Final Master Thesis

MASTER'S DEGREE IN INDUSTRIAL ENGINEERING

Generative Adversarial Network based Machine for Fake Data Generation

MEMORY

Author : Esteban Piacentino

Supervisor : Prof. Dr. Cecilio Angulo

Date : September, 2019

Escuela Técnica Superior

de Ingeniería Industrial de Barcelona

UNIVERSITAT POLITÈCNICA DE CATALUNYA

BARCELONATECH

Abstract

This paper introduces a first approach on using Generative Adversarial Networks (GANs) for the generation of fake data, with the objective of anonymizing patients information in the health sector. This is intended to create valuable data that can be used both, in educational and research areas, while avoiding the risk of a sensitive data leakage. For this purpose, firstly a thorough research on GAN's state of the art and available databases has been developed. The outcome of the project is a GAN system prototype adapted to generate raw data that imitates samples such as users variable status on hypothyroidism or a cardiogram report. The performance of this prototype has been checked and satisfactory results have been obtained for this first phase. Moreover, a novel research pathway has been opened so further research can be developed.

Contents

List of Figures

List of Tables

Acronyms

AI Artificial Intelligence

CNN Convolutional Neural Network

CT Computed Tomography

ECG Electro Cardiogram

ETSEIB Escola Tècnica Superior d'Enginyeria Industrial de Barcelona

FCN Fully Connected Network

GAN Generative Adversarial Network

MRI Magnetic Resonance Image

NLP Natural Language Processing

NN Neural Network

TFM Treball Fi de Màster (Final Master Thesis)

UPC Universitat Politècnica de Catalunya

Chapter 1

Introduction

This chapter will expose the motivation and aims of the project along with the formulation of the main problem that will be approached along this dissertation.

1.1 Motivation

The motivation of this master thesis comes from the recent arrival of Generative Adversarial Networks (GANs) and the innovative possibilities these are opening in many different fields. GAN algorithms have arisen in 2014 [1] and, since then, have been highlighted as potential alternatives for data augmentation and missing data problems, among others, due to their outstanding capabilities on generating realistic data instances, specially images.

To date, many researchers keep studying their capabilities and expanding their potential application areas, which gives a positive outlook to this technology. In particular, a question raised has leaded to this project, which is about the feasibility of using GAN systems to generate fake data, not necessarily images, that imitates the attributes of a private data-set. If possible, this generated machine would be a very useful tool as it will enable unlimited similar-to-the-original data without compromising the privacy of the original elements, avoiding any potential leakage risk of sensitive information. The applications of this tool, as it will be seen later on, could range from educational purposes to scientific simulations and investigations, as sensitive data from any field could be available without a risk of private data leakage.

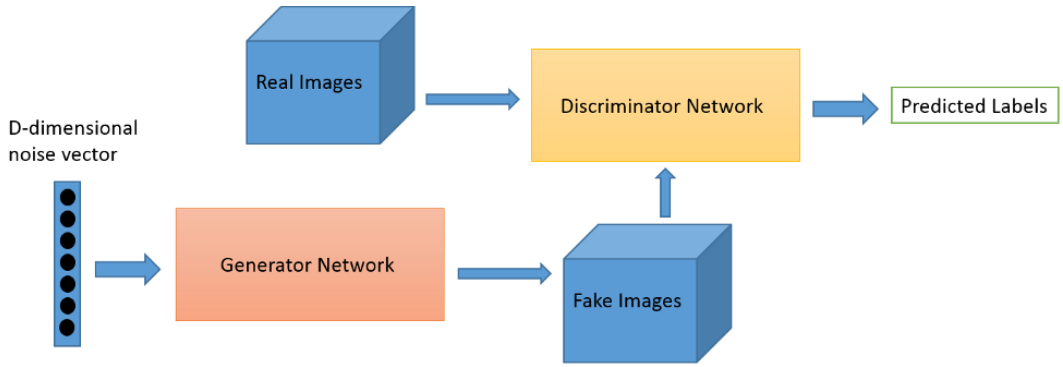


Figure 1.1: Basic GAN architecture. Source: `towardsdatascience.com/generative-adversarial-networks-gans-a-beginners-guide-5b38eceece24`

1.2 Problem Formulation

In order to answer the question exposed in the previous section, first of all it would be necessary to understand how a GAN works. Next, a thorough listing of general and specific objectives of the project will be feasible.

1.2.1 What is a GAN

Generative Adversarial Networks (GANs) are systems based on a min-max strategy where two algorithms are confronted: one algorithm generates data (the generator) and the other discriminates between fake and real data (the discriminator)

The generator's objective is to maximize the discriminator error while the discriminator wants to minimize it. This is an iterative process that ends when the discriminator error is 0.5, meaning that it fails 50% of the times, the baseline error in bi-classification. We can think of a GAN as a "Cat and Mouse Game" between a cop and a money counterfeiter [2], where the counterfeiter (the generator) tries to fool the cop (the discriminator) creating an endless loop where both players keep improving themselves by pure competition. Figure 1.1 shows a basic GAN system.

As described, in order to generate a fake image, we always need a source of "creativity" that in this case comes from a random noise vector (seed). On the other side, in order to be able to discriminate between real and fake images (so that the discriminator model can send to the generator model what is doing wrong) a database of real images is needed.

Hence, the objective function of the complete network is the following:

$$\min_G \max_D V(D,G) = E_{x\sim P_{data}(x)}[\log D(x)] + E_{z\sim P_z(z)}[\log(1 - D(G(z)))] \tag{1.1}$$

This expression represents value (V), which is a function of both, discriminator D and generator G. The goal is to maximize the discriminator (D) loss and minimize the generator (G) loss. Value V is the sum of expected log likelihood for real and generated data. Likelihoods (probabilities) are the discriminator outputs for real or generated images. Note that the discriminator output for a generated image is subtracted from 1 before taking the log. Maximizing the resulting values leads to optimization of the discriminator parameters such that it learns to correctly identify both real and fake data.

It is also necessary to explain that:

- P_{data}: Represents the distribution of real data.
- P_z: Represents the distribution of noise (usually a Gaussian distribution) from which we can generate a fake image.
- x and z: Represent the samples from each corresponding space.
- E_x and E_z: Represent the expected log likelihood from the different outputs of both real and generated images.
- D function outputs a real number ranged between 0 and 1 representing the probability for data being real (1) or fake (0). On the other hand, G function outputs a generated sample or instance.

Besides, in order to train generator and discriminator, Errors on their outputs are propagated back into the models. These errors are propagated as gradients of the following loss functions.

Update rule for the discriminator:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} [\log D(x^{(i)}) + \log(1 - D(G(z^{(i)})))] \tag{1.2}$$

Update rule for the generator:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(1 - D(G(z^{(i)}))) \tag{1.3}$$

where m represents the total number of samples tested in batch before updating both models, and θ_d and θ_g represents the weights of each model. It is worth to note that on following chapters of this project we will be referring to these errors that we are calculanting the gradient as *losses*.

As a GAN is just a system structure, the choice of the elements to compose this system (generator and discriminator) are up to the user. In this case, it is going to be used the most popular option: using a convolutional neural network (CNN) for the discriminator and a transposed CNN for the generator. In order to keep this project focused on the GAN structure, CNNs will not be explained in detail.

1.2.2 GAN's Iterative Steps

Now that purpose, elements and objective function have been described, the steps of a training process will be enumerated. Every loop of this training process is called an *epoch*. Inside this epoch, all real samples, usually images, available will be processed in batches: each batch is a random subset of a fixed size (called batch size) from the real samples data-set. After each batch, errors from the discrepancy between the discriminator and the generator outputs and their expected values are back-propagated to the models and, therefore, used to train the models. Steps inside each batch are:

1. Batch real images are feed them into the discriminator.
2. Errors from the discriminator output are calculated knowing that, ideally, these outputs should be all 1's as these images are all real.
3. Batch fake images are generated with batch noise vectors.
4. Batch fake images are feed into the discriminator.
5. Errors from the discriminator output are calculated knowing that, ideally, these outputs should be all 0's as these images are all fake.
6. Both errors calculated from the discriminator are now propagated to its model.

7. Steps number 3 and 4 are repeated with new noise vectors.

8. This time, Errors are calculated for the generator knowing that, ideally, all outputs from the discriminator should be 1 as these images should perfectly imitate the real ones.

9. These errors are now propagated to the generator's model.

10. Start again with another batch from this epoch.

As it will be shown in the following chapter, these systems can increase its complexity by adding new elements and features to the basic structure. The most common extra element is an encoder (usually another CNN) to feed the generator not with a random noise vector but with another source of information, allowing the system to generate real-like images from a descriptive text or another image, for example.

Another factor that adds complexity to the system is the need of fine-tuning configurations for the different elements of a GAN in order to ease its training and optimization process.

1.3 General Objectives of the Thesis

As a first approach, this project aims primarily to deepen into the applications and limitations of GANs around fake data generation. This entails the necessity of accomplishing the following elements:

1. Exploring the state of the art of GANs.

2. Carrying out a deep study of the available datasets and its attributes.

3. Developing a system based on GAN for the creation of believable new data from different data-sets.

4. Carrying out a proof of concept by using this system on some data-sets and reporting the obtained results.

1.4 Specific Objectives of the Thesis

By breaking down the general objectives previously exposed, the specific objectives for each of the parts are listed below.

1. **State of the Art**
    - Understanding the main areas in which GANs are being used.
    - Applicability for this project.
2. **Available Databases**
    - Getting familiarized with the sources and types of variables a GAN system should work with.
    - Observe the quantity of samples available in each of the explored databases.
3. **Developing a GAN-based System**
    - Getting familiarized with the tools used to code and execute a GAN system.
    - Getting familiarized with the application of both CNN and Fully Connected Networks (FCNs) for the GAN system.
    - Code a basic GAN system to generate fake data.
    - Modify the system to correctly read each of the available databases.
    - Modify the GAN system to allow the usage of raw data instead of image data.
4. **Proof of Concept**
    - Pre-process each data-set for its correct manipulation.
    - Train a GAN system for each of the data-sets and save the output results.
    - Analyze the results and define the next steps of the investigation.

1.5 Project Scope

Being this project a very first approach on using GANs for the generation of fake data in the health domain, avoiding as far as possible images as samples, is very important to clarify from the beginning which is the scope of the project to avoid covering more topics than the essentially needed:

This master thesis intends to open new lines of investigation for further studies on the matter about data-sets in the health domain. Hence, it does not aspire to have a final solution to a problem, but rather a first evidence of what this technology is capable of doing. Furthermore, as it has been explained in Section 1.2.1, this project will not cover the basic operating principles of the artificial neural networks used for the GAN system as it is not part of the project aims.

On the chronological side, this master thesis will be carried out during the second semester of 2018/2019.

Chapter 2

State of the Art

In order to understand what is the settling level of this technology and how is the market orienting it, during this chapter the most common applications of GANs will be revised. In order to do so, these applications will be grouped in 4 different types according to the problem they are trying to solve. These four groups are:

- Data Augmentation
- Data Anonymization
- Missing Data
- Cross-Domain Transfer GANs

2.1 Data Augmentation

Data augmentation refers to a technique for extracting the most amount of information from a database in order to get better results in a training task. In computer vision, this has usually been performed by rotating, flipping, cropping the original images in order to obtain a bigger data-set to train an algorithm. An example of these basic data augmentation techniques can be observer in Figure 2.1.

As a common problem in machine learning, data augmentation systems have been around for a while. In this case, some Generative Adversarial Network systems have been

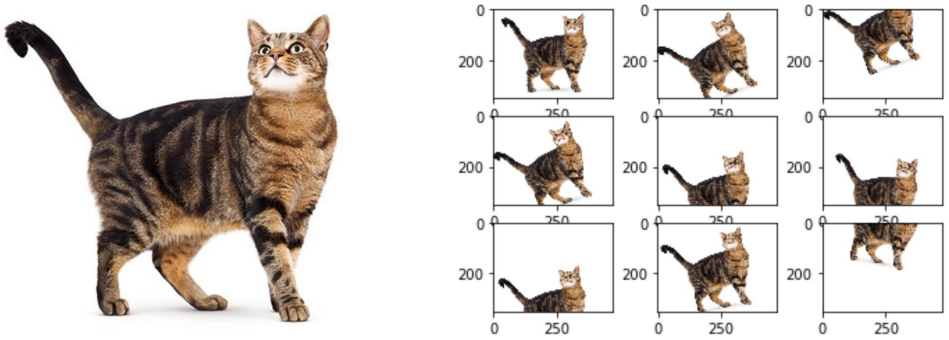


Figure 2.1: Example of basic data augmentation applied to an image. Source: `https://towardsdatascience.com/image-augmentation-for-deep-learning-histogram-equalization-a71387f609b2`

designed to generate new samples of a data base. Below are listed three articles that not only explore with success the usability of GANs for data augmentation, but also serve to present some of the main topics of this thesis:

GAN Augmentation: Augmenting Training Data using Generative Adversarial Networks by Christopher Bowles et al. [3], explores how to solve the lack of availability of large, labelled data-sets in medical imaging through GANs. This technique can offer a novel way to unlock additional information from a dataset by generating synthetic samples with the appearance of real images.

Medical Image Synthesis for Data Augmentation and Anonymization Using Generative Adversarial Networks by Hoo-Chang Shin et al. [4], offers a potential solution to two of the largest challenges facing machine learning in medical imaging, namely the small incidence of pathological findings, and the restrictions around sharing of patient data. The last problem will be brought out in other following articles.

Generative adversarial networks for data augmentation in machine fault diagnosis by Siyu Shao, Pu Wang and Ruqiang Yan [5], where data augmentation techniques through GANs are revisited about a different topic: learning from mechanical sensor signals for further applications in machine fault diagnosis.

2.2 Data Anonymization

Usually in medical environments, data privacy is a critical aspect when using patients databases. As an unchangeable information, if a patient's ADN or medical history gets leaked, there is no way to reverse this action unlike a user password leakage, where a simple password reset can resolve the problem. This is why anonymizing techniques has been very important in data processing and analysis problems.

As it has been already seen in the previous section, both health data-sets and anonymization problems are recurrent topics in Generative Adversarial Networks. Given our main goal, in this project we will focus on them, taking as a reference the following articles:

Learning Anonymized Representations with Adversarial Neural Networks by Clément Feutry et al. [6]. Statistical methods protecting sensitive information or the identity of the data owner have become critical to ensure privacy of individuals as well as of organizations. This paper investigates anonymization methods based on representation learning and deep neural networks. The training procedure aims at learning representations that preserve the relevant part of the information (about regular labels) while dismissing information about the private labels which correspond to the identity of a person.

AnomiGAN: Generative adversarial networks for anonymizing private medical data by Ho Bae, Dahuin Jung and Sungroh Yoon [7]. Similarly to the previous paper, a GAN structure is introduced to improve the maintenance of privacy of personal medical data, while also maintaining high prediction performance in their application.

2.3 Missing Data

Similar to data augmentation, another popular application for Generative Adversarial Networks is generating new data, not as a tool to increase the number of samples in a data-set, but for full filling the incomplete ones.

In this case, GANs are not just generating similar images to the fed ones, but producing real-like data that must comply some restrictions given by the circumstances of the samples. In order to do so, new elements should be added to the basic GAN system. These elements, usually encoders, which won't be explained in this project as they are outside its aims, allow the generator to modify its output according to some boundary

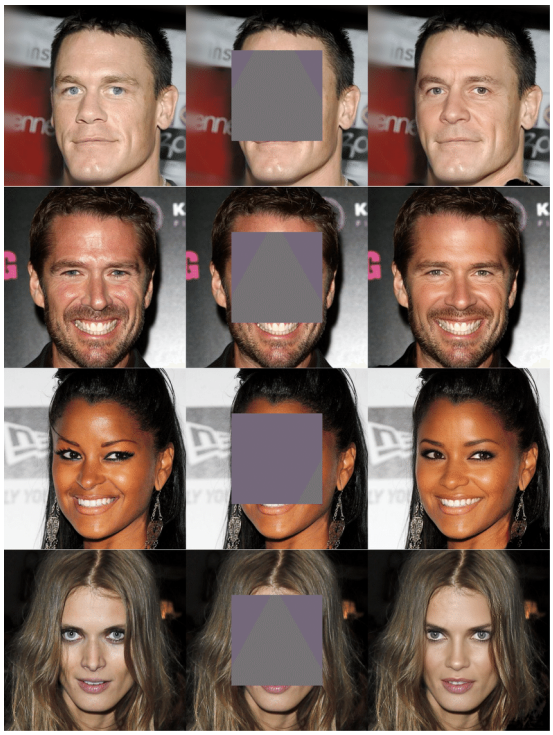


Figure 2.2: Example of missing data in an image: GAN's results (right) compared to ground truth image (left). Source: `https://www.researchgate.net/figure/Comparison-with-Context-Encoder-on-high-resolution-face-completion-The-top-row-are_fig1_322674898`

conditions. As it is shown in Figure 2.2, a common usage of missing data completion is image restoration, especially in image editing soft-wares.

