# Data Augmentation Techniques using Generative Adversarial Neural Networks on Side Channel Analysis

Achilleas Vlogiaris

# Data Augmentation Techniques using Generative Adversarial Neural Networks on Side Channel Analysis

by

Achilleas Vlogiaris

to obtain the degree of Master of Science in Computer Science
at the Delft University of Technology,
to be defended publicly on Friday April 16, 2021 at 10.00 AM

| | | |
|---|---|---|
| Student number: | 4875974 | |
| Thesis committee: | Dr. ir. S. Picek | TU Delft, Supervisor |
| | Prof. dr. ir. R. L. Lagendijk | TU Delft, Chair |
| | Dr. ir. Mottaqiallah Taouil | TU Delft |
| Cosupervisor: | Dr. ir. I. Buhan | Radboud University |

# Abstract

Side-channel Attacks can be performed in various ways by measuring the power consumption, the electromagnetic emission, or even by measuring an algorithm's execution time on the targeted device. More or less sophisticated methods can be used to utilize this information in order to perform a Side-channel attack, more specifically by training Machine Learning models with traces acquired from a profiling target or by directly performing statistical analysis to a black box target. Of course, performing a successful attack in a black box scenario requires large amounts of traces, even for unprotected implementations. On the other hand, with the state of the art Machine Learning methods, a successful Side-channel attack can be performed by acquiring a smaller number of traces.

Nevertheless, for every case, in order to perform a Side-channel Attack, first traces have to be acquired from the target. This acquisition phase is a time-consuming and computationally expensive procedure that requires hardware tools and human supervision. For this reason, in this thesis, we investigate if synthetic Side-channel traces can be generated when the Hamming Weight (HW) model is used as a power model. To generate synthetic traces, we use Generative Adversarial Neural Networks (GANs), a Deep Learning approach that is the state-of-the-art augmentation method in many other fields, especially in Computer Vision. Therefore, we introduce a pipeline to define the architecture of the GANs but also to improve and evaluate it. In other words, we investigate novel ways to introduce new software tools to minimize the time and processing power needed to perform a successful Side-channel attack. Since the HW model is used, the Side-channel datasets follow a specific binomial class distribution, and by using GANs, we can augment the dataset by altering this distribution according to our will. For this reason, before performing data augmentation, we investigate how the profiling models' performance is affected by the trace's class distribution.

Next, we use our pipeline to define two GAN architectures: a shallow one able to generate high-quality synthetic traces for unprotected implementations and an enhanced second architecture that can generate synthetic traces even when mild countermeasures such as jittering are in place, by being more stable but also more computationally expensive.

We apply the pipeline's methodology on unprotected datasets and on datasets with mild countermeasures such as jittering and we made some steps leveraging GANs to solve the task of augmenting dataset that consist of Side-channel traces. In this way, we managed to compare the GANs with the well established Synthetic Minority Oversampling Technique (SMOTE), and we observed that GANs achieved similar performance with SMOTE and, in some cases, better.

This thesis showed that GANs can generate high-quality synthetic traces for unprotected implementations and implementations with mild countermeasures, which means that there is space for future investigation in this direction that might lead to more sophisticated architectures able to perform data augmentation in the Side-channel Analysis field.

# Preface

Before you lies my thesis with the topic "Data Augmentation Techniques using Generative Adversarial Neural Networks on Side Channel Analysis," a work of more than one year that should complete my master's degree in TU Delft. I started my thesis on the 20th of January 2020 by joining Riscure as an intern. The first months were struggling because of my limited knowledge in the Side-channel Analysis field, as in the previous year of my studies, I focused more on Machine Learning and Deep Learning projects. After making my first steps in this new domain, the world changed as everyone was forced to work from home. Consequently, completing a thesis project became an even more challenging task. Thankfully, many people helped me overcome the academic challenges and, more importantly, to overcome the difficulties of working isolated to the same place, and I would like to take this opportunity to thank them.

Firstly, I would like to thank my supervisor, Stjepan, for guiding me through the challenges, for making the correct decisions, and for sharing his innovative ideas that proved to be essential in this thesis. Secondly, I would also like to thank Ileana, who guided me in my thesis project by spending a respectable amount of her time helping me with many challenges that I faced.

Finally, I am thankful to Stelios, Agapi, my brother Haris and my parents for helping me through all my studies, especially for the moral support during this thesis project.

*Achilleas Vlogiaris*
*Delft, April 2021*

# Contents

# 1 Introduction

During the mid of the last century, a new historical period began known as the Information Age. The economy has stopped being exclusively dependent on the traditional industry established by the Industrial Revolution and started to shift to Information technology. This new Information Age started to affect society exponentially, and the main reason for this economic structural change was the fast-growing computer processing power, every eighteen or so months, computer processing speed was getting doubled. Furthermore, the Information technological vast advancements during the last few decades seemed enough to ensure that this shift will affect society's evolution further in the future.

The information revolution led humanity to build large communication networks where massive amounts of data could be shared. These networks evolved into the well-known global system of interconnected computer networks, namely the Internet, which gave new companies space to grow. These information technology companies, i.e., Microsoft, IBM, Oracle, mainly produced software, hardware, or semiconductor equipment, while others focused on providing Internet or other related services. The Internet's fast expansion pace and the semiconductor growing rate led to an era where people are heavily dependent on Information technology devices. Since the last five years, the global population of smartphone users has risen from 25.3 percent to 44.9 percent [1]. Consequently, it is common sense to assume that the security of embedded devices like smartphones plays a significant role in society. Therefore various cryptographic algorithms were introduced in order to establish secure networks among the embedded devices. Along with the classical cryptanalysis that focuses on the mathematical structure of the cryptographic algorithms, one other security field evolved, called Side-channel Analysis (SCA).

Side-channel Analysis focuses on the information gained by studying any possible output of the device, which can be transformed into valuable information in order to study the device's cryptographic algorithms. For example, an information leak could be the power consumption, or the electromagnetic emissions of the device, or the time usage during encryption or decryption. To make it more transparent, we can think about a Side-channel Attack on the processor's cache memory. This is an active area that researchers focused on performing a Side-channel Attack during the last decade [3], [25], [59], [24]. Cache memory is employed in a regular computer system between the CPU and the RAM, which cache memory is relatively faster but smaller than the RAM. More specifically, during the execution of a program, the most recently used data are stored in the cache memory as it is more likely that they will be used again in the near future. During a program's execution, if the data used by the program are stored in the cache memory, we can say that we have a cache hit, and the CPU can quickly retrieve them. Therefore, a computer algorithm execution time depends a lot on the cache hits and cache misses. These time variations of the algorithm are used as information leakage to perform various Side-channel Attacks. This cache memory attack is one example of a Side-channel Attack family, while many others exist. Of course, embedded engineers and security analysts tried to minimize the data leakages by introducing hardware and software countermeasures. For that reason, the data acquisition phase became a time-consuming and challenging task in the SCA field.

[1]https://www.statista.com/statistics/203734/global-smartphone-penetration-per-capita-since-2005/

To study the limitations that countermeasures introduced in SCA, researchers focused on performing powerful attacks, and one primary factor for performing successful attacks is the quality and quantity of the acquired information leak. Therefore, the data acquisition phase plays a significant role in performing a powerful Side-channel Attack as large high-quality datasets are needed to build robust models.

Furthermore, during the last 20 years, the vast amount of information transferred through the Internet led to one more field of Information technology, namely machine learning and later deep learning, becoming very popular. This field focuses on building algorithms that are capable of understanding patterns that exist in the data. We could say that the machine learning objective is to build computer algorithms that can take advantage of massive amounts of data in order to learn and predict these patterns. By studying machine learning and the Side-channel Analysis fields, we can see that machine learning can work as an intermediate step between the SCA's data acquisition and attack phase. The machine learning models' ability to understand possible patterns in the extracted data makes them valuable tools for analyzing an embedded device's security properties. Therefore, we can theoretically assume that it is possible to convert the time-consuming data acquisition phase into a mathematically software-based procedure.

Many researchers currently focus on finding new models and improving the previous ones by using machine learning and deep learning techniques [4], [37], [20]. At the same time, limited research has been done on studying the nature of the leaking data. Traditional augmentation techniques do not seem to respond well in the SCA field, while other more complex methods, consisting of deep learning models like the Generative Adversarial Networks (GANs), are relatively unstable for the moment. GANs showed [50], [31], [23], [7], [26], [62] exemplary behavior in many applications in the Information technology field; therefore, it could be an exciting option to simulate the data acquisition's complex task. More specifically, in this thesis project, we investigate the performance of the Generative Adversarial Networks (GANs) under different setups and different datasets with and without introducing countermeasures. The first focus point is to understand the nature of the extracted data in order to see how the GANs' new synthetic data may improve a model to perform an SCA attack and furthermore to understand how the machine learning field can be combined efficiently along with the SCA field as both fields have different evaluation methods. The second point consists of two parts; the first part focuses on training the GANs by using a large dataset to produce high-quality synthetic data that can be evaluated by comparing the leakage areas between synthetic and original data. While the second part is to reduce that number of training data to investigate if GANs can generate plausible outputs when trained in a real scenario where the acquired dataset's size could be limited. By showing that a relatively small dataset is sufficient to train the GANs but insufficient to perform a successful SCA attack, we can conclude that GANs can be an alternative to the time-consuming data acquisition phase.

By taking into account all the above, we propose to investigate two main questions.

1. Is it feasible for Generative Models such as complex Generative Adversarial Neural Networks (GANs) to generate traces that will improve the performance of a Side-channel Analysis (SCA) attack when the training set is limited?
2. How can GANs be evaluated from a machine learning perspective in order to achieve parameter optimization for these models by considering that visual representations of the generated traces are not an option?

In Chapter 2, we address the essential theory behind Side-channel Analysis, Augmentation techniques, Machine Learning, Deep Learning, and Generative Adversarial Neural Networks. Next, in Chapter 3, we briefly mention relevant profiling models used in SCA and the limited relevant related work that focuses on SCA trace-sets' data augmentation problem. In chapter 4, we do a brief investigation about the nature of the SCA data and especially about the class distribution of the SCA data as it seems to be a necessary investigation in order to make the correct decision about the design of the GANs. Moving to Chapter 5, we present a comprehensive report of the proposed pipeline of this project. This pipeline has several steps; in the first step, the profiling and the GANs models are trained, then pre-processing and post-processing methods are applied in the given trace-sets, and lastly, the evaluation of the generated traces takes place. Next, in Chapter 6, an extensive analysis of the experimental results produced using the proposed pipeline occurs. Lastly, in Chapter 7, we sum up the fifth's Chapter outcomes and analyze their performance. Furthermore, we discuss the limitations of this pipeline and the alternative solutions to improve it.

# 2 Background

In this Chapter, the necessary background for understanding this thesis is introduced. First, we start with some basic machine learning concepts and a brief explanation of the machine learning models and methods. Then follows a brief explanation for Side-channel analysis, and lastly, data augmentations and data generation methods are addressed.

## 2.1. Side-channel Analysis

Side-channel Analysis has its roots back in 1943 as recent NSA declassified documents showed that for a certain encrypting teletype of a Bell phone device, an engineer managed to decipher it by observing spikes on an oscilloscope associated with the decrypted output [1]. Nowadays, Side-channel Analysis (SCA) analysis plays a significant role in the security evaluation of embedded systems, and unlike regular cryptanalysis, it does not focus on software errors but on any form of leakage that a device could emit. SCA takes advantage of any information that can be extracted from the implementation of a computer system. Most effective Side-channel attacks use as information the electromagnetic emissions and power consumption of the targeted device. Other methods can be found in the literature like acoustic emission and timing.

All this information can be modeled using leakage models; the most common are the Hamming Weight (HW), the Hamming Distance (HD), and the identity leakage model. The purpose of the HW model is to capture the trace's differences in power consumption of the device, and the idea is based on the assumption that in one byte of information, a one-bit flip indicates a change in the power consumption. Therefore, in one byte, the hamming weight is a value ranging between zero and eight following a binomial distribution with the most common value to be four while the rarest to be zero and eight. This phenomenon might cause the SCA attack more difficult as it makes the trace-set imbalanced. The alternative model, which is the identity model, solves this problem as the dataset is balanced but with 256 different classes, compared to using the HW model, which has only nine classes and is used for software implementations. Lastly, the HD model has the same characteristics as the HW model. The only difference between the HD model with the HW model is that HD is used for hardware implementations, and in order to calculate the HD, we look at the bit-differences of the byte before and after the byte goes through the crypto-algorithm's targeted area.

SCA evaluations can be divided into two categories the non-profiled and the profiled SCA. In this project, we focus more on profiled SCA, as in this category, we can produce potent attacks, and therefore GANs can demonstrate their full potential. However, impressive results can be expected even with non-profiling methods, but this can be addressed as future work for this project. Lastly, two more parts are addressed, a part that describes the two famous metrics of the field and a second part that describes two necessary countermeasures that can be exploited in order to make the targeted device or software algorithm less prone to Side-channel attacks.

[1]https://www.wired.com/2008/04/nsa-releases-se/

### 2.1.1. Non-Profiling Side-channel Analysis

The most straightforward Side-channel attack can be achieved mainly in a non-profiling context where we can collect measurements from a targeted device without having a clone device. More specifically, non-profiling methods can be divided into two categories; the first is the Simple Power Analysis (SPA) [42] which is the simplest one, while the second is more powerful than the first one and it is called the Differential Power Analysis (DPA). The SPA attacks use only the visual information gained by observing multiple traces under different combinations between random plaintexts and keys, and this attack method can be treated with very primitive countermeasures. In contrast, with DPA, we perform statistical analysis to exploit the correlation between the hypothetical power consumption and the actual traces extracted by the device, and for this reason, this statistical analysis is called Correlation Power Analysis (CPA). More specifically, in order to perform CPA, we first have to approximate the hypothetical power consumption by getting the hypothetical intermediate values and converting them to a value that characterizes the power consumption of the device. In order to do that, we use a power model such as the models mentioned above that can approximate the device's power consumption. The most common model and the one we use is the Hamming Weight power model. Next, if we have extracted a large set of traces from the targeted device, the CPA will probably find a correlation between the hypothetical power consumption and the extracted traces, leading to a successful Side-channel attack.

### 2.1.2. Profiling Side-channel Analysis

As mentioned above, the most potent Side-channel attack can be made when the hardware architecture or the software encryption algorithm is known. In other words, when the attacker owns a profiling device, a leakage profiling model can be determined. When Suresh Chari et al. [9] first introduced the profiling models idea, mathematical profiling models, namely Template Attacks were used in order to perform Side-channel attacks. Furthermore, by studying the literature [22], [47] we can see that the Machine Learning-based profiling models showed exciting results. More specifically, the Random Forest (RF) and the Support Vector Machine (SVM) classifiers worked well in the context of SCA (Side-channel Analysis). More computationally expensive models based on deep learning architectures were used and showed that neural network implementations like Convolutional Neural Networks (CNN) and Multilayer Perceptron (MLP) are the state-of-the-art profiling models in the SCA field. More specifically, MLP showed excellent performance when unprotected implementations occurred [5], [41] while CNN can perform sufficiently even when severe countermeasures were applied [8]. This can be understood better in the next section, where the CNNs are explained more extensively. We could briefly say that CNNs can work satisfactorily without the need for preprocessing techniques that other machine learning algorithms require. We can also point out that CNNs show a shift-invariant behavior, and thus, they can handle the task of profiling devices with countermeasures like masking and random delays.

### 2.1.3. Side-channel Analysis Metrics

In this project, we use Generative Adversarial Neural Networks (GANs) and Machine Learning-based (ML-based) profiling models. For the implementation of both GANs and ML-based profiling models, we usually try to tune them by studying the machine learning metrics, such as accuracy. In SCA's case, such kind of model performance measuring might be misleading since we can only estimate the profiling model's performance from the classification point of view by measuring the accuracy. Consequently, we cannot measure the attack's performance in general. A better way to measure the profiling model's performance is to use an SCA metric that takes into account the number of traces required to break the key.

For this reason, two SCA metrics are used and can be found in the literature [54], as the Guessing Entropy (GE) and the Success Rate (SR). Both of them are very similar, and by defining the first one, we can easily define the second one.

#### Guessing Entropy (GE)

In order to calculate the GE, we need to define the key ranking algorithm. The key ranking algorithm works as follows: first, we get the probability vector as a result of the profiling model. Each element of this vector is a value ranging from 0 to 1; the element with the highest probability value indicates the predicted class. To find the key's ranking, we try every possible key value. Specifically, when we are attacking the key's first byte, we try 256 different possible keys, and we store these 256 probability values into a vector. Sorting this vector from the highest value to the lowest one and then finding the element's position with the same probability value as the predicted one resulted from the profiling model, we can finally estimate how many guesses we have to make to break the key.

This is the case when we deal with only one trace; while dealing with multiple traces, the procedure changes as follows:

For each trace, we update the 256 element vector by taking into account the previous trace. More specifically, we add to the 256 element vector the 256 element vector's logarithmic values of the previous trace; that is how the key ranking algorithms take into account the information obtained by every trace each time [55].

Eventually, if the profiling model performs relatively well, the key can be retrieved after profiling some traces. That means that if the profiling model manages to classify the traces slightly better than randomly, the key can be retrieved after a large number of traces.

### 2.1.4. Countermeasures

In this subsection, three different types of countermeasures are addressed. The first one is the addition of random delay interrupts [11], a countermeasure that can be characterized as a trivial one like the second one which is called Desynchronization or Jittering, while the third one is the masking countermeasure, is the most common and effective one that can be found in the literature[45]. The main purpose of introducing countermeasures is to reduce the linear correlation between the Side-channel trace-points and the power consumption of the target. Therefore the task of retrieving the key by exploiting the statistical dependencies between the power consumption and the Side-channel traces is getting more complicated or even unfeasible.

#### Random Delay Interrupts

The Random Delay Interrupts countermeasure is introduced by Clavier et al. [11], and in their paper, they focus on the hardware implementation of this countermeasure. However, it can also be implemented easily in a software-based AES implementation by adding random delays to the AES algorithm to change the order of the most critical features and make the algorithm more resistant to Side-channel attacks.

#### Desynchronization (Software Jitter)

The Desyncrhonization countermeasure is a software simulation of clock jitter, and it can be performed by adding a simulated jitter effect on the original synchronized traces. This jitter effect is performed in every clock pattern of the trace by removing points and introducing new points. In this way, the traces are not aligned to each other. By introducing this countermeasure, the dataset is more prone to Side-channel attacks as it is more difficult for the security analysts to perform statistical analysis methods to detect the points of high leakage information in the traces.

#### Masking

Masking is a very effective Side-channel countermeasure technique used to protect sensitive information that can be captured by studying the statistical dependencies between the traces and the secret key. More specifically, this masking is usually Boolean and is applied by using the XOR function during the encryption of the data. In this way, it is difficult to detect specific points of interest where the Side-channel traces leak sensitive information. For example, in Section 2.4 later in this Chapter, we will see that in order to de-mask the dataset, we have to calculate the label again by performing an XOR operation, as can be seen in the following equation.

- $HW(Sbox(p_i \oplus k_i) \oplus mask)$

## 2.2. Machine Learning

Arthur Samuel of IBM firstly introduced machine learning algorithms in the 1950s [2]. He developed a computer program for playing checkers. More specifically, he designed a score function that aimed to measure each side winning's chances. Next, combining the information provided by the score function and John von Neumann's minimax strategy, the program chooses the right action. Of course, only the reward value combined with the minimax strategy cannot indicate the optimum next move. For this

[2]http://infolab.stanford.edu/pub/voy/museum/samuel.html

reason, during every action, Arthur Samuel stored the position and the reward value provided by the score function, and he used the minimax algorithm. Eventually, it seems that the minimax algorithm needs some training first to act positively. Concluding, we can say that Arthur Samuel's idea consists of an algorithm, a score function, and some memory. In other words, after some actions, we can say that the algorithm became robust enough to make the right decision. In the checkers game, the minimax algorithm is getting better and better while it acts. More specifically, the algorithm manages to learn how to play on the fly; in that case, we can say that the minimax algorithm worked in an unsupervised way. Of course, this is not the only way for the algorithm to learn how to choose the right action. In other cases, a different concept is suggested, where the machine learning algorithm has two operational phases, the first phase is to learn, and the second one is to act. More precisely, the name of these two algorithmic phases is the training and test phase. During the training phase, the algorithm is getting more robust and becomes able to act precisely in the test phase. In that case, we can say that the algorithm worked in a supervised way. Unsupervised learning might sound more practical as it is free of training datasets, but this is not easily applicable in all cases. For that reason, machine learning pipelines have been proposed [6] to combine one small set with labeled data with a larger one consisting of unlabeled data; this type of solution is called semi-supervised machine learning. Next, in the following two subsections, two essential types of machine learning models have been explained: the discriminative and the generative models, and then the deep learning concept is introduced.

### 2.2.1. Discriminative Models

The discriminative models usually perform well with classification tasks; more specifically, they manage to capture the conditional probability $P(X|Y)$, where $X$ is the training set samples, and $Y$ is the label set. In other words, a discriminative model is a model built to distinguish samples belonging to different categories.

The most straightforward classification task is binary classification, where the algorithm tries to classify the samples into two categories. In order to do that, the algorithm addresses a set of two probabilities, one for each class, and the class with the highest probability is the predicted label. This may sound simple, but entering more in-depth, we can see that machine learning algorithms are trained under many different configurations. These configurations are called hyperparameters, and the goal is to tune them as much as possible to achieve maximum performance. The hyperparameter optimization task sometimes is straightforward, as some more simple algorithms might need less optimization. In some other cases, when the algorithms are more complicated, like neural network implementations, the hyperparameter optimization might be a painful and time-consuming task, but it probably leads to more accurate models. One example of a robust discriminative model in classification tasks is the RF (Random Forest) classifier explained in Section 2.2.2.

### 2.2.2. Generative Models

The generative models usually perform well with unsupervised learning tasks like data generation tasks. More specifically, they can be explained by thinking of a model that can generate new samples belonging to a specific distribution. In some cases, the generative models can also be used to solve classification problems instead of discriminative models. For example, the Naive Bayes classifier is a generative model that often is used in classification tasks in a supervised way. Identifying the differences between the two models is essential to understand how Generative Adversarial Neural Networks operate. Specifically, the generative models try to capture the joint probability distribution $P(x, y)$, where $x$ is the samples of the dataset, and $y$ is the label of the dataset. The most straightforward generation task is to predict the next word in a sequence, and a solution to that task can be provided with an Auto-Regressive (AR) model [1]. The AR model is a simple generative model widely used in statistics, econometrics, and signal processing. Briefly, we can say that AR is a stochastic process and works as linear regression and makes a short-term prediction based on the short-term past, but it should be avoided for tasks that need long-term prediction.

The choice between supervised and unsupervised learning is closely related to the nature of the machine learning algorithm and the nature of the problem. In the next subsection paragraph, the RF's essential components are addressed as this algorithm is the main machine learning algorithm that has been used for SCA profiling in this thesis project.

### Random Forest

Random Forest (RF) is widely used in this project as a profiling model as it seems to perform well in SCA, especially for unprotected implementations. However, it is a model that can be trained relatively fast and can be used to make assumptions and run fast simulations to make comparisons between different GANs models and other augmentation techniques. For these reasons, it is essential to mention the necessary components of the RF classifier. In brief, the RF is a supervised learning algorithm, and it is a collection of decision trees that can be used in classification and regression tasks.

More specifically, RF performs better than decision trees as multiple decision trees are used simultaneously, which are trained on different parts of the same training set. In order to construct different parts of the same training set, we use the bagging method. The bagging, which is also called bootstrap aggregating, is used in RF, but other profiling models use it like artificial neural networks as it makes the training procedure more stable and reduces the variance. Therefore, it helps avoid overfitting, but it is not always beneficial for stable models like the nearest neighbors classifier, as the bagging might reduce the algorithm's performance in that case. This technique works in two steps; the first one is the bootstrapping step, where we split the original dataset into several subsets, and each subset consists of unique samples and other samples that might already exist in the rest of the sets. Every model is trained to its corresponding set, and then an ensemble learning process takes place where a voting scheme between the models predicts the chosen class. This procedure also can be seen visually in Figure 2.1

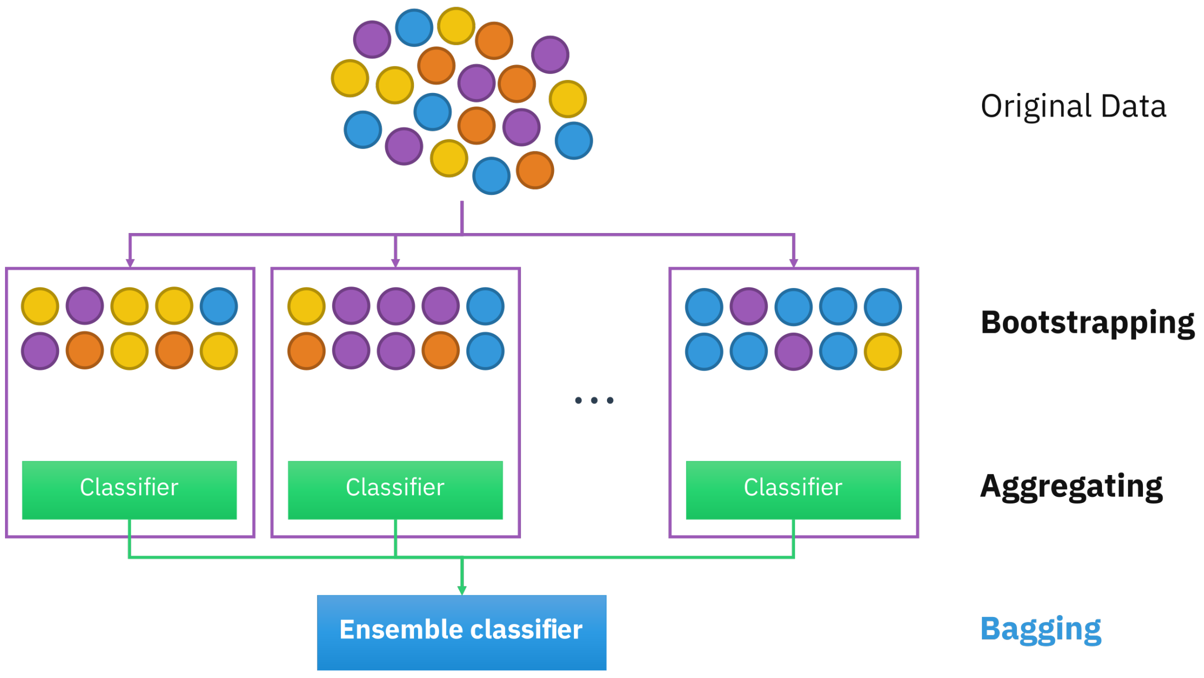


Figure 2.1: Bagging Algorithm [3]

Next, the RF classifier performs a different bagging method called feature bagging. This kind of bagging is the main difference from the regular decision trees. The way this bagging method works is to look for the most significant feature among all, and every decision tree takes into account a random subset of the feature set. As a result, models can capture a wide diversity of the dataset, and therefore the RF avoids overfitting.

In conclusion, RF combines the bagging technique and the feature randomness, and therefore the algorithm can construct statistically correlated decision trees, which are much more accurate than a single decision tree implementation. The RF's slight disadvantage is that the pre-trained models are relatively slow during the test phase.

[3]figure source: https://en.wikipedia.org/wiki/Bootstrap_aggregating

### 2.2.3. Deep Learning

In SCA, various machine learning algorithms like Random Forest(RF) and support vector machines (SVM) have been used in order to profile the clone device. However, recently, very effective profiling methods have been addressed in the literature [5] that built with deep learning algorithms like Multilayer Perceptron (MLP) and Convolutional Neural Networks (CNN). We can say that neural network models show exemplary behavior because they can deal effectively with multidimensional datasets. That means we can avoid time-consuming data preprocessing tasks, especially when CNNs have been used as these models showed good performance when countermeasures like features desynchronization or masking had been applied [41]. These promising results show that there is an intuition that Generative Adversarial Neural Networks (GANs) might give valuable generated traces.

In the next subsection paragraph, the Multilayer Perceptron's and Convolutional Neural Networks' essential components are reported as they are mainly used as profiling models in this project along with the random forest classifier.

#### Multilayer Perceptron

In order to explain the basics of deep learning, we have first to look at the first deep learning supervised algorithm. Maybe neural networks sound like a new domain of machine learning, but in order to look to the first neural network model, we have to go back to 1967 when Ivakhnenko and Lapa developed the first deep feedforward multilayer perceptron [27]. In their paper [27], they described a deep network with eight layers, and each one of the layers was a set of many components. More specifically, the perceptron's idea came by studying the human's brain anatomy; in other words, the perceptron is a simulation of a natural biological nerve cell, and as each nerve cell has a neuron, so the same we can say for the perceptron. By looking into the Figure 2.2 it can be seen multiple inputs in the neuron; these inputs are multiplied with the neuron's weight. Finally, this value goes through a non-linear activation function that acts as a threshold. If the neuron's value is over that threshold, this value is converted to a value that depends on the used activation function. A single layer Perceptron may be able to deal with linear classification tasks while complex classification tasks always require non-linear classifiers. This can be achieved by adding multiple layers and non-linear activation functions, which will introduce the required non-linearity to the classifier.

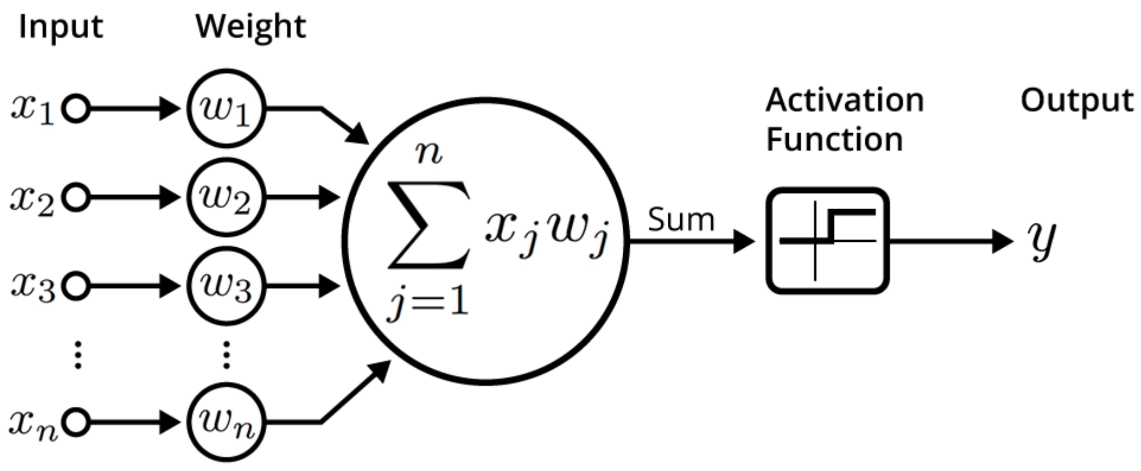


Figure 2.2: Neuron [4]

#### Convolutional Neural Networks

Like the Multilayer Perceptron algorithm, the Convolutional Neural Networks (CNNs) were first introduced in the 50s, and their initial idea came by studying the visual cortex of animals. We can say that artificial convolution neural networks try to mimic animals' vision as it is known that its classification abilities are extraordinary. This was understood by the machine learning society when Yann LeCun in 1994 [34] showed that for computer vision tasks like recognizing handwritten zip code numbers, the

[4]figure source: https://becominghuman.ai/artificial-neuron-networks-basics-introduction-to-neural-networks-3082f1dcca8c

CNNs are very accurate. More specifically, he proposed a pioneering Convolutional Neural Network, namely, LeNet5 that can be seen in Figure 2.3 which was the most successful version of a series of LeCun's CNNs since the year 1988. Lecun showed that convolutional layers could capture similar features that are distributed across the entire image, and therefore CNNs are very efficient on shape recognition tasks. In other words, CNNs show a shift-invariant ability that worked well for image classification tasks, and for the same reason, in the field of SCA, CNNs showed promising results when datasets with countermeasures occurred as the regions that show high data leakage can be located successfully.

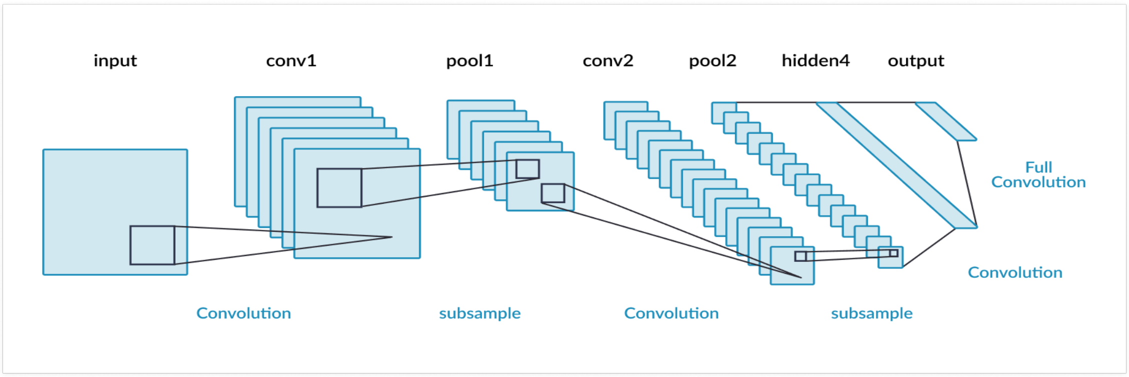


Figure 2.3: LeNet5 [5]

By studying the literature in most of the CNN architectures, we can find several convolutional layers, followed by one or more fully connected layers and, lastly, a softmax layer. Each of the convolutional layers consists of a filter, a non-linear activation function, and a pooling layer. The input and the output of every hidden convolution layer are called feature maps, and in practice, there are multidimensional arrays. Each feature map is the dot product operation's outcome between the image and the filters when we are in the first convolutional layer. For the rest of the convolutional layers, the feature maps are the output of the dot product between the filters and the previous feature maps. In order to understand the convolution operation between the feature maps and the filters, we can see in Figure 2.4 that every filter works like a window that goes through the data and calculates the element-wise product of this region, which is called the receptive field. Then these dot products are summing into one value for every sliding action. Then we can see in the LeNet5 architecture 2.3 the next layer is the pooling layer, which down-samples the features in order to keep the most important of them. Next, it passes through the non-linear activation function that is explained previously in the MLP part. Then, this whole operation ends up with fully connected layers combined with a Flattening layer, which transforms the multidimensional array into a single vector to combine all these features. Lastly, it goes through a softmax layer responsible for giving a probability value to every class respectively.

Now that it is clear what happens when an input goes through the neural layers, we have to talk about the Neural Network training. In order to do that, we need to adjust the neurons' weight according to the output of the last layer, and therefore, we have to define the Loss function, the optimization algorithm, and lastly, the Training algorithm.

