designed to optimize the number of operations, as detailed in section 1.3.3. Conversely, PhiNet prioritizes scalability to meet specific hardware requirements. While the micro-factorization technique used in MicroNet (discussed in section 1.3.3 and [33]) could potentially be applied to PhiNet, this avenue has not yet been explored.

Our findings demonstrate a clear trade-off between reducing parameters and operations (and thus power consumption) and overall performance. However, depending on the specific use case, the advantages of edge architectures, such as a 99.9% reduction in operations, can outweigh the minor performance decrease of 0.1% observed on pre-social images. MicroNet demonstrated exceptional effectiveness in the Real vs Fake task, particularly in controlled environments. These findings strongly support the use of such networks for computationally intensive tasks. Given the impressive accuracy achieved in supervised methods, it's conceivable that a future scenario could involve an edge architecture like MicroNet pre-validating posts on devices before they are uploaded to social media.

	MACC	**Number of parameters**
ResNet50	85.4 B	23.5 M
Phinet	4.07 B	1.14 M
Micronet	7.62 M	1.44 M

Table 4.7: Number of operations and parameters for each model. The number of operation is influenced by the resolution of the input image

Conclusions

The proliferation of deepfakes poses significant ethical concerns, particularly regarding their potential to erode trust in digital media, manipulate public opinion, and perpetuate harmful stereotypes or biases. As deepfake technology becomes increasingly sophisticated and accessible, the risks of malicious use escalate, threatening to undermine the credibility of authentic content and exacerbate existing societal divisions. Additionally, the potential for deepfakes to be used in cyberbullying, harassment, and other forms of personal attack raises serious concerns about individual privacy and security. Addressing these ethical challenges requires a multifaceted approach involving technological advancements in detection and mitigation, as well as broader societal conversations about media literacy and the responsible use of AI-generated content.

In recent years, deepfake detection has become increasingly critical due to advancements in generative models like StyleGANs and Diffusion Models, which produce synthetic media virtually indistinguishable from real content. Much research focuses on developing robust models that generalize across various generative architectures by leveraging telltale artifacts introduced during the creation process. However, these artifacts are often significantly altered when media is uploaded and shared online, undergoing compression and resizing.

This thesis investigates the trade-offs that arise when deepfake detection models need to perform consistently on both pristine and compressed images, a common scenario in real-world social media environments. We explore two primary strategies: the usage of Self-supervised learning to achieve robustness among social media compression, and the usage of edge architectures to explore the potential of future on-device validity checks of images before uploading.

We examined the potential of SSL to enhance model robustness to social media compression artifacts, specifically targeting platforms like Telegram, Facebook, and Twitter. This is done by leveraging SimCLR, a contrastive learning framework proposed by Google, which aims at teaching the model to distinguish between similar (augmented views of the same image) and dissimilar pairs of images. By doing so before the supervised learning phase on the Real vs Fake task, the models can capture essential features of the images and become less sensitive to variations caused by compression. Our research demonstrates that SSL can effectively bridge the performance gap between pre- and post-social media content. However, this robustness comes at the cost of overall accuracy, suggesting that a supervised approach may be preferable in situations where accuracy is paramount.

Moreover, we evaluated the performance of lightweight neural networks (MicroNet and PhiNet) for on-device deepfake detection, aiming to enable real-time validity-check. Our findings reveal that, in controlled settings, MicroNets can match the accuracy of larger models like ResNet50 while significantly reducing computational complexity. MicroNet accuracy of 99.8% on pre-social images is a stunning result. Although post-social for MicroNet are not that close yet to bigger models, pre-social results open up possibilities for efficient deepfake detection directly on user devices before the uploading.

Future research can extend this work in multiple directions. As Supervised Learning proved to be the one providing higher accuracy on post-social images, it is interesting to further explore its use. For example, it might be worth it to analyze a picture and try to understand which social networks have operated on that image previously, to use a classification head specifically trained for those levels of compression. This would help to fix the problem explored in section 4 of having different classification heads, each one specialized on a certain social. However, this would require further processing that needs to be optimized to allow edge devices to afford the computational burden. Alternatively, approaches of this kind might also be analyzed on different social media compression and generative models like Diffusion Models [12] or StyleGAN3 [29]. As deepfake technology continues to evolve, it is crucial to address the ethical implications and raise public awareness about the potential risks and harms associated with deepfakes. Future research could focus on developing educational

resources and tools to help individuals identify and critically evaluate deepfakes, empowering them to make informed decisions about the information they consume online.