Results and thorough explanations of this problem can be found in the following papers:

MisGAN: Learning from Incomplete Data with Generative Adversarial Networks. by Steven Cheng-Xian Li, Bo Jiang and Benjamin M. Marlin [8].

GAIN: Missing Data Imputation using Generative Adversarial Nets. by Jinsung Yoon, James Jordon and Mihaela van der Schaar [9].

2.4 Cross-Domain Transfer GANs

Last group of related applications refers to data transformation: manipulating an input in order to get some specific results. In this case, just as in Section 2.3, the input in

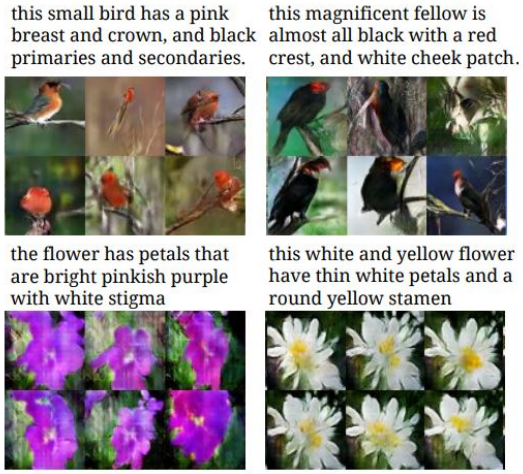


Figure 2.3: Examples of generated images from text descriptions. Left: captions are from zero-shot (held out) categories, unseen text. Right: captions are from the training set. Source: `https://arxiv.org/pdf/1605.05396v2.pdf`

the generator model is not a noise vector but a rich sample that should be modified. Applications for this type of problem are quite diverse, below are listed some examples:

Generative Adversarial Text to Image Synthesis by Scott Reed et al. [10]. In this work, it is demonstrated the capability of a GAN model to generate plausible images of birds and flowers from detailed text descriptions (see Figure 2.3) thanks to both text and image modeling advances.

SEGAN: Speech Enhancement Generative Adversarial Network by Santiago Pascual, Antonio Bonafonte and Joan Serra [11]. In this case it is shown the viability of enhancing voice recordings by reducing noise levels with GANs instead of classic speech enhancement methods such as spectral subtraction, Wiener filtering, statistical model-based methods. This can be useful for cochlear implants, where enhancing the signal before amplification can significantly reduce discomfort and increase intelligibility. Speech enhancement has also been successfully applied as a preprocessing stage in speech recognition and speaker identification systems. [12] [13] [14]

Medical Image Synthesis with Context-Aware Generative Adversarial Networks by Dong Nie et al. [15]. Last but not least, authors propose a way to generate computed tomographies from magnetic resonance images (CT). CT exposes radiation during acquisition, which may cause side effects to patients. Compared to CT, magnetic resonance imaging (MRI) is much safer and does not involve radiations. Therefore, recently researchers are greatly motivated to estimate CT image from its corresponding MR image of the same subject for the case of radiation planning (see Figure 2.4).

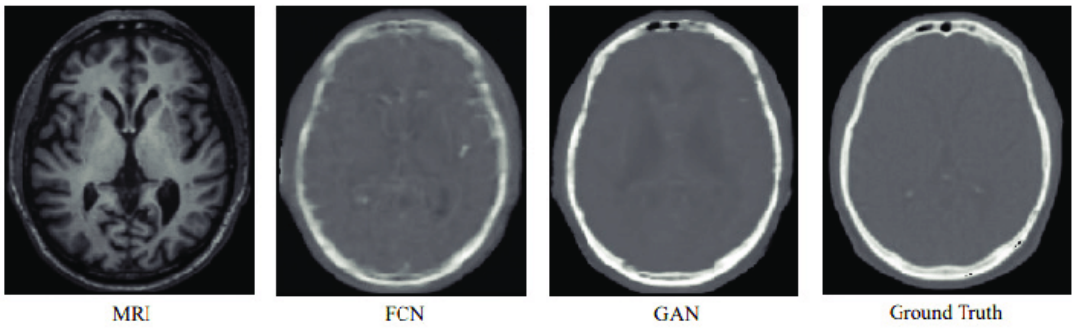


Figure 2.4: Generated CT images from an MRI scan. Second image has been generated with a Fully Connected Netowrk (FCN) and third by a GAN. Last image is the original CT image of the patient. Source: `https://www.researchgate.net/figure/Conversion-of-MRI-to-CT-using-GAN-38-Adapted-with-permission-Fig-5-Example-of-a_fig3_322657913`

2.5 Future Applicability

As a final section of this chapter, a more extended look on which are the future applications will be completed. As explained in Section 2.2, this project will be focusing on data anonymization applications. For this area, the possible implementations might be very vast: as the main issue in this situation comes through data privacy, if it feasible to by-pass it by anonymizing the original data what would suppose saving many sources of health information from important restrictions. This would proliferate the study of such data by making it more available.

On a different matter, the possibility of generating an infinite number of samples through a GAN is also very coveted. This virtue could allow the continuous simulation of certain variables from a patient for research purposes. This could help as well in the training process of other types of models by augmenting the number of samples, which are usually scarce in private data-sets. It could also be convenient in educational areas to avoid misconceptions in the subject because of a scarcity in examples.

Chapter 3

Project Management

In this chapter the tools and course of action to assure the success of the project will be introduced.

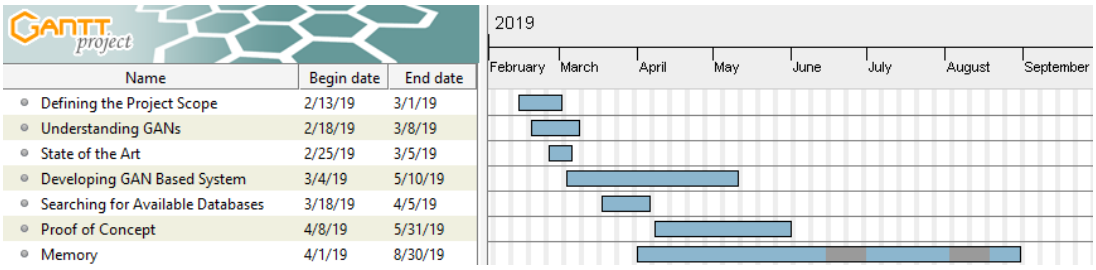
3.1 Procedure and Tools for Project Monitoring

In order to control the evolution of the work development, both in content and timing, some methodologies have been used. On the timing side, a GANTT structure has been elaborated. On the content side, a Github repository has been created for the evaluation of the code. For the reporting, an online software, called Overleaf, has been used. On the following sections these software tools and methodologies will be explained in detail.

3.1.1 GANTT

The GANTT structure followed by this project has been the one shown in Figure 3.1.

As it can be seen, most demanding tasks are writing down the memory, developing the GAN system and carrying out the proof of concept, in that order.



Name	Begin date	End date
Defining the Project Scope	2/13/19	3/1/19
Understanding GANs	2/18/19	3/8/19
State of the Art	2/25/19	3/5/19
Developing GAN Based System	3/4/19	5/10/19
Searching for Available Databases	3/18/19	4/5/19
Proof of Concept	4/8/19	5/31/19
Memory	4/1/19	8/30/19

Figure 3.1: GANTT scheme for the timing monitoring of this project. On the memory task, it is marked in grey the second half of June and first half of August as this weeks were not available for the project. Software used fort his purpose: *GANTT Project, https://www.ganttproject.biz/*.

3.1.2 Github Repository

In order to publish and maintain the code of this project, a Github repository has been created. Github[1] is an online platform focused on hosting code-based files and its versions in order to keep track of changes, secure its integrity on the cloud and public a project when needed.

In this case, all code files are publicly available in the following URL: `https://github.com/EstebanPiacentino/GAN4DataAnonymization`

It is important to highlight that the strictly necessary code to be reviewed in this memory is also attached as an Annex. For any other code section, the Github repository will be available for its reading.

3.1.3 Overleaf

Finally, for the monitoring of the report content and its versions a LaTeXonline editor has been used. This editor is called Overleaf[2] and has the ability of saving versions and share them between collaborators in real time. This ease considerably the reviewing process of the document. This platform also saves image and bibliographic repositories on the cloud without the need of a local software nor file to progress on it.

[1] More information about this platform can be found at `https://github.com`
[2] More information about this platform can be found at `https://www.overleaf.com`

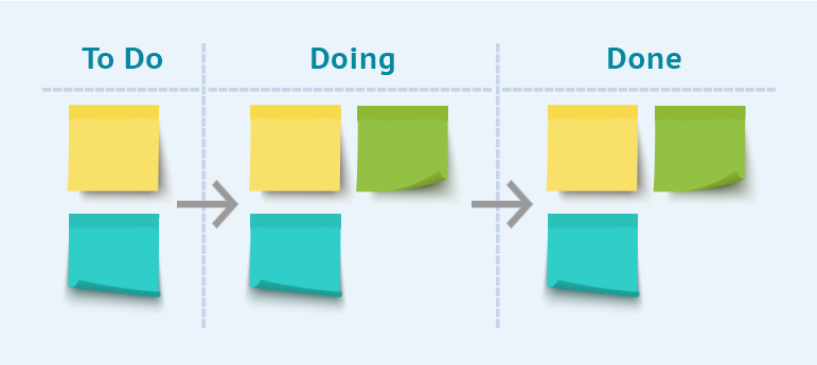


Figure 3.2: Kanban flow representation. The column in which is positioned each task –represented as a post-it– determines the status of the process. The standard status are "To Do", "Doing" and "Done". Source: `https://www.digite.com/kanban/what-is-kanban/`.

3.2 Validation Tools

For the validation of the objectives and tasks exposed above, another software, called Trello, has been used. Trello[3] is an online platform for tracking task completion based on Kanban flows. This allows the project to keep track between collaborators of which tasks are completed, which are active and which are to be initiated. In combination with the project monitoring tools exposed in the previous section, this software is a key element to ensure all checks and milestones are reached.

3.3 SWOT

Following on the Future Applicability (Section 2.5), taking into account the potential outcomes previously exposed for this technology and also its limitations a SWOT (strengths, weaknesses, opportunities, and threats) structure –called DAFO in Spanish– has been performed. Elements positioned in each of the clusters are shown in Figure 3.3.

Leaving aside the strengths of the technology, which have been already reviewed (at page 14), on the weaknesses side emphasis is made especially on the drawbacks of neural networks technologies. The fact that neural networks can only be verified by an extensive use of them, as its internal configuration cannot be deciphered, is a delicate factor that could be critical when facing the threats of this technology. On a secondary level, the long training times could be another weakness.

[3]More information about this platform can be found at `https://trello.com`

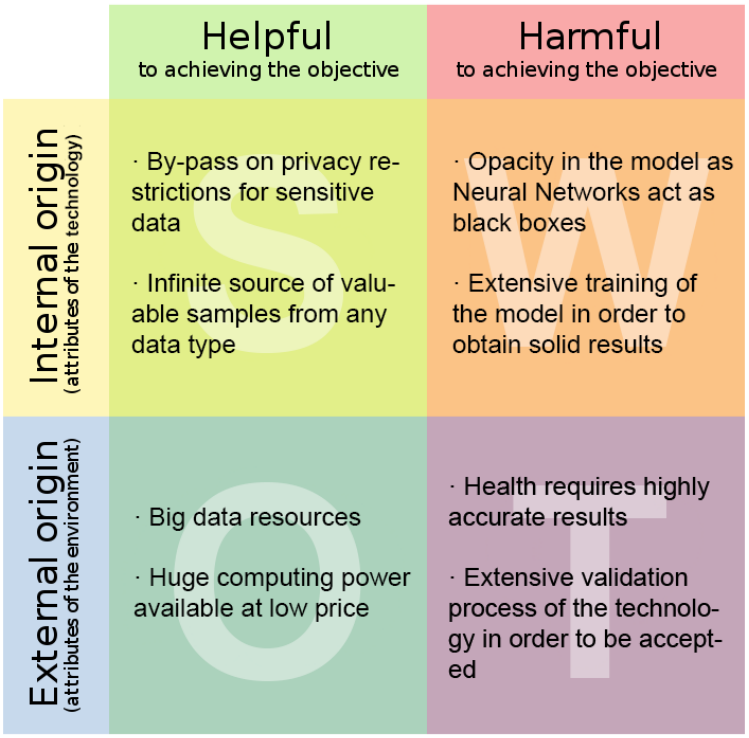


Figure 3.3: SWOT analysis for the exposed projects containing the main factors of each of the 4 parameters (Strengths, Weaknesses, Opportunities, and Threats). Structure template source:

On the opportunities area, the factors are highly influenced by the data revolution of the last decade. The increase of computing power at low prices together with an unprecedented volume of available data in every field has proliferated the usage of this artificial intelligence tools. However, as it is shown on the threats area, in the health sector highly demanding requirements will be encountered. This gives rise to the thought that, if first viability evidences are found, the following investigation steps will need to be thoroughly built on stability and reliability of the system.

3.4 Risks & Contingency Plan

As the system studied on this master thesis has different layers of complexity and a variety of configurable variables that can be changed to modify the output, the risks of finding problems throughout the process of generating fake data are very high. That is why identifying the risks that this project will be potentially facing along with a contingency plan to resolve them properly is very important.

The main risks that could lead to an unsatisfactory result on the proof of concept are the following:

- Insufficient original samples to train the GAN system.
- Errors in the neural networks models for either the generator or the discriminator.
- Errors in the conversion method from image to raw data in the GAN system.
- Non convergence of the generated output, leading to non realistic results.

Even though risks are many, the contingency plan will always be based on avoiding unnecessary difficulties and divide the main system elements in independent focus problems when possible to avoid dealing with the interaction of different issues at the same time.

In order to avoid an insufficient number of original samples, the criteria of the chosen databases will always prioritize its number of samples. This way, the chances of not having enough data will be low. It is not part of this first investigation to understand which is the minimum number of samples needed to satisfactorily generate fake data.

For the rest of possible errors, the key to dodge them will be built on testing the GAN based system developed for this project by parts: starting with the essential elements in its more basic form to ensure a solid structure and then add gradually more complexity and extra elements. With this strategy, issues will come one by one and debugging phases should be easier. It is also important to mention that, when changing any parameter of the system to increase its performance, modifications will be done one by one and tracking the changes applied to avoid confusion on the causality between the results and the changes involved.

3.5 Initial Cost Analysis

In the following section, the cost of this work will be calculated. Furthermore, as this first phase of the project hasn't supposed any cost outside the hours of dedication, an estimation cost of a second phase will be done. This second phase symbolizes the immediate next steps that should be overcome. The explanation of the next steps can be found at Chapter 7.

Initial Cost Analysis							
Category	Item	Quantity (1st Phase)	Quantity (2nd Phase)	Units	Price by Unit	Total (1st Phase)	Total (Both Phases)
Research	Project Manager Engineer	20	50	h	€18.00 (1)	€360.00	€1,260.00
	R+D Engineer	380	950	h	€17.00 (1)	€6,460.00	€22,610.00
	Papers purchase	0	2	items	€40.00 (4)	€-	€80.00
Hardware & Services	Cloud computing service	80 (2)	300	h	€30.77 (3)	€-	€9,231.00
	Cloud storage service	15 (2)	100	GB	€00.13 (3)	€-	€12.60
	Databases	4	6	items	€-	€-	€-
External Services	Medical specialist for the validation of results	0	12	h	€35.00 (1)	€-	€420.00
GAN System	Computer Science Engineer	80	120	h	€17.00 (1)	€1,360.00	€3,400.00
	Stability and improvements by specialist	0	16	h	€33.00 (1)	€-	€528.00
					Total	€8,180.00	€37,541.60

Table 3.1: Initial cost analysis table. (1) Salaries estimates have been gathered from: `https://www.experteer.es/salary_calculator`.(2) First phase cloud computing and services were free of cost as Google Colab has been used. However, in further phases more powered services will be used. (3) Prices gathered from Amazon Web Services. (4) Average price given other papers of similar topic and magnitude.

On the Table 3.1 it can be found the first estimate of the project cost.

The total cost, taking into account, both 1st and 2nd phases, is 37,541.60 €. As seen on the previous table, most important costs come from engineer hours of dedication and cloud computing services for the second phase. It is interesting to highlight that computational cost can be lower if there is a reduction on hardware specifications. The determined hardware specifications for the second phase are: 8x Nvidia Tesla V100 GPUs (128 GB) and 64 CPU cores in contrast with 1x Nvidia Tesla K80 GPU (12GB) and a single core CPU of a Google Colab environment.