### Loss Function, Optimization Algorithm, and the Training Algorithm

Every machine learning algorithm's goal is to find a way to tune the weights and the biases, so the algorithms' output for every training input is a successful approximation of the expected output. For example, in a classification task, we expect that the machine learning algorithm predicts the corresponding label correctly, and in order to do that, we have to define a cost function. This cost function is also called the objective function or loss function. The term loss is the most common one, as in every machine learning task, we try to minimize the distance between the predicted value and the expected value. Different loss functions can express this distance, and of course, this depends on the nature of

[5]figure source :https://missinglink.ai/guides/convolutional-neural-networks/convolutional-neural-network-architecture-forging-pathways-future/

[6]figure source: https://medium.com/@pallawi.ds/ai-starter-build-your-first-convolution-neural-network-in-keras-from-scratch-to-perform-a059eaa6d4ff

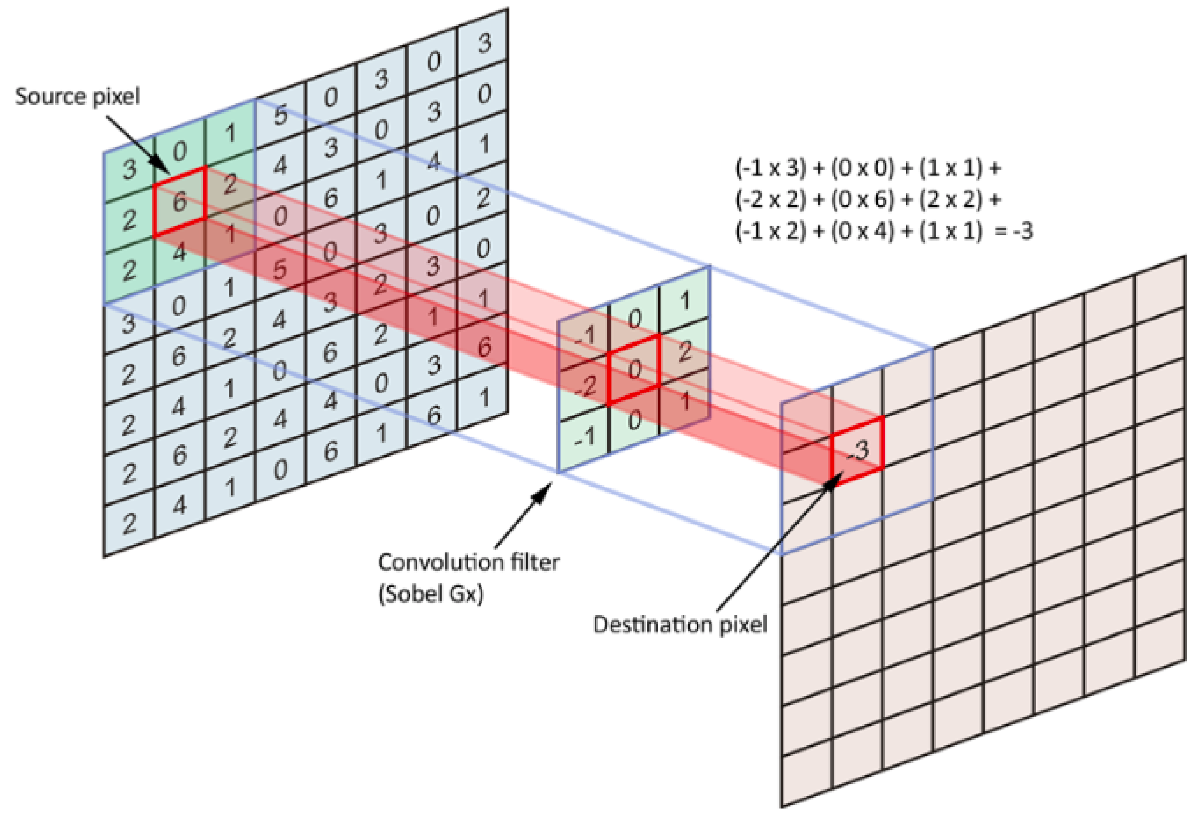

Source pixel
(-1 x 3) + (0 x 0) + (1 x 1) +
(-2 x 2) + (0 x 6) + (2 x 2) +
(-1 x 2) + (0 x 4) + (1 x 1) = -3
Convolution filter
(Sobel Gx)
Destination pixel

Figure 2.4: Feature maps [6]

the problem. For example, in a linear regression problem, the loss function is the Mean Squared Error (MSE), which can be expressed by the following formula:

$$L(w,b) = \frac{1}{n}\sum_{i=1}^{n}(y(x) - a)^2 \tag{2.1}$$

The term **w** and **b** corresponds to the weights and the bias, respectively, which need to be tuned in order to minimize the difference between the predicted value **a** and the expected value **y(x)** while the term **n** denotes the number of training samples. More specifically, this function is quite popular in simple tasks like linear regression because it can provide a convex function, meaning that there is one minimum, which is the global minimum.

In more complicated tasks like logistic regression, another loss function is used as the MSE does not provide a convex function but a function with multiple local minima and the optimization algorithm might fail to find the local minima with the lowest loss score. For these reasons, under different problems and algorithms, different loss functions can be used. In Neural Networks, the most common loss function is the Cross-Entropy loss. The Cross-Entropy loss is calculated as the difference between each predicted class probability with the expected class output. The mathematical formula can be found below.

$$L_{CE}(w,b) = -\sum_{i=1}^{n} t_i log(p_i) \tag{2.2}$$

Where **n** indicates the total number of classes, **i** is the corresponding class, **t** is the true class and the **p** is the predicted probability.

Without loss of generality, we can think about a binary problem, then the formula 2.2 is:

$$L_{CE}(w,b) = -\sum_{i=1}^{2} t_i log(p_i) = -[tlog(p) + (1-t)log(1-p)] \tag{2.3}$$

Now we can see that in the best case scenario where the class 1 has probability 0.9 and class 0 has probability 0.1 then $L_{CE}(w,b) = -[1log(0.9) + 0log(0.1)] = 0.05$ while when is the opposite $L_{CE}(w,b) = -[1log(0.1) + 0log(0.9)] = 1.$

From this example, it is clear that Cross-Entropy loss is minimal when the probability value of the correct class is close to 1 and large when the probability value of the correct class is close to 0.

Now we have to define the optimization algorithm. The most popular optimization algorithm and the one we use in this thesis project is stochastic gradient descent (SGD).

To minimize the loss by adjusting the weights, we have to find the loss function's global minimum. This global minimum can be found by adjusting the weights and looking at where the loss function derivatives are equal to zero. The way to do that is to make slight adjustments in the weights and calculate the derivative. If the derivative is negative, then it means that we are going towards down to the minimum, while if it is positive, we are going upwards up to the maximum. In other words, the derivative shows the gradient. Following this method sounds like the optimization procedure is a trivial task, but non-convex loss functions might lead this method to fail in practice. Further tuning or more sophisticated algorithms are required based on the gradient descent algorithm like the Stochastic Gradient Descent, the Adam Optimizing algorithm, and the RMSProp algorithm.

Lastly, we have to define how we update the weights of all the layers simultaneously. The well-established algorithm that achieves that weight updating is called the backpropagation algorithm. In a brief explanation, we can say that the backpropagation is an expression for the partial derivatives (that the SGD needs in order to minimize the loss function) of every weight in the network. The reason why backpropagation operates relatively fast is due to the fact that it is based on the chain rule technique.

## 2.3. Data Augmentation and Data Generation Methods

This section explains the Generative Adversarial Neural Networks (GANs) theory and the essential parts of a prevalent augmentation technique: Synthetic Minority Oversampling (SMOTE). We focus more on GANs by explaining what generator and discriminator models are. Furthermore, we make a small reference on how GANs are connected with the game theory, and then we mention two different categories of GAN failures.

### 2.3.1. Generative Adversarial Neural Networks (GANs)

In the previous section and within this section, the Generative Adversarial Networks are mentioned several times, while this project's goal depends on their performance. For this reason, in this subsection, we define more carefully what are GANs and how they work. In a short description, we could say that GANs are generative models consisting of two deep networks. The initial idea was introduced in the 2014 paper by Ian Goodfellow et al. [19]. Specifically, the first models' generator and discriminator were primarily based on MLP architectures and were unstable and difficult to train. However, later, CNN layers were introduced in the generator and discriminator, and therefore, GANs managed to generate very realistic images. Furthermore, researchers managed to profile the behavior of the GANs by introducing metrics like the Inception Score (IS) [51] and the Frechet Inception (FID) [21] score.

In general generative models are unsupervised learning tasks that manage to capture the dataset's distribution, in other words, to learn the specific regularities and patterns of the dataset and therefore are able to generate new samples that are indistinguishable from the original dataset. GANs follow the same tactic as every generative model but are more powerful and more unstable than classical generative models. On the other hand, GANs are the state-of-the-art solution in various data augmentation tasks, and the reason lies in the fact that they use deep learning models. More specifically, they use two neural network models that compete with each other, a generator and a discriminator. The generator tries to generate new fake samples of the dataset while the discriminator tries to classify the dataset samples as fake or real. The two models are two opponents in a zero-sum game, they compete with each other, and we can say that generator can generate plausible data samples when the discriminator model accuracy on classifying synthetic samples is about 50 percent, meaning in practice that the model cannot decide what is fake and what is real. In the literature, this procedure is usually characterized as an unsupervised learning task. However, if we look closely, we can see that we train two models. In the first model, the generator is trained from the feedback that it gets from the classified data samples of the discriminator, and the discriminator is trained by using traces that are real and traces generated by the generator. In other words, we approach an unsupervised learning task by using two supervised learning sub-tasks.

In the following four parts, the generator model and the discriminator model, and the competition between them is explained more extensively, and in the last subsection paragraph, we talk about the need for deep learning models in the field of generative models.

**Discriminator Model**

The discriminator model is trained by getting as inputs real data and synthetic data generated from the generator model. Therefore, we can say that we have a binary classification task between real and fake data samples. So the discriminator's learning process can be characterized 50 percent as supervised, and the rest 50 percent as unsupervised as the generated traces do not emanate from a pre-existing dataset. After the training phase and during the data generation phase, the discriminator does not play any role in generating the synthetic new traces and, therefore, is discarded. On the other hand, during the training phase, the discriminator plays a significant role as its performance shows how good are the generated data samples and, at the same time, by trying to get improved, forces the generator to produce better synthetic data.

Usually, the architecture of the discriminator is based on multilayer perceptron or convolutional neural networks. In this thesis project, the discriminator consists of two convolutional layers, and at the end, one fully connected layer, a more extensive explanation about the architecture of the discriminator can be found below in the fourth Chapter. In contrast, in this current Chapter, more theoretical claims and assumptions are addressed. To explain how the discriminator works, we can think of an example where the discriminator takes as an input a high-resolution image or a long time series like the extracted power traces in our case. With the convolutional layers' particular structure, the features are converted into a smaller set of more compressed and more representative ones. As explained previously in the CNN subsection, the discriminator manages to classify what is real and what is fake if the generator produces synthetic traces that are not very realistic. Thus, we should bear in mind that the more influential the discriminator model is, the more powerful the generator is.

**Generator Model**

The generator, on the other hand, is based on a pure unsupervised learning task, as the input for the generator is a vector of numbers that are ”drawn” randomly from a Gaussian distribution. In other words, we put noise in the generator's input, and the dot products between the weights of the generator and the features of the random noise give a structure to that noise and reform it into an image or into a power trace in our case. More specifically, we choose how large is this random vector, and this vector space is called latent space. Briefly, we can say that we put random noise as an input with a specific dimensional space, and we get as an output a meaningful output that belongs to a larger dimensional space. The random input vector size indicates how many features are needed to construct a meaningful output. Therefore, this latent space is one more extra hyperparameter of the generator model, and the tuning of this hyperparameter is crucial. Nevertheless, it is pretty challenging to choose the latent space's size as we need first to obtain some knowledge about the dataset's distribution.

**Game Theory**

Game Theory can be defined as [17] a mathematical approach to every competition game between rational decision-makers. Traditionally game theory is applied in various fields from social science to economics, but recently Ian Goodfellow [19] introduced it through GANs in the machine learning community. The modern game theory is based on the fundamental theorem of John von Neumann, which states that almost in every type of a zero-sum game with a finite set of actions, there is an equilibrium that all players benefit from converging to it. More specifically, GANs is a game of the two players that compete with each other, namely the generator and the discriminator, and it can be characterized as a finite zero-sum game between two-person that are aware of each other's gains and losses. More specifically, a zero-sum game means that the first player's gains are the losses of the second player and vice versa; also, the game has to be finite, meaning that the moves that a player can make are limited. Lastly, both players should obtain all the information about the game, the losses, and their opponents' gains. If all three conditions apply, then there is a mixed strategy that can lead to the desired equilibrium. In theory, the objective of GANs is to reach that equilibrium, and at the moment this is done, the generator produces samples that are indistinguishable from the real ones, and therefore we can say that the quality of the generated samples can be unlimited.

**Generator vs. Discriminator**

As mentioned above, the generative models work as an unsupervised learning task, although by framing the training procedure into a zero-sum game where the discriminator competes with the generator, the generative modeling task converts to a supervised learning problem, and the training procedure now works as follows:

We collect a batch of real samples and a batch of fake synthetic samples produced by the generator. Then the discriminator classifies these sets of two batches either as real or fake. The discriminator's weights are updated with the backpropagation algorithm to distinguish the real from the fake samples. Next, the generator is updated again using the backpropagation algorithm based on how well the fake samples fooled the discriminator. So we can say that when the discriminator manages to classify all the real and all the fake samples correctly, we do not have to update the weights. On the other hand, this means that the generated data samples are not as good as they should be, and the generator gets penalized in order to produce plausible samples. The training should be stopped when the goal is reached, meaning that the generator managed to produce plausible fake samples finally, and the discriminator has 50 percent accuracy.

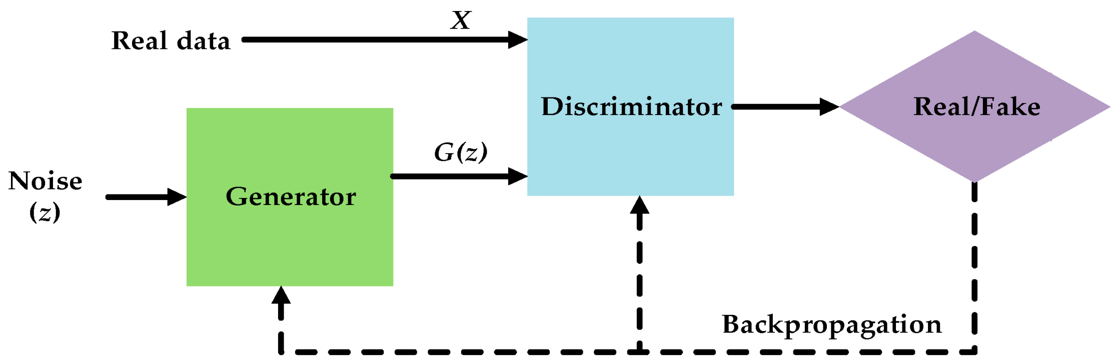


Figure 2.5: Generator vs Discriminator [16]

All this procedure might sound simple as we have to train the discriminator model to perform well, and then by using the discriminator, we can train the generator to produce plausible fake samples until the discriminator drops its accuracy to 50 percent. In reality, this is not how GANs work, as we have to be careful when we choose the generator's and the discriminator's architecture because these two models compete with each other, and one trains the other by making tiny steps. In other words, when the discriminator's performance is slightly getting improved, then almost instantly, the generator has to perform a bit better, then the discriminator's accuracy falls a bit, and for that reason, the weights are updated in order to become even better, and that way goes on. Simply, the point for both models is to compete with each other and at the same time to train each other step by step. In any other case, if the discriminator manages to learn very fast and get powerful, the generator will fail to compete. As in every competition game, the adversaries are getting better from each other when they are on the same level of performance, while in the case that one of them is much stronger than the other, the game is over. In practice, this fact can be noticed when we deal with GANs, and there are two specific failure cases, namely, mode collapse and convergence failure.

**Failure Modes**

As mentioned above, the failure cases are two, the mode collapse and the convergence failure. In the first failure case, the mode collapse can be identified when the generator model is "capable" of generating samples that belong in a finite set of outcomes. More specifically, in an image generation task, as mentioned before, the generator takes as an input a random vector that belongs to the latent space and transforms this random noise into an image. The generated images are images that actually belong to the dimensional space defined as latent space, and we choose the latent space to be large enough, so the generate synthetic images differ from each other in various ways, and this depends on the number of the dimensions. Now when a mode collapse failure happens, the generator cannot take advantage of the whole dimensional space, and eventually, the generated images belong to a minimal subset of the total possible images. In extreme mode collapse failure scenarios, we can see in the tutorial of GANs of Ian Goodfellow [51] that the generator always maps several different inputs resulting in the same output, meaning that the generator produces the same single sample.

In image generation tasks might be easy to identify a mode collapse visually, but in other tasks, like in our case, the evaluation of the generated traces is a difficult task, but there are also other ways of identifying this failure case. This can be done by studying the evolution of loss during the generator's and the discriminator's epochs. More specifically, oscillations in the generator's loss show that one snapshot of the model differs significantly from the previous. One way to force the generator into mode collapse failure is to change the hyperparameter that defines the size of the latent space into a minimal value; in that way, we can see that the generated samples come from limited dimensional space, and therefore, every sample will be identical to the others.

The second type of failure, called convergence failure, is the most common one. This is a classical failure that can be found in any neural network, machine learning, or any optimization algorithm as the loss might not converge towards zero during training. When we are dealing with GANs, this is not precisely the case; we do not care if the generator's and discriminator's loss is close to zero. Actually, this is not a desirable outcome and should be avoided because a sudden drop of loss in any of these two models is not something that we usually expect. Instead, we anticipate that both models will eventually converge to the equilibrium, as discussed in the Game Theory subsection paragraph. So we can say that convergence failure happens in two cases when the generator's loss is close to zero or continue to rises over the epochs. The main reason for this failure is that either the discriminator is very powerful or the generator is very weak. So, the generated data samples can be very easily perceived as fake by the generator; thus, in such a case, the model's training must be stopped from the beginning as the generator will never converge to the necessary equilibrium.

### 2.3.2. Synthetic Minority Oversampling Technique (SMOTE)

In 2002 Chawla et al. [10] proposed a pioneering augmentation method called the Synthetic Minority Oversampling Technique (SMOTE) focuses on the problem of the imbalanced Machine learning dataset. This method was the state-of-the-art for many machine learning problems with imbalanced datasets. Now we can say that GANs are the state-of-the-art methods for making augmented balanced datasets. For that reason, in this thesis, we compare the profiling methods' performance, which is trained in the augmented dataset by GANs and SMOTE.

More specifically, in order to tackle the class imbalance problem, SMOTE produces synthetic samples that were added to the minority class, while the most common method in the past to tackle this class imbalance problem was techniques of undersampling, meaning to discard samples from the majority class. Chawla et al. [10] showed that SMOTE combined with undersampling techniques could lead to classifiers with good performance. The basic idea was to generate new synthetic samples that do not depend on the dataset's application, while this generation of the samples takes place in the feature space and not in the data space. This is why the SMOTE is better than its predecessor, which is the minority oversampling with replacement. More specifically, the minority oversampling with replacement operates by replicating the data and not generating synthetic traces. With these replications, the data becomes less generalized as makes the machine learning algorithms' decision borders more narrow, and eventually leads them to overfitting phenomena.

The way the SMOTE algorithm works is the following: we define a set of neighbors for each sample; then, we multiply the difference between the sample and the nearest neighbor with a random number that is larger or equal to 0 and smaller or equal to 1. This new feature is stored in a vector, and we do the same for every feature. In that way, we generate a new synthetic sample that takes place somewhere between the original sample and its neighbor. This is why SMOTE forces the decision region of the minority class to become more general, and therefore the machine learning algorithm, which is trained in the new augmented dataset, avoids overfitting.

## 2.4. Dataset

For this thesis project's experiments, we used protected and unprotected software implementations of an Advanced Encryption Standard (AES) crypto-algorithm. The dataset used is the well-established ASCAD which is explained below. The Hamming Weight (HW) power model is used in order to construct the labels.

### 2.4.1. ASCAD

The ASCAD (ANSSI SCA Database) [7] is one of the most popular trace-set in the SCA community. In this thesis project, we use an unprotected version and a first-order boolean masked AES version. For both versions, a fixed subversion and a random key subversion of the dataset are used in order to perform various experiments on GANs and profiling models. For a protected implementation, the database's provided labels are constructed by targeting the Sbox output and taking into account the third byte of the plaintext and key. The ASCAD database does not provide the labels of an unprotected implementation; therefore, in order to perform experiments in an unprotected implementation, we have to use the provided scripts in the GitHub [8] repository to construct them by taking into account the first or the second byte of the key and the plaintext.

The labels are constructed by the following formula:

$$label = HW(Sbox[plaintext_i \oplus key_i]) \tag{2.4}$$

The fixed key dataset consists of 50,000 profiling traces and 10,000 attacking traces, while the random key dataset consists of 200,000 profiling traces and 100,000 attack traces, and only 77 percent of them are random while the rest are fixed.

[7] https://eprint.iacr.org/2018/053.pdf
[8] https://github.com/ANSSI-FR/ASCAD

# 3 Related Work

The performance of Machine Learning and Deep Learning models depends on many factors. One critical factor is the quality of the training datasets. The construction of a dataset is crucial for every Machine Learning problem as this usually requires much effort. The same and even more effort is also needed in the Side-channel Analysis field, where the SCA datasets are constructed by performing the time-consuming and challenging trace acquisition task. Recently the Side-channel Analysis researchers focused on Machine Learning techniques [47] and especially in Deep Learning techniques [41], and they successfully managed to perform powerful SCA attacks. On the other hand, to the best of our knowledge, very limited research [48], [56] is focused on data augmentation methods and especially on Generative Adversarial Neural Networks (GANs) that could be used in order to improve the performance of the (pre-existing) profiling models in SCA field.

## 3.1. Generative Adversarial Neural Networks

Generative Adversarial Networks (GANs) might be a relatively new concept but proved to be one of the most fast-growing concepts in the Deep Learning community [44], [39], [12]. The reason for this is the ability of GANs to be applied in a vast field of applications, to computer vision [14], [40], [58], to natural language processing [15], [28] and to time series synthesis [13], [38]. GANs are the state-of-the-art generative models for many applications in academia but also in the computer industry, as many software applications showed that GANs perform well in image generation tasks[1]. More specifically, artificial intelligence start-up companies[2][3] are taking advantage of the GANs' ability to generate sets of synthetic images with similar characteristics but with almost unlimited diversity between them. For example, Icons8 [4], an Argentina-based design firm, use various alternatives of the GAN model that Tero Karras et al. [29] proposed, namely StyleGAN to generate synthetic images of human faces so realistic that a human cannot detect their differences from a real image with a naked eye. The influence that GANs can exert in society can be perceived if we consider that GANs can be evolved into dangerous malicious software tools that could lead to terrifying results. One example is the deep-fake[5] software tool that can be easily converted to malicious software in order to operate as a crowd manipulation tool, i.e., by producing fake videos of politicians talking on behalf of some individuals.

We can say that GANs showed good performance when image generation tasks occurred; however, even if they were applied with some success in time series synthesis tasks, the research is limited compared to the GANs applied in computer vision. This is happening because there is the intuition that GANs might do not show good performance in sequential data. After all, this category of data has quite different properties than the image data. Kaleb Smith and Anthony Smith [52] try to take advantage of these 2-dimensional properties of the images. They use two WGANs [2], the first WGAN converts the

[1]https://www.washingtonpost.com/technology/2020/01/07/dating-apps-need-women-advertisers-need-diversity-ai-companies-offer-solution-fake-people/

[2]https://generated.photos

[3]https://www.rosebud.ai

[4]https://icons8.com

[5]https://medium.com/@lennartfr/deepfakes-and-the-world-of-generative-adversarial-networks-bf6937e70637

random vectors to spectrogram images while the second takes as input the spectrogram images and generates time series.

Furthermore, Jinsung Yoon et al. [60] proposed the TimeGAN to generate time series. The basic idea is that they use an auxiliary embedding network, which maps the high-dimensional features in lower-dimensional space, and in this way, they facilitate the training of the TimeGAN. The TimeGAN achieves promising performance, but it is mostly applied in stock prices analysis and appliances energy prediction and not in SCA.

Furthermore, worth to be mentioned is the work of Jerry Wei et al. [57] as it could be helpful when we use GANs for SCA datasets. The authors observed that usually collecting enough training data for rare diseases in order to train Deep Learning models is a challenging task. More specifically, they focused on building colorectal polyps synthetic images in order to balance the ratio between colorectal polyps images and healthy images. In order to achieve that, they used cycleGANs to produce a balanced augmented dataset and observed that synthetic colorectal polyps images looked healthy, and therefore the balanced augmented dataset could not improve the performance of the Deep Learning image detection models. For this reason, they focus on the problem where the samples of different classes look similar, and the proposed solution to this problem was to use a secondary Machine Learning algorithm to select the samples when the algorithm is very confident. More specifically, with this method, they collect samples that belong to the right class with a high probability value, meaning that these samples have robust precancerous features. In this way, the generated samples have been proved to be of higher quality than those before using this method.

We have to consider that the same might happen in SCA, while slight differences in the samples can be found across different classes. Therefore applying this method in SCA datasets could be very helpful in the training of GANs, but first, we have to consider that in order to apply this method, accurate profiling models are needed, something that is not expected when protected implementations are used. On the other hand, this method theoretically could work relatively well on unprotected SCA datasets. Furthermore, when the HW power model is used, this method could only work for the majority classes, i.e., 3,4,5 where the number of traces is relatively large. Instead, for the minority classes, this method should be avoided as it also works as a *"downsampling"* method.

## 3.2. Power-Trace Augmentation Methods in Side-channel Analysis

Cagli et al. [8] first introduce the augmentation techniques in the SCA field. They proposed two ad hoc data augmentation methods namely, *Shifting Deformation* and *Add-Remove*. More specifically, with the *Shifting Deformation*, the authors manage to simulate random time shifts in the traces. These random delays have been chosen by picking a number between 0 and a threshold $T$ randomly. Furthermore, random delays mostly occurred in the central part of the traces. In this way, the authors managed to produce new traces that consists of random deformations of the original ones. Next, they produced multiple augmented datasets for different thresholds $T$, and then they trained a CNN without altering its architecture. The performance of the CNN improved significantly as for large $T$ (500), the validation accuracy started to converge into the training accuracy, meaning that the CNN was not overfitting to the training data.

The second augmentations technique, the Add-Remove, worked as a simulation of a clock jitter effect. They inserted and suppressed a random number of time samples. Again they set a parameter R to define the maximum value of this number. Next, they randomly chose the inserted and suppressed time samples' position while the values of the inserted time samples have been chosen by calculating the mean between the previous time sample and the following one.

Lastly, they combined the two augmentation methods, and they observed that the overfitting phenomenon of the CNN was further reduced under a combination of T and R, but they noticed for large values of T and R, the performance of the CNN could deteriorate.

Picek et al. [48] investigated the class imbalance phenomenon of the SCA dataset when the Hamming Weight (HW) model is used. This phenomenon can lead to poorly performing profile models, as previous projects [47] show that Machine Learning methods are prone to fit accurately into the majority classes and, therefore, to perform well only in these classes. For this reason, balancing techniques were used. More specifically, Cost-Sensitive Learning and re-sampling techniques like Random Undersampling, Random Oversampling with Replacement, and Synthetic Minority Oversampling Technique (SMOTE) were applied in the dataset in order to improve the performance of the profiling models.

The augmented datasets' results were very similar to the original dataset when unprotected training datasets were used. Instead, when protected datasets were used, the results were promising. Nevertheless, in some cases, the balancing techniques did not manage to improve the profiling methods' performance in terms of Success Rate and Guessing Entropy. Conclusively, on average, SMOTE was the best balancing technique. In this way, the Random Forest (RF) classifier's performance when trained on the SMOTEd data appeared to be improved more than eight times in respect to the RF's performance when trained in the original imbalanced data. However, this was not always the case; in other profiling models like the pooled template attack model, the augmented dataset led to worse results.

Ping Wang et al. [56] investigated the problem of conducting efficient profiling attacks when the number of the extracted power-traces is limited. More specifically, they used Conditional Generative Adversarial Networks (CGANs) [44] to generate synthetic traces out of a limited set of real power-traces, which, as stand-alone, is not sufficient for a profiling attack. They emphasize that they do not focus on performing a powerful profiling attack as the authors do not seek to construct a powerful profiling model, but they investigate if CGANs can generate valuable traces that would further improve the profiling models' performance. They used four types of profiling models, namely, Random Forest (RF), Support Vector Machines (SVM), Multilayer Perceptron (MLP), and Convolutional Neural Networks (CNN). They concluded that when the Hamming Weight (HW) of the Sbox's output is used as a label, CGANs cannot generate valuable traces in order to conduct a successful attack. The authors claimed that this issue's main reason was some kind of noise in the generated traces, which could be observed in the Correlation Power Analysis (CPA) plots of the augmented trace sets, and they called this noise "leakage noise". This leakage noise phenomenon occurred because of the original's dataset imbalanced nature. They noticed that the training dataset's minority classes 8 and 0 do not have a sufficient number of traces in order to train the CGANs, and therefore the quality of the generated traces was low.

Thus, they used as a power model the Least Significant Bit (LSB), and therefore the label set was constructed by taking the LSB of the Sbox's output. Using this power model, they converted the multiclass problem to a binary class problem, and they observed that CGANs could generate traces that can improve the performance of the profiling models. They used protected and unprotected implementations, and they evaluated the generated traces by visually observing the shapes of the traces to see similarities between the original traces and the generated traces. They also demonstrated a comparison of the CPA results between the original and the generate traces to see if the generated traces have shown leakage picks in the exact locations as the original traces. Lastly, they evaluated the generated traces by looking at the GE improvements of the profiling models when trained on the augmented datasets (a fusion of the original and the generated dataset). However, they did not use Machine Learning (ML) evaluation metrics like the Inception Score (IS) and the Frechet's Inception Score (FID) as the ML metrics are not the best choice of evaluation in the SCA field [48].

In conclusion, they demonstrated that CGANs could work relatively well when the chosen power model is the LSB. Looking at their results, we can see that the LSB power model's choice leads to incompetent profiling attacks compared to the ones where the HW or the identity power model is used. Furthermore, they showed that in the best-case scenario, the augmented datasets produced by the CGANs improved the profiling models' performance regarding the Guessing Entropy (GE) by four times while Picek et al. [48] showed that in some cases, SMOTE could improve the performance of the profiling models by eight times. Lastly, they showed that with their CGAN model, they managed to construct fully synthetic datasets used to train a profiling model in order to perform an attack. This indicates that the generated traces can provide valuable information. However, they do not explain if there is any practical reason for generating and using fully synthetic trace sets.

## 3.3. Research Questions

In the Introduction Chapter 1 we came up with two abstract research questions; while now taking into consideration the Background Chapter 2 and the Related Work Chapter 3 we define in more detail the four following research questions.

1. Does the profiling model's performance depend on the dataset's class distribution, and more specifically, should the augmented datasets follow a binomial or a uniform class distribution?
2. Is it feasible for Generative Models such as complex Generative Adversarial Neural Networks(GANs) to generate valuable traces in order to construct synthetic datasets for Side-channel traces?

   (a) Is it feasible to perform dataset augmentation with GANs in order to improve the performance of a side-channel analysis (SCA) attack when the training set is limited?
   (b) Do GANs perform better than traditional augmentations techniques like the Synthetic Minority Oversampling Techniques (SMOTE) on SCA datasets?

3. How can GANs be evaluated from a Machine Learning perspective in order to achieve a parameter optimization for these models by considering that visual representations of the generated SCA traces are not an option?
4. Does the performance of the GANs depend on the complexity of the dataset, meaning that more countermeasures require deeper neural architectures?

# 4 Investigation of the Classes' Distribution

By studying the literature, [8] [48], and [56] it can be seen that researchers approach the problem of Side-channel traces augmentation from different perspectives. Cagli et al. [8] propose a more ad hoc way of performing augmentation by applying two augmentation techniques, namely *Shifting Deformations* and *Add-remove* on the acquired traces. With the two techniques explained more extensively in the Related Work Chapter 3.2, they managed to take advantage of the Neural Networks' ability to exploit large datasets without needing other preprocessing techniques. Next, Picek et al. [48] chose to perform balancing in the SCA datasets by using the SMOTE algorithm [10] something that proved to be very beneficial in performing SCA attacks in specific cases. On the other hand, Wang et al. [56] did not follow the same idea of performing balancing as the newly generated data produced by the CGANs, were added so that the augmented dataset followed its primary bell-shaped binomial distribution.

In this chapter, we investigate the effect of the data distribution on the profiling model's performance. In order to perform this investigation, we conduct a series of experiments in which we artificially change the available data distribution and observe the impact on the profiling model's performance.

As we are interested only in investigating if the profiling models' performance depends on the dataset's class distribution, we define a specific experimental setup. We choose three datasets based on the ASCAD dataset as explained in the Background Chapter 2.4.1; namely, the unprotected dataset, the desynchronized dataset, and the masked dataset. The label of each trace in the unprotected dataset is constructed by calculating the XOR operation between the first byte of the key and the plaintext; next, the XOR operations' output value goes through the substitution-box (Sbox) of the AES encryption algorithm. Finally, as we use the HW power model, the output of the Sbox is converted to an HW value.

Regarding the desynchronized dataset, we introduce random desynchronization to each trace of the unprotected dataset, also called jittering. The value of each trace jitter is a random number that lies between 0 and 100. This jittering effect leads to a desynchronized dataset being more prone to SCA attacks as now it is more difficult for the profiling model to detect the features that represent the information leakage. Lastly, to conduct experiments with the masked dataset, we target the 3rd byte of the key, as this byte is masked, meaning that we need to use a more strong profiling model to retrieve the key.

For the first two datasets, we used as profiling model the Random Forest (RF) classifier, while for the third dataset, a more strong profiling model is required. Therefore, we used the Convolution Neural Network proposed by Cagli et al. [8] in their paper as the CNN_best.

By studying the 50,000 traces of the fixed key ASCAD dataset, we observe that the traces' classes follow a binomial distribution. This binomial distribution is observed in every SCA dataset when the Hamming Weight (HW) power model is used. More specifically, we can see the exact class distribution of the ASCAD dataset below in Table 4.1.