Bibliography

[1] Alberto Ancilotto, Francesco Paissan, and Elisabetta Farella. Xinet: Efficient neural networks for tinyml. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 16922–16931, 2023.

[2] Ben Athiwaratkun and Keegan Kang. Feature representation in convolutional neural networks, 2015.

[3] Giulia Boato, Cecilia Pasquini, Antonio L. Stefani, Sebastiano Verde, and Daniele Miorandi. Trueface: a dataset for the detection of synthetic face images from social networks. In *2022 IEEE International Joint Conference on Biometrics (IJCB)*, pages 1–7, 2022.

[4] Sergi D Bray, Shane D Johnson, and Bennett Kleinberg. "testing human ability to detect 'deepfake' images of human faces". *Journal of Cybersecurity*, 9(1):tyad011, 06 2023.

[5] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale GAN training for high fidelity natural image synthesis. In *International Conference on Learning Representations*, 2019.

[6] L. Chai, D. Bau, S.-N. Lim, and P. Isola. What makes fake images detectable? Understanding properties that generalize. In *ECCV*, 2020.

[7] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations, 2020.

[8] Riccardo Corvi, Davide Cozzolino, Giada Zingarini, Giovanni Poggi, Koki Nagano, and Luisa Verdoliva. On the detection of synthetic images generated by diffusion models, 2022.

[9] D. Cozzolino, J. Thies, A. Rössler, C. Riess, M. Nießner, and L. Verdoliva. ForensicTransfer: Weakly-supervised domain adaptation for forgery detection. *arXiv preprint arXiv:1812.02510*, 2018.

[10] Davide Cozzolino, Diego Gragnaniello, Giovanni Poggi, and Luisa Verdoliva. Towards universal gan image detection, 2021.

[11] B. Dayma, S. Patil, P. Cuenca, K. Saifullah, T. Abraham, P. Lê Khàc, L. Melas, and R. Ghosh. DALL-E Mini, 7 2021.

[12] P. Dhariwal and A. Nichol. Diffusion models beat GANs on image synthesis. *Advances in Neural Information Processing Systems*, 34:8780–8794, 2021.

[13] M. Du, S. Pentyala, Y. Li, and X. Hu. Towards generalizable forgery detection with locality-aware autoencoder. *ACM CIKM*, 2019.

[14] R. Durall, M. Keuper, and J. Keuper. Watch your up-convolution: CNN based Generative Deep Neural Networks are failing to reproduce spectral distributions. In *CVPR*, 2020.

[15] Patrick Esser, Robin Rombach, and Björn Ommer. Taming transformers for high-resolution image synthesis, 2021.

[16] L. Nataraj et al. Detecting GAN generated fake images using co-occurrence matrices. In *IS&T EI, Media Watermarking, Security, and Forensics*, 2019.

[17] J. Frank, T. Eisenhofer, L. Schönherr, A. Fischer, D. Kolossa, and T. Holz. Leveraging Frequency Analysis for Deep Fake Image Recognition. In *CVPR*, 2020.

[18] Ian J. Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks, 2015.

[19] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[20] Diego Gragnaniello, Davide Cozzolino, Francesco Marra, Giovanni Poggi, and Luisa Verdoliva. Are gan generated images easy to detect? a critical analysis of the state-of-the-art, 2021.

[21] Hui Guo, Shu Hu, Xin Wang, Ming-Ching Chang, and Siwei Lyu. Eyes tell all: Irregular pupil shapes reveal gan-generated faces, 2022.

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[23] Mi-Young Huh, Pulkit Agrawal, and Alexei A. Efros. What makes imagenet good for transfer learning? *CoRR*, abs/1608.08614, 2016.

[24] Rakhee Kallimani, Krishna Pai, Prasoon Raghuwanshi, Sridhar Iyer, and Onel L. A. López. Tinyml: Tools, applications, challenges, and future research directions. *Multimedia Tools and Applications*, 83(10):29015–29045, September 2023.