Chapter 4

Available databases

In order to test the hypothesis of generating usable anonymized personal data, databases of personal data will be necessary. These databases will be used to train and test different GAN models that will imitate the instances of them.

These databases will be organized in two different groups: *Image* directories and *Data* directories. This division will ease the following chapters, where a first approach of GAN manipulation will be focused on image generation (which is the most common usage) and the second one on general raw data generation (which is the final goal of this project). This way, in the transition between images and data generation, it will be possible to focus entirely in the source of innovation of this project: how raw data can be feed into a GAN system in order to yield good results, without the need of explaining the basics of the system which will have been explained previously.

Before listing these databases, it is important to highlight that the main features used to choose the aforementioned data-sets have been the number of samples and the clarity/information about the data.

Figure 4.1: Data samples from the Fingerprint Database. Source: `http://dsl.cds.iisc.ac.in/projects/Anguli/`

4.1 Image Directories

4.1.1 Fingerprints Database

Set of high-definition auto generated fingerprints from the Anguli generator used as ground truth samples for inpainting and denoising tools (see Figure 4.1). In this case, it will be used as an example of private information that can be reproduced through a GAN.

Source ChaLearn (Anguli: Synthetic Fingerprint Generator)

Available Samples 75600

URL http://chalearnlap.cvc.uab.es/dataset/32/description/

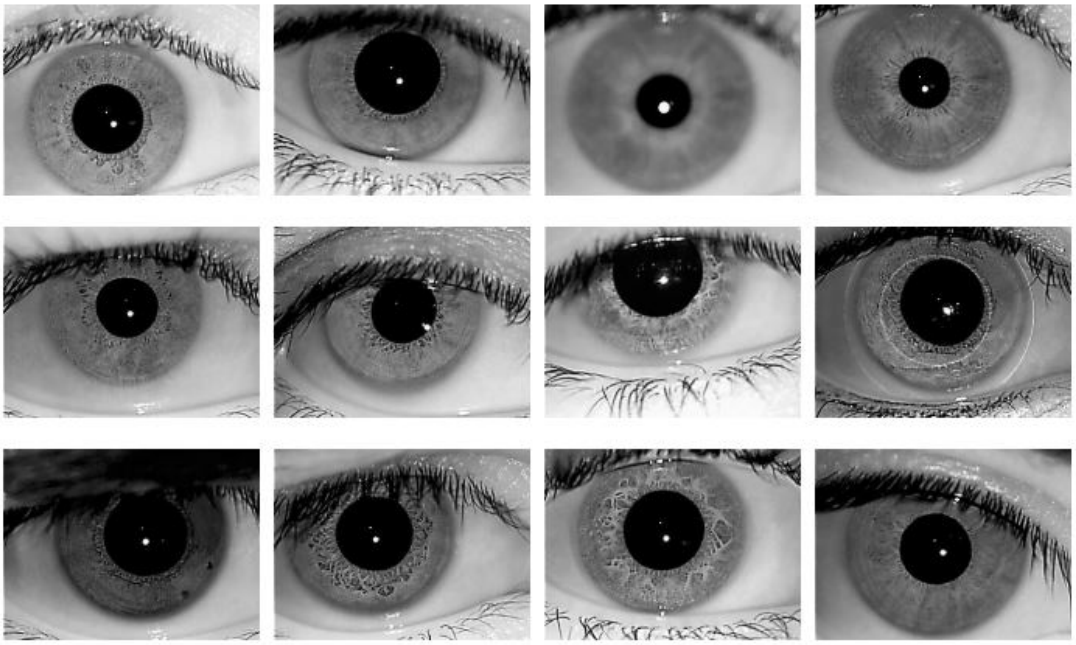


Figure 4.2: Data samples from the Iris Database. Source: `https://www4.comp.polyu.edu.hk/~csajaykr/IITD/Database_Iris.htm`

4.1.2 Iris Database

Set of left-and-right iris images collected from the students and staff at IIT Delhi, New Delhi, India (see Figure 4.2). This database is available in the public domain as a tool given the recent popularity of iris based personal identification systems. In this case, just like the fingerprint database, it will be used as an example of private information that can be reproduced through a GAN as well. Unlike fingerprints, these images contain more information and complexity.

Source IIT Delhi

Available Samples 2224

URL https://www4.comp.polyu.edu.hk/ csajaykr/IITD/Database_Iris.htm

4.2 Data Directories

4.2.1 Thyroid Database - Static Data

As the first data oriented set of the project, the thyroid database contains user features that are relevant for thyroid illnesses detection. All its variables are static, meaning that for each patient there's only one value per feature without contemplate its evolution. This database from KEEL has been released to identify if a given patient is normal, suffers from hyperthyroidism or hypothyroidism. The data contains both continuous and binary features (see Table 4.1) and they will all be imitated trough a GAN for a specific type of thyroid illnesses, hypothyroidism in this case.

Source KEEL

Available Samples 7200

URL https://sci2s.ugr.es/keel/dataset.php?cod=67

Features:

- Age (continuous)
- Sex (binary)
- On thyroxine (binary)
- Query on thyroxine (binary)
- On antithyroid medication (binary)
- Sick (binary)
- Pregnant (binary)
- Thyroid surgery (binary)
- I131 treatment (binary)
- Query hypothyroid (binary)
- Query hyperthyroid (binary)
- Lithium (binary)
- Goitre (binary)
- Tumor (binary)
- Hypopituitary (binary)
- Psych (binary)
- TSH (continuous)
- T3 (continuous)
- TT4 (continuous)
- T4U (continuous)
- FTI (continuous)
- Class (label): patient is normal (1), suffers from hyperthyroidism (2) or hypothyroidism (3)

Age	Binary Variables			TSH	T3	TT4	T4U	FTI	Class
0.73	0	...	0	0.0006	0.0150	0.1200	0.0820	0.1460	3
0.24	0	...	0	0.0003	0.0300	0.1430	0.1330	0.1080	3
0.47	0	...	0	0.0019	0.0240	0.1020	0.1310	0.0780	3
0.64	1	...	0	0.0009	0.0170	0.0770	0.0900	0.0850	3
...	...	...	...	...	...	...	...	...	...
0.23	0	...	0	0.0003	0.0260	0.1390	0.0900	0.1530	3
0.69	1	...	0	0.0003	0.0160	0.0860	0.0700	0.1230	3
0.85	1	...	0	0.0003	0.0230	0.1280	0.1040	0.1210	3
0.48	1	...	0	0.0021	0.0200	0.0860	0.0780	0.1100	3

Table 4.1: Data sample from the Thyroid database. Binary variables are listed above.

4.2.2 Cardiogram Database - Dynamic Data

In contrast with the previous data-set, the cardiogram database shows the evolution of its variables throughout time. The National Metrology Institute of Germany, has provided this compilation of digitized ECGs for research, algorithmic benchmarking or teaching purposes to the users of PhysioNet. The ECGs were collected from healthy volunteers and patients with different heart diseases by Professor Michael Oeff, M.D., at the Department of Cardiology of University Clinic Benjamin Franklin in Berlin, Germany.

Each record includes 15 simultaneously measured signals (see Figure 4.3): the conventional 12 leads placements (i, ii, iii, avr, avl, avf, v1, v2, v3, v4, v5, v6) together with the 3 Frank lead ECGs (vx, vy, vz). Each signal is digitized at 1000 samples per second, with 16 bit resolution over a range of ±16.384 mV.

The main objective of the usage of this database will be to simulate one of the ECG leads (i) for a specific range of time feeding the GAN previously with all the samples available from this data-set.

Source Physionet

Available Samples 549 (expandable by splitting each sample into multiple segments, see Chapter 6.2.2)

URL https://physionet.org/physiobank/database/ptbdb/

Figure 4.3: Data samples from the Physionet Cardiogram Database. Each row represents the data from the 16 input channels, (14 for ECGs, 1 for respiration, 1 for line voltage) in a 5 seconds time range. Source: `https://www.researchgate.net/figure/Conversion-of-MRI-to-CT-using-GAN-38-Adapted-with-permission-Fig-5-Example-of-a_fig3_322657913`

Chapter 5

Methodology & GAN System Description

In this section, the implementation of a GAN in a specific environment will be explained.

5.1 Methodology

The environment chosen to carry out this project has been:

- Execution environment: Google Colab
- Code Language: Python 3.6
- NN library: Pytorch

Google Colab offers free cloud processing capacity with dedicated graphics cards for training and inference tasks. On the other side, Python has been chosen because of previous knowledge of the language and Pytorch due to its high level configuration.

5.2 Code Sections

Below, the different sections of the GAN code model will be explained referencing them by the row numbers of the Fingerprint script (Annex 8.1), which is the base for the rest of GAN systems shown in this project. It is important to highlight that the base code and structure of this script has been extracted from a Medium article by Munesh Lakhey called *Generative Adversarial Networks Demystified* [16].

5.2.1 Dataset Class

After importing the required libraries into the environment, between line 45 and 69 (page 60), an extension of the *Dataset* class from Pytorch is implemented in order to import the Fingerprints database correctly. It defines how the database is indexed, how many samples there are and how to read each instance.

5.2.2 Loading the Database

Between lines 71 and 78 (page 61) the *Fingerprints Dataset* is instanced and fed into the *DataLoader* class (which is the class that administrates the training images) with a specific batch of 52 samples to train the discriminator and the generator for the case of the fingerprint model. Besides that, the transform function modifies samples to met the model specifications: it converts the image to a *tensor* object and normalize its values to range between -1 and 1.

5.2.3 Discriminator and Generator Models

Between lines 83 and 127 (page 61) the discriminator and generator models are configured and instanced. The number of layers, its size and the type of network (CNN or a linear FCN) will depend on the test. In all tests, however, the discriminator model always starts with an image-sized input and a single value output ranging between 0 and 1 (which is the one that specifies if the image fed is real or fake) and the generator starts with a noise vector (the seed for the generated image) of n dimensions (chosen by the user) and outputs a fake image of the same size as the original one. In order to restrict the output of

the discriminator between 0 and 1 the sigmoid activation function is used. Similarly, the hyperbolic tangent activation function is used in the generator model to range all pixel values between -1 and 1 (just as the original images do).

5.2.4 Noise Generator Function

Between lines 129 and 132 (page 62) the function for creating noise vectors is defined. As said before, this random vector is used as a seed for creating a new generated image. Each random number of the seed is normal distributed with mean 0 and variance 1 and the dimension of each seed vector is defined by the user. For this first test, seed vectors have 100 random numbers for each of the images on the batch.

5.2.5 Selection of the Optimizer Models

Between lines 134 and 139 (page 62) optimizers are defined for both, the generator and the discriminator. These optimizer models are the ones responsible of modifying the net weights in order to improve the output results of both generator and discriminator through the error gradient. The configuration of these optimizers has been left as-is from the base code of this script [16].

5.2.6 Training Loop

Between lines 148 and 223 (page 62), the training loop is found. This is the translation of Section 1.2.2 into code. Here, the previously listed elements are used altogether to form the GAN iterative system. In this case, the training loop is set to 100 epochs. In each of them, all the real images will be feed in random batches to the discriminator and train both generator and discriminator models.

5.2.7 Saving Step Results

Between lines 225 and 252 (page 64), which are actually still inside the training loop, both models and their respective current losses are saved in a specified frequency (every

5 epochs by default). This way the evolution of the losses can be then studied over time and a back-up copy of their models can be accessed if there's any forced stop of the script. After saving all the information, a 3-by-3 grid of images are displayed with current fake images of the generator, which allows to see how the model improves its output over the epochs passed.

5.2.8 Generating Final Results

Between lines 254 and 278 (page 65), a new 3-by-3 grid is rendered but this time with final results after passing all epochs.

Chapter 6

Project development: GAN Application

Now that the GAN system and its implementation in the working environment have been presented along with the databases that will be used, it is time to show the tests and its proofs done in this project on behalf of using Generative Adversarial Networks for anonymizing private health data.

As explained in Chapter 4, this GAN application will be organized in two separated groups: a first part where image-like private health data will be imitated (Section 6.1) – which should be the easier task as GANs are currently mainly used for visual applications– and a second one where the private data to imitate will be purely numeric features from a patient diagnose (Section 6.2).

6.1 Image Generation

In order to get a first approach on how a GAN system behaves, the initial applications tested on this project will be image-based, which is the usual case scenario of these kind of systems. This way, it will be easy to understand which are the capabilities and limitations of it and, especially, what changes it will bring when using raw data instead of images in the following chapter.

As explained in Section 5.2, for these applications the code has been structured adapt-

ing a previous work of Munesh Lakhey for generating digits from the MNIST data-set [16].

For both experiments below, the only changes that have been made to the original code are the following:

- Mounting Google Drive directory in the work environment to host training images on the cloud.
- An extension of the *Dataset* class (Section 5.2.1) to enable the correct reading of the specific data-sets of this project: this had enabled the possibility of working directly from a .zip file of all the images without the need of hosting every single sample on Google cloud by its own (which can cause reiterative errors when reading the data if all the images are in the same folder)
- Add-ons for visualizing and saving results live during the optimization process in order to see the evolution of discriminator and generator losses and reuse the models for further image generation.
- Specific generator and discriminator models for each database: both FCN and CNN models.

6.1.1 Fingerprints Data-set

Just as the rest of the tests in this project, before introducing the sample images for the first time in the system. These have been transformed to a grey-scale format where pixel values are transformed from [0:255,0:255,0:255] (RGB breakout) to [-1:1]. This simplifies and reduce the model size without losing any important information for the test (as all image data used already had only monochrome values). A visual example of this transformation can be found in Figure 6.1.

In this first test, the chosen model has been a linear one: the fingerprint images have been transformed from 275×400 (original size) to a single output ranging from 0 to 1 (discriminator) and the generator from a 100 dimensions noise vector to a 275×400 image through an FCN model. Both with the following layers setup (Figure 6.2). Regarding the batch size, it was decided to leave it on 52, which was the size for the original code of the GAN system.

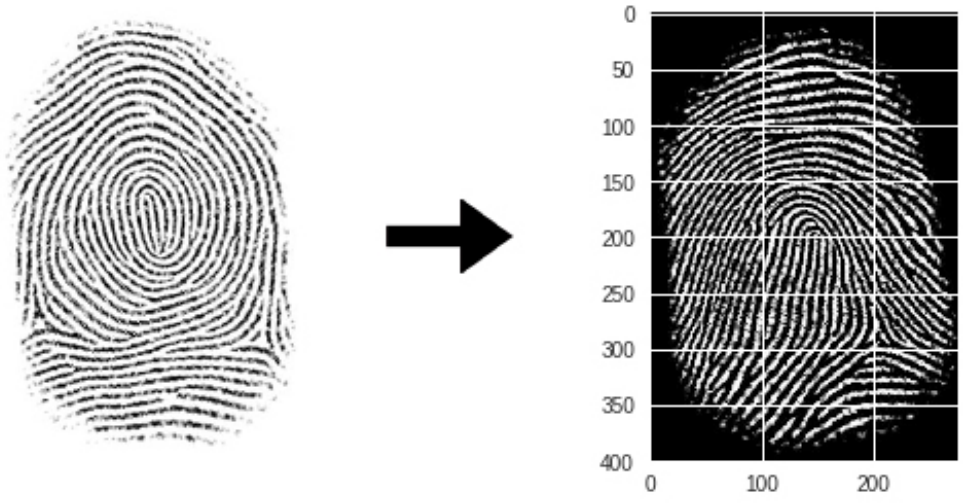


Figure 6.1: Representation of a raw sample of the fingerprint database (left) and a processed one where image values have been normalized to range between -1 and 1. Functions for this transformation are explained in Section 5.2.2.

The first findings of this test showed that trained FCN models are heavier than expected given the high number of links needed to connect all pixels from one layer to another ($\sim$ 1-2 GB). This was an important limitation, as layer's dimensions were highly limited to avoid excessive training times and because of hardware limitations. That's why, as seen in Figure 6.2, the first dimension reduction for the discriminator goes from 275×400 to 2048.

After a 100 epoch training process, first iterations of the problem showed that losses tend to stagnate after some epochs: sometimes before a clear output (first epochs) and other in a more evolved epoch. This can be spotted clearly in the losses of this test (see Figure 6.3) where after epoch 60, losses don't change any more and results don't improve. As found in some articles [17] [18], this is a concurrent problem with GAN systems and it will be found again in this project itself. For this first test, however, where the main goal is to get used to the system and get a reasonably good first fake generated image, this problem won't be inquired. In Chapter 6.2.2, a first solution for this problem it is found.