To investigate how the different class distributions affect the profiling models' performance, we construct datasets that follow three distributions: the binomial distribution, the U-shaped distribution, and the Balanced (uniform) distribution; which are presented in Figure 4.1 while the precise number of the traces for each class is described in Table 4.2.

| class | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| % | 0.4 | 3 | 11 | 22 | 27 | 22 | 11 | 3 | 0.4 |
| #traces | 200 | 1549 | 5486 | 11068 | 13554 | 10892 | 5494 | 1576 | 181 |

Table 4.1: ASCAD dataset 50,000 traces

| class | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| Binomial (%) | 0.4 | 3 | 11 | 22 | 27 | 22 | 11 | 3 | 0.4 |
| U-shaped(binomial) (%) | 20 | 16 | 12 | 1.5 | 1 | 1.5 | 12 | 16 | 20 |
| Balanced (uniform) (%) | 11.1 | 11.1 | 11.1 | 11.1 | 11.1 | 11.1 | 11.1 | 11.1 | 11.1 |

Table 4.2: Datasets: Binomial vs U-shaped(binomial) vs Balanced class distributions

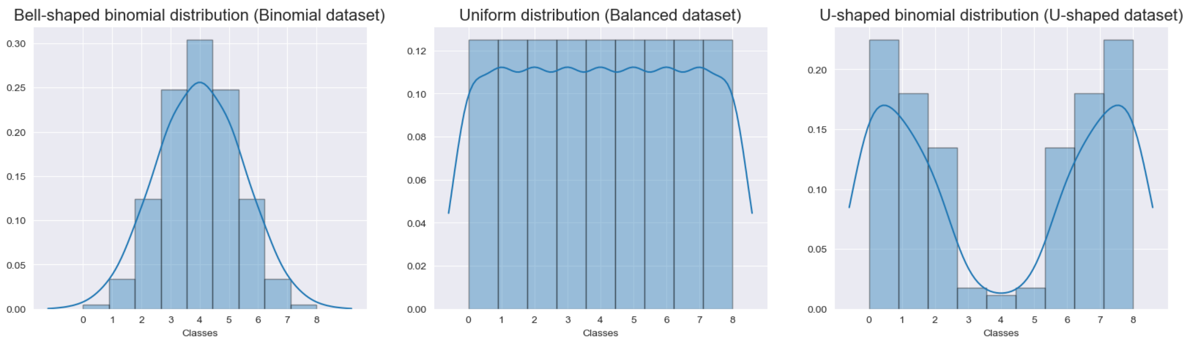


Figure 4.1: Binomial, Balanced and U-shaped dataset

Lastly, essential to be mentioned is that in this thesis, we focus on Side-channel attacks when the HW power model is used, and for this reason, the investigation in this Chapter is necessary. More specifically, this chapter focuses on a problem that the HW power model introduces. In case we choose a different power model like the identity model, the number of classes is 256 or the Least Significant Bit (LSB), the number of classes is limited to 2

This Chapter is divided into two sections; in the first one, we investigate the overall accuracy and the per-class accuracy that can be achieved by using the three above mentioned class distributions, while in the second section, we demonstrate and explain the results of the different GE curves representing the three different class distributions. Each section is divided into two subsections; the first subsection focuses on experimenting on unprotected implementations while the second one focuses on protected implementations.

## 4.1. Overall and Per-class Accuracy

The first step in order to understand the differences between the class distribution is to observe the overall accuracy and, more importantly, the per-class accuracy of the profiling models. When the Hamming Weight (HW) power model is used, the classification accuracy for the majority of classes 3,4,5 is expected to be higher than the accuracy of classes 0,1,2,6,7,8. At the same time, during the attack phase, the traces also follow the same binomial distribution. Therefore, the majority of the attack traces belong mainly in classes with higher accuracy, and generally, the classification accuracy of the attack traces can be characterized as successful even if the traces of minority classes (0,1,2,6,7,8) are misclassified. On the other hand, when we select the same amount of traces from each class in order to build a balanced dataset that follows a uniform distribution, we expect to achieve the same accuracy in every class, but lower accuracy for classes 3,4,5 in comparison with the dataset which follows a binomial distribution.

### 4.1.1. Unprotected ASCAD Dataset

The dataset now is unprotected, meaning that profiling models need very few traces to perform a successful attack; while using datasets with a relatively large number of traces, we do not expect to observe any differences between the class distributions. For this reason, we perform multiple experiments by using six different dataset sizes. The six datasets consist of 27, 36, 54, 63, 99, and 135 traces, respectively. For example, when we use 27 traces, the balanced dataset is a set of 3 traces per class (9 different classes). As a result, we observe that all the dataset sizes are a multiple of 9. For these six dataset sizes, the binomial and the U-shaped (binomial) datasets might not have traces that belong to the minority classes, meaning that the profiling model will not get trained for every class. Thus, we decided to select traces from a large dataset to construct datasets having at least one trace in each class. Experimenting with datasets with so few traces, we observe in every run different GE results. Subsequently, the average result is calculated by performing 100 runs of the same experiment in order to get reliable results.

| Class | Binomial / Balanced / U-shaped | Binomial / Balanced U-shaped | Binomial / Balanced U-shaped |
|---|---|---|---|
| | #27 = 44.5% / 38.9% / 16.8% | #36 = 47.9% / 42.7% / 16.8% | #54 = 53.2% / 48.7% / 17.1% |
| 0 | 12.9% / 72.2% / 84.5% | 10.3% / 73.8% / 86.7% | 6.4% / 82.2% / 91.5% |
| 1 | 4.0% / 44.0% / 48.0% | 2.6% / 47.7% / 56.6% | 9.4% / 51.7% / 58.7% |
| 2 | 24.9% / 39.4% / 37.7% | 29.3% / 40.5% / 42.3% | 33.2% / 48.8% / 52.7% |
| 3 | 53.2% / 38.1% / 5.3% | 53.8% / 41.6% / 3.4% | 59.6% / 47.2% / 1.6% |
| 4 | 55.8% / 37.1% / 7.4% | 62.6% / 41.8% / 5.9% | 68.0% / 48.4% / 3.0% |
| 5 | 53.4% / 39.4% / 6.4% | 55.6% / 43.0% / 4.2% | 60.7% / 47.5% / 2.0% |
| 6 | 24.9% / 39.9% / 41.0% | 30.3% / 44.0% / 43.7% | 36.4% / 50.1% / 50.2% |
| 7 | 4.5% / 40.3% / 47.4% | 2.9% / 45.0% / 53.9% | 8.9% / 52.4% / 57.6% |
| 8 | 11.1% / 66.8% / 77.5% | 8.7% / 73.0% / 85.4% | 6.7% / 78.7% / 94.2% |
| | #63 = 55.3% / 50.6% / 19.1% | #99 = 60.9% / 55.9% / 19.9% | #135 = 64.4% / 59.5% / 21.0% |
| 0 | 4.0% / 85.5% / 93.7% | 2.7% / 90.1% / 96.5% | 1.3% / 91.5% / 98.1% |
| 1 | 7.3% / 53.3% / 56.5% | 9.1% / 59.8% / 66.9% | 10.9% / 64.4% / 72.7% |
| 2 | 34.3% / 48.5% / 64.8% | 44.2% / 57.4% / 68.0% | 47.7% / 59.3% / 68.6% |
| 3 | 64.1% / 49.8% / 0.8% | 67.9% / 54.7% / 0.2% | 71.3% / 58.7% / 1.3% |
| 4 | 67.0% / 49.2% / 1.7% | 73.0% / 54.4% / 0.9% | 75.9% / 58.6% / 0.8% |
| 5 | 66.4% / 51.4% / 1.3% | 71.1% / 56.1% / 0.4% | 74.8% / 58.4% / 1.6% |
| 6 | 37.5% / 52.2% / 62.4% | 45.2% / 55.6% / 66.1% | 50.8% / 60.2% / 68.7% |
| 7 | 7.2% / 53.9% / 55.9% | 11.1% / 59.9% / 65.5% | 12.3% / 64.0% / 69.6% |
| 8 | 5.8% / 82.1% / 95.0% | 3.2% / 89.5% / 96.4% | 1.7% / 92.8% / 97.5% |

Table 4.3: Accuracy(%) of the profiling model (RF) for the Unprotected ASCAD Binomial vs Balanced vs U-shaped class distribution

In Table 4.3, the overall accuracy and per-class accuracy are demonstrated in percentages. We can observe that when the dataset classes follow a binomial distribution, the overall accuracy is slightly higher than the profiling models trained in balanced datasets and significantly higher than profiling models trained in the U-shaped datasets. Concerning the per-class accuracy, when the dataset is binomial, classes 3,4,5 achieve a higher accuracy score than classes 0,1,2,6,7,8. When the dataset follows a balanced (uniform) distribution, classes 0,1,2,6,7,8 achieve higher accuracy than classes 3,4,5 and not equal as expected. A likely interpretation of the above is that traces of classes 0,1,2,6,7,8 consist of more representative features, able to express the information leakage more efficiently. We observe the same trend regarding the U-shaped dataset as the majority classes 0,1,2,6,7,8 achieve the highest accuracy among the classes.

### 4.1.2. Protected ASCAD Datasets

Next, we conduct the same experiment for the desynchronized and the masked dataset. For the desynchronized dataset, we use different dataset sizes than before as we need more traces to perform an attack. More specifically, we use the following sizes: 450, 540, 621, 720, 801, 900. For the masked dataset, we use only one size of 1494 traces, and we perform a comparison only between the binomial and the balanced distribution.

#### Desynchronized Dataset

The RF is used again as a profiling model in order to make comparisons between the six sizes of the desynchronized dataset. In Table 4.4 we can observe that the overall and the per-class accuracy of each size is lower compared to the unprotected dataset, but both overall and the per-class accuracy follow the same trend. The binomial dataset's overall accuracy is again the highest for every different dataset size. When using the U-shaped dataset, a worse performance was observed. About the per-class accuracy, for the majority classes, the binomial and the U-shaped distribution, we observe higher accuracy than in the minority classes. Regarding the balanced dataset, classes 0,1,2,6,7,8 achieve higher classification accuracy, meaning that the traces belonging to these classes can be classified more easily than traces belonging in classes 3,4,5.

| Class | Binomial / Balanced / U-shaped | Binomial / Balanced U-shaped | Binomial / Balanced U-shaped |
|---|---|---|---|
| | #450 = 29.9% / 22.4% / 9.5% | #540 = 31.1% / 23.9% / 9.9% | #621 = 32.7% / 25.4% / 10.5% |
| 0 | 0.7% / 38.8% / 59.0% | 0.2% / 41.9% / 62.4% | 0.5% / 45.7% / 65.0% |
| 1 | 3.7% / 27.8% / 34.2% | 2.7% / 29.2% / 34.1% | 3.5% / 30.5% / 35.8% |
| 2 | 14.5% / 22.5% / 25.1% | 15.2% / 24.8% / 26.3% | 14.9% / 25.4% / 28.5% |
| 3 | 33.4% / 21.7% / 2.3% | 35.0% / 22.4% / 2.3% | 36.8% / 24.8% / 2.4% |
| 4 | 41.8% / 21.2% / 1.2% | 41.8% / 23.5% / 1.5% | 44.1% / 24.5% / 1.5% |
| 5 | 33.6% / 22.2% / 2.2% | 36.4% / 22.9% / 2.5% | 38.1% / 24.5% / 2.2% |
| 6 | 17.1% / 23.0% / 26.1% | 18.6% / 24.7% / 27.4% | 20.4% / 26.4% / 29.5% |
| 7 | 4.5% / 27.8% / 35.9% | 4.7% / 29.7% / 36.7% | 5.3% / 29.5% / 37.6% |
| 8 | 0.9% / 31.8% / 50.5% | 0.2% / 34.2% / 51.6% | 1.1% / 35.2% / 51.9% |
| | #720 = 33.9% / 26.4% / 10.7% | #801 = 34.9% / 27.4% / 11.2% | #900 = 36.2% / 28.5% / 11.6% |
| 0 | 0.3% / 48.6% / 30.5% | 0.3% / 49.2% / 70.8% | 0.5% / 50.8% / 71.9% |
| 1 | 3.0% / 30.3% / 30.5% | 3.1% / 32.0% / 35.7% | 2.5% / 32.62% / 37.7% |
| 2 | 15.7% / 26.3% / 30.5% | 16.4% / 28.0% / 32.4% | 16.1% / 27.9% / 32.4% |
| 3 | 39.1% / 25.6% / 30.5% | 39.1% / 26.6% / 2.4% | 41.4% / 27.6% / 2.9% |
| 4 | 45.5% / 25.6% / 30.5% | 47.6% / 26.5% / 1.7% | 48.1% / 27.5% / 1.9% |
| 5 | 39.4% / 25.9% / 30.5% | 39.9% / 27.1% / 2.1% | 42.1% / 28.1% / 2.6% |
| 6 | 20.8% / 26.9% / 30.5% | 22.0% / 27.5% / 30.7% | 23.9% / 29.8% / 31.3% |
| 7 | 5.7% / 32.0% / 30.5% | 6.7% / 33.3% / 40.3% | 6.1% / 32.8% / 41.2% |
| 8 | 1.0% / 38.2% / 30.5% | 1.0% / 42.2% / 51.7% | 1.0% / 40.5% / 55.1% |

Table 4.4: Desynchronized ASCAD Binomial vs Balanced vs U-shaped class distribution

**Masked Dataset**
For this dataset, we use a Convolution Neural network as a profiling model as the masked dataset is more complex than the two previous datasets, namely, the unprotected and the desynchronized datasets. Furthermore, we experimented only in one size and only between the binomial and the balanced distribution as the ASCAD database provides only 50,000 traces; consequently, the minority class consists of 166 traces. As a result of the above, the largest balanced dataset consists of $166 * 9 = 1494$ traces. On the other hand, it appears impossible to construct a dataset that follows a U-shaped binomial distribution large enough to train a profiling model on this dataset. Therefore we make only comparisons between the binomial and the balanced distribution.

| class | overall | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|---|
| Binomial | 22.1% | 0.0% | 4.1% | 10.9% | 26.1% | 30.4% | 24.8% | 12.0% | 3.0% | 4.7% |
| Balanced | 11.3% | 25.6% | 16.3% | 12.5% | 9.2% | 9.5% | 11.4% | 13.3% | 18.4% | 30.2% |

Table 4.5: Overall Accuracy and per-class Accuracy on Binomial vs Balanced class distributions on Masked dataset

In Table 4.5, we observe a similar trend with the two previous datasets as the overall accuracy of the binomial dataset is higher than the accuracy of the balanced dataset. The per-class accuracy is higher in the majority classes than in minority classes in the binomial dataset. Concerning the balanced dataset, we observe again higher per-class accuracy in classes 0,1,2,6,7,8 in comparison to classes 4,5,6. The only difference here is that the overall accuracy and per-class accuracy are significantly lower, and in this case, the performance of the profiling model in classes 0,1,2,6,7,8 plays a significant role, as can be seen in the next section.

## 4.2. Guessing Entropy Results Between Different Class Distributions

Investigating the profiling models' overall and per-class accuracy was necessary to understand how the different class distributions affect the profiling models' classification performance. However, in order to evaluate the actual performance of an attack, we conduct the same experiments by calculating the Guessing Entropy (GE), a metric that expresses how many traces are needed to retrieve the key.

It is essential to understand that the key-rank algorithm tries to retrieve the key by giving a set of correct candidate keys that correspond to the HW value (label) predicted by the profiling model. During the calculation of the key-rank, all the possible keys are tested as it is explained in the Background

Chapter 2.1.3 and the correct key is the key in which the profiling model provides the highest probability value. However, we have to consider that it might not be a single key but multiple candidate keys that correspond to the profiling model's highest probability value. Of course, we are not interested in guessing the correct key out of a very large key candidate set. Thus, the maximum likelihood estimation is used by taking into account more traces in order to limit the number of the candidate keys to a small number of more representative ones and eventually to the one correct key.

As we explained before in this Chapter, the HW values 3,4,5 are the most common, meaning that if we manage to train a profiling model that predict accurately the labels of traces that belong in classes 3,4,5, we will eventually retrieve the key. On the other hand, we have to consider that the number of possible key candidates for a random plaintext corresponds to an HW value 3,4,5 is larger compared to the number of possible key candidates that correspond to HW values 0,1,2,6,7,8. In other words, for the majority of random plaintext, the traces belong in 3 or 4 or 5 classes, but we need to make more guesses as the number of the candidate keys provided by the GE algorithm is relatively large. On the other hand, if the random plaintext combined with the correct key result to an HW value of 0,1,2,6,7,8, we might have profiling models with lower per-class accuracy in classes 0,1,2,6,7,8, but if the prediction will be accurate, then the number of the candidate keys is relatively small meaning that we can retrieve the key with fewer guesses. We can say that high accuracy in classes 3,4,5 leads to good classification results in general, but traces of 3,4,5 class provide less information to the maximum likelihood estimation compared to the information provided by the traces of the classes 0,1,2,6,7,8. In conclusion, we can say that it is difficult to make any hypothesis about the optimum class distribution that could improve the performance of the profiling models in SCA.

The GE experiments of this section follow the same setup used in the accuracy investigation Section 4.1; we use the unprotected version of ASCAD and the two protected datasets. We first compare the GE curves for the binomial datasets, the balanced datasets, and the U-shaped binomial datasets. Lastly, we perform one more extensive GE experiment in order to compare the ten different class distributions demonstrated in Table 4.8.

### 4.2.1. Unprotected ASCAD Dataset

The GE results can be observed in Figure 4.2. To make the GE curves more transparent, we demonstrate in a comparison Table 4.6 the GE results of the Binomial, the U-shaped(binomial), and the Balanced(uniform) datasets. More specifically, we demonstrate the required number of traces when we make 10 guesses to break the key.

| Size | 27 | 63 | 99 | 135 | 171 | 198 |
|---|---|---|---|---|---|---|
| Binomial | 12 | 10 | 4 | 4 | 3 | 2 |
| U-shaped(binomial) | >50 | 48 | 42 | 38 | 38 | 20 |
| Balanced | 12 | 6 | 4 | 4 | 3 | 2 |

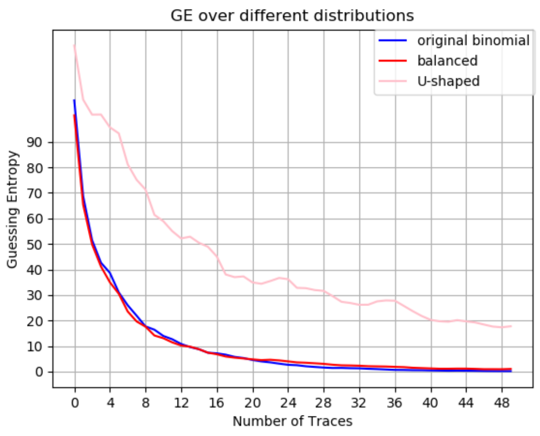
Table 4.6: Required number of traces in order to achieve GE=10: Unprotected dataset Binomial vs U-shaped (binomial) vs Balanced (uniform) class distribution

We notice that the balanced distribution is preferred in small dataset sizes instead of the binomial distribution, but the differences between GE results are minimal. Furthermore, we can observe that these differences tend to be eliminated as far as the dataset is getting larger, meaning that the two distributions perform similarly.

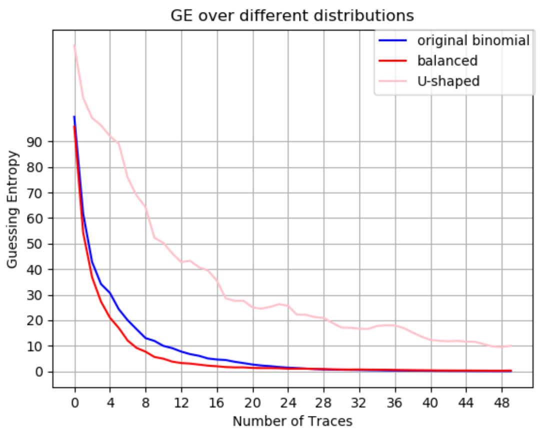
On the other hand, when we observe the GE curves of the U-shaped distribution, we can see that the GE metric captures the significant differences in the overall classification accuracy as the GE curve of the U-shaped dataset decreases significantly slower than the GE curve of the binomial and balanced dataset. Therefore we can say that the accuracy is not a very misleading metric when dealing with unprotected datasets. Lastly, we can claim that the dataset's class distribution plays a significant role in the profiling models' performance by observing all the above results.

### 4.2.2. Protected ASCAD Datasets

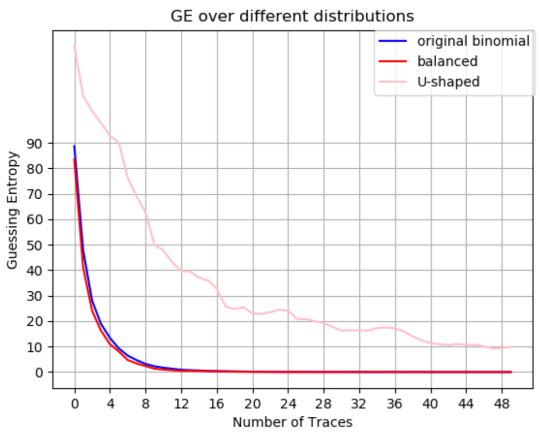
We conduct the same experiments with two protected datasets. The first dataset is the desynchronized dataset, and the second is the masked dataset.



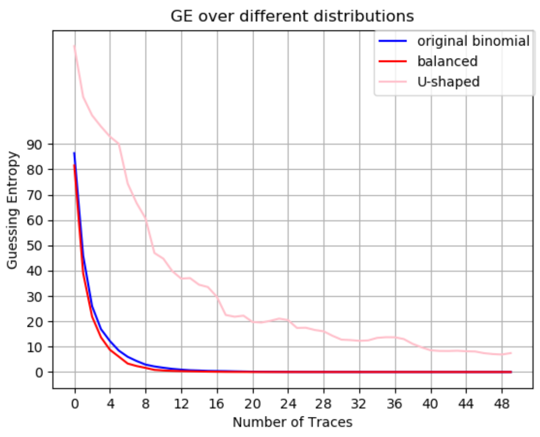
(a) 27 traces



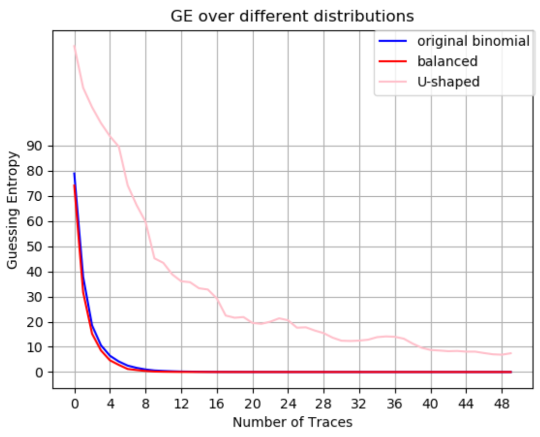
(b) 36 traces



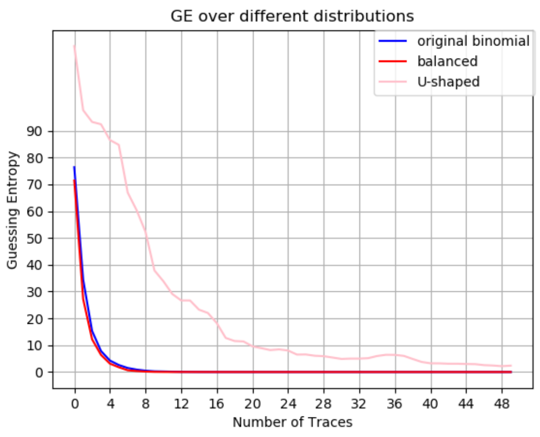
(c) 54 traces



(d) 63 traces



(e) 99 traces



(f) 135 traces

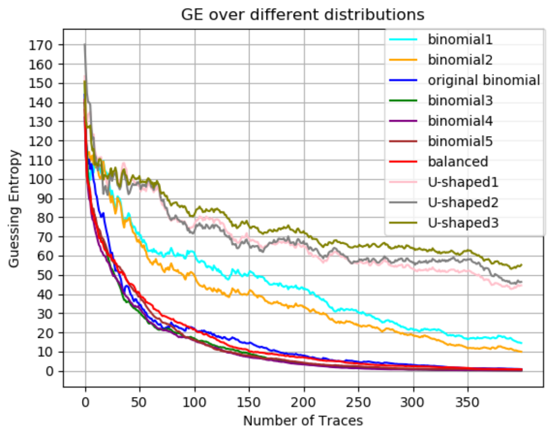
Figure 4.2: Binomial vs Balanced vs U-shaped distribution on unprotected ASCAD trace-set

### Desynchronized Dataset

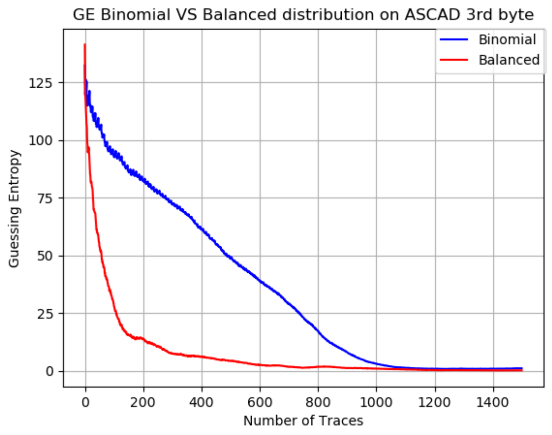
The GE results now can be observed in Figure 4.4. To examine the GE results more precisely, we demonstrate in the below Table 4.7 that compares the required number of traces that achieves GE equal to 10 when comparing the three different class distributions. We notice again that the GE curves of the balanced, binomial, and U-shaped binomial dataset follow the same behavior as with the unprotected

| Size | 450 | 540 | 621 | 720 | 801 | 900 |
|---|---|---|---|---|---|---|
| Binomial | 200 | 150 | 120 | 60 | 50 | 45 |
| U-shaped(binomial) | >500 | >500 | >500 | >500 | >500 | >500 |
| Balanced | 160 | 120 | 90 | 60 | 50 | 45 |

Table 4.7: Required number of traces in order to achieve GE=10: Protected dataset Binomial vs U-shaped vs Balanced class distribution



(a) 450 traces



(b) 1494 traces

Figure 4.3: (a): a comparison of 10 different binomial distributions (described in Table 4.8) for the desynchronized dataset, (b): a comparison between a binomial and a balanced (uniform) distribution for the masked ASCAD dataset

dataset, while the only difference is that we need more traces for the GE curve to converge to zero. Furthermore, a more extensive experiment is conducted as shown in Figure 4.3a where we compare the RF classifier GE performance under ten different class distributions that can be seen in Table 4.8.

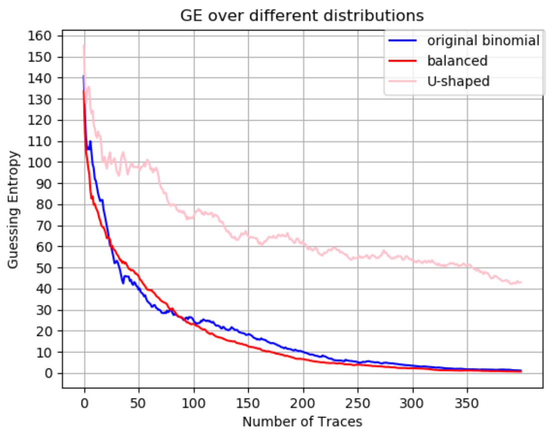
| class | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| Binomial 1 (%) | 0.4 | 0.5 | 3 | 27 | 38 | 27 | 3 | 0.5 | 0.4 |
| Binomial 2(%) | 0.4 | 0.5 | 8 | 25 | 33 | 25 | 8 | 0.5 | 0.4 |
| Original Binomial (%) | 0.4 | 3 | 11 | 22 | 27 | 22 | 11 | 3 | 0.4 |
| Binomial 3 (%) | 4 | 7 | 10 | 18 | 22 | 18 | 10 | 7 | 4 |
| Binomial 4 (%) | 6.5 | 9.5 | 11.5 | 14 | 17 | 14 | 11.5 | 9.5 | 6.5 |
| Binomial 5 (%) | 8 | 10 | 11.5 | 13 | 15 | 13 | 11.5 | 10 | 8 |
| Balanced (%) | 11.1 | 11.1 | 11.1 | 11.1 | 11.1 | 11.1 | 11.1 | 11.1 | 11.1 |
| U-shaped 1 (%) | 20 | 16 | 12 | 1.5 | 1 | 1.5 | 12 | 16 | 20 |
| U-shaped 2 (%) | 22 | 18 | 9.2 | 0.5 | 0.5 | 0.5 | 9.2 | 18 | 22 |
| U-shaped 3 (%) | 24 | 20 | 5.2 | 1.5 | 1 | 1.5 | 5.2 | 20 | 24 |

Table 4.8: 10 different class distributions

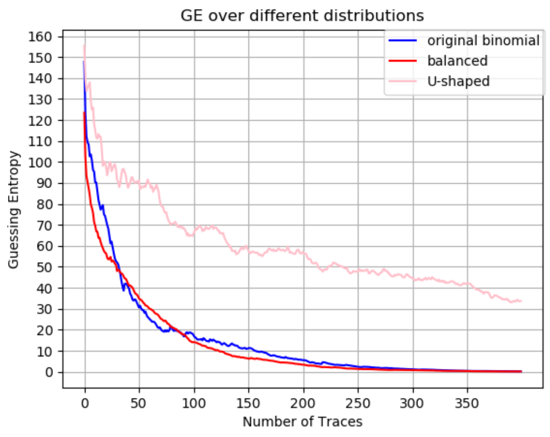
### Masked Dataset

Lastly, we conduct one more experiment with a masked dataset, and now we use a Convolutional Neural Network as we need a more powerful profiling model in order to perform an attack. We do this experiment only for the maximum balanced dataset that can be constructed out of the masked dataset. It consists of 166 traces per class, meaning that we have in total $166 * 9 = 1494$ traces. The balanced dataset is only compared with the binomial dataset and not with the U-shaped binomial as we do not have the required number of traces to perform an attack with a dataset that follows this type of class distribution.

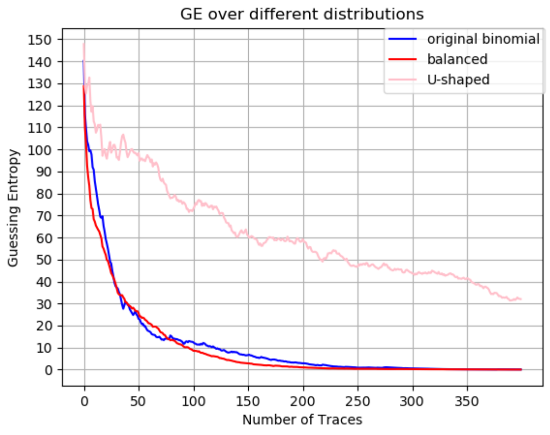
In this dataset, which can be characterized as the most prone to SCA attacks but also the most realistic out of these three (unprotected, desynchronized, and masked dataset), we can observe in Figure 4.3b a significant difference in GE performance between the binomial and the balanced dataset.



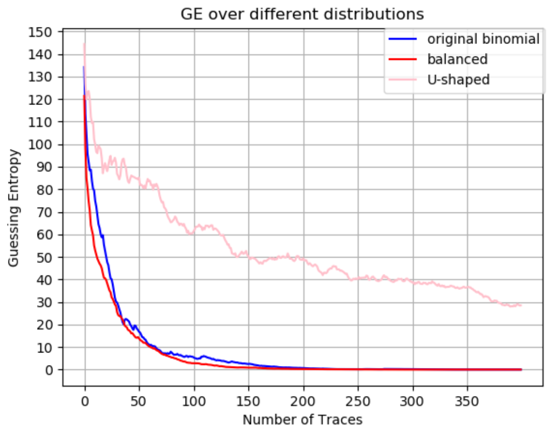
(a) 450 traces



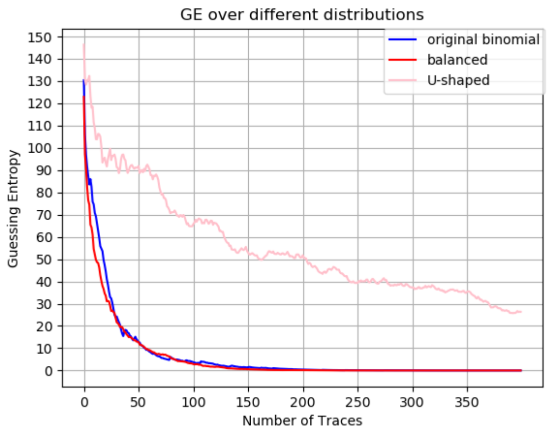
(b) 540 traces



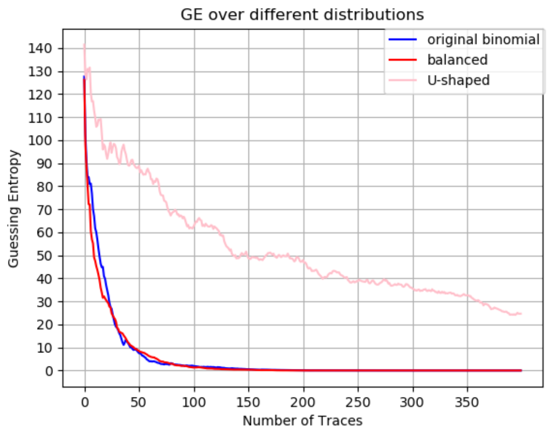
(c) 621 traces



(d) 720 traces



(e) 801 traces



(f) 900 traces

Figure 4.4: Binomial vs Balanced vs U-shaped distribution on protected ASCAD trace-set

The balanced dataset's GE curve decreases towards zero significantly faster than the GE curve of the binomial dataset. This can be explained by looking into the accuracy Table 4.5 of the previous section. In the binomial dataset's minority classes, the per-class accuracy is very low, which is explicable as there are not many traces for CNN to get trained. The problem lies in the fact that the accuracy is also relatively low for most classes, meaning that even with an unlimited number of traces in classes 3,4,5, there will always be a threshold number that cannot be surpassed. For this reason, balanced datasets could be favored under this setup as the optimum solution would be to push all classes in order to approach as much as possible this per-class accuracy threshold number rather than trying to overcome this threshold number in classes 3,4,5.

## 4.3. Conclusions

Based on the results described in this Chapter, we conclude that the ideal class distribution that a dataset should follow is the balanced (uniform) distribution as under the most realistic scenario of the masked dataset, the profiling model's performance seems to be the optimum. On the other hand, for the rest of the datasets (unprotected and desynchronized datasets), we can also suggest constructing datasets that follow the binomial distribution. More specifically,

- The task of choosing the class distribution of the dataset is essential as different class distribution types like the U-shaped binomial distribution can lead the profiling model to poor classification and GE performance.
- The optimal class distribution of the dataset depends on the nature of the dataset. More specifically, it is pointless to perform balancing on unprotected datasets or to datasets with mild countermeasures as the bell-shaped binomial datasets lead to similar GE performance.
- In more realistic scenarios where serious countermeasures are implemented, balancing is the best solution as the overfitting phenomenon is extreme, meaning that it is important to increase the classification accuracy of classes 0,1,2,6,7,8.

