UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Master in Deep Learning for Audio and Video Signal Processing

MASTER THESIS

Learning supervised by synthetic data for Chest X-ray images

Author: Eric Morales Agostinho
Advisor: Juan Carlos San Miguel Avedillo

September 2022

Eric Morales Agostinho
Learning supervised by synthetic data for Chest X-ray images

Eric Morales Agostinho
C\ Francisco Tomás y Valiente Nº 11

PRINTED IN SPAIN

LEARNING SUPERVISED BY SYNTHETIC DATA FOR CHEST X-RAY IMAGES

Eric Morales Agostinho
Advisor: Juan Carlos San Miguel Avedillo

Video Processing and Understanding Lab
Departamento de Tecnología Electrónica y de las Comunicaciones
Escuela Politéctnica Superior
Universidad Autónoma de Madrid

June 2021

Trabajo parcialmente financiado por la Consejería de Educación e Investigación de la Comunidad de Madrid bajo el proyecto SI1/PJI/2019-00414 (Aiding diagnosis by self-supervised deep learning from unlabeled medical imaging)

Abstract

The use of Artificial Intelligence is changing our way of life. We can find it everywhere, from our virtual assistant making an appointment for us in a restaurant to algorithms that can diagnose diseases better and faster than a doctor. The two key problems with this type of technology are, firstly, the excessive computing power it needs and, secondly, the massive amount of data required to make it work properly. However, in the case of medical images, the focus of our research, there are even further difficulties, as we must take into account privacy issues along with the fact that right now, health organizations are not sharing information at all. The use of synthetic data can provide us with large datasets without any privacy issues at a reduced cost. To achieve that, we have developed a synthetic data generator based on Generative Adversarial Networks. Ideally, these artificially generated images should not contain sensitive personal information while maintaining statistical features similar to the original images. This way, a machine learning model will be able to learn from it. In our project, we demonstrate that this approach works with certain caveats: It requires the data to be representative enough - which in our simplified case it is. But, as the complexity increases, the algorithm struggles to solve the task. This indicates that the data generated by the GAN does not have enough statistical power to solve complex problems.

Keywords

Generative Models, Generative Adversarial Networks, GANs, synthetic data, medical images, x-ray, Unsupervised Domain Adaptation, Transfer Learning, privacy

Table of Contents

Lists

List of Equations

List of Figures

List of Tables

1 Introduction

1.1 Motivation

According to [Chen et al., 2021], a key challenge for applying AI in the medical field is the representativeness of the data employed for training AI models. Hence, it becomes essential to look for and to eliminate biases and errors in the trained models. These biases may be due to differences among the data captured in different hospitals, demographics (gender, ethnicity...), or simply by the class imbalance that is common in this type of data. In general, we need large datasets so representative data is less affected by biases. However, such datasets present an increasing privacy concern and therefore, rules and regulations are implemented to ensure only authorised individuals and organisations may access to the data.

The above-mentioned issues can be overcomed by using synthetic data, and one of the most widespread ways to generate such synthetic data are Generative Adversarial Networks (GANs), as can be seen in [Schütte et al., 2021], the baseline of this research work. However, we could use any other technique to generate this data, be it another type of generative model such as autoencoders [Wan et al., 2017] or even extracting images using medical simulators [Súja r et al., 2019]. Once we have these synthesised data generated with their corresponding labels, we can apply the dataset for supervised or self-supervised learning.

For supervised learning using synthetic medical data, it is typical to combine the available real and synthetic labelled data for increasing the accuracy of the trained model as compared to employing only real data [Çalli et al., 2021].

Self-supervised learning has been recently proposed in the context of medical imaging to address the limitations of data availability, annotation and privacy concerns. One type of approaches does self-supervised learning by training a model using just unlabelled data (or very few labelled data and the rest of the data being unlabelled) [Gazda et al., 2021]. This requires a pretext task (e.g. predict a given rotation), a task which if a model is trained to solve will also learn visual features that can be adapted to solve the real task. In a similar context of few labelled data, other alternative less explored is to perform Unsupervised Domain Adaptation (UDA). UDA may be applied as a two stage approach by doing fully

supervised training using only synthetic images and then performing domain adaptation with real data without annotations [Csurka, 2017].

These unsupervised learning techniques differ from supervised learning in that supervised learning uses real data that is all annotated, typically by a human. In contrast to unsupervised learning, which, as already explained, its labels are generated automatically using pretext tasks (self-supervised learning) or adapted from synthetic data (unsupervised domain adaptation).

1.2 Goals

The main objective of this thesis is to explore the use of synthetic data for self-supervised learning based on convolutional neural networks in the context of chest X-ray imaging [Çalli et al., 2021]. This objective is divided into the following stages:

1.– Study the available datasets for chest X-ray images [Irvin et al., 2019, Wang et al., 2017].

2.– Analyse the existing approaches to generate synthetic chest X-ray images [Schütte et al., 2021].

3.– Train and adapt a selected approach for generating synthetic chest X-ray images for different classes (i.e. pathologies).

4.– Analyse the existing approaches for unsupervised domain adaptation (i.e. domain transfer) from synthetic to real data for classification tasks [Csurka, 2017].

5.– Train and adapt a selected approach for unsupervised domain adaptation from synthetic to real data for classification tasks.

6.– Generate a large dataset of synthetic data based on the model developed in stage (3) with a dataset for chest X-ray images [Irvin et al., 2019, Wang et al., 2017].