[25] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation, 2018.

[26] Tero Karras, Miika Aittala, Samuli Laine, Erik Härkönen, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Alias-free generative adversarial networks, 2021.

[27] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. *CoRR*, abs/1812.04948, 2018.

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

[29] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. *CoRR*, abs/1912.04958, 2019.

[30] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan, 2020.

[31] Federica Lago, Cecilia Pasquini, Rainer Böhme, Hélène Dumont, Valérie Goffaux, and Giulia Boato. More real than real: A study on human visual perception of synthetic faces. *CoRR*, abs/2106.07226, 2021.

[32] H. Li, B. Li, S. Tan, and J. Huang. Detection of deep network generated images using disparities in color components. *Signal Processing*, 174, 2020.

[33] Yunsheng Li, Yinpeng Chen, Xiyang Dai, Dongdong Chen, Mengchen Liu, Lu Yuan, Zicheng Liu, Lei Zhang, and Nuno Vasconcelos. Micronet: Improving image recognition with extremely low flops, 2021.

[34] Alexandra Sasha Luccioni, Sylvain Viguier, and Anne-Laure Ligozat. Estimating the carbon footprint of bloom, a 176b parameter language model, 2022.

[35] F. Marra, C. Saltori, G. Boato, and L. Verdoliva. Incremental learning for the detection and classification of gan-generated images. In *IEEE WIFS*, 2019.

[36] Michael McCloskey and Neal J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. volume 24 of *Psychology of Learning and Motivation*, pages 109–165. Academic Press, 1989.

[37] S. McCloskey and M. Albright. Detecting GAN-Generated Imagery Using Saturation Cues. In *IEEE ICIP*, 2019.

[38] Thanh Thi Nguyen, Quoc Viet Hung Nguyen, Dung Tien Nguyen, Duc Thanh Nguyen, Thien Huynh-The, Saeid Nahavandi, Thanh Tam Nguyen, Quoc-Viet Pham, and Cuong M. Nguyen. Deep learning for deepfakes creation and detection: A survey. *Computer Vision and Image Understanding*, 223:103525, October 2022.

[39] A. Nichol, P. Dhariwal, A. Ramesh, P. Shyam, P. Mishkin, B. McGrew, I. Sutskever, and M. Chen. GLIDE: Towards photorealistic image generation and editing with text-guided diffusion models. *arXiv preprint arXiv:2112.10741*, 2021.

[40] Francesco Paissan, Alberto Ancilotto, and Elisabetta Farella. Phinets: a scalable backbone for low-power ai at the edge, 2021.

[41] Francesco Paissan, Alberto Ancilotto, and Elisabetta Farella. Phinets: A scalable backbone for low-power ai at the edge. *ACM Trans. Embed. Comput. Syst.*, 21(5), dec 2022.

[42] David Patterson, Joseph Gonzalez, Urs Hölzle, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean. The carbon footprint of machine learning training will plateau, then shrink, 2022.

[43] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, and M. Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125v1*, 2022.

[44] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer. High-resolution image synthesis with latent diffusion models. In *CVPR*, pages 10684–10695, 2022.

[45] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer. Stable diffusion. `https://github.com/CompVis/stable-diffusion`, 2022.

[46] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks, 2019.

[47] L. Verdoliva. Media forensics and deepfakes: an overview. *IEEE Journal of Selected Topics in Signal Processing*, 14(5):910–932, 2020.

[48] S.-Y. Wang, O. Wang, R. Zhang, A. Owens, and A. Efros. CNN-generated images are surprisingly easy to spot... for now. In *CVPR*, 2020.

[49] Zhisheng Xiao, Karsten Kreis, and Arash Vahdat. Tackling the generative learning trilemma with denoising diffusion gans, 2022.

[50] X. Xuan, B. Peng, W. Wang, and J. Dong. On the generalization of GAN image forensics. In *Chinese Conference on Biometric Recognition*, pages 134–141, 2019.

[51] X. Zhang, S. Karaman, and S.-F. Chang. Detecting and Simulating Artifacts in GAN Fake Images. In *IEEE WIFS*, pages 1–6, 2019.

[52] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning, 2020.