By doing this investigation, we studied how the different class distributions of the dataset affect profiling models' performance. Consequently, we can make the first steps towards understanding what type of class distribution the GANs' generated traces should follow when the HW model is used.

# 5 Methodology for Generative Adversarial Networks in Side-channel Analysis

This chapter describes the methodology for generating synthetic traces. More specifically, we propose a pipeline to build datasets of pure synthetic traces. We use synthetic traces to build augmented datasets consisting of a combination of real and synthetic traces generated by the GANs. Our pipeline was first designed as a set of performance evaluation steps that helped us tune the network and find more potent architectures, which are addressed in Chapter 6. Furthermore, this pipeline is used to maximize shallow architectures' performance to save time and computational power rather than performing a Neural Architecture Search. This pipeline consists of many components, shown in Figure 5.1. The components of our pipeline are 5 in number and are explained briefly in the following list:

1. Window Selection: N windows of m features each are selected to train the GANs.
2. Architecture Selection: Specific architectures are built in order to improve the performance of the GANs by looking into the complexity of the dataset.
3. GANs Training Phase: 9 different GANs, one for each class, are train evaluated on several evaluation mechanisms in order to perform a hyperparameter optimization.
4. Snapshot Selection: Two Snapshot Selection mechanisms are used to improve the performance of the GANs and simultaneously evaluate the synthetic traces produced by GANs.
5. Feature Concatenation: A concatenation of the seven datasets (100 features each) in order to construct a more robust synthetic dataset.

Therefore this chapter is divided into five sections; in the first one, the window selection procedure is described in detail, where we select the Points of Interest to train the GANs and the profiling models. In the second section, the training procedure of GANs and the profiling model is described. Also, in the same section, we demonstrate the basic architecture of the GANs. In the next section, the two different snapshot selection mechanisms are described, where we select the most potent snapshots of the GANs. The fourth section explains the dataset feature concatenation and why this step is essential, especially for more complex datasets. In the last section, the steps mentioned above are combined into a single pipeline. This last section describes how this pipeline can work well in unprotected datasets, its limitations, and how its components should be altered to deal with more complex datasets.

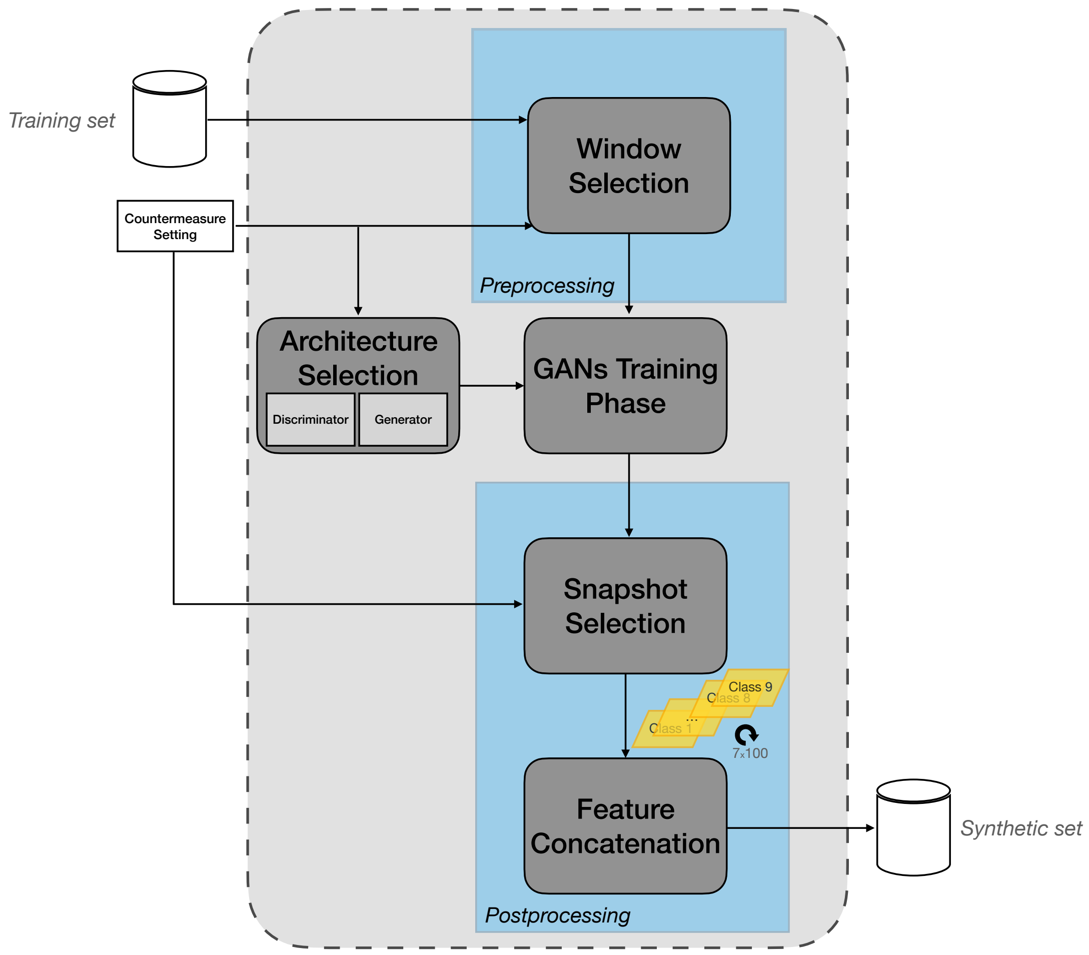


Figure 5.1: Synthetic Dataset Construction Pipeline

## 5.1. Windows Selection

In the SCA field, in order to perform an attack through large datasets, Points of Interest (PoIs) should be first detected. These PoIs can be detected using methods like the Simple Power Analysis [32] accompanied with a Test Vector Leakage Assessment (TVLA) [53] which experienced Security Analysts can perform. On the other hand, Machine Learning profiling models like CNNs can extract information through a large feature space [5] even in the SCA field. Nevertheless, in this thesis project and in many scientific papers [5], [61] this PoIs detection is applied before the profiling models' training. More specifically, a specific window is selected where the time-samples (features) are highly correlated with the targeted device's power consumption. This way, models with lower complexity can be used to perform an SCA attack. We can easily choose this window for the unprotected dataset by calculating Pearson's correlation coefficient, indicating the leakage areas. In Figure 5.2 we compute the Pearson's Correlation Coefficients of the ASCAD dataset[1] for byte 0 (unprotected) when the HW power model is used. For a dataset with countermeasures, more potent PoIs detection methods should be used, but we leave this for future work. In this project, we use only 100 features (time-samples) to train the GANs; further details about the reasons we choose 100 features as PoIs can be found in the Discussion section 5.6 of this Chapter.

We can see three large peaks in Figure 5.2; it is noticeable that the last peak where it consists of 2000 features, is the densest of the three. We choose the densest peak as we expect to find the most considerable information leak in this peak. More specifically, in our case, we chose 700 features around the last peak, as shown in Figure 5.3a. Next, to train the GANs into a 100 features dataset, we split these 700 features into 7 windows of 100 features each. For example, a Correlation Power

[1] https://github.com/ANSSI-FR/ASCAD

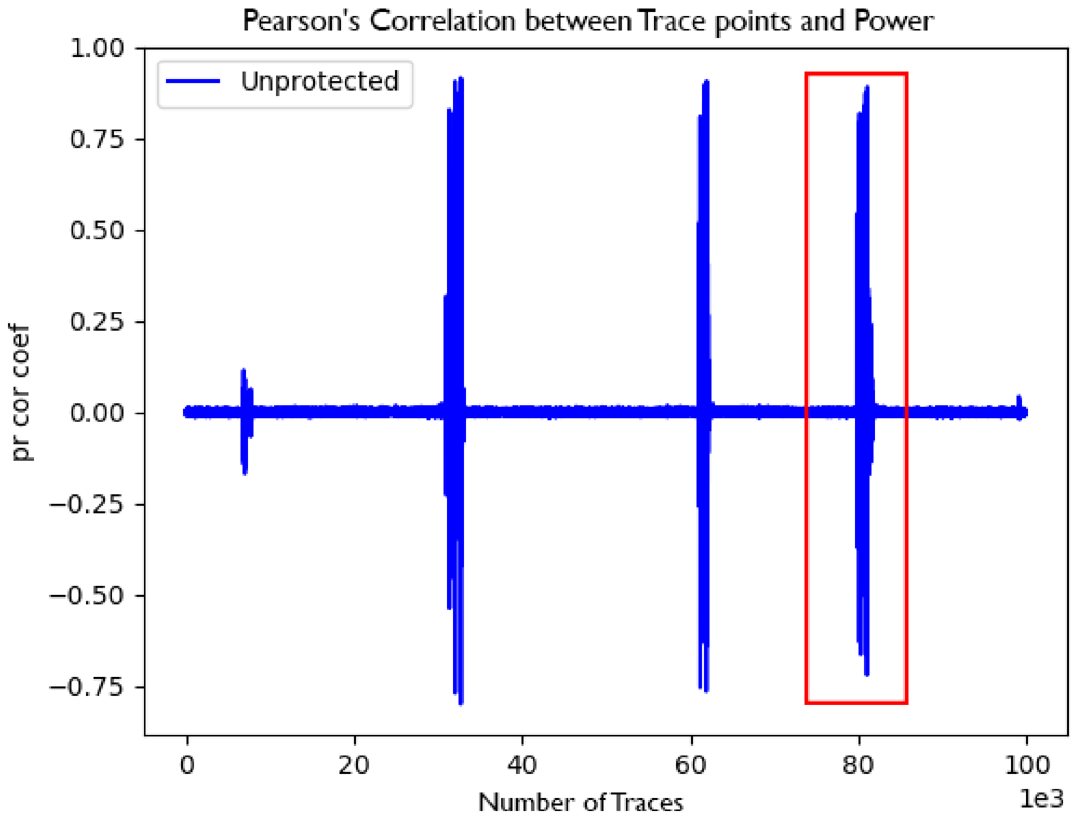
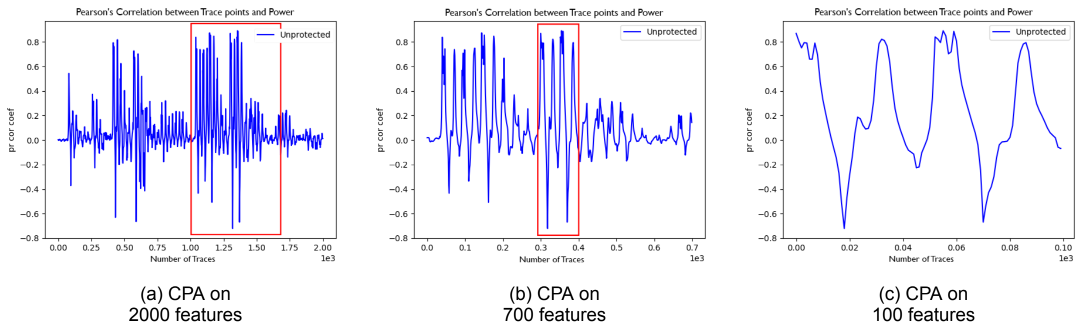


Figure 5.2: CPA on ASCAD byte 0 (HW power model) 100,000 features

Analysis (CPA) of a 100 feature window between the 300 and 400 features of Figure 5.3b can be seen in Figure 5.3c.



(a) CPA on 2000 features
(b) CPA on 700 features
(c) CPA on 100 features

Figure 5.3: Unprotected dataset

Lastly, we apply the same Window Selection technique for a desynchronized dataset of jitter 10. We can observe in Figures 5.4, 5.5a, 5.5b and 5.5c the Pearson's Correlation between trace-points and Power of 100000, 2000, 700 and 100 features window when this desynchronization of jitter 15 is applied in the dataset. Moreover, we can see that Pearson's correlation value reduced a lot compared to the Figures of the unprotected dataset mentioned above, meaning that using a CPA for more datasets with more complex countermeasures like larger jittering or masking to select the desired window will be a more difficult task.

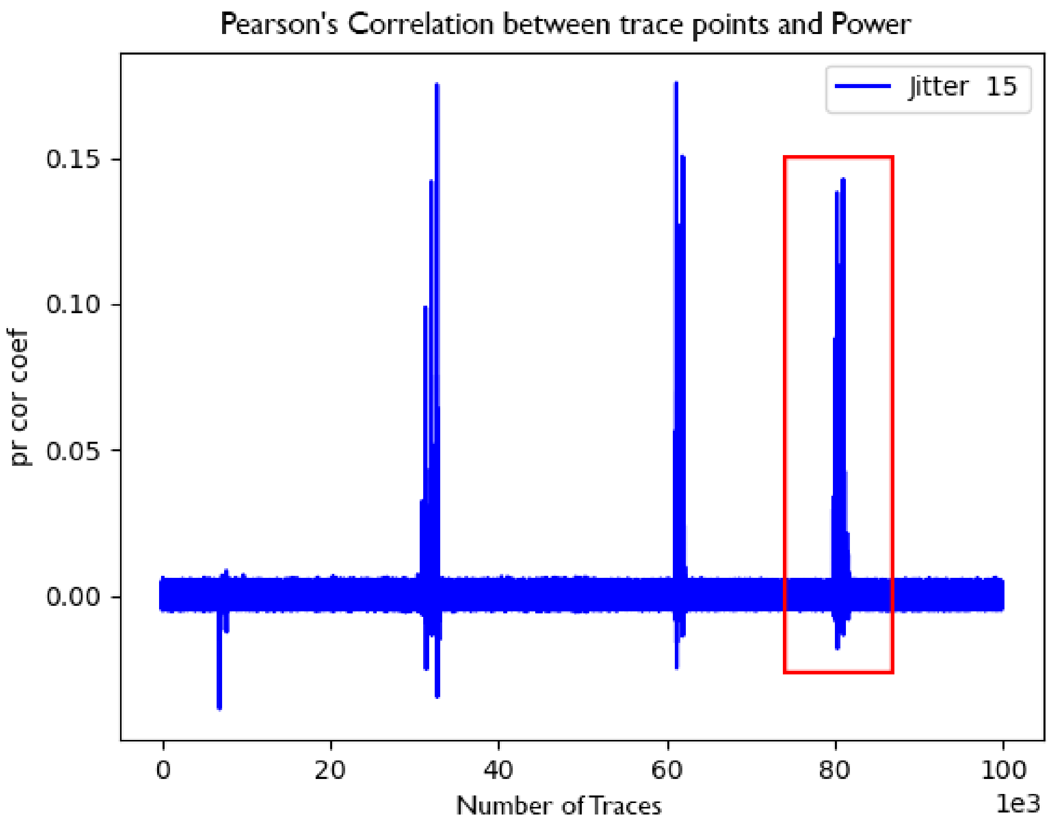
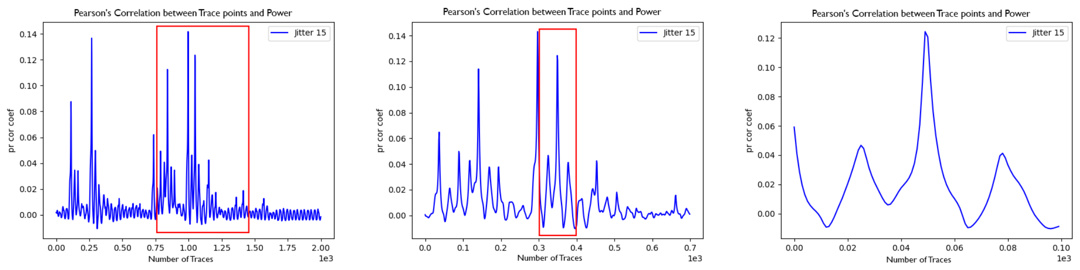


Figure 5.4: CPA on ASCAD byte 0 desynchronized (jitter 50) (HW power model)



(a) CPA on 2000 features

(b) CPA on 700 features

(c) CPA on 100 features

Figure 5.5: Desynchronized dataset of jitter 10

## 5.2. Generative Adversarial Neural Network Architecture

In this section, we define the base of Generator's and Discriminator's architecture as it can be seen in Figure 5.6. We expose the number of kernels (k), channels (n), and layers (l) as parameters to be tuned.

### 5.2.1. Design Considerations

It is common knowledge that performing hyperparameter optimization for a deep learning model such as a Convolutional Neural Network is challenging; the same is expected for GANs. GANs have plenty of hyperparameters and two different deep learning models that compete with each other, meaning that every time we want to change the Generator's architecture, the Discriminator's architecture should be changed as well. This section proposes two architectures, one for each dataset (unprotected and

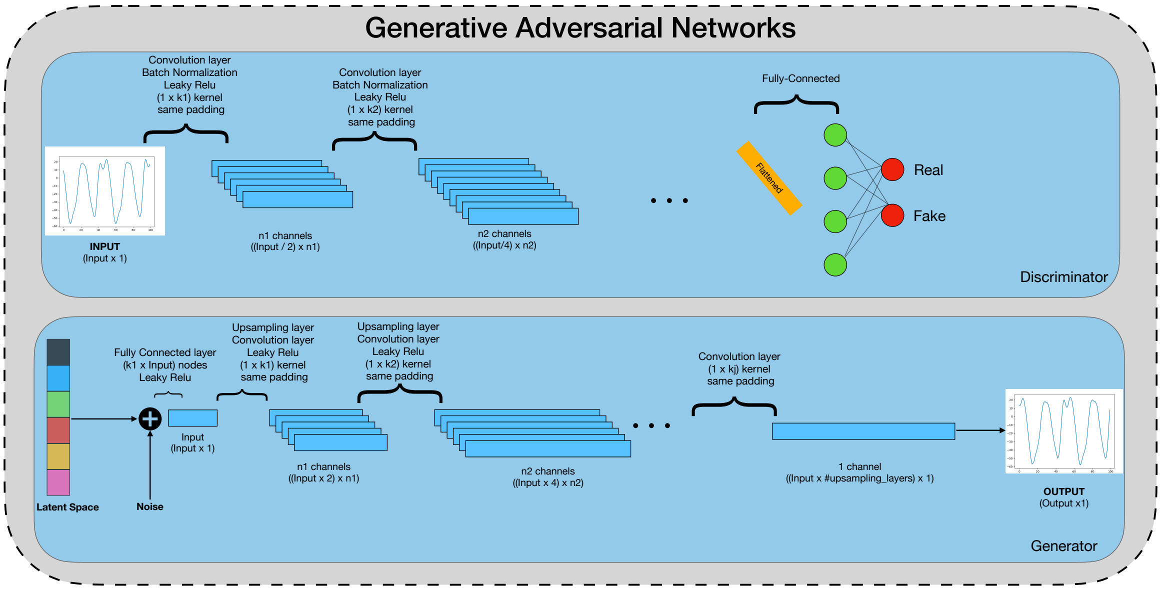


Figure 5.6: Generative Adversarial Networks

desynchronized), based on the idea that the Discriminator must be equally powerful with the Generator, as Tero Karras et al. [30] point out in their paper. Figure 5.6 shows the Discriminator and the Generator architecture's parameterized base. More specifically, the Discriminator's base architecture consists of l_d convolutional layers accompanied with a Batch Normalization after every convolutional layer except the first one. Furthermore, the activation function that is used is the Leaky Relu and has been added after every Convolutional Layer. For the Generator, the input layer is a Fully Connected layer, and l_g combinations of an Up-sampling Layer and a Convolutional Layer. Every layer of the Generator is accompanied by a Batch Normalization layer and a Leaky Relu. The architecture inspired this network's idea that Alec Radford et al. [49] proposed where they introduce the Convolutional Layers in GANs. Furthermore, they use the Batch Normalization Layer and the Leaky Relu, something common in GANs architectures as it is known that these two layers have the ability to reduce the variance of the Generator's and the Discriminator's loss, something which is very crucial on training GANs which usually show a very unstable behavior [30].

The reason for choosing convolutional layers in both Generator and the Discriminator for generating power traces comes from the same reason why Eleonora Cagli et al. [8] choose a CNN as a profiling model. The theoretical assumption is that convolutional layers that were introduced first in CNNs have the ability to learn the shift-invariant features. Consequently, we hope that convolutional layers will manage to learn interesting patterns even when datasets with countermeasures like desynchronization or masking are used.

### 5.2.2. Architecture Tuning

In practice choosing an architecture is always based on performing multiple experiments, mostly when dealing with different datasets. This is common in computer vision while completely different architectures or altered architectures are used on different datasets, and we can assume the same with GANs on the SCA dataset. The best architecture selection in Deep Learning models can be achieved by using a Neural Architecture Search [63], a solution that is not always preferred as a large amount of computational power must be used. On the other hand, the Neural Architecture Search has been applied successfully on GANs, where Royson Lee et al. [36] apply Neural Architecture Search on the Discriminator while Xinyu Gong et al. [18] perform it on the Generator. In this project, we experiment with different architectures only by using the theoretical assumptions mentioned before, and we propose two architectures explained in detail in Chapter 6.

We will see later in Chapter 6 that choosing the GANs right architecture is a difficult task but crucial in every Deep Learning model. For example, we can say that the larger the number (l), the deeper is the architecture, but this does not lead necessarily to a more robust architecture as the number of the channels (n) and the size of the kernels play a significant role in tuning an architecture. More specifically, channels (filters) mean more and different types of features can be extracted from the traces, while the size of the kernels depends on what hypothesis we made about the nature of the essential features of the traces. The right combination of these parameters will lead to high-performance GANs.

## 5.3. GANs Training Phase

In this section, we address how many GANs are needed for the pipeline in order to construct synthetic datasets, the chosen hyperparameters, and how we tune them.

### 5.3.1. Specialization and Models

We choose to train GANs only on 100 features as we observed that using more features, the GANs suffer from mode collapse and convergence failure. We choose to build a dataset that consists of 700 features meaning that we need to train 7 GANs on 100 features, and the 100-feature synthetic for each of the 7 GANs are concatenated into a 700 feature dataset as it is explained in Section 5.5. Furthermore, each 100-feature GAN consists of 9 different GANs, one for each class. More details about the reasoning of this design are discussed in Section 5.6.

### 5.3.2. Hyperparameters

We conducted a set of experiment with different learning rates and batch sizes by using the grid search technique, and we concluded to set the batch size as a proportion of the size of the dataset as the performance of each class's GAN depend on the size of the dataset. For example, when we use the 50,000 traces of the ASCAD dataset, the minority classes 0 and 8 have only around 100 traces, meaning we cannot use large batch sizes, e.g., 50 or 100. The batch size we used was 5% of the size of the dataset. The chosen learning rate was 0.0002. The optimizer used is the Adam optimizer with the following parameters: beta_1: 0.5 (The exponential decay rate for the first moment estimate), beta_2: 0.999 (The exponential decay rate for the second-moment estimates).

The hyperparameter tuning was based on examining the variance of the Machine Learning metrics and whether these metrics were close to the rule of thumb metrics mentioned in Section 5.4.

On the other hand, choosing and tuning the right architecture and selecting the optimal hyperparameters are not the only steps to generate plausible synthetic traces as one effective procedure that should be taken into account is to select when to stop the training of the GANs and choose a trained model. This is something similar to the early stopping in CNNs, and MLPs [46] as training the neural networks for a large number of epochs, the model will learn only patterns that belong to the training dataset. In Section 5.4, we introduce the Snapshot Selection Mechanism.

## 5.4. GANs Snapshot Selection Mechanism

GANs sometimes show a cyclic behavior in a variety of applications [43], [33]; the same applies for GANs when applied in SCA. In Figure 5.9 we can see an example where the GANs of class 3 periodically converge and diverge from the desired accuracy, meaning that for some epochs, the model perform better while for others perform very poor and we need to define a method for identifying when that happens. For this reason, we proposed two mechanisms, one that is based on the Machine Learning metrics and a second that is based on a pre-trained profiling model.

As a rule of thumb, it is expected[2] to get high-quality synthetic data when the Discriminator's accuracy on real and fake data is somewhere between 0.6 and 0.8, while it is acceptable that Discriminator's loss on real data and fake data to fluctuate between 0.5 and 0.8. On the other hand, we expect to get a higher loss for the Generator, around 1.0. Of course, we cannot claim that this is the expected behavior case in every application of GANs, but we can follow this rule of thumb to select the epoch where the training should be stopped. More specifically, we do not stop the training process, but we save one snapshot of the GAN model every ten epochs. We do that for 1000 epochs, meaning that we have at our disposal 100 snapshots of the GAN model. Finally, we examine the ML metrics of the Generator and the Discriminator, and we keep only the Snapshots that losses follow the above-mentioned rule in order to reduce the set of candidate snapshots.

[2]https://machinelearningmastery.com/practical-guide-to-gan-failure-modes/

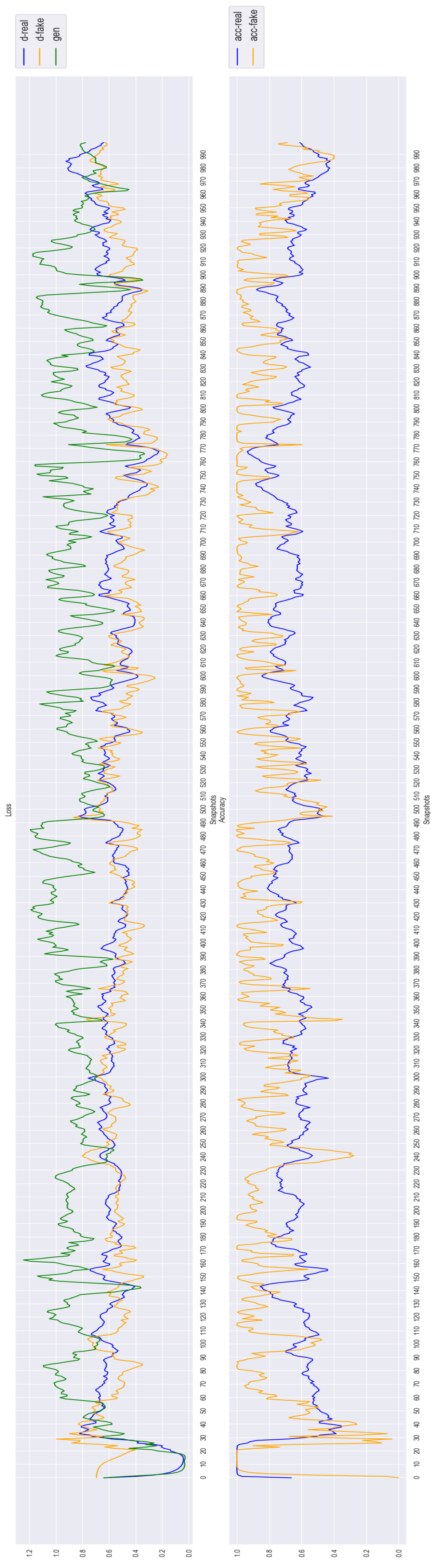


Figure 5.7: class 3 Machine Learning metrics

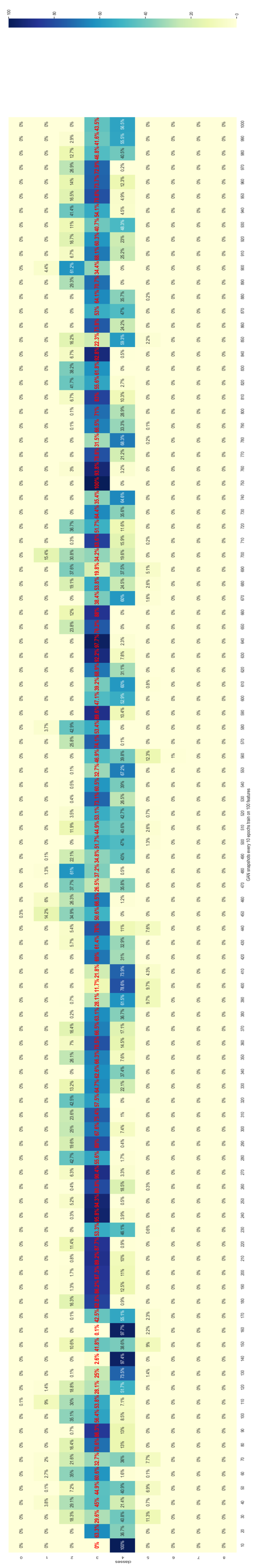


Figure 5.8: class 3 heatmap

Figure 5.9: (a): ML metrics for GANs on class 3 (b): Classification prediction of the pre-trained profiling model (RF) on synthetic traces for class 3

To evaluate if the above method works when GANs are applied in the SCA field, we trained a profiling model on real traces and evaluated each model's generated traces. This way of evaluating the traces is based on observing the results that GANs produced; this is a common way to evaluate GANs in image generation tasks with the difference that observing the results visually by a human is no longer an option. For this reason, we trained a profiling model, a Random Forest, in our case, and we classified for each snapshot a set of generated traces. In this way, we propose a second Snapshot Selection mechanism based on a profiling model's performance. We can see an example of how this snapshot mechanism works in Figure 5.8 where we can see a heatmap that represents the classification results for the generated traces of class 3. We notice a cyclic behavior of GANs as for the first epochs, the generated traces of class 3 are classified as traces that belong in other classes, meaning that the Generator's snapshots performance for these epochs is low. Next, after a couple of epochs, we can see that the Generator produces plausible traces that belong in the correct class 3. In this example, the pre-trained profiling model is the Random Forest (RF), and we have to take into account the possibility that GANs generate traces that are not plausible but able to fool the RF. This is a limitation that we have to consider and not assume that this mechanism is 100 percent reliable.

## 5.5. Dataset Feature Concatenation

The pipeline's last step is to combine a number of GANs sets to generate a dataset with traces of larger traces. In our case, we combine 7 GANs set to generate a dataset with traces of 700 time-samples each. The reason for implementing this concatenation is that by performing a set of experiments in the next Chapter, we can see that for a dataset with countermeasures, 100 features is an insufficient number in order to train a profiling model and break the key. Consequently, we show in the experiments Section 6.1.2 that this feature concatenation contributes to training sufficiently profiling models. This method can be applied to build an even larger synthetic dataset than 700 by following the same idea of concatenating 100 features.

## 5.6. Discussion

The experiments in this thesis that take place in the next Chapter 6 follow the methodology that this Chapter introduces. This thesis focuses on performing data generation when the HW power model is used. For this reason, we choose to have nine different GANs, one for each class, and train them individually. This problem might sound like an ad hoc solution that will not become generalized to other power models like the identity model where the classes are 256, but in machine learning, it has been shown that ad hoc solutions are preferred in many applications. Especially in industry where very robust neural models are needed, machine learning pipelines are built to get the desired result. One example is when Samsung[3] used 16 different neural networks to upscale the picture quality. These 16 different neural models are trained in different datasets, and in this way, they build a single pipeline able to produce high-resolution pictures. The whole idea of training and using different neural models for different inputs is also addressed and explained more extensively from Royson Lee et al. [35] where they observed that using tailored CNNs for specific video and image can lead to better performance in comparison to a more generalized model.

One alternative solution to build a more generic model for data generation is the Conditional GAN used by Wang et al. [56] where they conclude that CGANs is not a suitable model to perform data augmentation when the HW model is used. The failure of CGANs to generate plausible traces can be easily explained if we consider the heavy overfitting phenomena which appear when other machine learning models like CNNs are trained on datasets with countermeasures that aim to perform an SCA attack. These overfitting phenomena indicate that correctly classifying traces with countermeasures is a problem that, according to our knowledge, is not solved yet. On the other hand, this is not an obstacle when we try to retrieve the key; even with the low accuracy and heavy overfitting profiling models, we can still retrieve the key using the maximum likelihood estimation explained in 2.1.3. Now, in our case, Conditional GANs do not attempt to break the key but to generate valuable traces that belong to a specific class, meaning that we need to force the Generator and the Discriminator to distinguish the classes accurately, something that is not achieved by more stable deep learning models like CNNs yet. Of course, we must be careful and not compare generative models like GANs with discriminative

[3]https://news.samsung.com/za/samsung-debuts-2021-neo-qled-micro-led-and-lifestyle-tv-lines-highlighting-commitment-to-sustainable-and-accessible-future

models like the CNNs one used in [8], but we can assume that generating traces that correspond to the correct class is a difficult task. Therefore under this theoretical assumption, we chose to train one GAN model for each class.

The Snapshot Selection will be an essential component of the pipeline that is demonstrated in the last section of this Chapter; of course, we can claim that if we choose the right architecture and perform a proper tuning, we might get a very stable GAN that will not need any Snapshot selection mechanism, and this is true as it can be seen in Section 6.2 of the next Chapter 6. However, we can not claim that we will always be able to find an architecture that is stable in every epoch.

## 5.7. Conclusions and Limitations

In the following list, we briefly present the components of our pipeline, and we address its limitations from a theoretical point of view:

1. Window Selection: we choose a specific area of 700 features, and from these, we split the training dataset into seven sets of 100 features to train the profiling and the GAN models
   - When the unprotected dataset is used, the linear correlation between the power consumption (represented by the labels) and the features (time-samples) of the traces is large, and the Pearson's correlation coefficient can sufficiently indicate the information leakage windows
   - When we use a dataset with countermeasures, the linear correlations are getting hidden, meaning that other statistical methods should be used instead [5].
2. Architecture Selection: we carefully choose the Discriminator and the Generator's architecture, specifically the number, type, and width of layers that should be used.
3. GANs Training Phase: a phase that requires as the previous one serious tuning, meaning that multiple hyperparameters must be tuned in order to achieve high-quality traces
   - When the unprotected dataset is used, we can use as an evaluation method the pre-trained model of the Snapshot selection phase in order to classify the generated traces, also as it is explained in Section 5.4, the machine learning metrics must be studied in order to understand the behavior of the model.
   - Unfortunately, when a dataset with complex countermeasures has been implemented, the usage of a pre-trained model is no longer an option; for this reason, only the machine learning metric observation and Pearson's Correlation between trace-points and Power methods can be used for performing parameter optimization to the model.
4. Snapshot selection: a crucial component of the pipeline as the right choice of the snapshots can boost the performance of the GANs.
5. Feature Concatenation: a component that can be easily performed without any need for studying the complexity of the dataset.

# 6 Experiments and Results

In the previous chapter, we defined the pipeline for constructing synthetic datasets, and we addressed its limitations and capabilities from a theoretical point of view. In this Chapter, we exploit our pipeline to build two architectures. The first architecture is a shallow architecture capable of generating plausible synthetic traces for the unprotected dataset. The second is an enhanced version of the first one and focuses on generating synthetic traces for the two desynchronized datasets of jitter 10 and 15, respectively. Therefore, the reason for implementing an enhanced and, therefore, deeper architecture when more complex datasets are presented can be understood if we look at Figure 5.4 where we can see for the desynchronized dataset of jitter 15, Pearson's correlation coefficient magnitude reduces a lot, meaning that linear relationships between the features and the traces' power are minimized. For this reason, we need a more potent Discriminator model in order to push the Generator to generate more plausible synthetic traces.