Regarding the output results seen in Figure 6.4, it can be perceived how in the first epoch there is already a defined contour of the fingerprint, where the outside of it is purely black (as in the original figures) but the inside is just noise without any kind of grooves. On following epochs, these grooves start to be defined but fingerprints are still very bright as there is not a clear separation between finger lines. Finally, between 60 and 100 epochs, results have improve significantly: fingerprints are darker (because of more

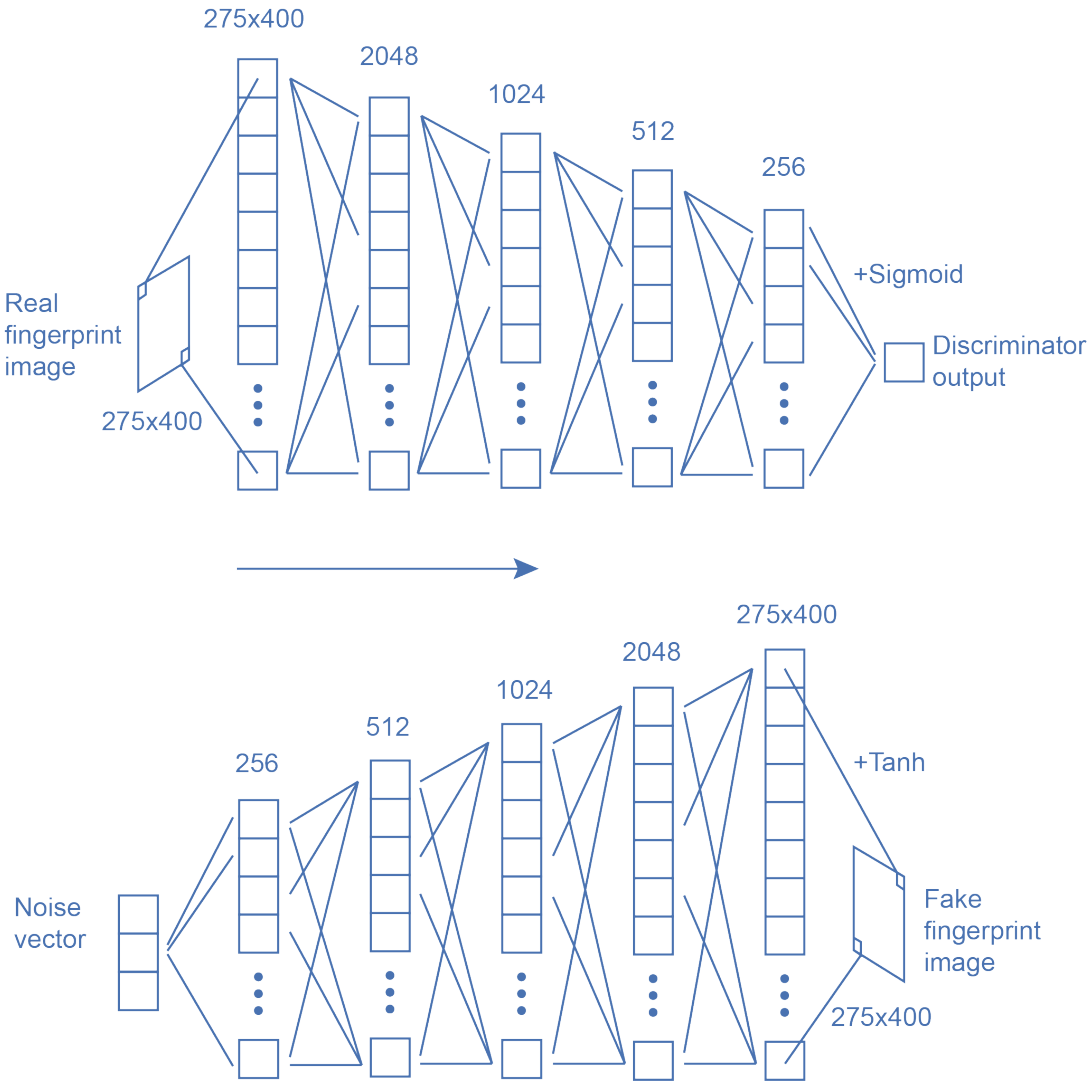


Figure 6.2: Schematic representation of the FCN models used for the Fingerprint test for both discriminator (top) and generator (bottom). As explained in Section 5.2.3, sigmoid and hyperbolic tangent functions are used to range output values between the desired values.

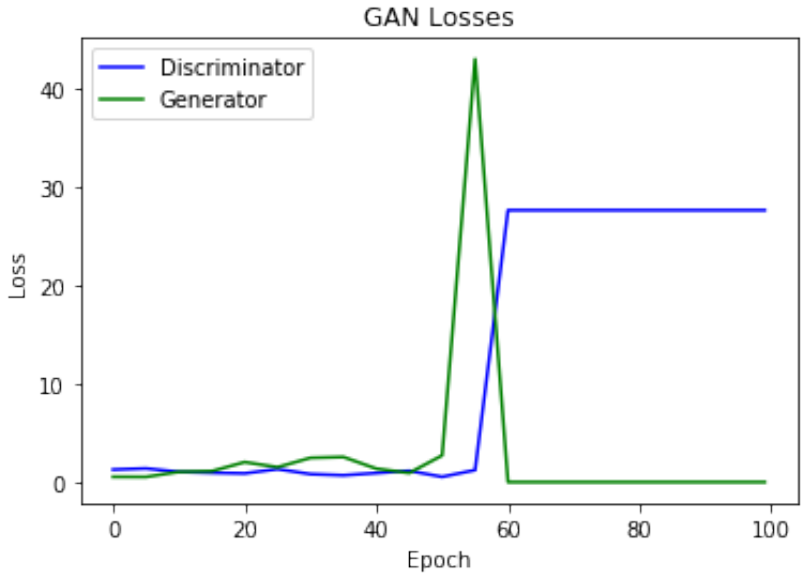


Figure 6.3: Loss values for both discriminator and generator throughout the 100 epoch training. As said before, there's a clear stagnation of the optimization process around the 60th epoch.

defined grooves) and some finger patters can be identified. However, results are still blurry and finger lines are not as defined as the real samples. These problems could be caused by the nature of FCN models: as fully connected layers are based on the aggregation of linear regressions, results tend to resemble more to an average of all real samples rather than a brand new generated sample.

In the following Section 6.1.2, where same experiments will be carried out for the Iris data-set, in order to mitigate this problem, CNN models will be used a part from the already known FCN ones. CNN models are non-linear and, because of its methods to extract features (in the case of discriminator) and create patterns (in the case of the generator) they tend to give better results. Plus, as it's not a fully connected architecture, models are significantly lighter ($\sim$ 30 MB)[1].

6.1.2 Iris Data-set

For the case of Iris data-set, the procedure has been the same: Images have been transformed and fed into the model aiming to observe the output performance of fake images. This time however, tests have been carried out for both FCN and CNN models to compare results between them.

[1] As it has been explained on Chapter 1.2.1, it will not be explained how CNN's nor FCN's work as it is outside the aims of this project.

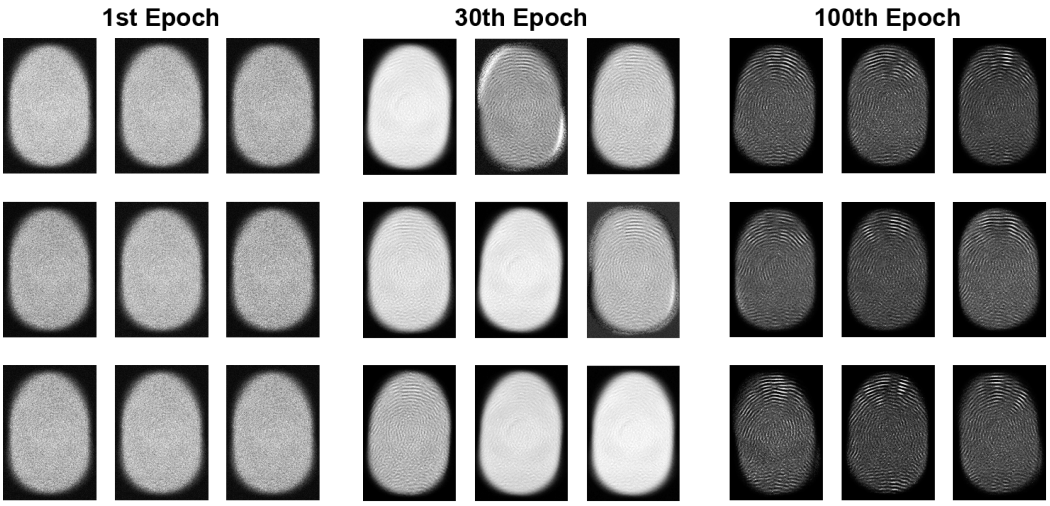


Figure 6.4: Generated fingerprint images samples throughout the 100 epoch training. 30 epoch has been chosen as a middle point due to loss stagnation around the 60th epoch.

FCN model

For the linear model, the main change from the fingerprint one has been the batch size, which has been reduced from 52 to 20 as the total database is significantly smaller (from 75600 to 2224). Besides from that, as the image size of the samples has changed as well (from 275×400 to 320×240), FCN model has slightly changed from last test to fit input and output images. The structure, however, remains the same from Figure 6.2.

Comparing losses between fingerprint and iris test (see Figure 6.5), it is seen that in this case the model doesn't stagnate: Discriminator and generator values are still oscillating. However, there's a slight tendency of the discriminator going up. The generator however, even though it has improved from epochs 40 to 60, it seems to be increasing its loss again.

Regarding the visual results, after 100 epochs, fake samples have some interesting attributes (see Figure 6.6): Images seem to be more unique between them than fingerprint results; each eye has a slightly different orientation, eyelids are sometimes closer than others and the light source changes position between samples.

On the other side, both fingerprint and iris results share the blurry effect due to the FCN model, which gives the sensation of a brush painted picture. As a medical oriented test, this can be a problem if details such as the iris membrane or pupil shape are important. That is why, a "pattern" oriented model as CNNs could be more suitable for this kind of task.

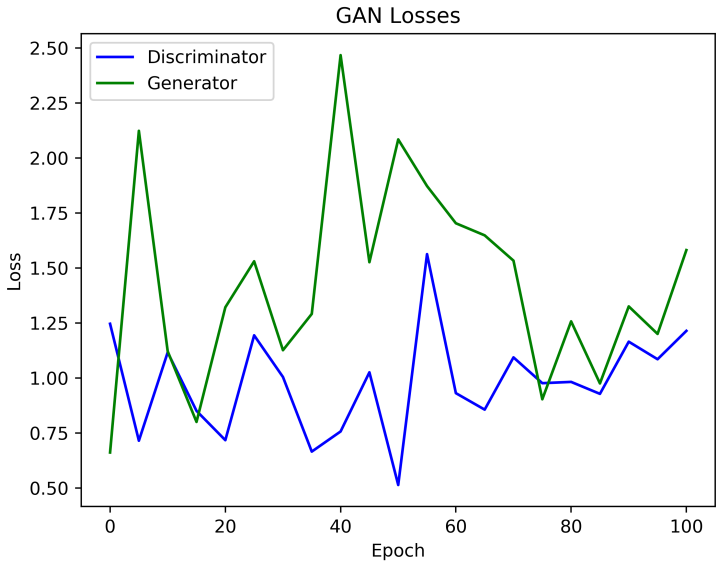


Figure 6.5: Loss values for both discriminator and generator throughout the 100 epoch training for the linear model pm the Iris data-set.

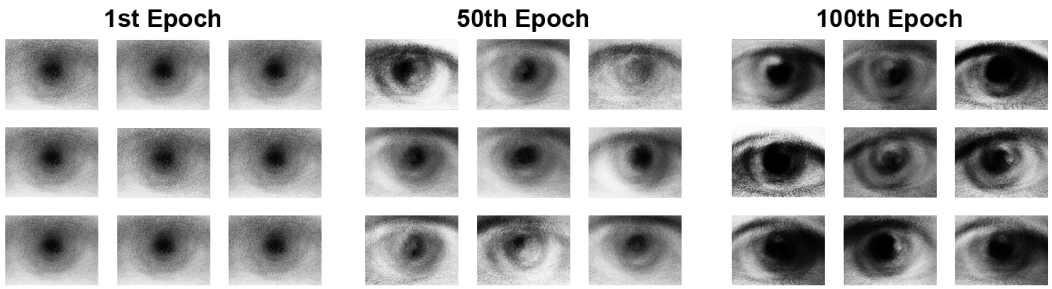


Figure 6.6: Generated iris images samples throughout the 100 epoch training with the FCN model.

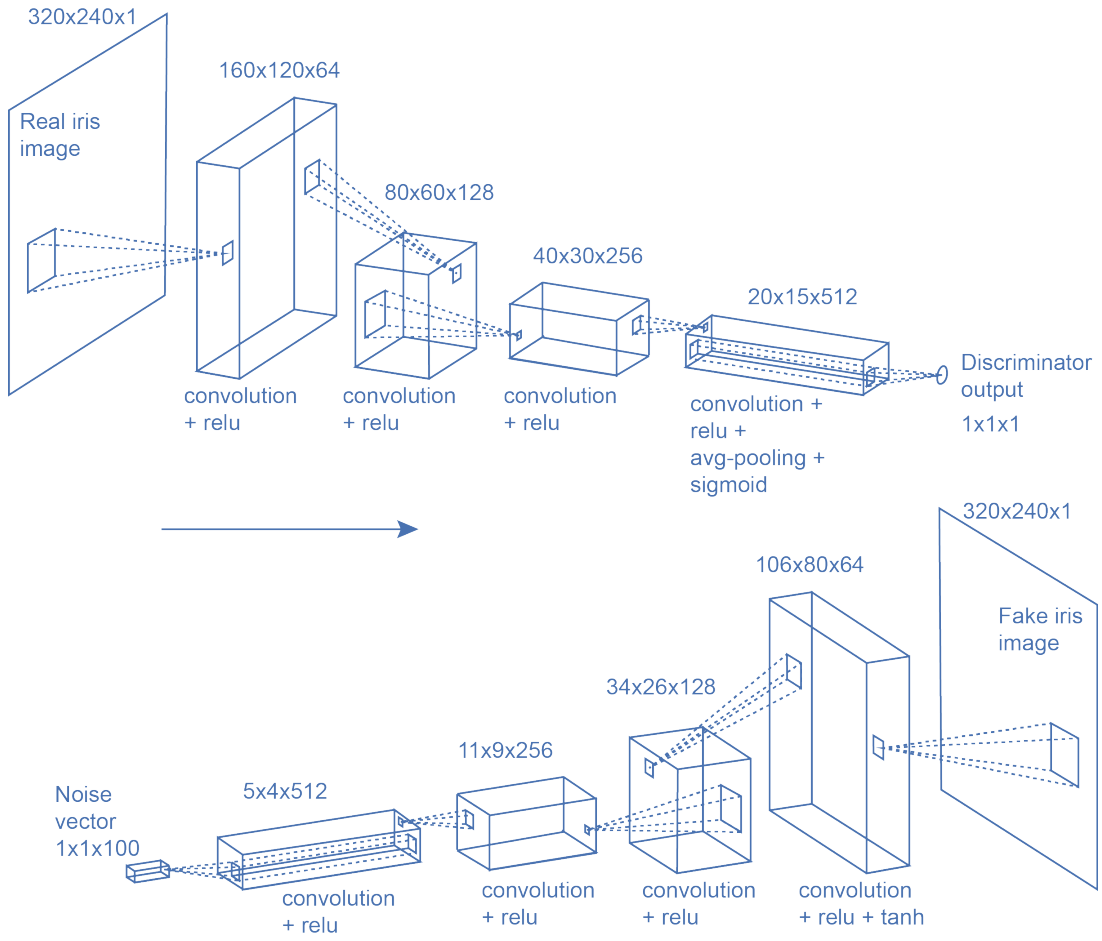


Figure 6.7: Schematic representation of the CNN models used for the Iris test for both discriminator (top) and generator (bottom). Sigmoid and hyperbolic tangent functions have the same functionality as in FCN models (Section 5.2.3). Regarding ReLU's, they are used to increase the non-linearity on generated images, which makes them more natural. Lastly, the avg-pooling function has been used to average the last output matrix into a single value ($1 \times 1 \times 1$).

CNN model

That being said, a CNN model for the Iris data-set was created. The structure of it, both for the discriminator and the generator is displayed in Figure 6.7. Due to the non-square nature of the iris database, CNN layers need to be set up meticulously to smoothly shift from a square image (noise vector) to a rectangular one (generated image) or vice versa. CNN layers configuration for this test are shown in Annex 8.2. As explained in page 35, CNN models are lighter, which makes the training process significantly faster. Because of that, the model has been trained for 200 epochs instead of 100. Visually, decent results seemed to take longer to rise than FCN models.