In order to evaluate the quality of the synthetic traces, the two following methods are used:

- Pearson's Correlation between trace-points and Power (PC) comparison
- Guessing Entropy (GE) of the profiling models trained on the synthetic dataset

After evaluating datasets that consist of synthetic traces, we perform data augmentation by using the generated synthetic traces of the second architecture. In order to evaluate the quality of the augmented traces, the following two methods are used:

- GE curves calculation for two different dataset sizes and comparison between real augmented and replicated traces.
- Comparison of the GE curves between GANs' augmented datasets and SMOTEd augmented datasets.

Consequently, we divide this Chapter into three sections. In Section 6.1, we explore the shallow architecture results and how the proposed pipeline of the previous chapter improves its performance. In Section 6.2, we demonstrate the second architecture results and how we concluded to this more potent architecture by using the proposed pipeline. In these two Sections, we evaluate the two architectures' performance by generating only synthetic traces when the GANs are trained in the full dataset of 50,000 traces. In Section 6.3, we perform an investigation on data augmentation by using only the second (enhanced) architecture.

## 6.1. Investigation Using Shallow GANs

In this section, we evaluate various aspects of the shallow GAN architecture (Figure 6.1); detailed information about the number of parameters of the Discriminator's, the Generator's and the whole system can be found in Table 6.1, 6.2 and 6.3 respectively.

In this section, two datasets are used the Unprotected and the desynchronized dataset of jitter 10. For both datasets, we use the 50,000 traces of the ASCAD dataset[1] to train the GANs of the shallow architecture. In the list below, we address the two datasets.

[1]https://github.com/ANSSI-FR/ASCAD

- Unprotected (ASCAD dataset of 50,000 traces of 100 features each)
- Jitter 10 (constructed by introducing jittering of 10 on the Unprotected dataset)

We evaluate their performance by demonstrating the results of the two evaluation methods mentioned above, namely Pearson's Correlation between trace-points (PC) and Power and GE, and at the same time, we investigate how the proposed pipeline 5.1 of Chapter 5 affects the performance of the GANs by conducting experiments on the different Snapshot Selection Mechanisms 5.4, and by performing the Feature Concatenation method 5.5.

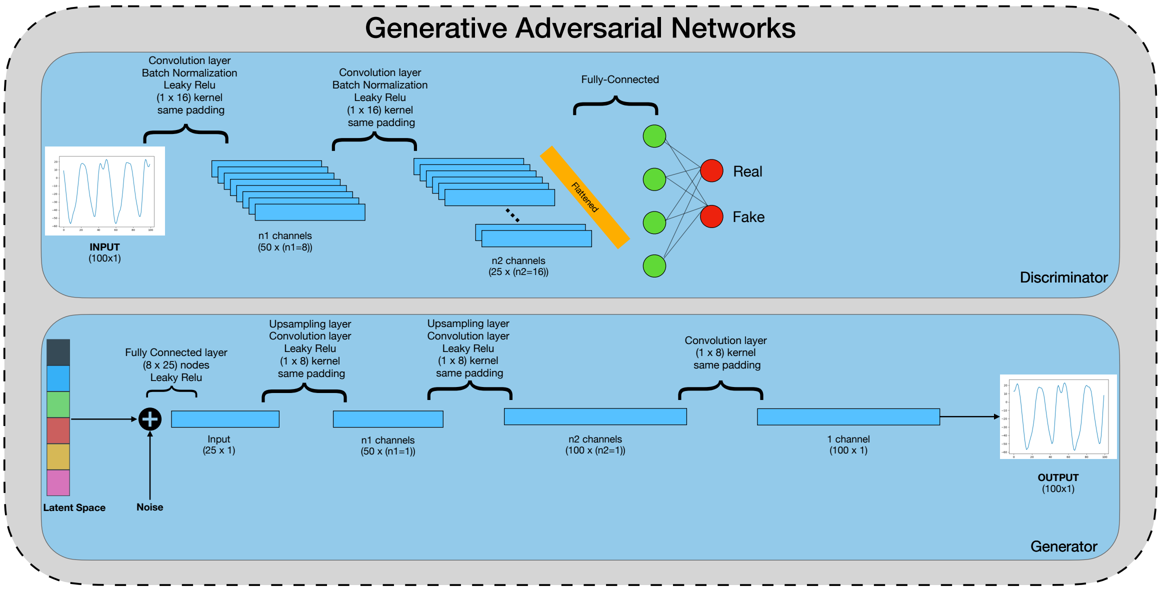


Figure 6.1: Generative Adversarial Networks (shallow architecture)

| Layer (type) | Output shape | Parameters |
|---|---|---|
| Convolutional Layer | (None, 50, 8) | 136 |
| Leaky ReLU | (None, 50, 8) | 0 |
| Convolutional Layer | (None, 25, 16) | 2064 |
| Batch Normalization Layer | (None, 50, 8) | 64 |
| Leaky ReLU | (None, 50, 8) | 0 |
| Flatten | (None, 400) | 0 |
| Fully Connected Layer | (None, 1) | 401 |

Table 6.1: Discriminator Configuration

### 6.1.1. Snapshot Selection Mechanism Evaluation with Pearson's Correlation between Trace Points and Power Comparison

In this subsection, we conduct a set of experiments to investigate how the different Snapshot Selection Mechanisms (look in Section 5.4 for more information about the Snapshot Selection Mechanisms) performance by looking into the differences of PC (Pearson's Correlation between trace-points and Power) distributions between the synthetic and the real datasets. Therefore we train the shallow GAN architecture with two datasets, namely the unprotected and the desynchronized with jitter 10, and for each real dataset, we build three synthetic datasets, one for each Snapshot Selection Mechanism, namely the Best Snapshot Selection based on the ML metrics, the Best Snapshot Selection based on the RF classifier and the Worst Snapshot Selection.

| Layer (type) | Output shape | Parameters |
|---|---|---|
| Fully Connected layer | (None, 200) | 20200 |
| Leaky ReLU | (None, 200) | 0 |
| Reshape | (None, 25, 8) | 0 |
| UpSampling Layer | (None, 50, 8) | 0 |
| Convolutional Layer | (None, 50, 1) | 129 |
| Batch Normalization Layer | (None, 50, 1) | 4 |
| Leaky ReLU | (None, 50, 1) | 0 |
| UpSampling Layer | (None, 100, 1) | 0 |
| Convolutional Layer | (None, 100, 1) | 17 |
| Batch Normalization Layer | (None, 100, 1) | 4 |
| Leaky ReLU | (None, 100, 1) | 0 |
| Convolutional Layer | (None, 100, 1) | 26 |

Table 6.2: Generator Configuration

| Model (type) | Output shape | Parameters |
|---|---|---|
| Generator | (None, 100, 1) | 20380 |
| Discriminator | (None, 1) | 2665 |

Table 6.3: Overall shallow GAN configuration

Figure 6.2b shows the PC comparison between the real unprotected dataset and a synthetic unprotected dataset of 50,000 traces each. The PC is calculated by measuring the Pearson's correlation coefficient between the features and the dataset labels. We can notice that, for this window of 100 features, the synthetic traces follow almost the same correlation distribution. Theoretically, the PC visualization may not be the optimal evaluation metric for the quality check of the synthetic traces as it shows the leakage peaks and not whether this similarity of the peaks will lead to sufficient Side-channel attack, which could be fully demonstrated by calculating the GE metric. Nevertheless, we can hypothesize that if the new synthetic traces follow the same PC distribution as the real traces, then the synthetic traces' features will have the same characteristics as the real traces. In other words, PC is a method that visually represents the leakage information of the traces. Therefore, the PC can be an intermediate step for evaluating the new synthetic traces, and in the case that synthetic traces follow the same PC with the real traces, we can claim that the new traces contain valuable leakage information.

For this specific PC comparison of Figure 6.2b, we chose one snapshot of the 100 available snapshots for each class by observing the Machine Learning metrics (Generator's and Discriminators' loss and accuracy) of each class in Figures A.1, A.4, A.7, A.10, A.13, A.16, A.19, A.22 and A.25. We notice that robust candidate snapshots are the Best Snapshot (ML) of Table 6.4. By choosing this set up of snapshots, the synthetic dataset's PC of features 300 to 400 successfully follows the real traces' PC analysis.

Next, the question is: "what happens if we choose a different set of snapshots". The answer is that GANs usually have a cyclic behavior [43], [33], meaning that GANs performance fluctuates through the epochs. For some periods, the snapshots can generate valuable synthetic data, while for other periods, the snapshots cannot generate something better than noise. In order to demonstrate the necessity of performing snapshot selection, we conduct the following three comparisons.

- We choose the best snapshots according to the ML metrics
- We choose the best snapshots according to a pre-trained profiling model (RF)
- We choose the worst snapshots according to a pre-trained profiling model (RF)

In Table 6.4 we indicate the corresponding snapshots of each selection method for the unprotected dataset.

| class | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| Best Snapshot (ML) | 80 | 50 | 640 | 520 | 210 | 650 | 230 | 80 | 110 |
| Best Snapshot (RF) | 740 | 170 | 660 | 640 | 190 | 550 | 370 | 620 | 730 |
| Worst Snapshot | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 |

Table 6.4: Snapshot selection by using three different mechanisms on unprotected dataset

In Table 6.5 we indicate the corresponding snapshots of each selection method for the desynchronized dataset of jitter 10.

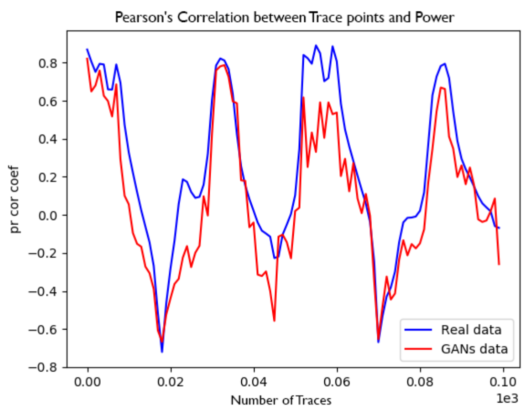
| class | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| Best Snapshot (ML) | 100 | 850 | 400 | 690 | 800 | 480 | 750 | 500 | 500 |
| Best Snapshot (RF) | 680 | 950 | 480 | 810 | 740 | 500 | 880 | 830 | 420 |
| Worst Snapshot | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 |

Table 6.5: Snapshot selection by using three different mechanisms on desynchronized dataset with jitter 10

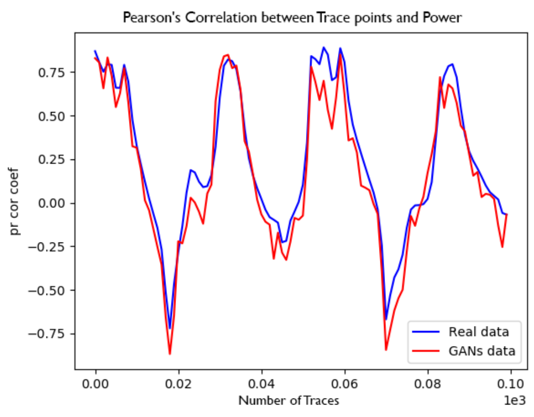
In Figure 6.2 we observe the comparisons of the three snapshot selection mechanisms explained in Table 6.4 for the unprotected dataset. We notice that the Pearson's correlation coefficient distribution of the synthetic traces fits better the correlation distribution of the real dataset when we choose the best snapshot selection according to the pre-trained RF profiling model's classification results. The PC distribution of the Best Snapshot (ML) and Worst Selection (RF) synthetic datasets also follow the PC distribution of the real dataset quite well, with the first one providing a more accurate distribution than the second, but both distributions seem to be less accurate than the PC distribution of the Best Snapshot (RF) synthetic dataset.

In Figure 6.3 we observe the comparisons of the three snapshot selection mechanisms explained in Table 6.5 for the desynchronized dataset of jitter 10, and we notice the same trend about the PC distribution between the real dataset and the three synthetic datasets again.

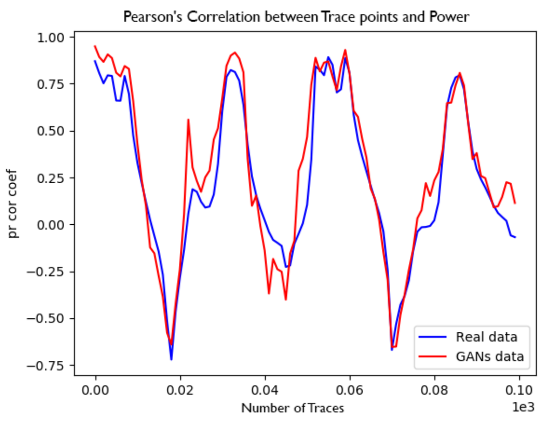
Choosing a set of snapshots where the pre-trained profiling model characterizes them as traces that do not belong in the correct class leads to a PC distribution that does not follow the real dataset's PC distribution (6.2a, 6.3a). We expected this observation as we know that a trace with a very low signal-to-noise ratio leads to PC with random peaks, and furthermore, we know that a wrong snapshot selection leads to traces that consist more of random noise rather than valuable information. On the other hand, Figures 6.2c and 6.3c show that the PC distributions (real and synthetic dataset) are very similar, but we cannot claim that this synthetic dataset will perform equally well with the real dataset in a Side-channel attack. The only assumption that we can make is that similar PC distribution indicates that the synthetic trace features correlate similarly with the labels compared to the real traces' features.



(a) Worst Snapshot Selection based on profiling model (RF)

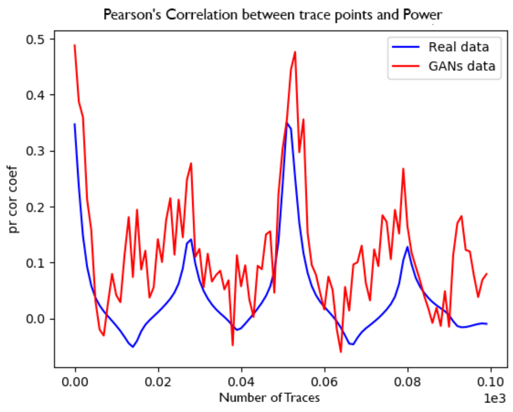


(b) Snapshot Selection based on Machine Learning metrics

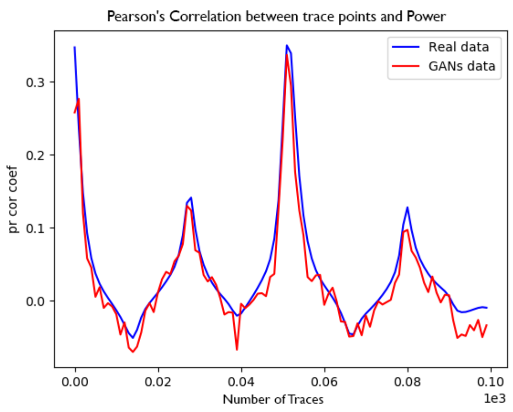


(c) Best Snapshot Selection based on profiling model (RF)

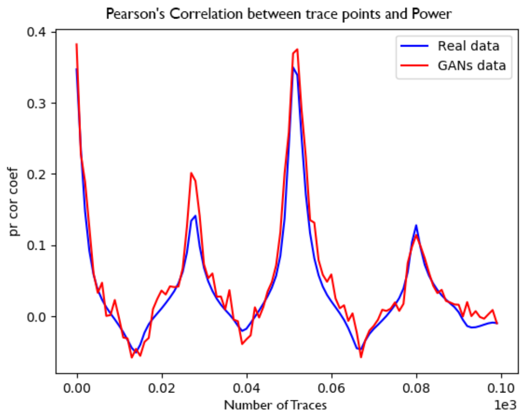
Figure 6.2: A comparison of three different types of snapshot selections on unprotected dataset



(a) Worst Snapshot Selection based on profiling model (RF)



(b) Snapshot Selection based on Machine Learning metrics



(c) Best Snapshot Selection based on profiling model (RF)

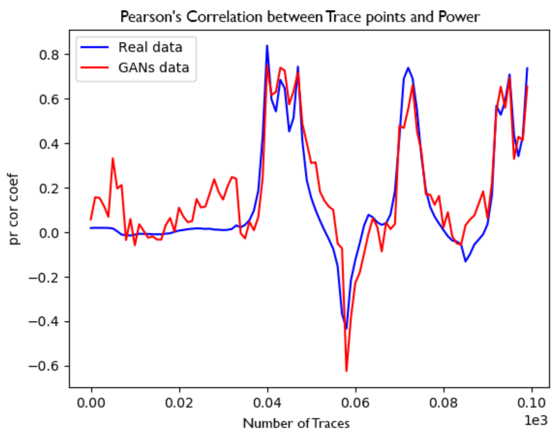
Figure 6.3: A comparison of three different types of snapshot selections on desynchronized dataset of jitter 10

### 6.1.2. Feature Concatenation Experiment

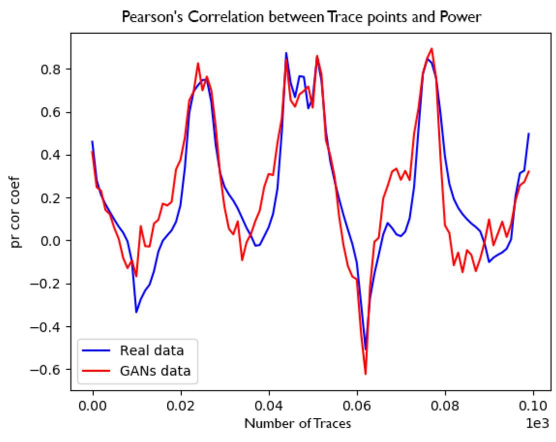
Next, we demonstrate the PC for the two datasets (unprotected and desynchronized of jitter 10) of the 700 features window. In order to construct a synthetic dataset of 700 features, we follow the procedure that is explained in section 5.5. Briefly, we train seven different sets of GANs, and we concatenate the generated traces in order to get a new synthetic dataset of 700 features. The individual performance of each 100 features window is demonstrated in Figure 6.4 for the unprotected dataset and in Figure 6.5 for the dataset with jitter 10. Both Figures 6.4 and 6.5 show clearly for windows (a): 0-100, (b): 100-200, (c): 200-300, (d): 300-400, (e): 400-500 that the PC distribution of the synthetic dataset follows the PC distribution of the real dataset while, for windows (f): 500-600 and (g): 600-700 the PC peaks of the real dataset are lower, meaning that GANs cannot capture the distribution of the real dataset. Furthermore, using a profiling model for evaluating the generated traces now is not an option as the profiling models failed to achieve validation accuracy higher than 30 percent, and therefore it is pointless to conduct this selection method. Both Figures 6.4 and 6.5 show the same trend that for some of the windows, the PC distribution of the synthetic follow more accurately the PC distribution of the real dataset, meaning that some windows have more robust features than others and this can also be understood if we look at the difference between the two Figures (6.4, 6.5) where the features of the jittered dataset are less robust, and therefore the shallow architecture struggles a lot to generate a synthetic dataset where its PC distribution follows the PC distribution of the real dataset.

The fact that the PC distribution of the synthetic traces on windows (f) 6.4f, 6.5f and (g) 6.4g, 6.5g do not follow the real traces' correlation distribution is something that we expected. The correlation peaks in these two windows are very low, meaning that we need GANs that can extract the patterns out of these traces. The shallow architecture that was used for these experiments cannot capture these patterns meaning that we need a higher capacity architecture to generate traces with such low PC peaks. This was achieved by introducing the enhanced architecture of Figure 6.10, described in section 6.2.

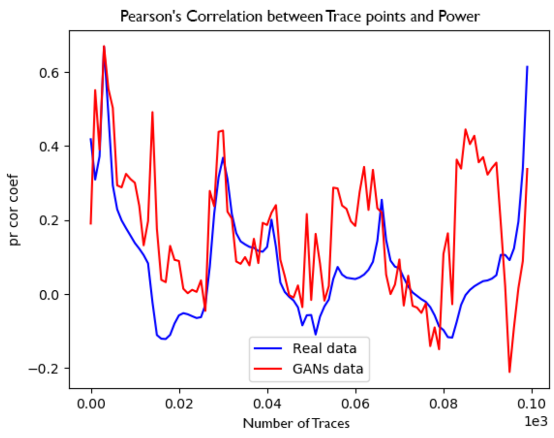
Lastly, in the following Figures 6.6a 6.6b and we can see the PC of the 700 features synthetic dataset vs. the PC distribution of the 700 features real unprotected dataset and the desynchronized dataset of jitter 10, respectively.



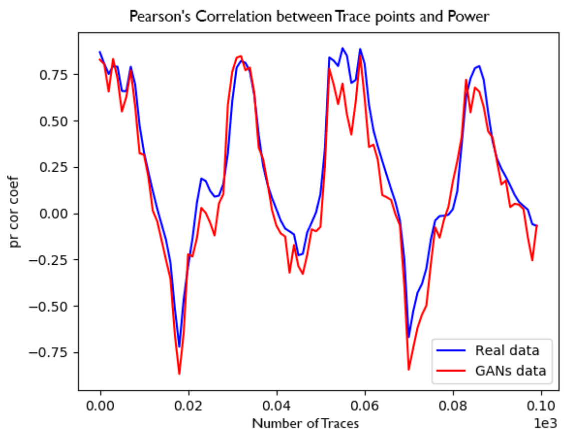
(a) features 0 - 100, 50,000 traces



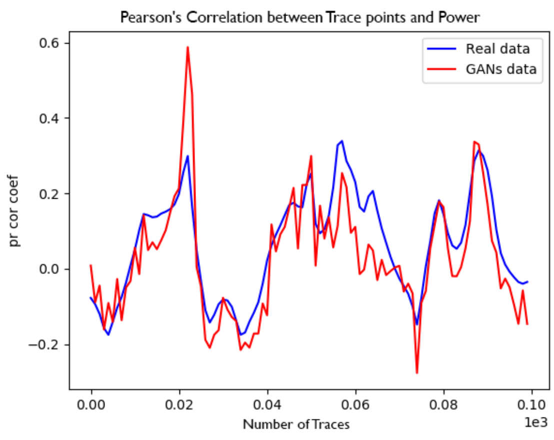
(b) features 100 - 200, 50,000 traces



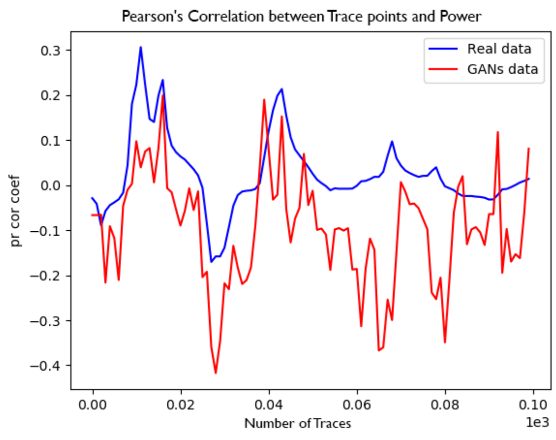
(c) features 200 - 300, 50,000 traces



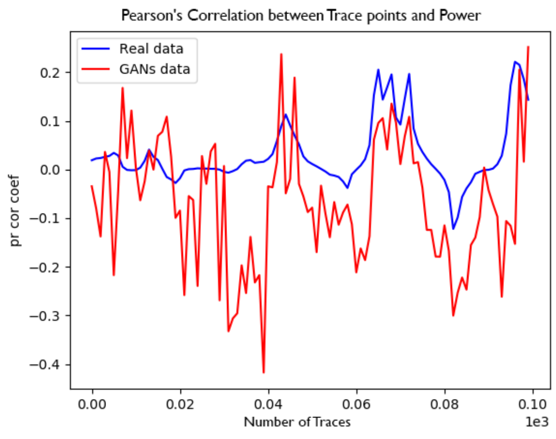
(d) features 300 - 400, 50,000 traces



(e) features 400 - 500, 50,000 traces

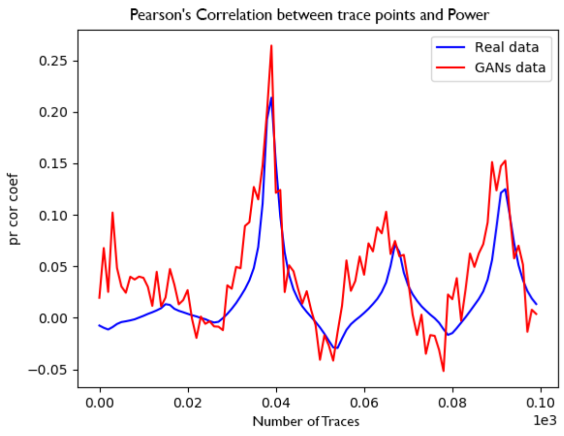


(f) features 500 - 600, 50,000 traces

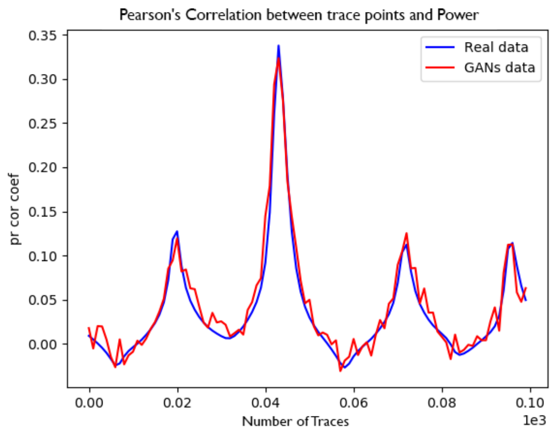


(g) features 600 - 700, 50,000 traces

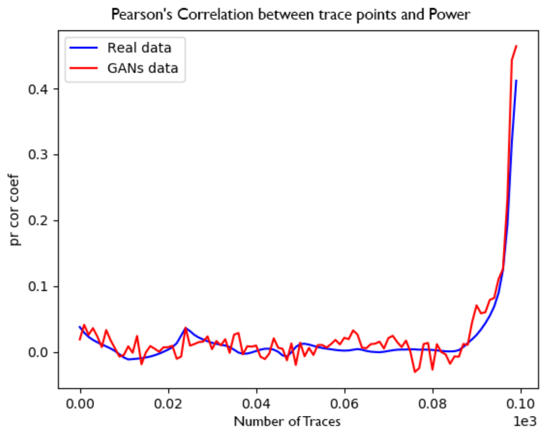
Figure 6.4: Correlation Power Analysis on unprotected dataset



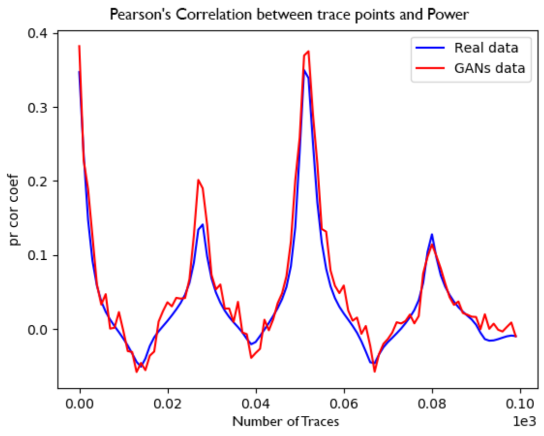
(a) features 0 - 100, 50,000 traces



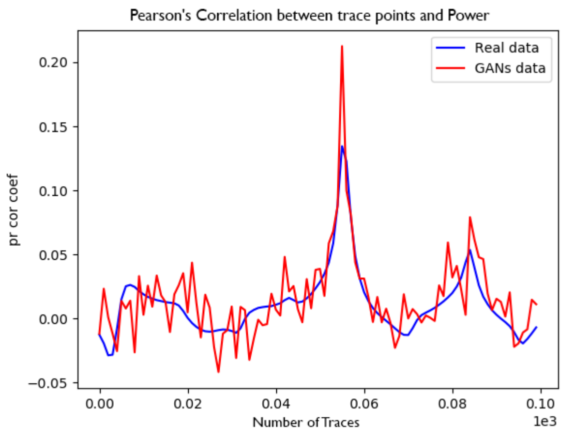
(b) features 100 - 200, 50,000 traces



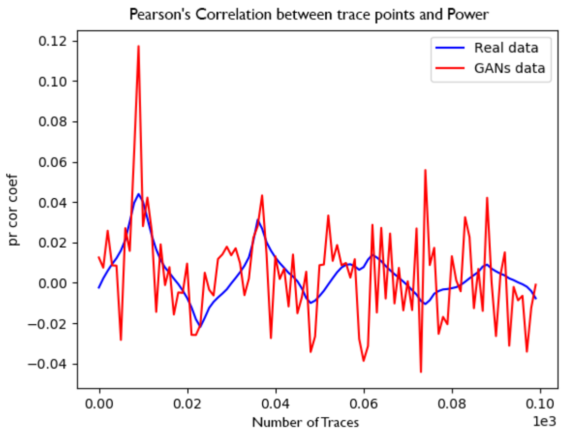
(c) features 200 - 300, 50,000 traces



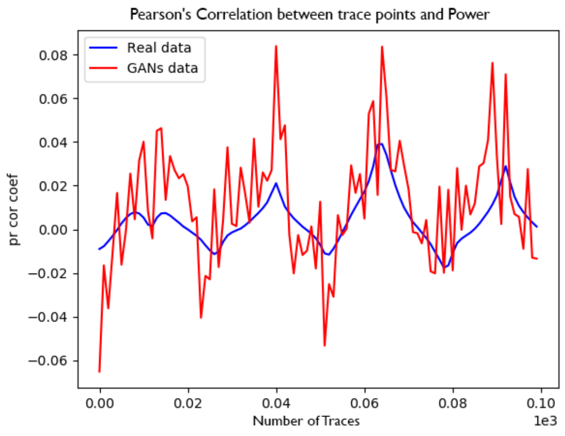
(d) features 300 - 400, 50,000 traces



(e) features 400 - 500, 50,000 traces

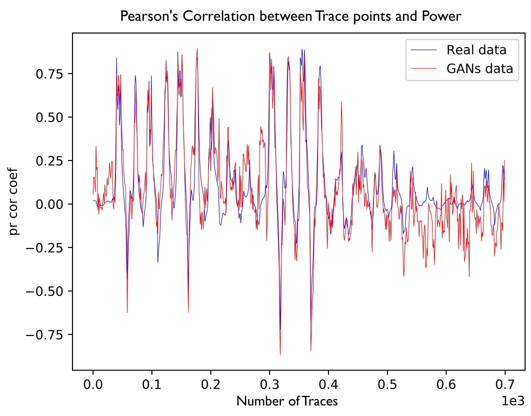


(f) features 500 - 600, 50,000 traces

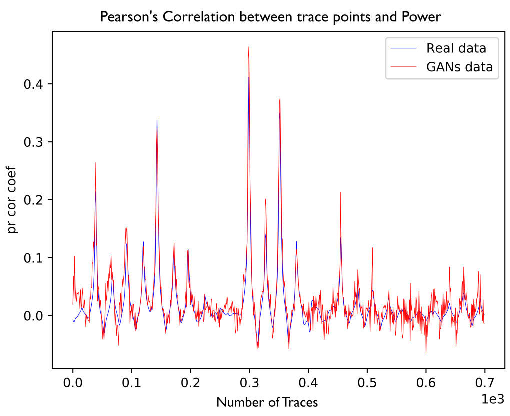


(g) features 600 - 700, 50,000 traces

Figure 6.5: Correlation Power Analysis on desynchronized dataset of jitter 10



(a) 700 concatenated features of unprotected dataset



(b) 700 concatenated features on desynchronized dataset of jitter 10

Figure 6.6: CPA on 700 features for two datasets (a): unprotected dataset, (b): desynchronized dataset of jitter 10

### 6.1.3. Guessing Entropy on Synthetic Dataset

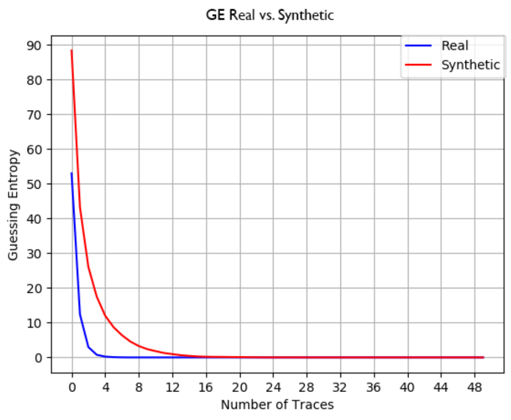
Previously we evaluated the synthetic dataset by visualizing the PC of the synthetic traces vs. the real traces. We concluded that GANs could generate traces that follow the same PC as the real traces. Furthermore, we showed that snapshot selection contributes significantly in finding GAN's snapshots that follow the PC distribution more accurately.

Here we evaluate the synthetic traces by training the RF with the aforementioned synthetic dataset (of 100 features and 700 features), and next, we demonstrate the GE curves of the profiling model trained on synthetic traces vs. the RF model trained on real traces.

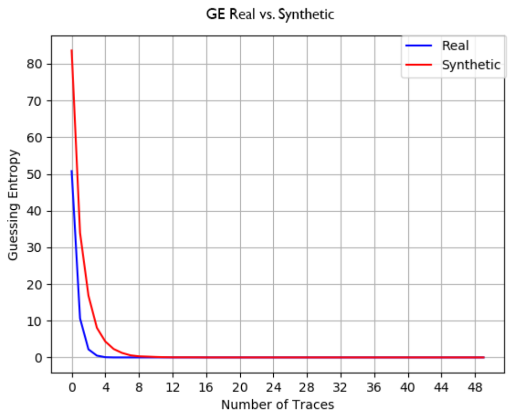
We trained the RF profiling model on the synthetic unprotected dataset, and the validation accuracy was almost 35% for the 100 feature synthetic dataset, while with the 700 feature synthetic dataset, we managed to achieve almost 45% validation accuracy, meaning that the overfitting phenomenon was heavily reduced after the Feature Concatenation method. Figure 6.7a, shows that by training the profiling model (RF) with the 100 feature synthetic unprotected dataset generated by 7 GANs, the key can be retrieved by using 12 traces, while for the 700 feature synthetic unprotected dataset, the key can be retrieved by using only seven traces not many more than the number of traces required when the RF model is trained on a real dataset as it can be seen in Figure 6.7b. More specifically, Figure 6.7c shows clearly this GE performance improvement when the Feature Concatenation method is used.

We performed the same experiments now for the desynchronized dataset with jitter 10. Figure 6.8a and 6.8b shows the GE performance between the synthetic dataset and the real dataset for the 100 and 700 feature synthetic desynchronized datasets with jitter 10, respectively, while Figure **??** shows the GE performance comparison between the two synthetic datasets namely, the 100 and the 700 feature synthetic desynchronized datasets with jitter 10. For the desynchronized dataset, we observe the same trend with the unprotected dataset in the GE performance with the only difference that now we need slightly more traces to break the key, more specifically 30 traces for the 100 feature synthetic desynchronized dataset and 20 for the 700 feature synthetic desynchronized dataset.

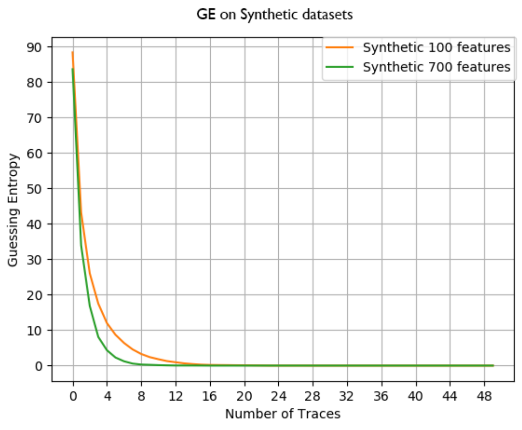
Furthermore, in Figure 6.9 we make a GE comparison between the different types of snapshot selection mentioned in the 6.1.1. We can notice that selecting snapshots according to the best method the faster the GE curve converges towards the real dataset's GE curve. The difference between selecting the worst (RF-based) and the best (RF-based) snapshots is noticeable, meaning that we should not randomly choose snapshots. On the other hand, although the GE difference is not that large between the best selection (ML-based) and the best selection (RF-based), the synthetic traces based on the pre-trained RF model still give better results and therefore are preferred. With this information, we can argue that we prefer synthetic datasets that their PC distribution follows the real dataset's PC distribution, meaning that PC comparison between the synthetic dataset and the real dataset is a valid intermediate evaluation step.



(a) 100 features window
(unprotected)

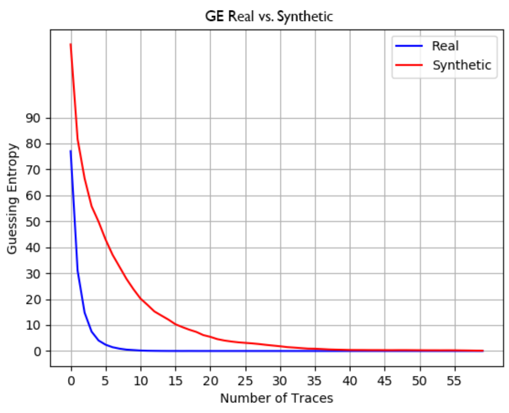


(b) 700 features window
(unprotected)

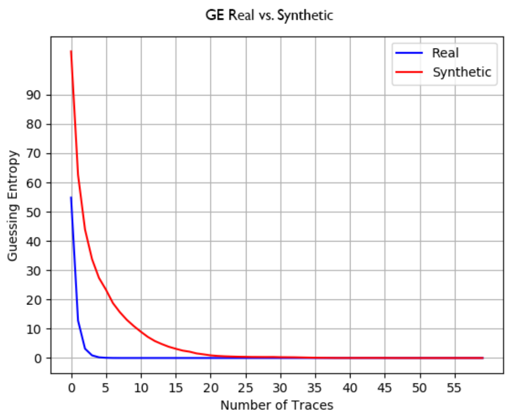


(c) 100 feature synthetic dataset
vs. 700 feature synthetic dataset
(unprotected)

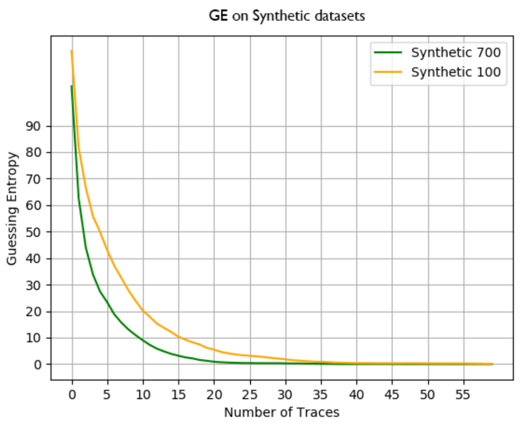
Figure 6.7: (a): a comparison between real and synthetic traces on a 100 features window, (b): a comparison between real and synthetic traces on a 700 features window, (c): a comparison between synthetic on 100 features window and synthetic traces on a 700 features window



(a) 100 features window (jitter 10)

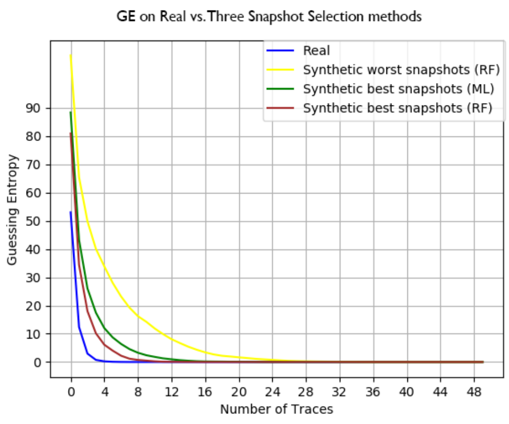


(b) 700 features window (jitter 10)

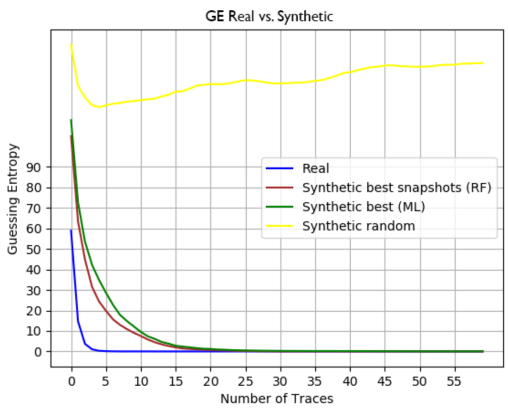


(c) 100 feature synthetic dataset
vs. 700 feature synthetic dataset
(jitter 10)

Figure 6.8: (a): a comparison between real and synthetic traces on a 100 features window, (b): a comparison between real and synthetic traces on a 700 features window, (c): a comparison between synthetic on 100 features window and synthetic traces on a 700 features window on desynchronized dataset of jitter 10



(a) GE between 3 different
snapshot selection mechanisms
on unprotected dataset



(b) GE between 3 different
snapshot selection mechanisms
on desynchronized dataset of
jitter 10

Figure 6.9: 3 Different Selection Mechanism on two datasets (a): unprotected and (b): desynchronized with jitter 10

### 6.1.4. Discussion on the Synthetic Traces

In this Section, we used all the 50000 traces of the ASCAD dataset to train the GANs on two datasets, namely, the unprotected dataset and the desynchronized dataset of jitter 10. Moreover, in order to train the RF for making the snapshot selection again, all the 50000 traces have been used for both datasets. In this section, we investigated if the GANs can generate high-quality synthetic traces by using the shallow architecture 6.1 and how the proposed pipeline improve the performance of the shallow GAN. We have to keep in mind that we did not attempt yet to solve the data augmentation problem. In the next four bullets, we briefly discuss the limitations and capabilities of the proposed pipeline based on the results of the presented experiments.

- For the real dataset's (both unprotected and jittered) areas with high correlation peaks (windows (a:6.4a, 6.5a), (b: 6.4b, 6.5b), (c:6.4c, 6.5c), (d:6.4d, 6.5d)), the synthetic traces sufficiently follow the correlation distribution of the real dataset, while for the rest of the windows GANs do not perform that well, meaning that GANs' performance depends on how robust are the features of the dataset and therefore its complexity.
- By using the GANs' Machine Learning metrics, we can sufficiently choose a snapshot that can generate valuable synthetic traces and therefore to retrieve the key.
- We also show that snapshot selection is a procedure that is essential as the worst selection of snapshots leads to synthetic traces of lower quality. Furthermore, we notice that more careful snapshot selection like the one we performed using a pre-trained profiling model can lead to GANs snapshot models able to generate new plausible traces. Therefore with such a dataset, we are able to train a profiling model in order to perform a successful Side-channel attack.
- Lastly, we showed that using GANs models for several windows and then concatenating the features (time-samples) of the synthetic dataset leads to a more powerful high-dimensional synthetic dataset which can further improve the profiling models' performance and, therefore, to a faster recovery of the key.

## 6.2. Enhancing the GANs Architecture

In this section, we evaluate various aspects of the enhanced GAN architecture (6.10). The reason for designing an enhanced version of the shallow architecture can be understood by looking at PC Figures 6.7c and 6.8c we can definitely see that moving from the unprotected dataset to a dataset with mild countermeasures such as jittering the performance for the architecture of Figure 6.1 reduces a lot. For this reason, we introduced an enhanced architecture and conducted a set of experiments aiming to achieve better results when the desynchronization countermeasures are present. In this section, we use three datasets to conduct experiments; the first two also used in the previous section, the unprotected and the desynchronized dataset of jitter 10, while in this section, we increment the jitter by 5 and we construct a desynchronized dataset of jitter 15 in order to see the limitations of the new enhanced architecture. Therefore in the following list, we address the three datasets.

- Unprotected (ASCAD dataset of 50,000 traces of 100 features each)
- Jitter 10 (constructed by introducing jittering of 10 on the Unprotected dataset)
- Jitter 15 (constructed by introducing jittering of 10 on the Unprotected dataset)

Figure 6.10 shows the enhanced architecture. The only difference in the layers between the shallow and the enhanced architecture is the addition of one layer in the Discriminator of the enhanced architecture, while the number of layers of the Generator is the same. The addition of this one layer makes the Discriminator more robust in discriminating what is synthetic (fake) and what is real. On the other hand, this had as a result that the Generator during the training phase became very weak even from the first epochs and not able to take advantage of the Discriminator's robustness as the Generator's loss remained the same during the training phase. For this reason, we had to construct a more robust Generator able to compete with the enhanced Discriminator. We kept the same number of layers for the Generator, but we increased the number of channels. More specifically, we added 7 channels (n1=8) to the first Convolution layer and 15 channels (n2=16) to the second one. About the

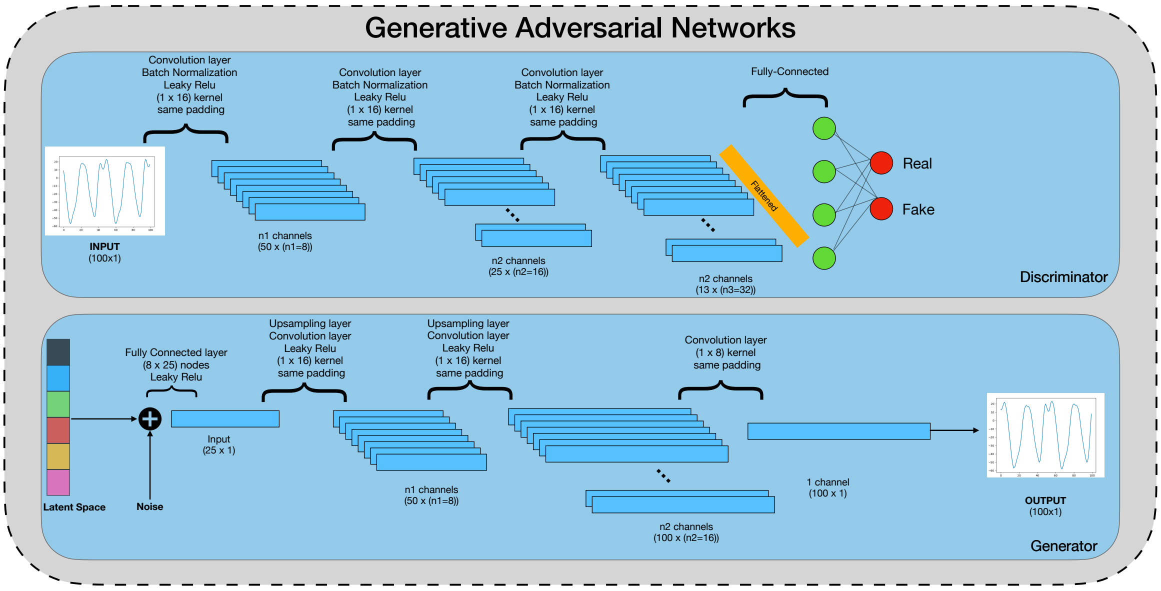


Figure 6.10: Generative Adversarial Networks (enhanced architecture)

Discriminator, the number of channels for each layer remained the same as before, while the number of channels for the new third layer was n3=32. Furthermore, about the Generator, we change the kernel size of the Convolutional from 8 to 16 as a more stable performance was observed with this kernel size. In the following two parts of this section, we demonstrate the enhanced architecture results by only showing the results after the Feature Concatenation step. After running experiments on architectures with multiple layers for a different set of learning rates and batch sizes, we have concluded to this architecture. The architecture in Figure 6.10 is the architecture with the best performance. Furthermore, we managed to get good results for the desynchronized dataset of jitter 15, a dataset that was too difficult for the previous section's shallow architecture. About larger jitters, the performance of this enhanced architecture was very low.

A more detailed description of the enhanced architecture where the number of the parameters of the Discriminator, the Generator and the whole system can be seen below in Tables 6.6, 6.7 and 6.8 respectively.

| Layer (type) | Output shape | Parameters |
|---|---|---|
| Convolutional Layer | (None, 50, 8) | 136 |
| Leaky ReLU | (None, 50, 8) | 0 |
| Convolutional Layer | (None, 25, 16) | 2064 |
| Batch Normalization Layer | (None, 25, 16) | 64 |
| Leaky ReLU | (None, 50, 8) | 0 |
| Convolutional Layer | (None, 13, 32) | 8224 |
| Batch Normalization Layer | (None, 13, 32) | 128 |
| Leaky ReLU | (None, 13, 32) | 0 |
| Flatten | (None, 416) | 0 |
| Fully Connected Layer | (None, 1) | 417 |

Table 6.6: Discriminator Configuration

| Layer (type) | Output shape | Parameters |
|---|---|---|
| Fully Connected layer | (None, 200) | 20200 |
| Leaky ReLU | (None, 200) | 0 |
| Reshape | (None, 25, 8) | 0 |
| UpSampling Layer | (None, 50, 8) | 0 |
| Convolutional Layer | (None, 50, 8) | 1032 |
| Batch Normalization Layer | (None, 50, 8) | 32 |
| Leaky ReLU | (None, 50, 8) | 0 |
| UpSampling Layer | (None, 100, 8) | 0 |
| Convolutional Layer | (None, 100, 16) | 2064 |
| Batch Normalization Layer | (None, 100, 16) | 64 |
| Leaky ReLU | (None, 100, 16) | 0 |
| Convolutional Layer | (None, 100, 1) | 401 |

Table 6.7: Generator Configuration

| Model (type) | Output shape | Parameters |
|---|---|---|
| Generator | (None, 100, 1) | 23793 |
| Discriminator | (None, 1) | 11033 |

Table 6.8: Overall GAN Configuration of Enhanced Architecture

### 6.2.1. Handcrafted Neural Architecture Search

We concluded to the above neural architecture of Figure 6.10 by performing a series of experiments with different number of layers, number of kernels, kernel sizes, and learning rate by evaluating first the Machine Learning metrics and secondly the classification results provided by the pre-trained profiling model explained on section 6.1.1. More specifically, in Figure 6.11 we can see the Generator's and the Discriminator's loss and accuracy for class 3 for the desynchronized dataset of jitter 15. We can notice that this Figure does not look like the Figure 5.7 of Chapter 5 where we performed Snapshot selection by studying the ML metrics of the GANs. In this Figure 6.11 we did not perform smoothing among the batches as now we are interested in the variance of the Generator's and Discriminator's loss as the variance indicates the stability of the GANs. We can definitely see that the enhanced architecture is more stable as Generator's and Discriminator's losses and accuracy oscillations reduced a lot. Furthermore, by looking at the heatmap Figures 6.12, we can see that the cyclic behavior is eliminated for most classes, meaning that we can choose the snapshot model less carefully than before and get rid of the Snapshot selection component of the proposed pipeline. Of course, this component contributed a lot to find the best architecture, and for trying trace generation in a more complex dataset like the desynchronized dataset of jitter 20, this component will boost the performance of the GANs. The ML metrics of the GANs and the classification results of the pre-trained profiling model for the rest of the classes can be found in the Appendix Chapter A

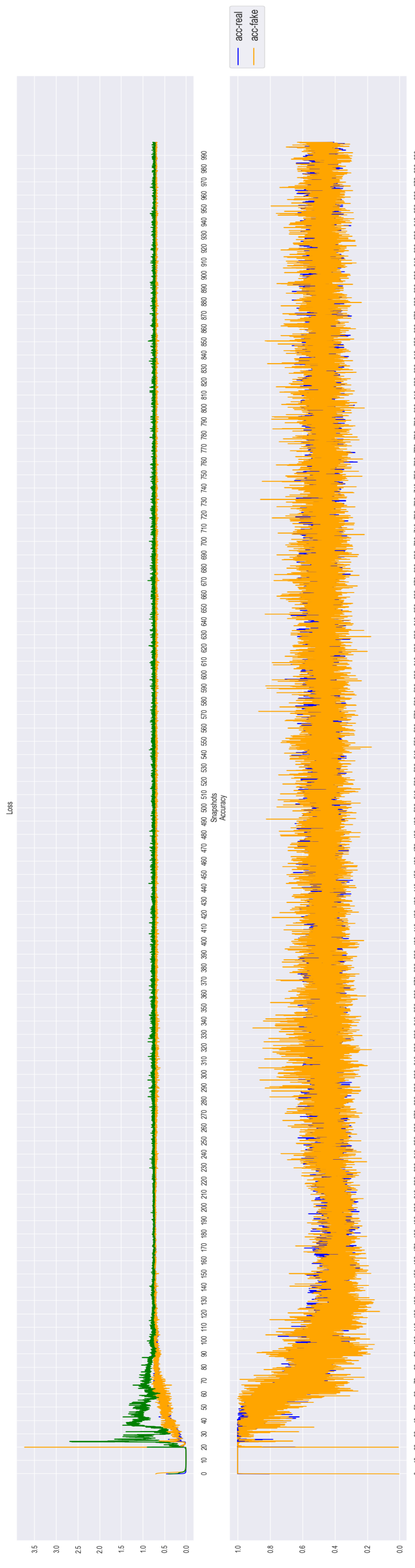


Figure 6.11: class 3 Machine Learning metrics

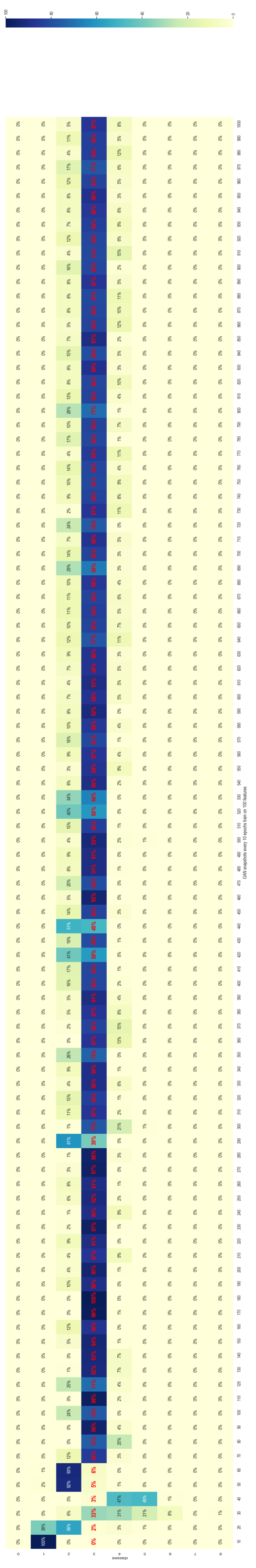
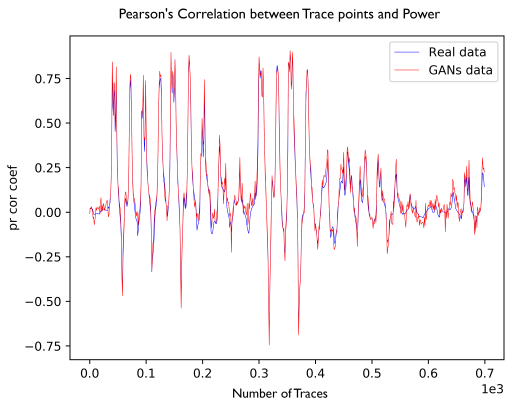


Figure 6.12: class 3 heatmap

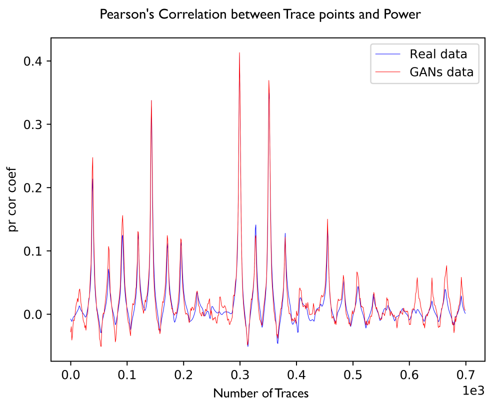
Figure 6.13: (a): ML metrics for GANs on class 3 on desynchronized dataset of jitter 15(b): Classification prediction of the pre-trained profiling model (RF) on synthetic traces for class 3 on desynchronized dataset of jitter 15

### 6.2.2. Pearson’s Correlation between trace-points and Power (PC) on Synthetic Dataset (Unprotected, Jitter 10, Jitter 15)

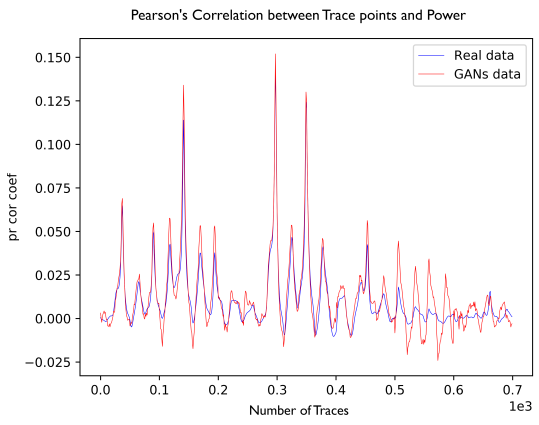
The CPA of the unprotected, Jitter 10, and Jitter 15 dataset can be seen in Figures 6.14a, 6.14b, 6.14c respectively. We can see that the synthetic datasets’ PC distributions perfectly match the real PC distribution, indicating that the enhanced architecture generates high-quality synthetic traces for the three datasets. Furthermore, worth mentioning is that now GANs can generate high-quality synthetic traces even for features window 200-300 of the unprotected dataset as it can be seen in Figure 6.15a, a window in which the shallow architecture had poor performance as we can see in Figure 6.4c of the previous section 6.1. A similar difference in performance can be observed in the 0-100 feature window of the desynchronized dataset with jitter 10, as in Figure 6.15b we can see that the synthetic’s dataset PC distribution fits quite well the PC distribution of the real dataset while, the shallow architecture had very poor performance (Figure 6.5a). Figure 6.14c shows PC comparison between the Jitter 15 real and the Jitter 15 synthetic dataset of the window of 300-400 features.



(a) 700 concatenated features of unprotected dataset for the enhanced architecture

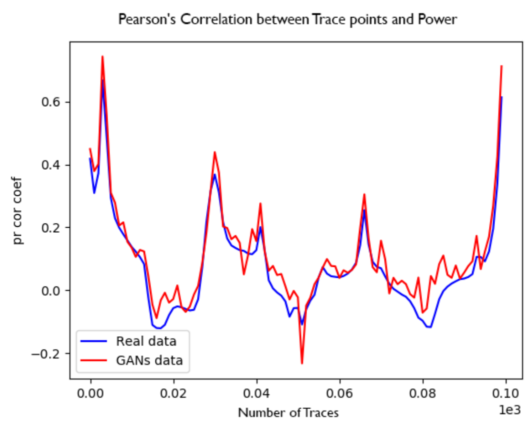


(b) 700 concatenated features of desynchronized dataset with jitter 10 for the enhanced architecture

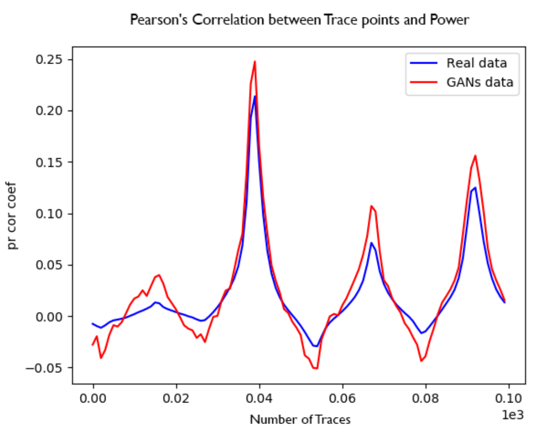


(c) 700 concatenated features of desynchronized dataset with jitter 15 for the enhanced architecture

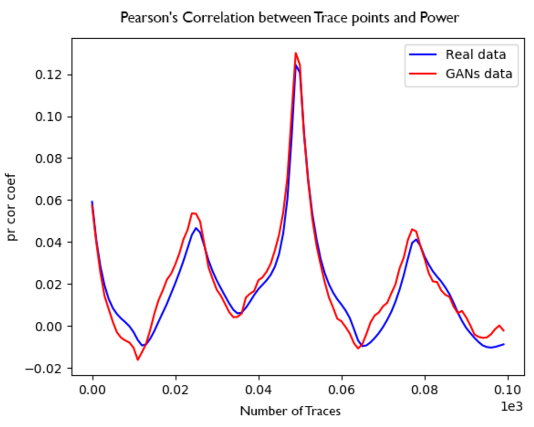
Figure 6.14: PC on 700 features for the enhanced architecture on three datasets (a): unprotected, (b): jitter 10, (c): jitter 15



(a) PC for 100 concatenated features (200-300 window) of unprotected dataset

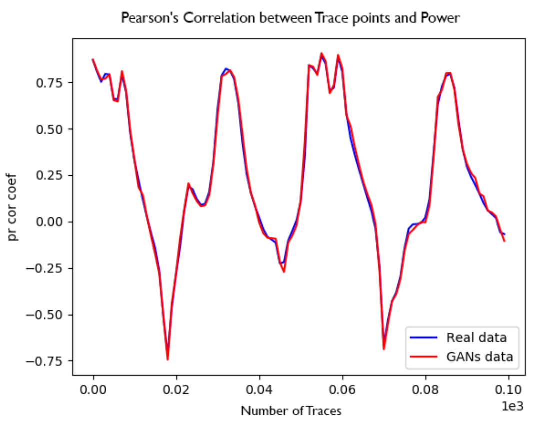


(b) PC for 100 concatenated features (0-100 window) of desynchronized dataset of jitter 10

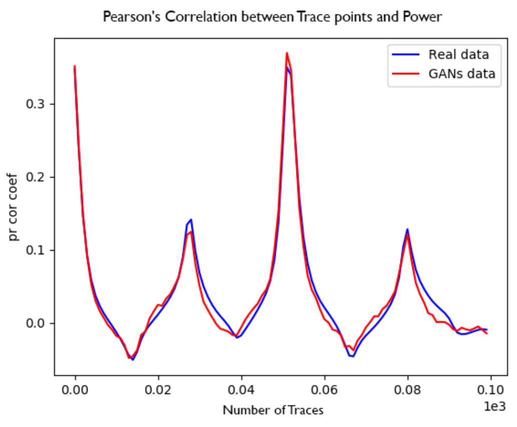


(c) PC for 100 concatenated features (300-400 window) of desynchronized dataset of jitter 15

Figure 6.15: A PC comparison for real vs synthetic dataset on the enhanced architecture for difficult windows on the (a): unprotected, (b): desynchronized jitter 10 and (c): desynchronized jitter 15



(a) PC for 100 concatenated features (300-400 window) of unprotected dataset

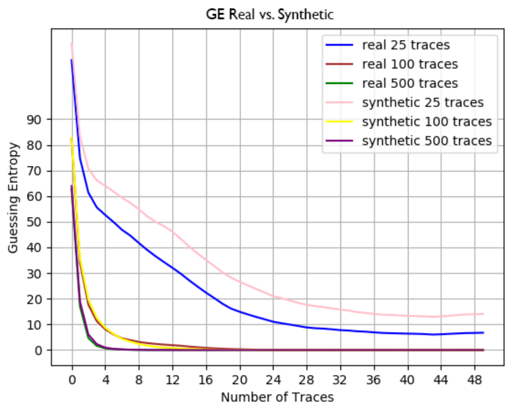


(b) PC for 100 concatenated features (300-400 window) of desynchronized dataset of jitter 10

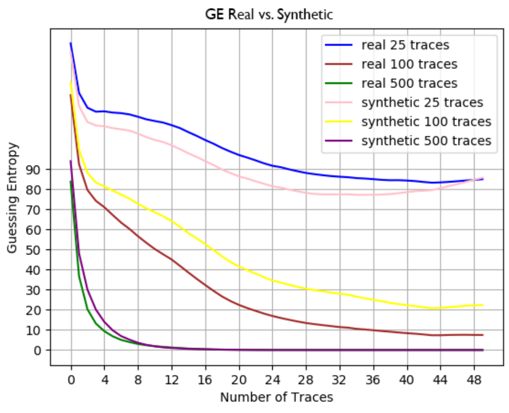
Figure 6.16: A PC comparison for real vs synthetic dataset on the enhanced architecture for easy windows on the (a): unprotected, (b): desynchronized jitter 10

### 6.2.3. GE on Synthetic Dataset (Unprotected, Jitter 10, Jitter 15)

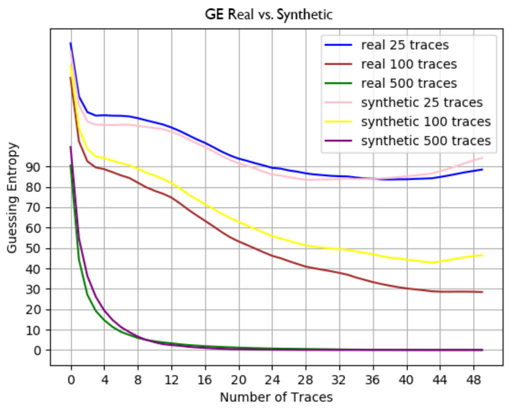
Now we evaluate the enhanced architecture by calculating the GE performance of the synthetic dataset when the GANs are trained on the three datasets (unprotected, Desynchronized with jitter 10 and jitter 15), and we compare them with the GE performance of the real dataset. More specifically, in Figures 6.17a 6.17b and 6.17c we can see the GE curves of the real and the synthetic datasets for three different sizes, namely 25, 100, and 500 traces. It is noticeable that the synthetic datasets' GE curve is very close to the real GE curve and sometimes identical. Next, in Figures 6.18a 6.18b and 6.18c we perform the same experiment but with larger datasets in order to show that for large datasets, this trend remains the same and that large synthetic dataset generated by GANs can lead to GE curves that reduce to zero very fast.



(a) GE on 700 concatenated features of unprotected dataset

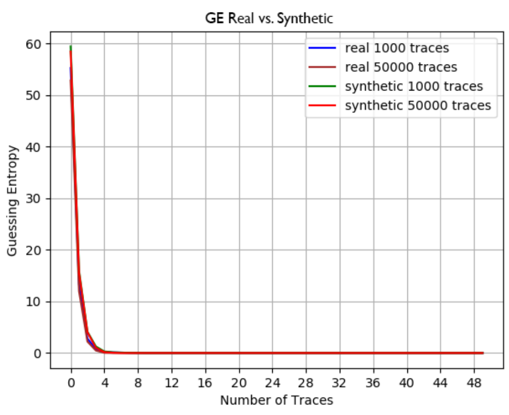


(b) GE on 700 concatenated features of desynchronized dataset with jitter 10

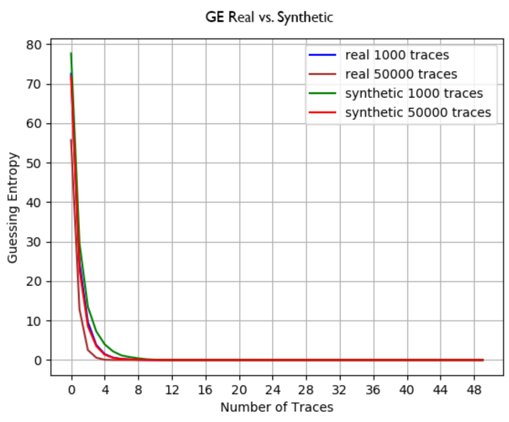


(c) GE on 700 concatenated features of desynchronized dataset with jitter 15

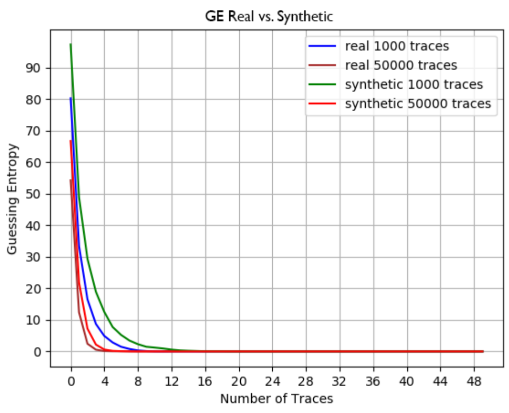
Figure 6.17: A GE comparison between real and synthetic traces on three different datasets (a): unprotected, (b): desynchronized with jitter 10, (c): desynchronized with jitter 15



(a) 700 concatenated features of unprotected dataset



(b) 700 concatenated features of jitter 10 dataset



(c) 700 concatenated features of jitter 15 dataset

Figure 6.18: A GE comparison between real and synthetic traces on three different datasets (a): unprotected, (b): desynchronized with jitter 10, (c): desynchronized with jitter 15

### 6.2.4. Discussion on Enhanced Architecture

In this section, we showed that using a deeper GANs architecture can improve the performance. More specifically, we showed that for the unprotected dataset and the desynchronized dataset with jitter 10 and 15, the PC distribution of the synthetic dataset was almost identical to the PC distribution of the real dataset. Similarly, the synthetic datasets' GE performance was almost identical to the real dataset's GE performance. Concluding, we can see that the general architecture of Chapter 5 can be enhanced by performing an architecture exploration based on trial and error by using the evaluation methods of the proposed pipeline of Figure 5.1.

## 6.3. Trace Augmentation on Unprotected and Jitter 15 Dataset

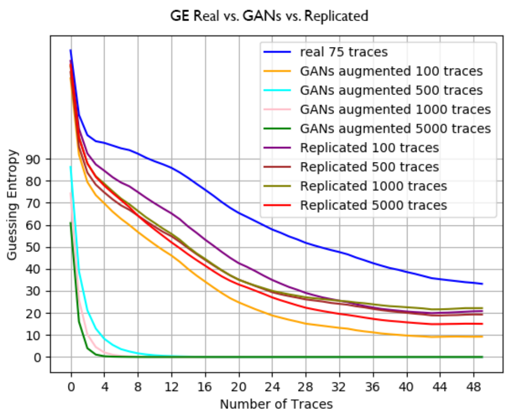
In this section, we conduct a set of data augmentation experiments. We choose a dataset that follows the binomial distribution, and we add synthetic traces produced by GANs. The experimental setup for these experiments is discussed below in 6.3.1. Next, we demonstrate a comparison between the real vs. augmented datasets by GANs vs. replicated datasets. Lastly, we compare binomial augmented datasets produced by GANs with balanced datasets produced by the SMOTE algorithm. For this subsection experiments, we use the enhanced architecture of Figure 6.10.