Regarding losses (see Figure 6.8), it is shown here one of the popular problems about

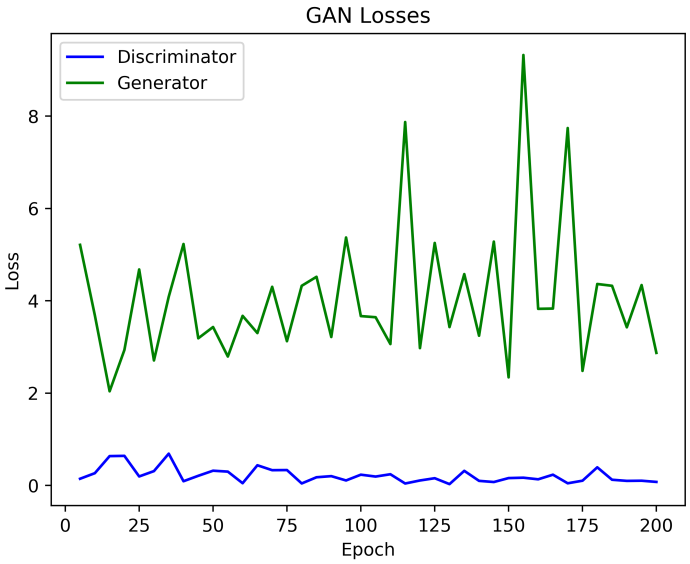


Figure 6.8: Loss values for both discriminator and generator throughout the 200 epoch training for the convolutional model om the Iris data-set.

GAN systems [17]: the discriminator gets too successful that the generator gradient vanishes and learns nothing, which it will be called *Diminished Gradient*.

However, throughout the training process eyes kept turning more realistic (see Figure 6.9). One of the key improvements over the 200 epochs has been the renderization of a single pupil in an image: as seen on the 100th epoch samples, the pupil was still split in two or even nonexistent in some of the generated images. On the final results, it is important to observe that fake images are remarkably more detailed than the ones from Figure 6.6. However, the overall shape of the eye seems less natural than on the previous iteration, which leads us to the main problems of the CNN model: as a patterned model, all samples are less unique between them. One example of this can be found on the eyelashes of the final fake images, which are mostly all made of the exact same pattern. Just like the blurriness of the FCN samples, having the same patterns for the iris membrane or the pupil shape is neither good. Actually, another of GAN's problems is the *Mode Collapse* [17], where the generator collapses producing a very narrow varieties of samples, which could be an important problem for health environments.

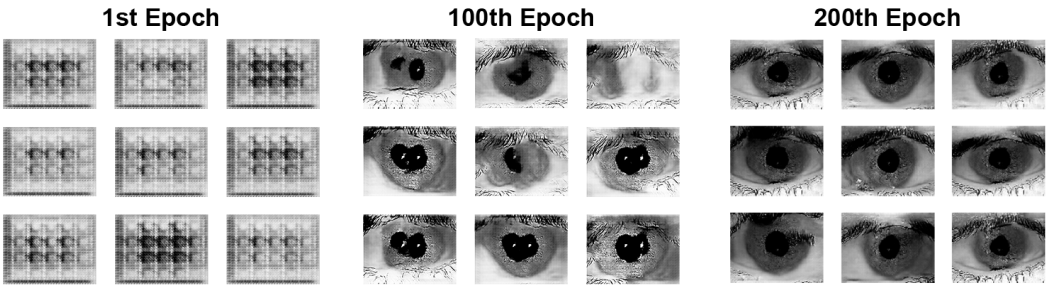


Figure 6.9: Generated iris images samples throughout the 200 epoch training with the CNN model.

6.2 Data Generation

On this section of the project, focus will shift from images to raw data generation. This conversion is not trivial, however. Data, unlike images, can have many different dimensions and contain values within many different ranges. Images, on the other side, are mainly represented in a range value of 0 to 255 for every pixel and with just 2 or 3 dimensions depending if it is RGB or not.

Hence, in order to use the same tools as for image samples, changes will be needed to uniform data and represent them in a *visual* way to the GAN system. This issue will be divided in two tasks:

- **Data standardization**: Bringing all features of the data-set to the same range value, which will be from 0 to 255, and trimming all samples to same same amount of features if needed.
- **Data arrangement**: Deciding the layout of the data, meaning how features are placed, in a square shaped space.

A domestic parallelism of this problem could be how humans visualize data in order to understand it themselves better and quicker. Feeding the original data to the GAN system in the correct way could be as important as representing an electrocardiogram data to a doctor in a proper –and visual– way.

That being said, on the code level, the main changes that will be done to the previous scripts will be:

- Data wrangling processes on the original samples in order to address the issues

explained above.

- Creating converters from processed data to image files and vice versa.

Besides that, the Following sections will be divided in two: static and dynamic data. This separation is necessary to differentiate the type of data it is being simulated. On the first section, static data, each feature is an independent variable that it is supposed, within the scope of the project, to be constant over time. Opposed to that, the dynamic data section contains features that are different values over time of the same variable. In other words, on dynamic data each row is the sequence of values from a time-series sample. While on static data each row is a list of independent attributes that describes the sample itself.

6.2.1 Static Data

The first data-set that will be imitated is Thyroids' from KEEL's repository. As explained in Section 4.2.1, this database contains both continuous and binary features from patients thyroid-related health status. The main objective of using a GAN on this repository is to generate new anonymous users with the same conditions as the original patients. In order to do so, a method to translate the data into an image format needs to be designed. This way it will be possible to pass all patients information to the GAN system and then receive the generated patient samples accordingly.

Following the requirements presented in Section 6.2, below are the steps followed before training the GAN system with the original data:

Data standardization

In order to give the same importance to all variances within each of the continuous features, all units need to be standardized to a common range. This is because the range of each feature in the data-set varies widely and this can be a problem in the objective function domain due to the weight of each variable range. This action is also called feature scaling or normalization.

In order to do so, the distribution of all continuous variables has been studied to check its max and min values and evaluate if there are anomalous extreme values (*outliers*)

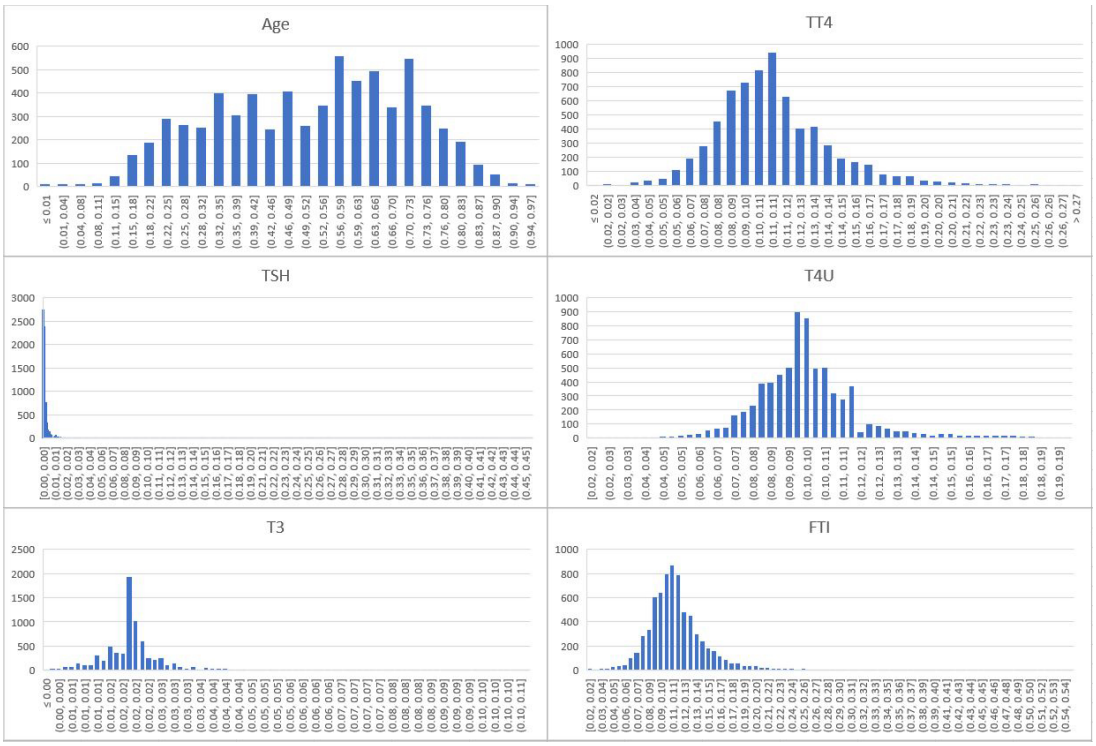


Figure 6.10: Thyroid's features distribution histogram. Every vertical axis represents the total amount of samples in each feature sub-range.

that can be removed from the data-set to avoid losing resolution on the variability within the most populated range. An example for this could be the following: given a feature where its values are within 0 and 10 but there is one anomalous value at 100, if the feature is standardized to a [0,1] range taking into account the anomalous sample, the most populated range –and therefore them most important– will be reduced to [0,0.1] so the value at 100 can be placed at 1.

Before analyzing the features distribution, it is important to say that for this test only class 3 data has been used. This means that the GAN system will be trained only with hypothyroidism patients in order to simulate just one type of diagnostic. It has been chosen hypothyroidism because it was the condition with a greater number of samples in the data-set (6666) in contrast with hyperthyroidism (368) and normal condition (166).

That being said, on Figure 6.10 are plotted the histograms of all continuous features in the Thyroids data-set.

As it can be seen, TSH, T3 and FTI features are the ones with greater long tales. Taking into account this experiment is being carried out without medical knowledge, and therefore it is unknown the importance nor reason of these isolated values, the trim of the outliers has been done visually with the histograms in Figure 6.10. On Table 6.1 are

Feature normalization limits					
Age	TSH	T3	TT4	T4U	FTI
0.01	0	0.002	0.017	0.04	0.002
0.97	0.02	0.05	0.27	0.18	0.26

Table 6.1: Feature range limits taking into account histograms from Figure 6.10.

Age	Binary Variables			TSH	T3	TT4	T4U	FTI	Class
192	1	...	1	9	70	104	77	143	3
62	1	...	1	4	149	127	170	105	3
123	1	...	1	25	117	86	166	76	3
168	255	...	1	12	80	61	92	83	3
...	...	...	...	...	...	...	...	...	...
59	1	...	1	4	128	123	92	150	3
181	255	...	1	4	75	70	55	120	3
223	255	...	1	4	112	112	117	118	3
125	255	...	1	27	96	70	70	107	3

Table 6.2: Data sample from the Thyroid database once min-max normalization have been applied.

represented the determined limits for each of the features exposed above. After removing the outliers samples and filtering by hypothyroidism condition, the total remaining samples are 6330.

Now that ranges have been specified, the feature normalization will begin. For this project, a min-max normalization will be used. The formula for which is:

$$\left\lfloor \frac{x - \min(x)}{\max(x) - \min(x)} \cdot (255 - 1) + 1 \right\rceil \tag{6.1}$$

This equation will output every feature between 1 and 255, which is the usual range for image files. The number 0 is excluded from the range so that can be used exclusively as a NULL value. This will be useful in the following section. Applying Eq. (6.1) to Thyroid's data-set, the first examples shown in Section 4.2.1 look like as in Table 6.2.

Data arrangement

Now that the data is clean and standardized, we can proceed to the arrangement of itself in an image format. Taking into account the two types of features –binary and continuous– and their number –6 and 15 respectively– many arrangements can be done in a 7×7 grid. In Figure 6.11 are displayed some of them. The reason why the arrangement of all features should be done in a square area is to avoid complications in the system configuration if models such as CNN are used, just like it happened in Section 6.1.2 for the Iris data-set. For this first iteration, the arrangement used will be the first one (left,

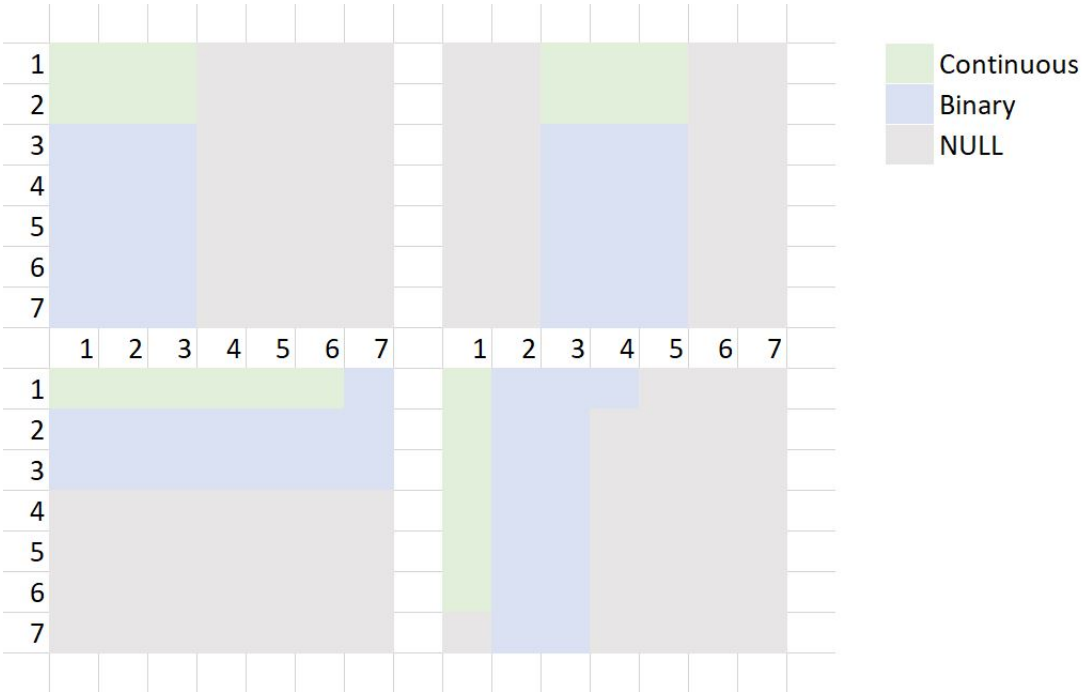


Figure 6.11: Possible Thyroid's features arrangements separating binary and continuous features.

top scheme). It is important to note that for the NULL area, all values will be set to 0. As brought in Section 6.2.1, the value 0 will be exclusively used to determine a value as NULL, this way it will be easier for the neural network to distinguish between feature areas and NULL areas.

In Figure 6.12, there is an example of real thyroid sample turned into image with the chosen arrangement.

Figure 6.12: Original sample from the Thyroid's data-set converted into an image. This sample is following the first arrangement scheme from Figure 6.11.

Model, Training and Results

As a first test, the Thyroid images will be feed into a FCN. Just as the other FCN models in this project, it will be trained for 100 epochs. Batch sizes will be of 20 images. Regarding FCN layers, as this time images are only 7×7 pixels, their number and size has been reduced considerably. The structure keeps being the same as Figure 6.2 but with just one hidden layer of size 64 between the input and the output for both the discriminator and the generator. The last change that has been done to the original structure is the size of the noise vector for the generator, which is now of 5 values instead of 100.

After training, losses this time have come the other way around: as seen in Figure 6.13, the generator overtook the discriminator from the beginning. The discriminator trend is positive and the generator seems stable between 0.6 and 0.8. Similar to the CNN Iris model (page 38), there seems to be a diminished gradient [17] on the discriminator side that is causing it an inability to improve because the generator is too strong.

On the output results side, it can be seen a great improvement between epochs on Figure 6.14. It is perceived how between epoch 1 and 50, the values on the binary zone start to get contrasted values –around 1 or around 255– and on the continuous values there is still a variety of values, which is correct. However, on epoch 50 there are yet values on the NULL side. Finally, on epoch 100 this problem is solved: all values of the NaN side are 0. It is important to note that on epoch 100 there are some values on the binary area that are not exclusively 1 or 255. This problem however, can be solved by

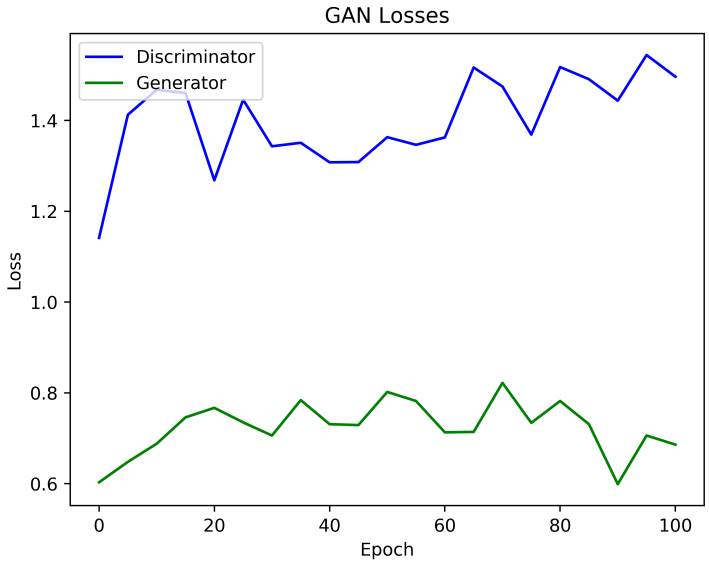


Figure 6.13: GAN system losses evolution throughout 100 epochs on a FCN model for the Thyroids data-set.

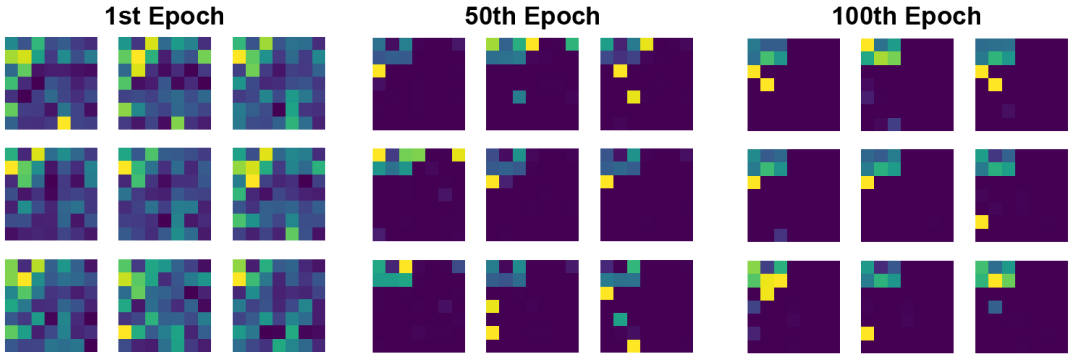


Figure 6.14: Thyroid's fake images evolution throughout 100 epochs on a FCN model.

applying a logic statement to round up this values to 1 and 255.

Subsequently, in order to understand if fake generated data is following the same distributions as the original, 6330 data samples will be generated and compared to the original ones. After generating these samples and converting the image files into numeric data, all continuous features data –both original one and generated– have been visualized through box-plots in Figure 6.15.

In all cases, the generated samples are positioned inside the ranges of the ground truth values for all continuous features. However, the dispersion of the values is different: Medians does not match neither Q1 and Q3 percentiles or min and max values. The feature that could be more in-line with ground truth samples is T3. This difference in the distribution of the features might be explaining that a simple GAN with FCN models

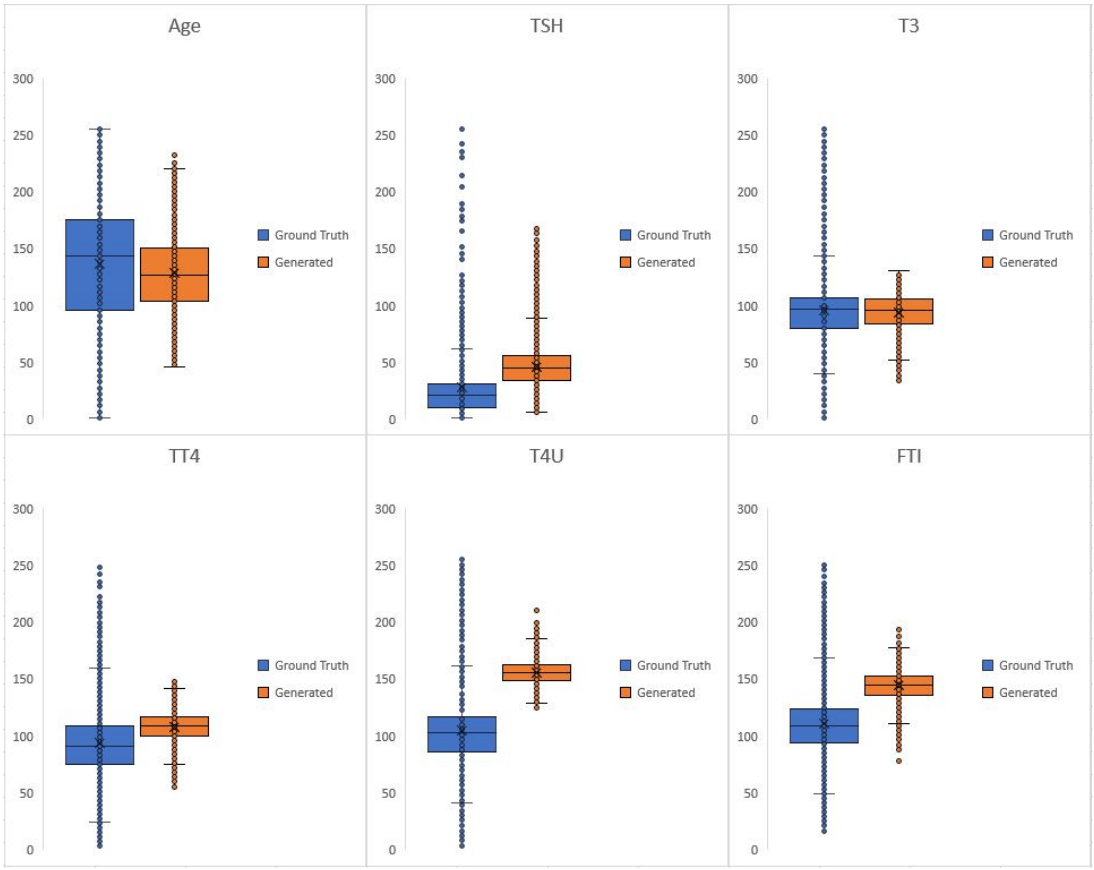


Figure 6.15: Thyroid's continuous data features box-plots: comparison between fake generated data and original data.

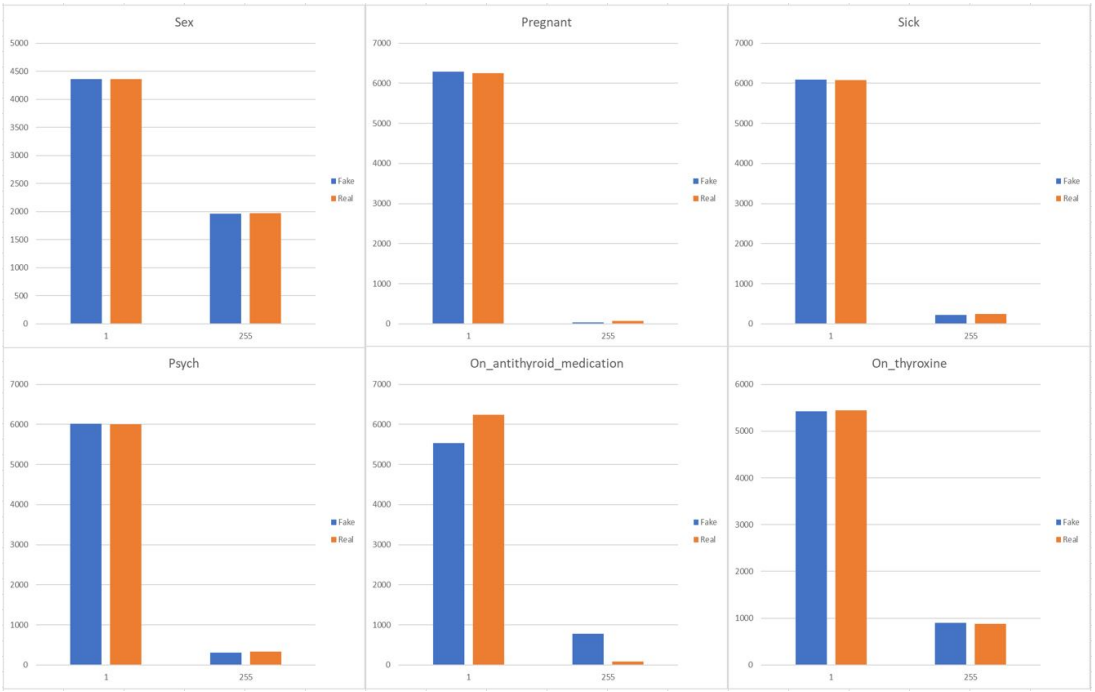


Figure 6.16: Thyroid's binary data features count by output value: comparison between fake generated data and original data. These are 6 of the 15 binary variables, the rest of the plots can be found at page 67.

is not enough to replicate this kind of data-set. This rises the possibility of future lines of investigation related with this project around the study of new indicators to be used as loss values for the discriminator and generator. One example could be comparing standard deviations between ground truth batch samples and generated samples for each of the continuous features. This, however, will not be analyzed in this document as it moves away from the project goals.

On the side of binary variables, it has been analyzed if, for the same amount of samples of fake and real data, the total amount of positive and negative values in all features is similar between both groups. In Figure 6.16 is rendered this comparison for some of the binary variables. The rest of the features binary plots are placed in the Annex at page 67.

Results show that counts between real and fake data are similar for almost all features. However, for the feature *On_antithyroid_medication* a more pronounced difference between groups exists. This could be related with the round up process to get mid values in the fake binary data output (e.g. 150) to 1 or 255 depending on which is the nearest one. In any case, results are satisfactory for this first state of the investigation. Because of that, the data-set will not be tested on CNN models as it is thought that for the goals of the project the FCN solution is good enough.

6.2.2 Dynamic Data

The second data-set that will be imitated is the Cardiogram database from Physionet. As explained in Section 4.2.2, this database contains records of 15 different signals from patient's electrical impulses from the heart for a certain amount of time. The main objective using a GAN on this repository is to generate new anonymous cardiograms samples that look similar to the original ones. Specifically, in this case it will be simulated a single signal from the 15 available, lead "i". Same as on static data, a method to translate the data into an image format should be designed. This way it would be possible to pass all patients information to the GAN system and then receive the generated patient samples accordingly.

Following the requirements presented in Section 6.2, below are the steps followed before training the GAN system with the original data:

Data standardization

As explained in Section 6.2 the Cardiogram data-set, as an example of dynamic data, has a single variable –the "i" lead chosen– but as many features as sequential temporal values are available for each patient. Similar to the static data section, for this cardiogram signal it will be necessary to know which are the minimum and maximum values it can get in order to standardize the features values into the [0,255] range[2].

Due to the size of the database, it was not possible to use tools such as excel to analyze the signal values distribution –saving just all "i" lead values from all users weighted around 1GB, ∼59.000.000 values–. Hence, the database has been analyzed with Python and Pandas' library. An extract from the used code and its output is being displayed in Figure 6.17. On the description of the data (`Out[3]` in Figure 6.17) it can already be seen how 25% and 75% quantiles values are already far away from max and min values. This leads to think that there are probably outlier samples that should be removed from the data-set. In order to decide which samples to remove, the limit values that cover between 1% and 99% quantiles has been gathered (`Out[4]` and `Out[5]` in Figure 6.17). These are around -2 and 2.

Now that the range has been specified, the feature normalization will begin. Just as

[2]For this case, as it will be seen on the data arrangement section, it is not necessary to isolate the 0 value to use it as a NULL element.

```
In [1]: import pandas as pd
        import matplotlib

In [3]: data=pd.read_csv('histogram.csv',index_col=0)

        %matplotlib inline
        #data.plot.hist(alpha=0.5,bins=15)
        data.describe()

Out[3]:
```

	0
count	5.852655e+07
mean	-7.684476e-04
std	6.351073e-01
min	-9.115000e+00
25%	-1.830000e-01
50%	-1.600000e-02
75%	1.720000e-01
max	1.594300e+01

```
In [4]: data.quantile(0.99)
Out[4]: 0    2.022
        Name: 0.99, dtype: float64

In [5]: data.quantile(0.01)
Out[5]: 0   -1.9835
        Name: 0.01, dtype: float64
```

Figure 6.17: Extract from the Cardio data-set analysis with Python and Pandas. The *histogram.csv* was the file in which all "i" lead values from all users was saved. Last two rows show that 1% and 99% quantiles are between -2 and 2 values.

for the static data section, a min-max normalization will be used. Now, however, as the output range will be from 0 to 255, the formula used is:

$$\left\lfloor \frac{x - \min(x)}{\max(x) - \min(x)} \cdot 255 \right\rceil \tag{6.2}$$

Unlike the static data, however, the Cardiogram data-set has another issue that needs to be tackled: the number of features is not the same for all samples –as not all patients recordings have the same length– and the total amount of samples is significantly smaller that previous data-sets used in this project.

A solution to both problems is trimming each user sample into equal-sized features pieces. This way the number of samples will be greater and they will all the same amount of features. The choice of what size –and therefore number of features– should have each sample needs to be balanced between enough duration length and a good amount of total samples. In this case it has been decided to trim each patient recording into blocks of 2025 features –removing the tail if needed–: this will multiply the number of samples from 549 to 28902 while maintaining around 2 seconds of recording in each sample, which

0001	0002	0003	0004	0005	...	0041	0042	0043	0044	0045
0046	0047	0048	0049	0050	...	0086	0087	0088	0089	0090
					.					
					.					
					.					
1936	1937	1938	1939	1940	...	1976	1977	1978	1979	1980
1981	1982	1983	1984	1985	...	2021	2022	2023	2024	2025

Table 6.3: Data arrangement for cardiogram sample.

represent between 2 and 3 heart beats most of the times. The reason of choosing 2025 features is related with the next chapter: as it was presented on Thyroids and Iris datasets, it is preferred to have squared images when dealing with some models such as CNNs. Therefore, the total amount of features needs to be a perfect square number, and 2025 it is one of them.

Finally, to apply this two modifications exposed above on the original data, a Python script has been coded. This way it is easier to automatize the trim process and avoid the data size limitations of tools such as Excel. The code from this script it is placed in the Annex 8.4.

Data arrangement

Taking into account that this time there is not a second type of features such as the binary group on the static data, all the square area available for the data to image transformation can be filled up with features. That is why in this case there is no necessity of NULL values.

That being said, the only factor that needs to be determined is the path that will follow consecutive features in the image. For this test is has been decided to follow a "Z" movement from left to right. An example of all 2025 feature disposition can be seen in Table 6.3. Furthermore, in Figure 6.12 there's an example of a ground truth cardio sample turned into image with the chosen arrangement.

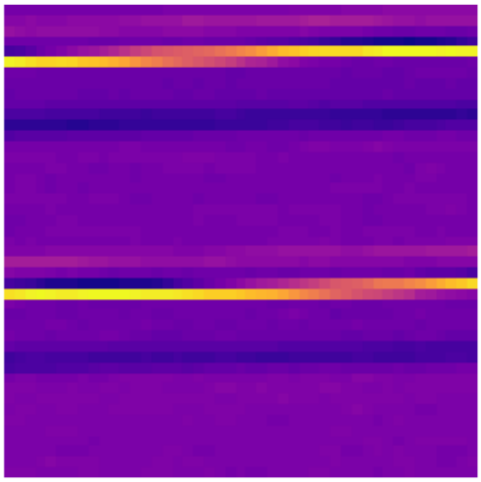


Figure 6.18: Original sample from the Cardio's data-set processed and converted into an image.

Model, Training and Results

Same as for the static data, the Cardio data-set will be first feed into a FCN model training it for 100 epochs. Compared to the 6330 samples available for the Thyroids database, now for the Cardio database there are available 28902 samples. Because of that, both the batch size and the hidden layers of the model can be larger compared the previous configuration: the batch size will be 52 and there will be 3 hidden layers of size 1024, 512 and 256 respectively for both the discriminator and the generator. The structure of the FCN model remains the same as in Figure 6.2 and the noise vector is again sized 100.

After training, the losses for both the discriminator and the generator seemed to be more unstable than previous cases (see Figure 6.19): values were similar at some epochs between models and both had positive trends along the first 100 epochs. Because of that, it was decided to extend the training up to 165 epochs in order to see if the generator's losses trend could change. But it was not the case.

However, the main evidences of the system not being able to reproduce a cardiogram signal were the fake sample results that were being generated. As seen in Figure 6.20, around the 50th epoch and beyond, all generated images were very similar and far away from the original samples. Fake outputs did not have coherent values between contiguous features and there were not identifiable pulse signals, just very noisy waves.