### 6.3.1. Experimental Setup on Data Augmentation

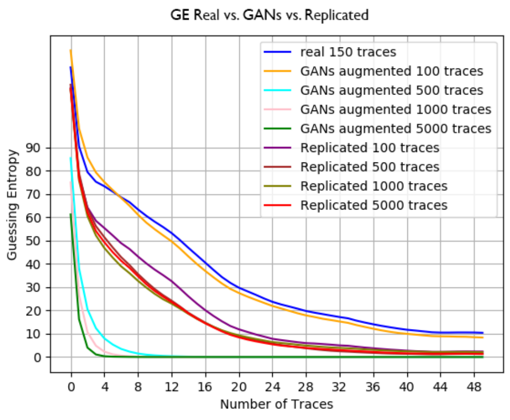
In order to demonstrate the performance of GANs in data augmentation, we have to choose first a dataset with a number of traces that is not large enough to train the profiling models to break the key but large enough to train the GANs. Under this case, it can be shown that GANs can use a rather small dataset and therefore augment it to a larger one that can be used for retrieving the key. In this project, the dataset with the most potent countermeasures is the desynchronized dataset with jitter 15, where we only need 50 traces for training a Random Forrest in order to retrieve the key. This means that that we need to use less than 50 traces for training the GANs, an insufficient number for training most of the deep learning models that usually require large datasets to perform well. For this reason, we use a large balanced dataset of 900 traces (100 traces for each class) for training the GANs, and we compare the GE performance of the augmented dataset with replicated datasets out of this small dataset. Lastly, we show the GE performance of the dataset produced by GANs vs. the datasets produced by using the SMOTE algorithm. In this way, we can investigate if the synthetic traces' ability to train the profiling model sufficiently remains the same when we augment a real dataset. Furthermore, we show that by finding a more potent architecture in the future, we will be able to perform a real case data augmentation scenario.

### 6.3.2. GE Comparison GANs vs Real vs Replicated dataset

In Figure 6.19 the GE performance of the profiling model trained on the real, the GANs (augmented by GANs), and the replicated datasets is presented. The difference between the two sub-Figures (a) and (b) is that in Figure (a), we use 75 real traces to perform augmentation, while in Figure (b), we use 150 real traces. In both experiments (a) and (b), we augment the real traces by adding 100 (gold line), 500 (aqua line), 1000 (pink line), and 5000 (green line) synthetic traces constructing four augmented datasets. Furthermore, we construct four replicated datasets, meaning that these datasets consist of multiplication of the original real traces. The sizes of these replicated datasets are 100 (purple line), 500(brown line), 1000 (dark green line), and 5000 (red line) traces as well. In both plots, it is noticeable that for large augmented datasets by GANs, we need only a few traces to break the key (less than four), while even for large replicated datasets, we need more than 40 traces. We can see the two datasets (100, 500 traces) perform better than the 75 and 150 real traces, but we can see that the GE performance does not keep growing when we use a large replicated dataset of 5000 traces. This is because the profiling model is over-fitted to the same real traces and consequently cannot manage to classify the attack traces correctly. Lastly, since the GE of the augmented datasets surpasses the replicated datasets, we can claim that our GAN model produces synthetic traces with a large variety, meaning that the GAN model avoided the mode collapse failure state.



(a) 75 real traces vs augmented traces by GANs vs replicated traces



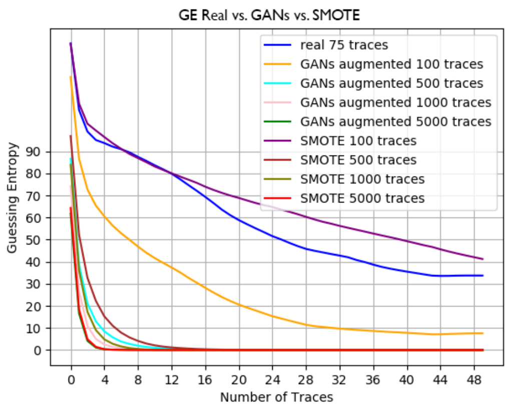
(b) 75 real traces vs augmented traces by GANs vs replicated traces

Figure 6.19: GE comparison between real, augmented by GANs and replicated traces when for real dataset the desynchronized dataset with jitter 15 is used

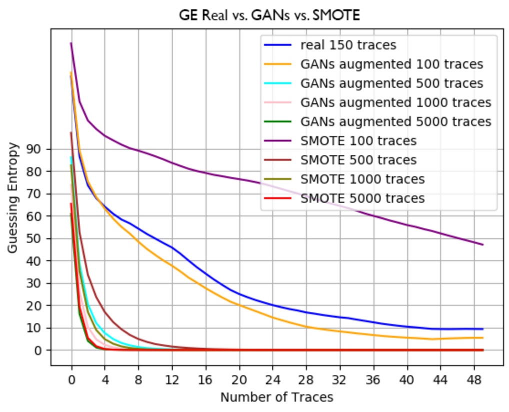
### 6.3.3. Guessing Entropy Comparison GANs vs SMOTE

The second experiment for the data augmentation investigation is a comparison between a balanced (augmented) dataset constructed by SMOTE with a binomially distributed augmented dataset by GANs.

In Figure 6.20 we can see this comparison for the desynchronized dataset of jitter 15 for two dataset sizes; we follow the same set-up again with the previous experiment as the first dataset consists of 75 real traces, while the second consists of 150 real traces. To experiment with the two data augmentation methods (GANs and SMOTE), we fit the SMOTE algorithm into the same dataset we used to train the GANs and randomly select the synthetic traces that we need to construct balanced augmented datasets. In both Figures 6.20a and 6.20b we can see the same trend that for small datasets, the GANs perform better than SMOTE while for larger datasets, the GANs and SMOTE GE curves seem to converge to the same value. We conclude that GANs perform slightly better than SMOTE in this data augmentation scenario, especially for small datasets.



(a) 75 real traces vs augmented traces by GANs vs SMOTEd traces



(b) 150 real traces vs augmented traces by GANs vs SMOTEd traces

Figure 6.20: GE comparison between real, augmented by GANs and SMOTEd traces when for real dataset the desynchronized dataset with jitter 15 is used

### 6.3.4. Discussion on Data Augmentation

In this section, we performed a set of experiments on data augmentation scenario by using two small datasets of 75 and 150 real traces, each augmented by GANs trained on a large dataset of 900 traces. We can notice that augmenting the dataset by using synthetic traces generated by GANs improved the profiling models' GE performance compared to the dataset replication method and balancing method (SMOTE). More specifically, when we trained the RF on the augmented dataset generated by GANs, we get better GE performance than the replicated datasets, which was expected as the replicated dataset contain the same information as the original real dataset. Furthermore, the GE performance is slightly improved when GANs were used in comparison to the balanced dataset generated by the SMOTE algorithm. In the experiments of Section 6.20, we compared the binomially distributed augmented dataset by GANs with the balanced augmented dataset by SMOTE. The reason we did not perform balancing with GANs and we kept its binomial distribution is based on the associated conclusions of Chapter 4, where for datasets desynchronized with jitter smaller than 100, the GE performance is similar regardless of their class distribution.

## 6.4. Conclusions

In this chapter, we used the pipeline 5.1 that introduced in Chapter 5 to perform a set of experiments in two architectures. The first shallow architecture containing fewer weights is able to get trained fast but is accurate only on the unprotected dataset. The second architecture is an enhanced deeper version of the first with a larger number of weights, capable of generating plausible synthetic traces for the desynchronized dataset of jitter 10 and 15. More specifically, in the first section (6.1) of this chapter, we show the performance of the shallow GANs on the unprotected dataset by comparing the PC and the GE of the synthetic traces with the real traces. Furthermore, we demonstrated the need for the Snapshot selection mechanism and the Feature Concatenation step, while in the second section (6.2), we showed the need for the architecture selection component of the pipeline as more potent architectures could generate plausible synthetic traces for more complicated datasets and we performed a dataset augmentation, and we show that GANs perform better than the dataset replication method and slightly better than the SMOTE algorithm.

On the other hand, for the two GANs architectures, we did not manage to perform a data augmentation under a realistic scenario meaning that we did not use a small dataset to train the GANs and we generate more GANs to improve the performance of the profiling model. The main reason that we did not manage to perform a data augmentation under this scenario is the fact that the most complex dataset consisted of countermeasures of jitter 15. Even if the proposed architectures were potent enough to generate traces trained on a more complex dataset like the desynchronized dataset of jitter 100 or a masked dataset in order to perform a dataset augmentation under a realistic scenario we could not use, the most accurate Snapshot Selection Mechanism as it is based on a pre-trained profiling model.

After taking all the above into account, we propose a single pipeline where the steps above can be combined. Furthermore, we analyze the pipeline's limits when we use a dataset with countermeasures, and we propose future solutions for a more generalized pipeline. The following diagram 5.1 presents a visual representation of this pipeline.

In the previous chapter, we addressed the limitations of the pipeline from a theoretical point of while now we address the limitation of the pipeline by taking into account the experiments of this chapter again.

In the following list, we address the limits and how this pipeline should be parameterized:

1. Window Selection: we choose a specific area of 700 features to train the profiling and the GAN models
   - When the unprotected dataset is used, the linear correlation between the power consumption (represented by the labels) and the features (time-samples) of the traces is large, and the Pearson's correlation coefficient can sufficiently indicate the information leakage windows
   - When we use a dataset with countermeasures, the linear correlations are getting hidden, meaning that other statistical methods should be used instead [5].
2. Architecture Selection: we carefully choose the Discriminator and the Generator's architecture, more specifically the number and the type of layers that should be used.
   - The handcrafted architecture search can provide more robust architectures but requires a lot of effort to find one.
   - We did not manage to construct robust architectures for the dataset with jitter larger than 15.
   - By using the unprotected dataset, we concluded to the specific architecture of subsection 5.2: the Discriminator consists only of 2 convolutional layers, powerful enough to discriminate between real and fake traces produced by the Generator. On the other hand, the Generator consists of one fully connected layer and two combinations of up-sampling and convolutional layers, expanding the input noise of 25 features valuable traces of 100 features.
   - In order to use a more complex dataset like the desynchronized dataset, we need to alter the architecture as the valuable patterns in the traces are now hidden. Consequently, a more potent Discriminator is required, which is described in Chapter 6.
3. GANs Training Phase: a phase that requires as the previous one serious tuning, meaning that multiple hyperparameters must be tuned in order to achieve high-quality traces
   - When the unprotected dataset is used, we can use as an evaluation method the pre-trained model of the Snapshot selection phase in order to classify the generated traces, also as it is explained in Section 5.4, the machine learning metrics must be studied in order to understand the behavior of the model.
   - Unfortunately, when a dataset with complex countermeasures has been implemented, the usage of a pre-trained model is no longer an option; for this reason, only the machine learning metric observation and Pearson's Correlation between trace-points and Power (PC) methods can be used to perform parameter optimization to the model.

4. Snapshot selection: a crucial component of the pipeline as the right choice of the snapshots can boost the performance of the GANs.
    - When the unprotected dataset is used, the two mechanisms are available; the first one studies the machine learning metrics while the second one uses a pre-trained profiling model.
    - When protected datasets are used, the only option of selecting the right snapshot is based on studying the machine learning metrics.
5. Feature Concatenation: a component that can be easily performed without any need for studying the complexity of the dataset.

# 7 Conclusions and Future Work

In this thesis project, we investigated the task of data augmentation on the SCA dataset and the possibility of constructing synthetic SCA datasets by using Generative Adversarial Networks (GANs) when the HW power model is used. In Chapter 4, we investigated the optimal class distribution that a dataset should follow to improve the SCA field's profiling models' performance. More specifically, we performed a set of experiments on three datasets; namely, the ASCAD unprotected, the ASCAD desynchronized with jitter of 100, and the masked ASCAD dataset. We concluded that we could choose between a balanced and a binomially distributed dataset for the first two datasets without expecting significant differences in the Guessing Entropy (GE) performance of the profiling models trained on these two distributions. In contrast, when stronger countermeasures are used, such as masking, the balanced dataset is preferred. The reason for performing such an investigation is based on the fact that with GANs, we can control the class distribution of the synthetic dataset and therefore the class distribution of the augmented datasets, meaning that the first research question that is made when such kind of problem is approached is: "How does the class distribution affect the performance of the profiling models".

In Chapter 5 we propose a pipeline 5.1 consisting of several components: Window Selection, Architecture Selection, GANs Training Phase, Snapshot Selection, and Feature Concatenation. With this pipeline, we managed to improve and simultaneously evaluate the performance of the GANs. Furthermore, in the same Chapter, the basic idea of the two neural architectures that are used in Chapter 6 is introduced.

Finally, in Chapter 6 two neural architectures are constructed and tuned by using the proposed pipeline of Chapter 5. For these two architectures, we perform a set of experiments; with the first one, we show the importance of the pipeline's component by constructing synthetic traces, while with the second one, we augment real datasets with synthetic traces and compare them with augmented dataset produced by using a data replication method and a balanced augmented dataset produced by using the SMOTE algorithm.

Lastly, in this Chapter, we answer the research question defined in Chapter 3, we address all the scientific contributions in the SCA field of this project, and then we talk about the future work that can be done in this project and the future applications where GANs can be used in the SCA field.

## 7.1. Reflection on Research Questions

1. **Research Question:** Does the profiling model's performance depends on the dataset's class distribution, and more specifically, should the augmented datasets follow a binomial or a uniform class distribution?

   **Answer on Research Question:** From the experiments of Chapter 4, we can see that for unprotected datasets or with desynchronized datasets the selection between the binomial and the uniform distribution does not play a significant role. However, we can notice that if a U-shaped distribution is used, the GE performance is getting low, meaning that we have to be careful about what class distribution we choose. When more potent countermeasures are used, such as masking, the overfitting phenomenon is so strong during the profiling models' training phase, and therefore, the balancing produces better results. In conclusion, we answer that the profiling models' performance depends on the class distribution of the dataset, and when a dataset with mild countermeasures is used, we can choose between a balanced and a binomial dataset. In contrast, for potent countermeasures, we can reduce the overfitting phenomena for the minority classes by using a balanced dataset, leading to the better overall accuracy and GE performance.

2. **Research Question:** Is it feasible for Generative Models such as complex Generative Adversarial Neural Networks(GANs) to generate valuable traces in order to construct synthetic datasets for Side-channel traces?

   **Answer on Research Question:** Yes the quality of the synthetic traces especially when the enhanced architecture was used was high almost indistinguishable from the real ones while the shallow architecture can generate synthetic traces in order to train fast a profiling model and effectively retrieve the key when the unprotected dataset is used.

   (a) **sub-Question:** Is it feasible to perform dataset augmentation with GANs in order to improve the performance of a side-channel analysis (SCA) attack when the training set is limited?

   **Answer on sub-Question:** According to our two proposed architectures the answer is: "No", as the two proposed architectures of the GANs seem to need more data than the profiling models, but it has be seen that GANs can generate high-quality synthetic traces when they are trained in large datasets.

   (b) **sub-Question:** Do GANs perform better than traditional augmentations techniques like the Synthetic Minority Oversampling Techniques (SMOTE) on SCA datasets?

   **Answer on sub-Question:** Better than trivial data augmentation methods like replication and sightly better than SMOTE.

3. **Research Question:** How can GANs be evaluated from a Machine Learning perspective in order to achieve a parameter optimization for these models by considering that visual representations of the generated SCA traces are not an option?

   **Answer on Research Question:** In order to perform parameter optimization and create more potent architectures for GANs the two Snapshot Selection Mechanism of the pipeline 5.1 are used namely, Snapshot Selection Mechanism based on the loss and the accuracy of the generator and the discriminator (ML-based) and a Snapshot Selection Mechanism based on a pre-trained profiling model, the Random Forest classifier (RF) in our case (RF-based). Furthermore, as an intermediate step of the GANs performance evaluation, the Pearson's Correlation between Trace Points and Power Comparison has been used. More specifically, we generated synthetic datasets, and we performed the Pearson's correlation coefficient between the features (time-samples) and the labels (HW). In this way, we expect to see that the synthetic traces leak the same way that real traces do. With these three methods (2 Snapshot Selection Mechanisms and the Pearson's Correlation between Trace Points and Power Comparison), we managed to observe the performance of the GANs (the GANs stability by looking at the variance of the loss and the accuracy) and to evaluate the quality of the synthetic traces (RF-based Snapshot Selection mechanism and Pearson's Correlation between Trace Points and Power Comparison between real and synthetic traces).

4. **Research Question:** Does the performance of the GANs depends on the complexity of the dataset, meaning that more countermeasures require deeper neural architectures?

   **Answer on Research Question:** Yes more complex datasets lead to datasets with less leakage, meaning that capturing interesting patterns of the data by using GANs is more difficult when shallow architectures have been used while the extended architecture can deal with desynchronized datasets of jitter 10 and 15 quite well.

## 7.2. Scientific Contributions

We investigated how the class distribution of the dataset affects the profiling models' performance on the SCA field when the Hamming Weight power model is used. We point out that the class distribution of the dataset plays a role in the profiling models' performance as when the dataset follows a U-shaped class distribution, the performance of the profiling models is low. On the other hand, we also point out that balanced datasets do not contribute more to the profiling models' performance than the dataset that follows the natural binomial distribution when unprotected datasets or desynchronized datasets are used. Lastly, when potent countermeasures are used like masking the balanced dataset are preferred, since by using the state of the profiling models (CNNs), the per-class accuracy is so low, and we prefer to push all classes to reach that threshold rather than achieving higher accuracy to the majority classes.

We proposed a pipeline that can be used to train fast GANs with limited computational power, which can generate plausible synthetic traces trained on an unprotected dataset by following a series of steps. Furthermore, we showed that with this pipeline, we could tune and construct deeper architectures that can generate high-quality synthetic traces even for datasets with countermeasures (desynchronized jitter 10 and 15), with the disadvantage this architecture is computationally more expensive than the shallow architecture. With the proposed pipeline, we did not manage to solve the data augmentation under a real scenario, but we showed that GANs could generate high-quality synthetic traces. Furthermore, we showed that enhanced architectures could generate traces for more complex datasets by capturing the traces' leakage information more effectively. Therefore we showed that more enhanced architectures could be proposed by using the proposed pipeline to solve a data augmentation task under real-life conditions.

Lastly, with the proposed pipeline, we provide a labeled (HW values) synthetic dataset that can be used for further experimentation in the future; one example for exploiting synthetic dataset generated by GANs is addressed in the Future Work Section 7.4 of this Chapter.

## 7.3. Limitations

The first requirement to solve the data augmentation problem is to train the GANs for a limited dataset that cannot efficiently train a profiling model. The second requirement is to train the GANs and then the profiling model with a dataset with countermeasures in order to show the crucial differences between the real dataset and the augmented dataset in the Guessing Entropy plot. In this project, we managed to train sufficiently the GANs with an unprotected dataset and a desynchronized dataset of 10 and 15 jitter but for a large dataset. Therefore, we cannot perform a real data augmentation scenario as the two proposed architectures can only generate plausible synthetic traces for desynchronized datasets with mild countermeasures (jitter of 10 and 15). In order to show big differences in the GE performance of the models, we have to generate plausible synthetic traces trained on a desynchronized dataset of jitter 100 or to used a tiny dataset to train the GANs, which is not an option even for the extended architecture.

Moreover, in the case of a real data augmentation scenario where the profiling model over-fits too much for a given dataset, we cannot use the Snapshot Selection Mechanism that is based on a pre-trained profiling model, meaning that the whole performance of the proposed pipeline depends on the Snapshot Selection Mechanism based on observing the Machine Learning metric of the GANs, something that requires expertise and is not very reliable.

Lastly, by observing the performances of the profiling models of Chapter 4 when the masking countermeasure is used, we can observe that the per-class accuracy is very low for almost every class and, therefore again, the use of the Snapshot Selection mechanism based on pre-trained profiling models is not an option and again we have to use the Snapshot Selection Mechanism based on observing the Machine Learning metric.

## 7.4. Future work

By using the proposed pipeline, deeper handcrafted architectures can be defined in the future. In Chapter 6 we showed the potential of a deeper architecture and how much higher the synthetic traces' quality is when used. For this reason, we believe that new deeper and handcrafted architectures can produce synthetic traces with countermeasures like jittering of 100 or even masking for a limited training dataset. Eventually, we expect that using the proposed pipeline in the future, we will be able to perform a real data augmentation scenario where the GANs require fewer traces to get trained sufficiently in comparison with the state-of-the-art profiling models.

We can think one step further and perform a Neural Architecture Search to find robust GANs architectures based on the proposed one of Chapter 5. Of course, a Neural Architecture Search requires an evaluation mechanism in order to produce valuable GANs architectures. According to the literature evaluating the SCA dataset's profiling models and according to this project, evaluating GANs with the SCA dataset is difficult. One possible solution to this problem would be to use the proposed pipeline 5 or some components like the Snapshot Selection Mechanism to evaluate the Neural Architecture Search's new architectures.

Furthermore, in the future, we can use the same pipeline to propose architectures for different power models such as the identity model with the limitation that we need to build a large number of GANs, namely 256 GANs one for each class where for each GANs a small amount of traces corresponds.

Synthetic datasets can be used in other applications rather than performing a data augmentation task. One example is to train the GANs on traces that belong to specific crypto-operations like an RSA or an AES. Furthermore, we can zoom more in the traces and not only look at what happens in the first round around the Sbox but to focus on the assembly instructions that are used for a crypto-operation and train the GANs with a different dataset that will include the information of each assembly instruction. In this way, we can build a set of generators that will work as simulators and produce synthetic traces that belong to specific assembly instructions. We expect these traces to be very clean with high correlation peaks meaning the complexity of these traces will be very low, and even with the shallow architecture, we generate high-quality traces. Therefore, by having at our disposal the assembly instructions, we can construct a set of generators that will produce synthetic traces based on assembly instruction to perform a real trace acquisition simulation without the need for hardware devices like oscilloscopes.

# Bibliography

[1] R. Aguiar and M. Collares-Pereira. “TAG: A time-dependent, autoregressive, Gaussian model for generating synthetic hourly radiation”. In: *Solar Energy* 49.3 (1992), pp. 167–174. ISSN: 0038-092X. DOI: https://doi.org/10.1016/0038-092X(92)90068-L. URL: https://www.sciencedirect.com/science/article/pii/0038092X9290068L.

[2] Martin Arjovsky, Soumith Chintala, and Léon Bottou. *Wasserstein GAN*. 2017. arXiv: 1701.07875 [stat.ML].

[3] C. Ashokkumar, R. P. Giri, and B. Menezes. “Highly Efficient Algorithms for AES Key Retrieval in Cache Access Attacks”. In: *2016 IEEE European Symposium on Security and Privacy (EuroS P)*. 2016, pp. 261–275. DOI: 10.1109/EuroSP.2016.29.

[4] Ryad Benadjila et al. “Deep learning for side-channel analysis and introduction to ASCAD database”. In: *Journal of Cryptographic Engineering* 10.2 (2020), pp. 163–188.

[5] Ryad Benadjila et al. “Study of deep learning techniques for side-channel analysis and introduction to ASCAD database”. In: *ANSSI, France & CEA, LETI, MINATEC Campus, France. Online verfügbar unter https://eprint. iacr. org/2018/053. pdf, zuletzt geprüft am* 22 (2018), p. 2018.

[6] David Berthelot et al. “MixMatch: A Holistic Approach to Semi-Supervised Learning”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc., 2019, pp. 5049–5059.

[7] Andrew Brock, Jeff Donahue, and Karen Simonyan. “Large scale gan training for high fidelity natural image synthesis”. In: *arXiv preprint arXiv:1809.11096* (2018).

[8] Eleonora Cagli, Cécile Dumas, and Emmanuel Prouff. “Convolutional Neural Networks with Data Augmentation Against Jitter-Based Countermeasures”. In: Aug. 2017, pp. 45–68. ISBN: 978-3-319-66786-7. DOI: 10.1007/978-3-319-66787-4_3.

[9] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. “Template Attacks”. In: *Cryptographic Hardware and Embedded Systems - CHES 2002*. Ed. by Burton S. Kaliski, çetin K. Koç, and Christof Paar. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 13–28. ISBN: 978-3-540-36400-9.

[10] Nitesh V Chawla et al. “SMOTE: synthetic minority over-sampling technique”. In: *Journal of artificial intelligence research* 16 (2002), pp. 321–357.

[11] Christophe Clavier, Jean-Sébastien Coron, and Nora Dabbous. “Differential Power Analysis in the Presence of Hardware Countermeasures”. In: *Cryptographic Hardware and Embedded Systems — CHES 2000*. Ed. by Çetin K. Koç and Christof Paar. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 252–263. ISBN: 978-3-540-44499-2.

[12] A. Creswell et al. “Generative Adversarial Networks: An Overview”. In: *IEEE Signal Processing Magazine* 35.1 (2018), pp. 53–65. DOI: 10.1109/MSP.2017.2765202.

[13] Chris Donahue, Julian McAuley, and Miller Puckette. “Adversarial audio synthesis”. In: *arXiv preprint arXiv:1802.04208* (2018).

[14] Gintare Karolina Dziugaite, Daniel M. Roy, and Zoubin Ghahramani. *Training generative neural networks via Maximum Mean Discrepancy optimization*. 2015. arXiv: 1505.03906 [stat.ML].

[15] William Fedus, Ian Goodfellow, and Andrew M. Dai. *MaskGAN: Better Text Generation via Filling in the______*. 2018. arXiv: 1801.07736 [stat.ML].

[16] Jie Feng et al. “Generative Adversarial Networks Based on Collaborative Learning and Attention Mechanism for Hyperspectral Image Classification”. In: *Remote Sensing* 12.7 (2020). ISSN: 2072-4292. DOI: 10.3390/rs12071149. URL: https://www.mdpi.com/2072-4292/12/7/1149.

[17] Thomas S Ferguson. *A Course in Game Theory*. World Scientific, 2020.

[18] Xinyu Gong et al. *AutoGAN: Neural Architecture Search for Generative Adversarial Networks*. 2019. arXiv: 1908.03835 [cs.CV].

[19] Ian Goodfellow et al. “Generative Adversarial Nets”. In: *Advances in Neural Information Processing Systems*. Ed. by Z. Ghahramani et al. Vol. 27. Curran Associates, Inc., 2014, pp. 2672–2680.

[20] Benjamin Hettwer, Stefan Gehrer, and Tim Güneysu. “Applications of machine learning techniques in side-channel attacks: A survey”. In: *Journal of Cryptographic Engineering* (2019), pp. 1–28.

[21] Martin Heusel et al. “GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium”. In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017, pp. 6626–6637. URL: https://proceedings.neurips.cc/paper/2017/file/8a1d694707eb0fefe65871369074926d-Paper.pdf.

[22] Gabriel Hospodar et al. “Machine learning in side-channel analysis: a first study”. In: *Journal of Cryptographic Engineering* 1.4 (Oct. 2011), p. 293. ISSN: 2190-8516. DOI: 10.1007/s13389-011-0023-x. URL: https://doi.org/10.1007/s13389-011-0023-x.

[23] Rui Huang et al. “Beyond face rotation: Global and local perception gan for photorealistic and identity preserving frontal view synthesis”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 2439–2448.

[24] Mehmet Sinan Inci et al. “Cache attacks enable bulk key recovery on the cloud”. In: *International Conference on Cryptographic Hardware and Embedded Systems*. Springer. 2016, pp. 368–388.

[25] Gorka Irazoqui et al. “Wait a minute! A fast, Cross-VM attack on AES”. In: *International Workshop on Recent Advances in Intrusion Detection*. Springer. 2014, pp. 299–319.

[26] Phillip Isola et al. “Image-to-image translation with conditional adversarial networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1125–1134.

[27] A. G. Ivakhnenko. “Polynomial Theory of Complex Systems”. In: *IEEE Transactions on Systems, Man, and Cybernetics* SMC-1.4 (1971), pp. 364–378. DOI: 10.1109/TSMC.1971.4308320.

[28] Nikolay Jetchev, Urs Bergmann, and Roland Vollgraf. *Texture Synthesis with Spatial Generative Adversarial Networks*. 2017. arXiv: 1611.08207 [cs.CV].

[29] Tero Karras et al. “Analyzing and Improving the Image Quality of StyleGAN”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2020.

[30] Tero Karras et al. *Progressive Growing of GANs for Improved Quality, Stability, and Variation*. 2018. arXiv: 1710.10196 [cs.NE].

[31] Tero Karras et al. “Progressive growing of gans for improved quality, stability, and variation”. In: *arXiv preprint arXiv:1710.10196* (2017).

[32] Paul Kocher, Joshua Jaffe, and Benjamin Jun. “Differential Power Analysis”. In: *Advances in Cryptology — CRYPTO’ 99*. Ed. by Michael Wiener. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 388–397. ISBN: 978-3-540-48405-9.

[33] Naveen Kodali et al. “On convergence and stability of gans”. In: *arXiv preprint arXiv:1705.07215* (2017).

[34] Yann LeCun et al. “Comparison of learning algorithms for handwritten digit recognition”. In: *International conference on artificial neural networks*. Vol. 60. Perth, Australia. 1995, pp. 53–60.

[35] Royson Lee, Stylianos I. Venieris, and Nicholas D. Lane. *Neural Enhancement in Content Delivery Systems: The State-of-the-Art and Future Directions*. 2020. arXiv: 2010.05838 [cs.CV].

[36] Royson Lee et al. “Journey Towards Tiny Perceptual Super-Resolution”. In: *Computer Vision – ECCV 2020*. Ed. by Andrea Vedaldi et al. Cham: Springer International Publishing, 2020, pp. 85–102. ISBN: 978-3-030-58574-7.

[37] Liran Lerman et al. “Template attacks vs. machine learning revisited (and the curse of dimensionality in side-channel analysis)”. In: *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer. 2015, pp. 20–33.

[38] Dan Li et al. *MAD-GAN: Multivariate Anomaly Detection for Time Series Data with Generative Adversarial Networks*. 2019. arXiv: 1901.04997 [cs.LG].

[39] Ming-Yu Liu and Oncel Tuzel. “Coupled Generative Adversarial Networks”. In: *Advances in Neural Information Processing Systems*. Ed. by D. Lee et al. Vol. 29. Curran Associates, Inc., 2016, pp. 469–477.

[40] Liqian Ma et al. “Pose Guided Person Image Generation”. In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017, pp. 406–416.

[41] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. “Breaking Cryptographic Implementations Using Deep Learning Techniques”. In: Dec. 2016, pp. 3–26. ISBN: 978-3-319-49444-9. DOI: 10.1007/978-3-319-49445-6_1.

[42] Rita Mayer-Sommer. “Smartly Analyzing the Simplicity and the Power of Simple Power Analysis on Smartcards”. In: *Cryptographic Hardware and Embedded Systems — CHES 2000*. Ed. by Çetin K. Koç and Christof Paar. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 78–92. ISBN: 978-3-540-44499-2.

[43] Panayotis Mertikopoulos, Christos H. Papadimitriou, and Georgios Piliouras. “Cycles in adversarial regularized learning”. In: *CoRR* abs/1709.02738 (2017). arXiv: 1709.02738. URL: http://arxiv.org/abs/1709.02738.

[44] Mehdi Mirza and Simon Osindero. *Conditional Generative Adversarial Nets*. 2014. arXiv: 1411.1784 [cs.LG].

[45] M. Nassar et al. “RSM: A small and fast countermeasure for AES, secure against 1st and 2nd-order zero-offset SCAs”. In: *2012 Design, Automation Test in Europe Conference Exhibition (DATE)*. 2012, pp. 1173–1178. DOI: 10.1109/DATE.2012.6176671.

[46] Guilherme Perin, Ileana Buhan, and Stjepan Picek. “Learning when to stop: a mutual information approach to fight overfitting in profiled side-channel analysis.” In: ().

[47] S. Picek et al. “Side-channel analysis and machine learning: A practical perspective”. In: *2017 International Joint Conference on Neural Networks (IJCNN)*. 2017, pp. 4095–4102. DOI: 10.1109/IJCNN.2017.7966373.

[48] Stjepan Picek et al. “The Curse of Class Imbalance and Conflicting Metrics with Machine Learning for Side-channel Evaluations”. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2019.1 (Nov. 2018), pp. 209–237. DOI: 10.13154/tches.v2019.i1.209-237. URL: https://tches.iacr.org/index.php/TCHES/article/view/7339.

[49] Alec Radford, Luke Metz, and Soumith Chintala. *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*. 2016. arXiv: 1511.06434 [cs.LG].

[50] Alec Radford, Luke Metz, and Soumith Chintala. “Unsupervised representation learning with deep convolutional generative adversarial networks”. In: *arXiv preprint arXiv:1511.06434* (2015).

[51] Tim Salimans et al. “Improved Techniques for Training GANs”. In: *Advances in Neural Information Processing Systems*. Ed. by D. Lee et al. Vol. 29. Curran Associates, Inc., 2016, pp. 2234–2242.

[52] Kaleb E Smith and Anthony O Smith. “Conditional GAN for timeseries generation”. In: *arXiv preprint arXiv:2006.16477* (2020).

[53] François-Xavier Standaert. “How (not) to use welch’s t-test in side-channel security evaluations”. In: *International Conference on Smart Card Research and Advanced Applications*. Springer. 2018, pp. 65–79.

[54] François-Xavier Standaert, Tal G. Malkin, and Moti Yung. “A Unified Framework for the Analysis of Side-Channel Key Recovery Attacks”. In: *Advances in Cryptology - EUROCRYPT 2009*. Ed. by Antoine Joux. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 443–461. ISBN: 978-3-642-01001-9.

[55] François-Xavier Standaert, Tal G. Malkin, and Moti Yung. “A Unified Framework for the Analysis of Side-Channel Key Recovery Attacks”. In: *Advances in Cryptology - EUROCRYPT 2009*. Ed. by Antoine Joux. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 443–461. ISBN: 978-3-642-01001-9.

[56] Ping Wang et al. *Enhancing the Performance of Practical Profiling Side-Channel Attacks Using Conditional Generative Adversarial Networks*. 2020. arXiv: 2007.05285 [cs.CR].

[57] Jerry Wei et al. *Generative Image Translation for Data Augmentation in Colorectal Histopathology Images*. 2019. arXiv: 1910.05827 [eess.IV].

[58] C. Yang et al. "High-Resolution Image Inpainting Using Multi-scale Neural Patch Synthesis". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA: IEEE Computer Society, July 2017, pp. 4076–4084. DOI: 10.1109/CVPR.2017.434. URL: https://doi.ieeecomputersociety.org/10.1109/CVPR.2017.434.

[59] Yuval Yarom and Katrina Falkner. "FLUSH+ RELOAD: a high resolution, low noise, L3 cache side-channel attack". In: *23rd {USENIX} Security Symposium ({USENIX} Security 14)*. 2014, pp. 719–732.

[60] Jinsung Yoon, Daniel Jarrett, and Mihaela van der Schaar. "Time-series Generative Adversarial Networks". In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc., 2019, pp. 5508–5518. URL: https://proceedings.neurips.cc/paper/2019/file/c9efe5f26cd17ba6216bbe2a7d26d490-Paper.pdf.

[61] Yingxian Zheng et al. "How to Compare Selections of Points of Interest for Side-Channel Distinguishers in Practice?" In: *Information and Communications Security*. Ed. by Lucas C. K. Hui et al. Cham: Springer International Publishing, 2015, pp. 200–214. ISBN: 978-3-319-21966-0.

[62] Jun-Yan Zhu et al. "Unpaired image-to-image translation using cycle-consistent adversarial networks". In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2223–2232.

[63] Barret Zoph and Quoc V. Le. *Neural Architecture Search with Reinforcement Learning*. 2017. arXiv: 1611.01578 [cs.LG].

# A
# Machine Learning Metrics and Heatmaps

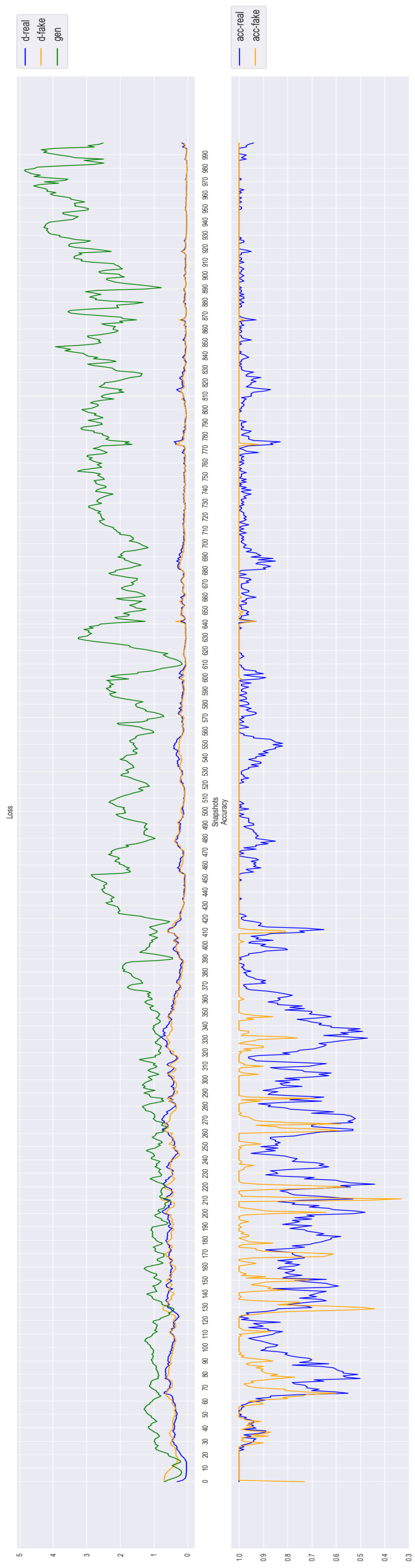


Figure A.1: class 0 Machine Learning metrics

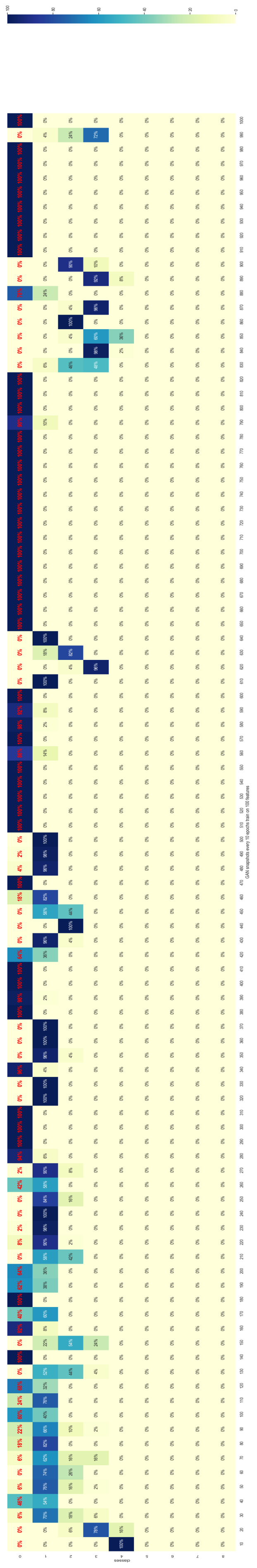


Figure A.2: class 0 heatmap

Figure A.3: (a): ML metrics for GANs on class 0 on Unprotected dataset (b): Classification prediction of the pre-trained profiling model (RF) on synthetic traces for class 0 on Unprotected dataset

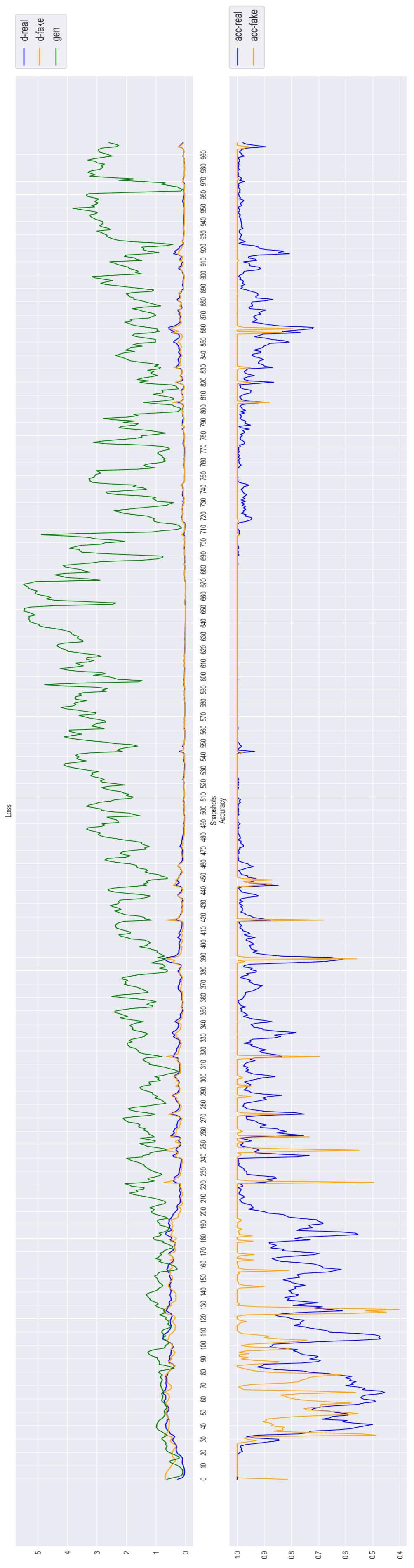


Figure A.4: class 1 Machine Learning metrics

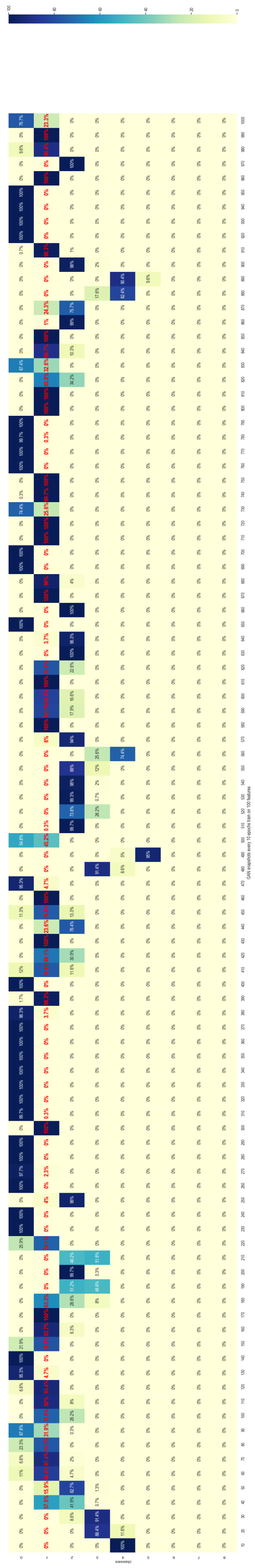


Figure A.5: class 1 heatmap

Figure A.6: (a): ML metrics for GANs on class 1 on Unprotected dataset (b): Classification prediction of the pre-trained profiling model (RF) on synthetic traces for class 1 on Unprotected dataset

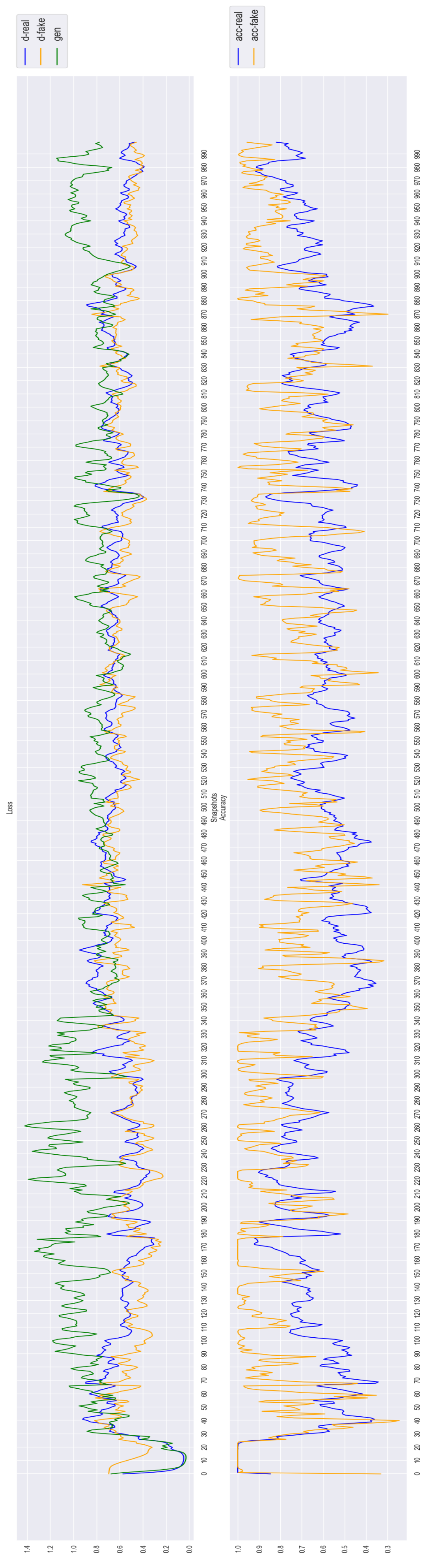


Figure A.7: class 2 Machine Learning metrics

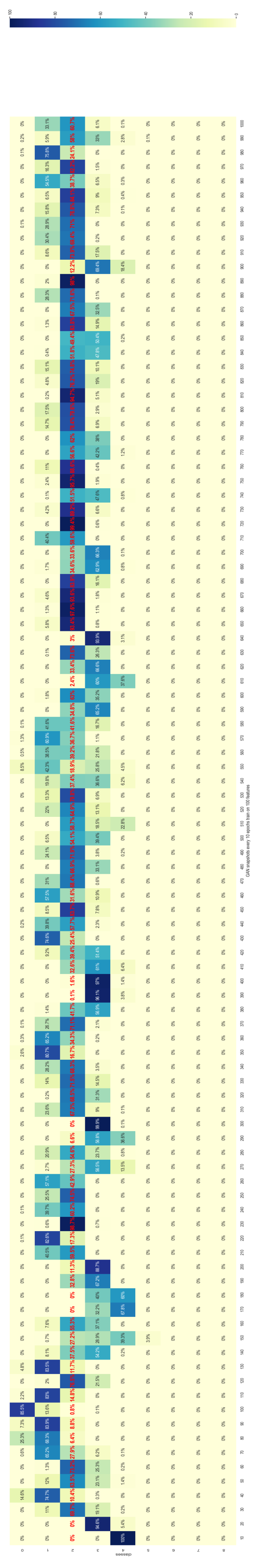


Figure A.8: class 2 heatmap

Figure A.9: (a): ML metrics for GANs on class 2 on Unprotected dataset (b): Classification prediction of the pre-trained profiling model (RF) on synthetic traces for class 2 on Unprotected dataset

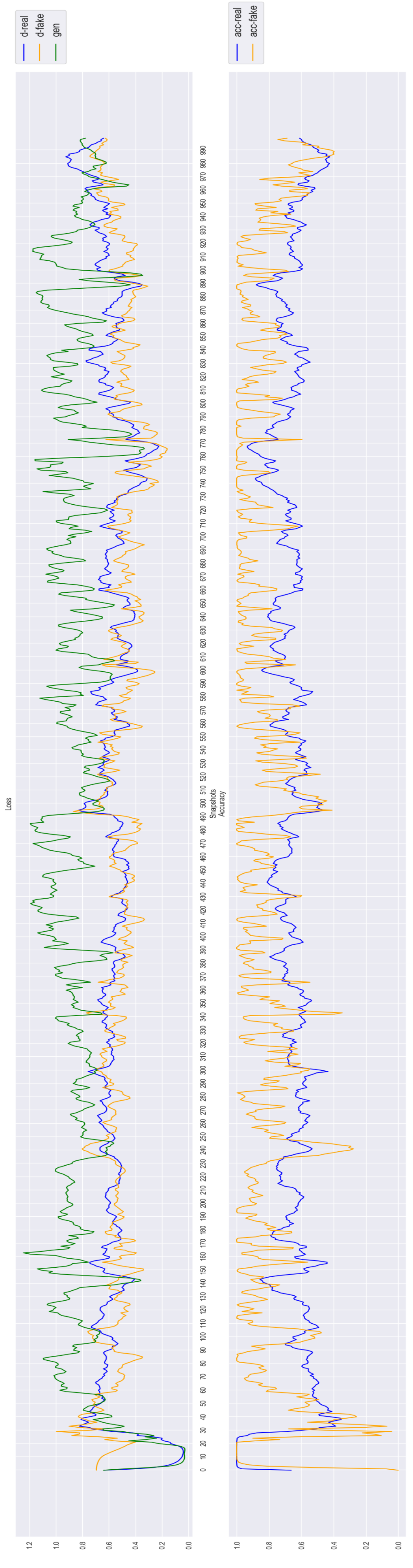


Figure A.10: class 3 Machine Learning metrics

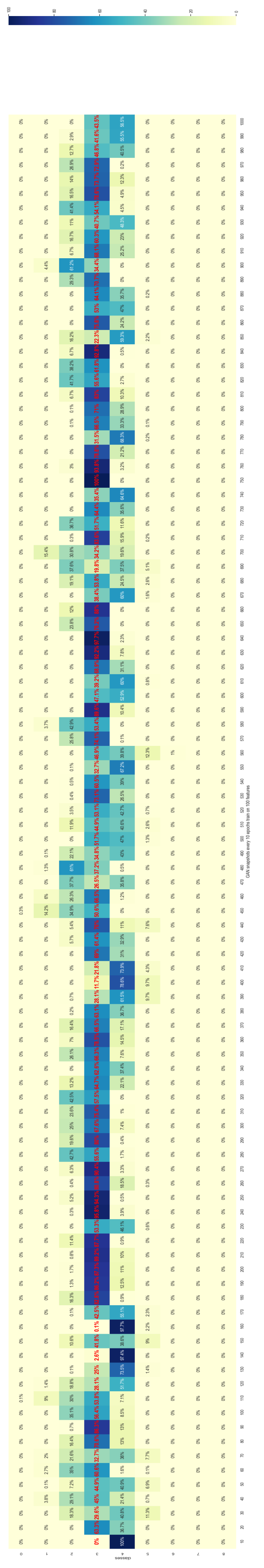


Figure A.11: class 3 heatmap

Figure A.12: (a): ML metrics for GANs on class 3 on Unprotected dataset (b): Classification prediction of the pre-trained profiling model (RF) on synthetic traces for class 3 on Unprotected dataset

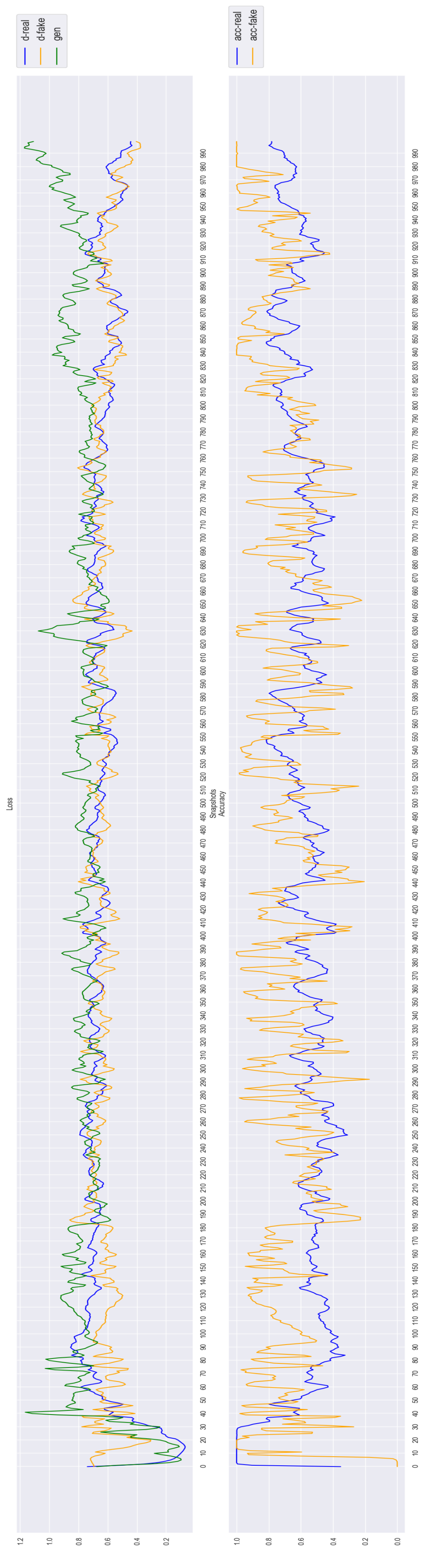


Figure A.13: class 4 Machine Learning metrics

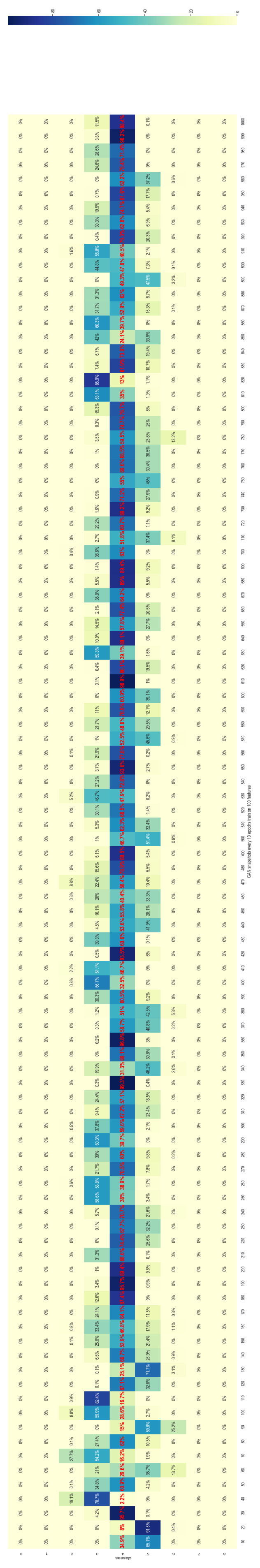


Figure A.14: class 4 heatmap

Figure A.15: (a): ML metrics for GANs on class 4 on Unprotected dataset (b): Classification prediction of the pre-trained profiling model (RF) on synthetic traces for class 4 on Unprotected dataset

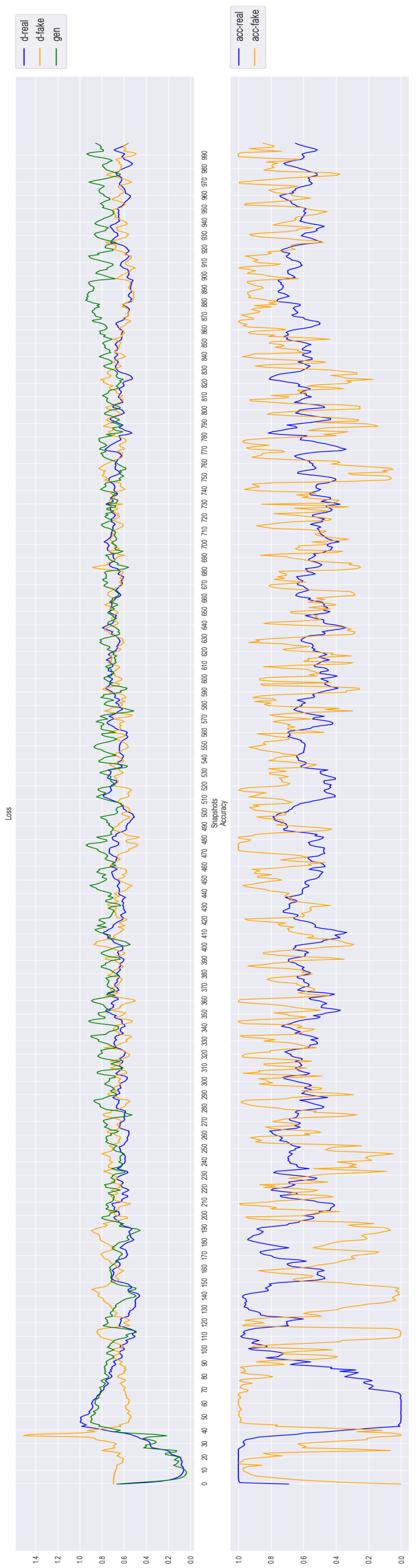


Figure A.16: class 5 Machine Learning metrics

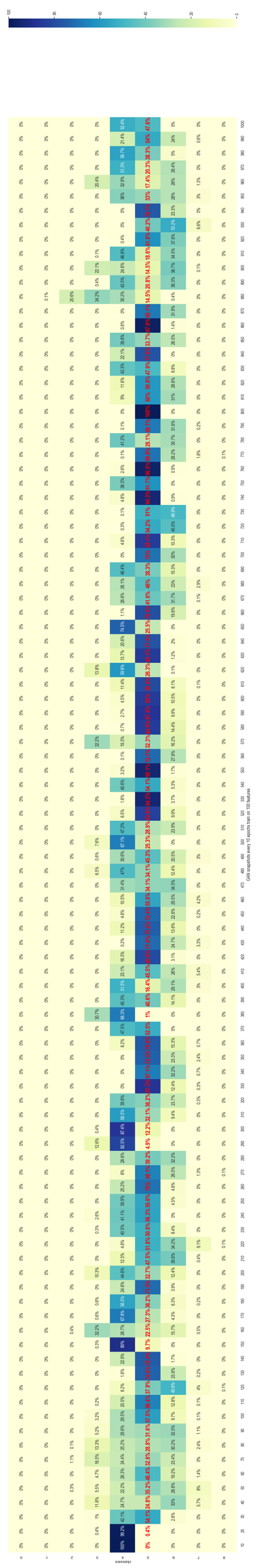


Figure A.17: class 5 heatmap

Figure A.18: (a): ML metrics for GANs on class 5 on Unprotected dataset (b): Classification prediction of the pre-trained profiling model (RF) on synthetic traces for class 5 on Unprotected dataset

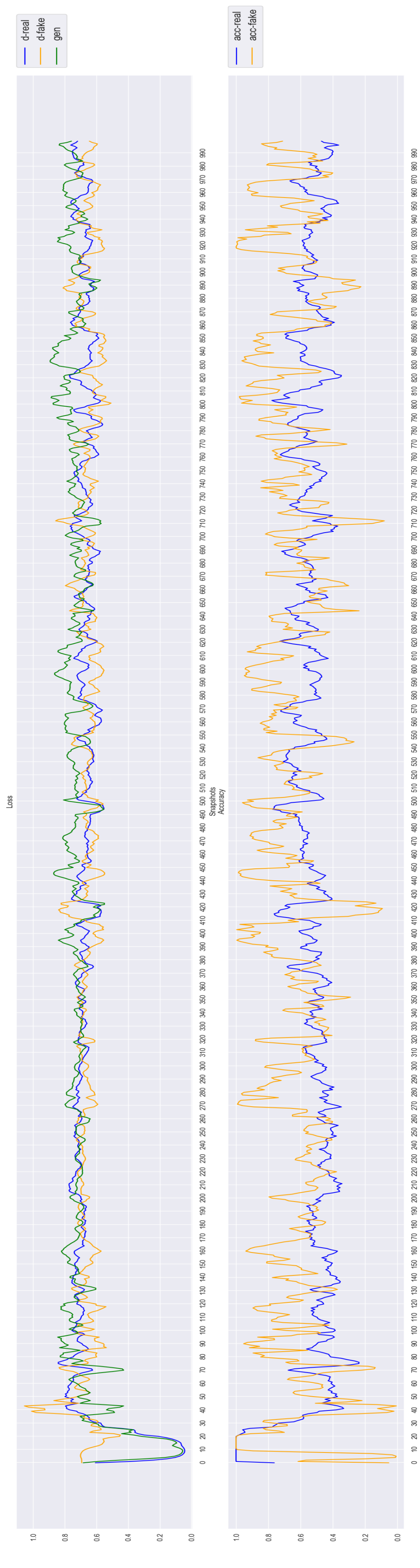


Figure A.19: class 6 Machine Learning metrics

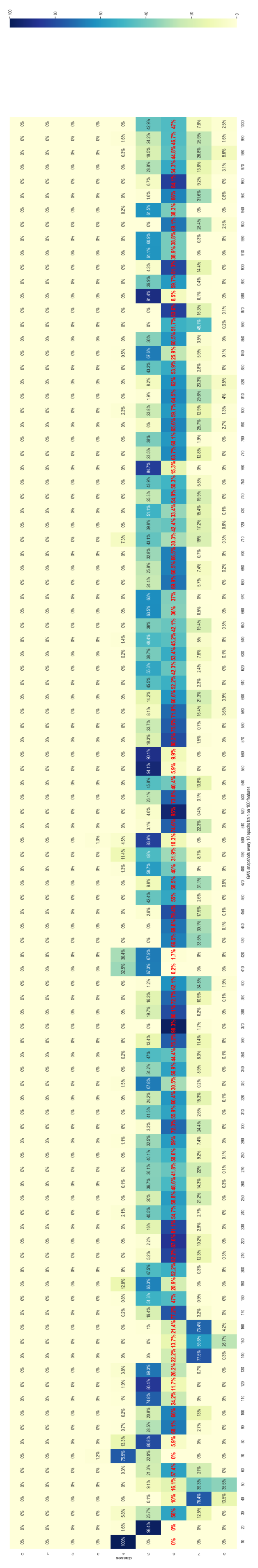


Figure A.20: class 6 heatmap

Figure A.21: (a): ML metrics for GANs on class 6 on Unprotected dataset (b): Classification prediction of the pre-trained profiling model (RF) on synthetic traces for class 6 on Unprotected dataset

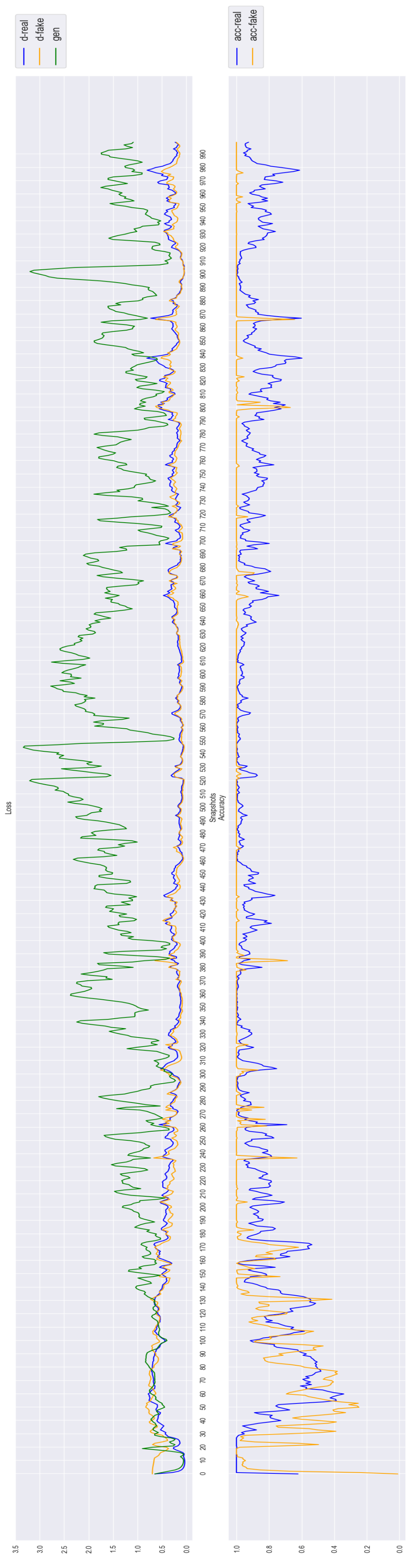


Figure A.22: class 7 Machine Learning metrics

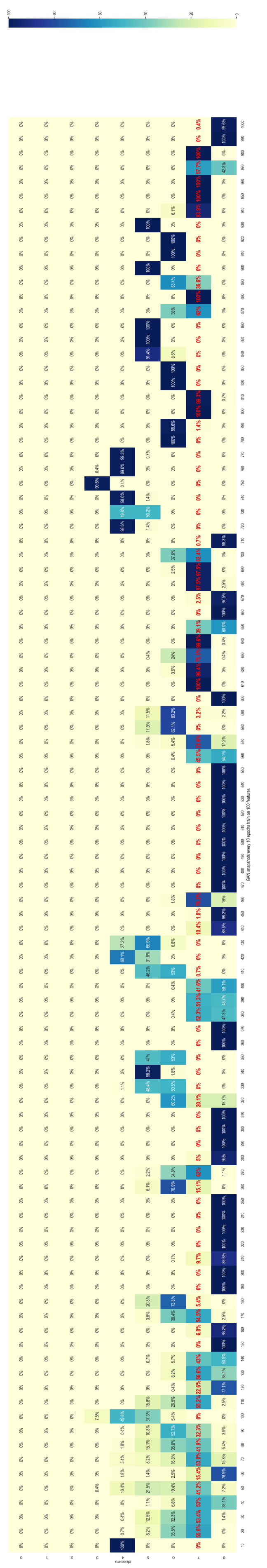


Figure A.23: class 7 heatmap

Figure A.24: (a): ML metrics for GANs on class 6 on Unprotected dataset (b): Classification prediction of the pre-trained profiling model (RF) on synthetic traces for class 7 on Unprotected dataset

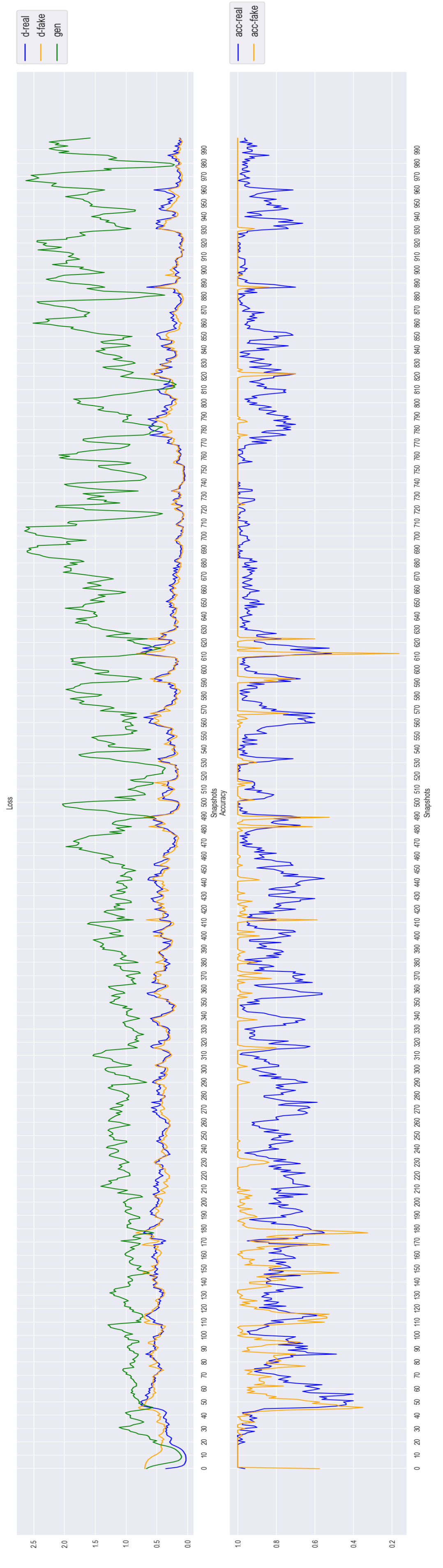


Figure A.25: class 8 Machine Learning metrics

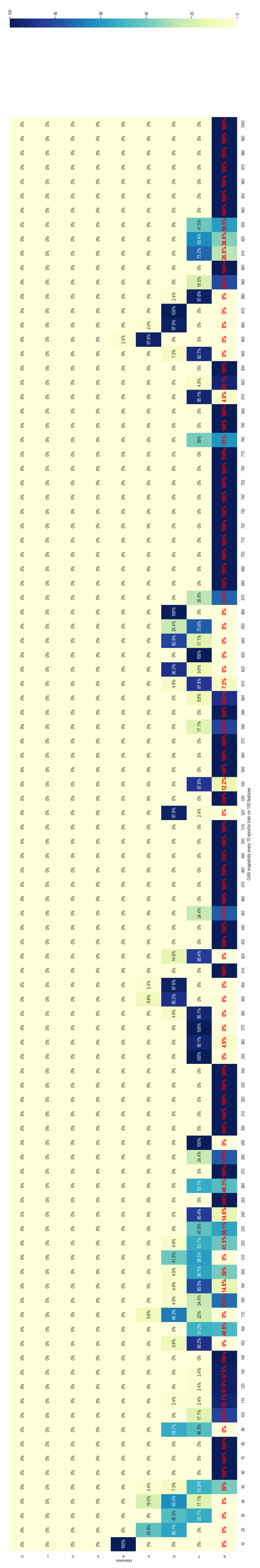


Figure A.26: class 8 heatmap

Figure A.27: (a): ML metrics for GANs on class 8 on Unprotected dataset (b): Classification prediction of the pre-trained profiling model (RF) on synthetic traces for class 8 on Unprotected dataset