These results led to think that a pattern based model –such as CNN– could work better in this kind of situation. At the end, for this experiment there is a need of a very

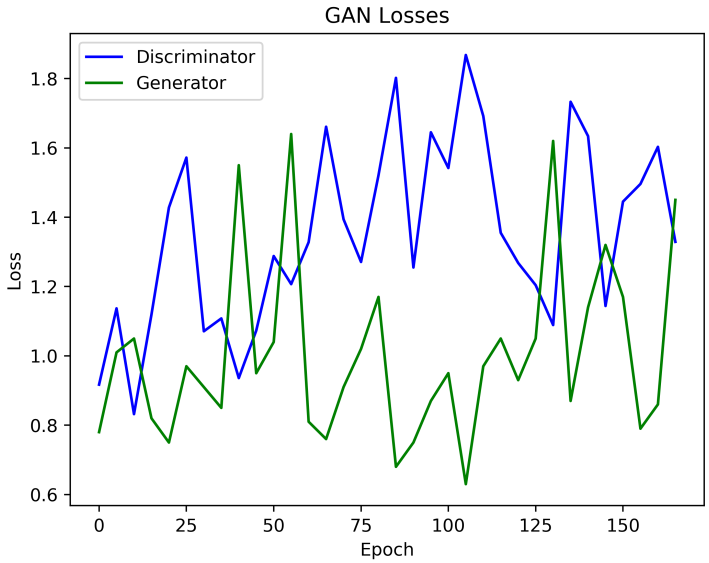


Figure 6.19: GAN system losses evolution throughout 165 epochs on a FCN model for the Cardio data-set.

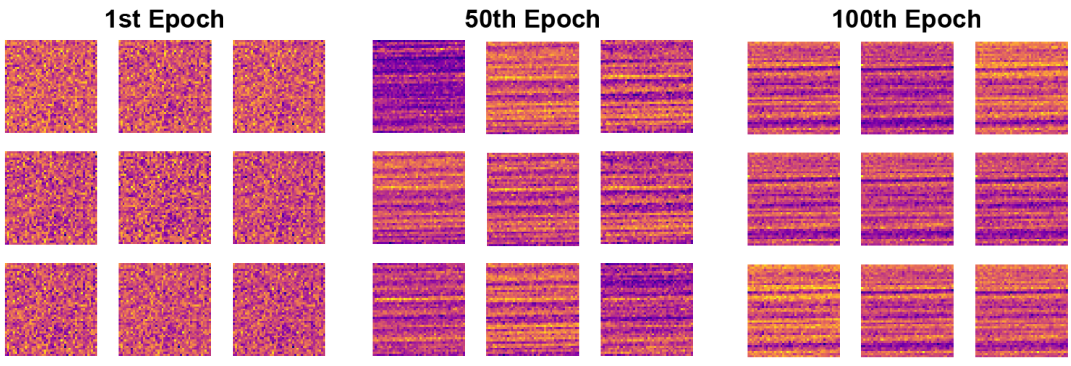


Figure 6.20: Generated Cardio images samples throughout the 100 epoch training with FCN models as discriminator and generator.

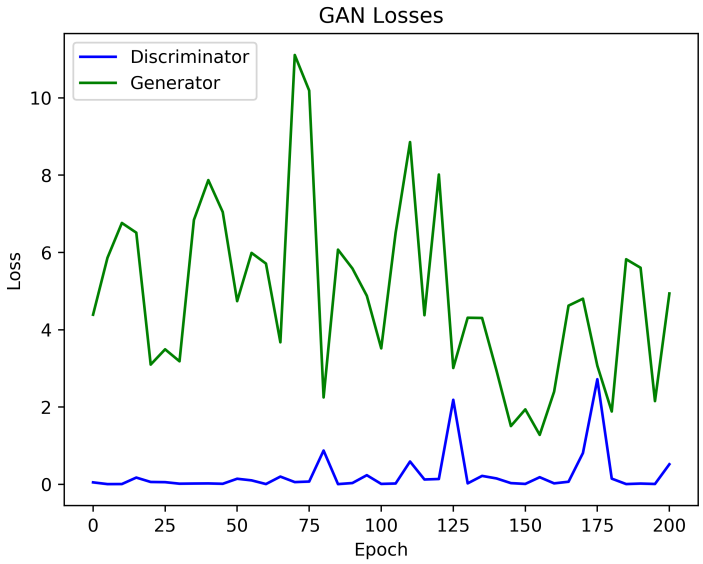


Figure 6.21: GAN system losses evolution throughout 200 epochs on a CNN model for the Cardio data-set.

defined chain of values with different patterns for both the heart beat and the steady state of the electric signal and not a sequence of diffused mean values throughout the recording. The differences between styles, especially those that are being searched for this test, can be observed in the generated outputs of the Iris experiment (see Figure 6.6 and Figure 6.9).

Because of this belief, a CNN model have been configured for the Cardio database: Similar to the Iris CNN model, the GAN will be trained for 200 epochs with a batch size of 30. Regarding the CNN configuration, as this time the images have a squared aspect ratio, the settings are easier than before. Given the structure of a CNN on Figure 6.7, the only difference is that there are 3 hidden layers of 64, 128 and 256 dimensions respectively. The configuration of each of this layers can be observed in Annex 8.5. The noise vector remains of size 100.

On the losses side, as it can be observed in Figure 6.21, there is again a problem of *vanished gradient* [17] as the discriminator gets too successful very quickly.

However, this time, to avoid a major saturation of the model –as happened on the Fingerprint data-set– a threshold value will be set up to avoid training discriminator more time when its loss is already very low. In order to do this, an *IF* condition will be placed before propagating the gradients to the discriminator in order to train the model only when its loss is above a specific value. Taking into account lines 189 to 192 from

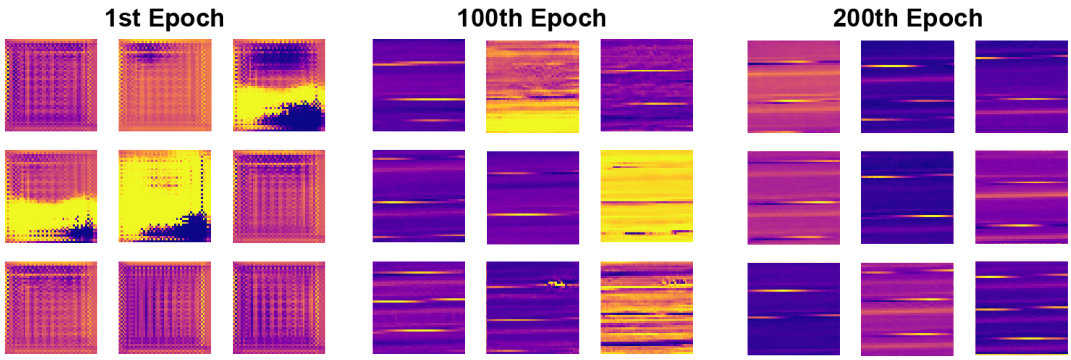


Figure 6.22: Generated Cardio images samples throughout the 200 epoch training with CNN models as discriminator and generator.

the original Fingerprints code in Annex 8.1, the modification that have been done is the following one:

Listing 6.1: Modification of the original GAN system for the Cardiogram data-set

```
    if loss_disc.item() >= 0.01:
          #computes the gradients
          loss_disc.backward()
          #update parameters using gradients & optimizer rules
          d_optimizer.step()
        else:
          pass
```

The threshold value has been set up to 0.01. Right at the beginning of the training process, it has been seen how the CNN model was having better results: after a few epochs the output images started to contain some patterns similar to the ones on ground truth samples. As shown in Figure 6.22, after the 100th epoch, some of the samples were already resembling to the original ones, however there were still some strange patters such as indefinite heart beats (mid bottom sample) or very high pitched images (mid and bottom right samples). On the 200th epoch, however, these problems were gone.

Given that now results are not static data, instead of analyzing the distribution of the studied variable, the output data will be compared to the original one visually –just how a doctor would do for this kind of information–. This way, it will be easy to see if signal samples seem to have the same patters, sizes and frequencies as the original ones. A random sub-group of real and fake cardiogram data can be observed in Figure 6.23. These results are very interesting on a first sight as both could seem real from a non-professional perspective.

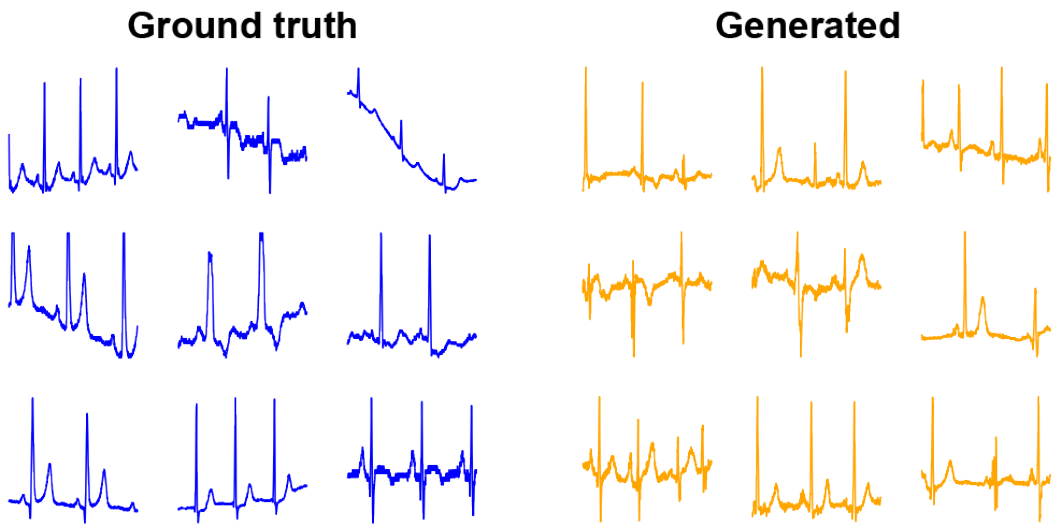


Figure 6.23: Ground truth Cardio signals vs generated samples after 200 epoch training on a GAN system with CNN models as discriminator and generator.

In a detailed comparison, there are some differences and similarities between both groups. Before mention them however, it is important to highlight that these evaluations are purely based on the visual differences between fake and real data. We are not aware if the fake data patters that seem unreal because they are not presented in the real data sample, could indeed resemble real conditions on human cardiograms. In order to make a better study of the results the consultant of a doctor would be required.

That being said, if we compare both groups, we can see that the generated cardiograms have quite equidistant heart pulses just as the real data does. Furthermore, overall patterns are very similar. Another similarity is that on all ground truth signals, the base slope is constant during all the time range. Same trend is observed on generated images.

Regarding differences, there are mainly two elements. There is a clear difference on spike lengths: ground truth heart pulses have consistently the same length, however this consistency is not solid on the generated side. Sometimes spikes are smaller or bigger than before. On the other side, the spike pattern within the same patient seems to be always the same but on the generated group patters have slightly variations throughout the sample. On the whole, the output images are not as defined and consistent as real samples. However, results seem to be satisfactory taking into account this is a first approach to the matter.

Chapter 7

Conclusions and Further Research

To start this chapter, it is fair to say that main objectives of the research have been accomplished. For each of the analyzed databases a first valid output has been yielded. On the whole, results show that data anonymization through Generative Adversarial Networks is feasible. However, further investigations will determine the accuracy levels that the data output can get and whether it is useful for health issues or not.

On the objectives side, it has been surprising to observe the ease of conversion from images to data for the GAN system and the results obtained in a small amount of time, both for linear and convolutional models. Another satisfactory detail was the solid coherence of final cardiogram's results throughout time: especially the smoothness of the curves and the similarity between spikes from a single fake patient recording.

Regarding the elements from the investigation that can be improved or studied further, it is important to highlight the following ones:

- An in-depth study of the data outputs from both Thyroids and Cardiogram data-sets by a specialist to verify the value and quality of the fake data.

- On Thyroids data-set, a more exhaustive comparison between generated an original data could be done: It could be interesting to check for correlations between variables on the original samples and check if these correlations are still visible in the fake data. Furthermore, in order to improve the similarity of the variables distributions –as explained at page 48–, an indicator of the deviation similarity between fake and original data could be fed into the objective function in following versions of the

system. Finally, a convolutional model for this data-set would be also enriching to train, in order to compare the performance between models.

- On the Cardiogram side, besides a better understanding of the results quality with the help of a specialist, a more complex system could be implemented to simulate cardiograms with specific characteristics: a concrete average heart rate, a concrete disease behavior. All this combined with longer generated recordings, as right now they are just 2 seconds long.

- On a general level, a recurrent problem that has been encountered is the non-convergence of results. Although all tests had a first valid output, losses graphs seemed not to be converging to stable values. This could be caused by a short number of training epochs or a bad system configuration. In any case, new iterations of the training process with longer computing periods and more powerful machines would be necessary.

- Finally, for a second phase of this project, a more mature system version would be appreciated. This means fine-tuning the configuration and standardizing its code to yield more solid and reliable results. This also entails further investigation on the creation of GANs, which could mean the purchase of some papers as has been noted in the initial cost analysis for the second phase (Section 3.5).

Chapter 8

Annex

8.1 Fingerprint GAN Code

Listing 8.1: Fingerprints GAN code in Python

```
# -*- coding: utf-8 -*-
"""GAN with PyTorch - Fingerprints.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1QrSDRXtQwyYkJaOX8-bb2Zo6yRjJYATu
"""

from google.colab import drive
drive.mount('/gdrive', force_remount=True)

cd /gdrive/My Drive/Colab Notebooks

#Dependencies
import numpy as np
import os
from skimage import io, transform
import torch
import matplotlib.pyplot as plt
# %matplotlib inline
from torchvision import transforms, datasets, utils
from torch.utils.data import Dataset, DataLoader
```

```
import torch.nn as nn
from random import choice

from zipfile import ZipFile
import io as ios

#Leer imagenes desde el zip

#from zipfile import ZipFile
#import io as ios
#
#archive = ZipFile("Fingerprints.zip", 'r')
#image_data = ios.BytesIO(archive.read("0.jpg"))
#out=io.imread(image_data)
#
#plt.imshow(out)

#cd /gdrive/My Drive/Colab Notebooks/Fingerprints
#!unzip 'Fingerprints.zip'
#rm finger_{1..2}.jpg

class FingerprintsDataset(Dataset):
    """Fingerprints dataset."""

    def __init__(self, root_dir, transform=None):
        """
        Args:
            root_dir (string): Directory with all the images.
            transform (callable, optional): Optional transform to be
                applied
                on a sample.
        """
        self.root_dir = root_dir
        self.transform = transform
        self.archive = ZipFile(self.root_dir, 'r')

    def __len__(self):
        return 75600 #Total number of images

    def __getitem__(self, idx):
        img_data = ios.BytesIO(self.archive.read(str(idx)+'.jpg'))
        image = io.imread(img_data)

        if self.transform:
            image = self.transform(image)

        return image
```

```
70
71 #Load
72 out_dir = './Fingerprints_dataset' #saves dataset here
73 transform =  transforms.Compose([transforms.ToTensor(),
74         transforms.Normalize((0.5,),(0.5,))])
75 #Instance data, save and apply transform
76 fingerprints_data =  FingerprintsDataset(root_dir='Fingerprints.zip',
       transform=transform)
77 #Data loader feeds data as inputs and labels, of fixed batch size
78 train_loader = torch.utils.data.DataLoader(fingerprints_data, batch_size
       = 52, shuffle = True)
79
80 #Get Device
81 device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
82
83 #Define discriminator
84 def disc_model():
85     discriminator_model = nn.Sequential(
86       nn.Linear(275 *400, 2048),
87       nn.LeakyReLU(0.2),
88       nn.Dropout(0.3),
89       nn.Linear(2048, 1024),
90       nn.LeakyReLU(0.2),
91       nn.Dropout(0.3),
92       nn.Linear(1024 , 512),
93       nn.LeakyReLU(0.2),
94       nn.Dropout(0.3),
95       nn.Linear(512 , 256),
96       nn.LeakyReLU(0.2),
97       nn.Dropout(0.3),
98       nn.Linear(256 , 1),
99       nn.Sigmoid()
100      ).to(device)
101     return discriminator_model
102 #create a discriminator network
103 discriminator = disc_model()
104 #if we want to start from a checkpoint model
105 discriminator.load_state_dict(torch.load('f_name_lastV2.pth')['
       dis_state_dict'])
106 discriminator.eval()
107
108 #Define generator
109 def gen_model():
110     generator_model = nn.Sequential(
111           nn.Linear(100, 256),
112           nn.LeakyReLU(0.2),
113           nn.Linear(256 , 512),
```

```
            nn.LeakyReLU(0.2),
            nn.Linear(512 , 1024),
            nn.LeakyReLU(0.2),
            nn.Linear(1024 , 2048),
            nn.LeakyReLU(0.2),
            nn.Linear(2048 , 275 *400),
            nn.Tanh()
            ).to(device)
    return generator_model
#create a generator network
generator = gen_model()
#if we want to start from a checkpoint model
generator.load_state_dict(torch.load('f_name_lastV2.pth')['gen_state_dict
    '])
generator.eval()

#Noise generator
from torch.autograd import Variable
def rand_vecs(batch_size):
    return Variable(torch.randn(batch_size, 100)).to(device)

#Define optimizer and loss criterion
import torch.optim as optim
#create separate optimizers, use BCELoss for both networks
d_optimizer = optim.Adam(discriminator.parameters(), lr = 0.0002)
g_optimizer = optim.Adam(generator.parameters(), lr = 0.0002)
criterion = nn.BCELoss()

#Get the labels
def ones_or_zeros(batch_size, labels):
    if labels == 1:
        return Variable(torch.ones(batch_size, 1)).to(device)
    elif labels == 0:
        return Variable(torch.zeros(batch_size, 1)).to(device)

#TRAIN
#Create empty lists to catch losses
losses_disc, losses_generator = [], []
print_freq = 5 #fix how frequently to print losses
for epoch in range(100): #100 by default
    #Networks are training hence, .train()
    discriminator.train()
    generator.train()

#Train discriminator
    tot_inputs_passed=0
    for inputs in train_loader:
```

```
160
161         #.to(device)--> work in either cpu or gpu
162         inputs = inputs.to(device)
163
164         batch_size = inputs.size(0)
165         #Note: inputs.shape ->([52, 1, 28, 28])
166         #reshape to (batch_size, features) as expected by network
167         real_data = inputs.view(inputs.size(0), -1)
168
169         #Flush the retained gradients
170         d_optimizer.zero_grad()
171         # probabilities given real data
172         pred_real = discriminator(real_data)
173
174         #Computing loss for real data
175         loss_real=criterion(pred_real, ones_or_zeros(batch_size, 1))
176
177         #Note grads for gen are detached
178         fake_data = generator(rand_vecs(batch_size)).detach()
179
180         fake_data = fake_data.to(device)
181         # probabilities given fake data
182         pred_fake = discriminator(fake_data)
183
184         #Computing loss for generated data
185         loss_fake=criterion(pred_fake, ones_or_zeros(batch_size, 0))
186
187         loss_disc = loss_real + loss_fake  #combined loss
188         losses_disc.append(loss_disc.item()) # log the losses
189         #computes the gradients
190         loss_disc.backward()
191         #update parameters using gradients & optimizer rules
192         d_optimizer.step()
193
194         #discriminator completes a loop
195 #Train generator
196         #Trains from scratch, newly generated data are used
197         g_optimizer.zero_grad()
198         #generates image/feature vector
199         fake_data_new = generator(rand_vecs(batch_size))
200
201         fake_data_new = fake_data_new.to(device)
202
203         pred_fake_new = discriminator(fake_data_new)
204         #Generator's loss minimized after, labels are set to 1
205         loss_generator = criterion(pred_fake_new,   ones_or_zeros(
              batch_size, 1))
```

```
206
207         losses_generator.append(loss_generator.item())
208         loss_generator.backward()
209
210         g_optimizer.step() #completes training loop for generator
211
212         #Printing the Evolution:
213         #image=generator(rand_vecs(batch_size))[0]
214         #image_host=image.cpu()
215         #sample=image_host.view(-1, 275).detach().numpy()
216         #plt.axis('off')
217         #plt.grid(b=None)
218         #plt.imshow(sample)
219         #plt.show()
220
221         tot_inputs_passed+=1
222
223         print('\r',str(round(float(tot_inputs_passed)/len(train_loader)
                *100,2))+"% Epoch passed",end='')
224
225 #Print and save epoch, generator_state_dict as chk_point
226     if epoch % print_freq ==0:
227         print("loss_disc:{:.6}.., loss_generator: {:.5}".format(loss_disc
                , loss_generator))
228         torch.save({
229             'epoch': epoch,
230             'gen_state_dict': generator.state_dict(),
231             'dis_state_dict': discriminator.state_dict(),
232             }, ('f_name_' + 'lastV2' + '.pth'))
233         torch.save({
234             'epoch': epoch,
235             'losses': "loss_disc:{:.6}.., loss_generator: {:.5}".format(
                    loss_disc, loss_generator),
236             }, ('f_name_' + str(epoch) + '.txt'))
237         fig=plt.figure(figsize=(8, 12))
238         columns = 3
239         rows = 3
240         for i in range(1, columns*rows +1):
241             image=generator(rand_vecs(batch_size))[choice(range(
                    batch_size))]
242             image_host=image.cpu()
243             sample=image_host.view(-1, 275).detach().numpy()
244             #can't convert CUDA tensor to numpy. Use Tensor.cpu() to copy
                     the tensor to host memory first.
245             fig.add_subplot(rows, columns, i)
246             plt.axis('off')
247             plt.grid(b=None)
```

```
248             plt.imshow(sample)
249             plt.imsave('Epoch_'+str(epoch)+'_sample_'+str(i)+'.png',
                    sample)
250         plt.show()
251     else:
252         print("New epoch: "+str(epoch+1))
253
254 #Generating Images
255
256 generator=gen_model()
257 generator.load_state_dict(torch.load('f_name_lastV2.pth')['gen_state_dict
        '])
258 generator.eval()
259
260 fig=plt.figure(figsize=(8, 12))
261 columns = 3
262 rows = 3
263 for i in range(1, columns*rows +1):
264     image=generator(rand_vecs(52))[choice(range(52))]
265     image_host=image.cpu()
266     sample=image_host.view(-1, 275).detach().numpy()
267     fig.add_subplot(rows, columns, i)
268     plt.axis('off')
269     plt.grid(b=None)
270     im=plt.imshow(sample)
271     im.set_cmap('Greys')
272 plt.show()
273
274 image=generator(rand_vecs(52))[0]
275 image_host=image.cpu()
276 sample=image_host.view(-1, 275).detach().numpy()
277
278 plt.imsave('Final_Result.png', sample, cmap='Greys')
```

8.2 Iris CNN Layer Configuration

Discriminator										
Layer ini	Layer Fin	Wini	Wfin	Hini	Hfin	kernel_w	kernel_h	stride_w	stride_h	pad_w	pad_h
1	64	320	160	320	160	4	4	2	2	1	1
64	128	160	80	160	80	4	4	2	2	1	1
128	256	80	40	80	40	4	4	2	2	1	1
256	1	40	19	40	19	4	4	2	2	0	0
			1	Avg Pooling	1						

Generator										
Layer ini	Layer Fin	Wini	Wfin	Hini	Hfin	kernel_w	kernel_h	stride_w	stride_h	pad_w	pad_h
100	256	1	8	1	8	8	8	4	4	0	0
256	128	8	34	8	34	8	8	4	4	1	1
128	64	34	105	34	105	8	8	3	3	1	1
64	1	105	320	105	320	8	8	3	3	0	0

Table 8.1: CNN layers and its configuration for the Iris data-set

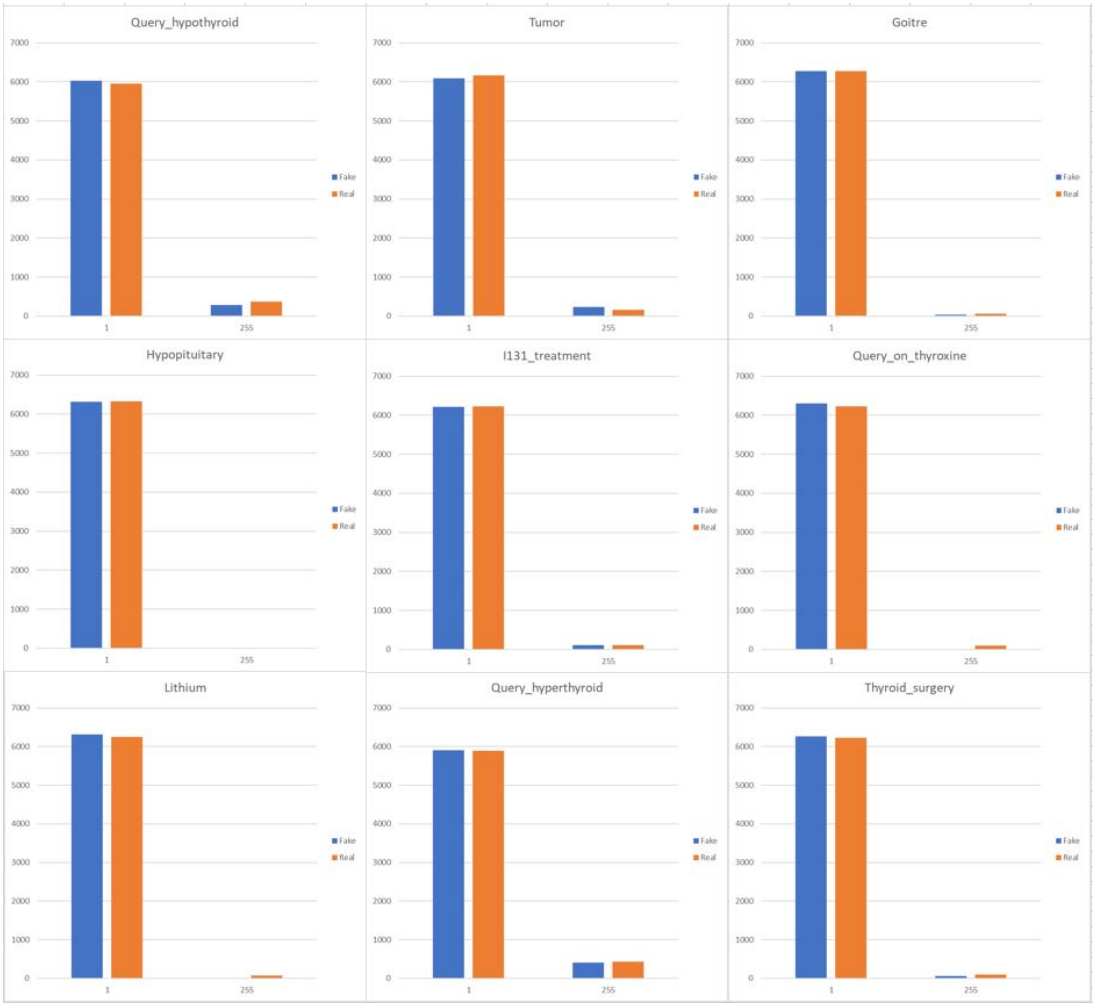


Figure 8.1: Thyroid's extra binary data features count by output value: comparison between fake generated data and original data. These are 9 of the 15 binary variables, the rest of the plots can be found at page 48.

8.3 Thyroid's Binary Extra Features

8.4 Cardiogram Data-Set Sample Trimming Process

Listing 8.2: Python's script for processing and trimming the data

```
1 import wfdb
2 import pandas as pd
3 import numpy as np
4
5 output_db=pd.DataFrame(columns=['signal'])
6
7 cardio_index=pd.read_csv('cardio_index.csv')
```

```
row=-1
chunk_size=2025

def remap(x,sample):
        A=-2
        B=2
        C=0
        D=255

        out=(x-A)/(B-A)*(D-C)+C

        if out>255:
            out=255
        if out<0:
            out=0

        return int(out)

for i in range(len(cardio_index)):
    folder=cardio_index.loc[i,'folder']
    file=cardio_index.loc[i,'file']
    record=wfdb.rdrecord(file,pb_dir='ptbdb/'+folder)
    sample_signal=record.p_signal[:,0]
    nr_chunks=len(sample_signal)/chunk_size
    if nr_chunks>=1:
        print(nr_chunks)
        limit_signal=int(len(sample_signal)-round(nr_chunks%1*chunk_size
            ,0))
        print(limit_signal)
        sample_signal=sample_signal[:limit_signal]
        n_chuncks=int(len(sample_signal)/chunk_size)
        splits=np.split(sample_signal,n_chuncks)
        #add splits as output_db instances
        for chunk in splits:
            row+=1
            row_sample=chunk.tolist()
            sample=[remap(x,row_sample) for x in row_sample]
            output_db.at[row,'signal']=sample
            print('row: '+ str(row))
            print('len chunck: '+str(len(chunk)))
    else:
        pass
output_db.to_csv('output_cardio_db_remapped.csv')
```

8.5 Cardio CNN Layer Configuration

Discriminator										
Layer ini	Layer Fin	Wini	Wfin	Hini	Hfin	kernel_w	kernel_h	stride_w	stride_h	pad_w	pad_h
1	64	45	42	45	42	6	6	1	1	1	1
64	128	42	20	42	20	6	6	2	2	1	1
128	256	20	9	20	9	6	6	2	2	1	1
256	1	9	2	9	2	6	6	3	3	0	0
			1	Avg Pooling	1						

Generator										
Layer ini	Layer Fin	Wini	Wfin	Hini	Hfin	kernel_w	kernel_h	stride_w	stride_h	pad_w	pad_h
100	256	1	8	1	8	8	8	2	2	0	0
256	128	8	19	8	19	7	7	2	2	1	1
128	64	19	41	19	41	7	7	2	2	1	1
64	1	41	45	41	45	7	7	1	1	1	1

Table 8.2: CNN layers and its configuration for the Cardio data-set

Acknowledgements

I should thank first of all to Cecilio Angulo for his interest and support during all the making process of this project. And especially his enthusiasm on the subject, which has awoke a great curiosity on me.

I also want to thank all the open source projects that have made this thesis possible by making its tools and knowledge available.

Finally, I would like to thank Laura and my loved ones for their unconditional support during this period.

Thank you.

References

[1] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014. URL `http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf`.

[2] Patrick L. Paper review on generative adversarial network(gan) part 1. *Medium*, 2017. URL `https://medium.com/@patrickhk/paper-review-on-generative-adversarial-network-gan-part-1-48597bcc96df`.

[3] Ricardo Guerrero Paul Bentley Roger Gunn Alexander Hammers David Alexander Dickie Maria Valdés Hernández Joanna Wardlaw Daniel Rueckert Christopher Bowles, Liang Chen. Gan augmentation: Augmenting training data using generative adversarial networks. *arXiv*, 2018. URL `https://arxiv.org/abs/1810.10863`.

[4] Shin HC. et al. Medical image synthesis for data augmentation and anonymization using generative adversarial networks. *Springer, Cham*, 2018. doi: https://doi.org/10.1007/978-3-030-00536-8_1.

[5] Ruqiang Yan Siyu Shao, Pu Wang. Generative adversarial networks for data augmentation in machine fault diagnosis. *Computers in Industry*, 106:85–93, 2019. doi: https://doi.org/10.1016/j.compind.2019.01.001.

[6] Yoshua Bengio Pierre Duhamel ClÃĭment Feutry, Pablo Piantanida. Learning anonymized representations with adversarial neural networks. *arXiv*, 2018. URL `https://arxiv.org/abs/1802.09386`.

[7] Sungroh Yoon Ho Bae, Dahuin Jung. Anomigan: Generative adversarial networks for anonymizing private medical data. *arXiv*, 2019. URL `https://arxiv.org/abs/1901.11313`.

[8] Benjamin Marlin Steven Cheng-Xian Li, Bo Jiang. Misgan: Learning from incomplete data with generative adversarial networks. *arXiv*, 2019. URL https://arxiv.org/abs/1902.09599.

[9] Mihaela van der Schaar Jinsung Yoon, James Jordon. Gain: Missing data imputation using generative adversarial nets. *arXiv*, 2018. URL https://arxiv.org/abs/1806.02920.

[10] Xinchen Yan Lajanugen Logeswaran Bernt Schiele Honglak Lee Scott Reed, Zeynep Akata. Generative adversarial text to image synthesis. *arXiv*, 2016. URL https://arxiv.org/abs/1605.05396v2.

[11] Joan Serrà Santiago Pascual, Antonio Bonafonte. SEGAN: Speech enhancement generative adversarial network. *arXiv*, 2017. URL https://arxiv.org/abs/1703.09452.

[12] J. Droppo J. Wu Y. Gong D. Yu, L. Deng and A. Acero. A minimum-mean-square-error noise reduction algorithm on melfrequency cepstra for robust speech recognition. *IEEE, ICASSP*, pages 4041–4044, 2008.

[13] T.M. O'Neil O. Vinyals P. Nguyen A.L. Maas, Q.V. Le and A.Y. Ng. Recurrent neural networks for noise reduction in robust asr. *INTERSPEECH*, pages 22–25, 2012.

[14] J. Ortega-Garcia and J. Gonzalez-Rodriguez. Overview of speech enhancement techniques for automatic speaker recognition. *Spoken Language*, pages 929–932, 1996.

[15] Nie D. et al. Medical image synthesis with context-aware generative adversarial networks. *Springer, Cham*, 10435, 2017. doi: https://doi.org/10.1007/978-3-319-66179-7_48.

[16] Munesh Lakhey. Generative adversarial networks demystified. *Medium, Data Driven Investor*, 2019. URL https://medium.com/datadriveninvestor/gans-demystified-f057f5e32fc9.

[17] Samuel A. Barnett. Convergence problems with generative adversarial networks (GANs), 2018. URL https://arxiv.org/abs/1806.11382.

[18] Jonathan Hui. Gan – why it is so hard to train generative adversarial networks! *Medium, Data Science*, 2018. URL https://medium.com/@jonathan_hui/gan-why-it-is-so-hard-to-train-generative-advisory-networks-819a86b3750b.