7.– Study the utility of the generated synthetic data to do self-supervised learning by unsupervised domain adaptation based on synthetic data. This utility will be measured by carrying out experiments and ablation studies using the X-ray dataset not employed for the synthetic data generated, and later comparing it with self-supervised approaches [Gazda et al., 2021].

1.3 Project Structure

Chapter 1. Introduction. In this chapter, we describe the problem around which the whole project revolves, the motivation for the work and the structure of the project.

Chapter 2. State of the art. In this chapter, we collect the information that has served as a theoretical basis for this work. We analyse the latest advances in Generative Models, the generation of synthetic medical images and the use of Unsupervised Domain Adaptation in classification problems.

Chapter 3. Datasets. In this chapter, we present the datasets used, as well as an exhaustive analysis required to successfully carry out the project.

Chapter 4. Algorithm. In this chapter, we present the different algorithms used in each of the two parts of our work, synthetic data generation and Unsupervised Domain Adaptation (UDA).

Chapter 5. Evaluation. In this chapter, we detail the experiments carried out about the algorithms used, as well as the results obtained in each of them.

Chapter 5. Conclusions and future work. In this chapter, we summarize the conclusions of this work with a brief review of possible improvements that can be further investigated in the future.

2

State of the art

This chapter will give a short introduction to the different key points behind this work, from an overview of generative models to the use of synthetic medical images in machine learning and the unsupervised domain adaptation. It is divided into three sections. First, one dedicated to the fundamental theory needed to understand the project, basic concepts about Generative Models. Then one section explains the actual work related to the generation of synthetic medical images. And finally a section dedicated to explaining what is Unsupervised Domain Adaptation and why it is interesting in this project.

2.1 Fundamentals

2.1.1 Generative models

In order to create synthetic data we will use a type of model called "generative model", specifically Generative Adversarial Networks (GANs) [Goodfellow et al., 2014]. All the content of this section is based on the NIPS tutorial [Goodfellow, 2017].

The term refers to any model that uses a train set that follows a distribution ρ_{data} learns how to estimate that distribution, resulting in a probability distribution ρ_{model}. There are two types of generative models, the ones that estimates ρ_{model} explicitly (Figure 2.1) and the ones that are able to generate samples from ρ_{model} (Figure 2.2). GANs can be both but are focused primarily on the second one, sample generation.

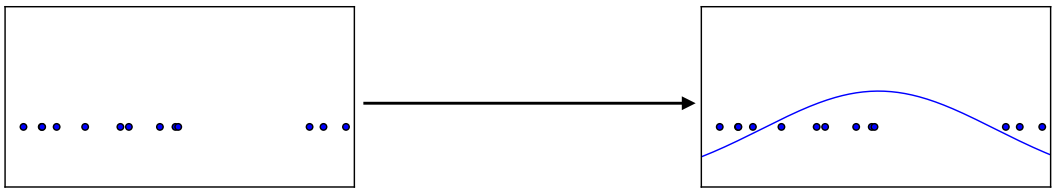


Figure 2.1: Example of a generative model that estimates Gaussian density distribution [Goodfellow, 2017].

The GANs are designed to solve many of other generative models problems:

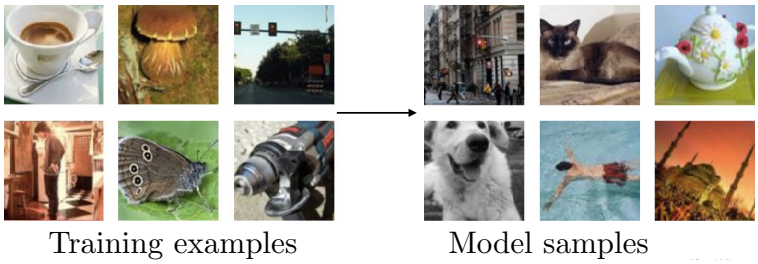


Figure 2.2: Example of a generative model that generates samples from the model distribution [Goodfellow, 2017]. Images extracted from ImageNet dataset [Deng et al., 2009, Deng et al., 2010, Russakovsky et al., 2015].

- They are able to generate samples in parallel.
- The generator architecture has almost no restrictions.
- They do not need Markov chains.
- They do not need variational bound.
- The results generated by GANs are subjectively better.

On the other hand, the GANs have a new disadvantage, the training is more difficult because it requires finding the Nash equilibrium [Ratliff et al., 2013], which is not as easy as optimizing a cost function.

Generative Adversarial Networks

The idea of the GANs can be represented as a MiniMax game [Rivest, 1987]. The two players are, on one hand, the **generator**, which is creating samples trying to follow the original distribution of the training date. On the other hand, the **discriminator** is trying to detect if the sample received is real (from the dataset) or fake (made by the generator). The generator is trained with the aim of fooling the discriminator and the discriminator is trained as a traditional binary classifier (real or fake) using supervised learning. More detail about the process is in Figure 2.3.

The two players can be represented as functions, G is the generator, that takes z as input and depends on the parameters $\theta^{(G)}$. D is the discriminator, that takes as input x or $G(z)$ and depends on the parameters $\theta^{(D)}$.

The cost function depends on the parameters of both players, $\theta^{(D)}$ and $\theta^{(G)}$. As the discriminator is being trained as a traditional binary classifier we can use the standard cross-entropy function as cost (Equation 2.1) but using two sources of data, one is the dataset, always with label 0, and the other is the generator, always with label 1. The objective of the discriminator is to minimize the cost function.

$$J^{(D)}(\theta^{(D)}, \theta^{(G)}) = -\frac{1}{2}E_{x\sim p_{data}} logD(x) - \frac{1}{2}E_z log(1 - D(G(x))). \tag{2.1}$$

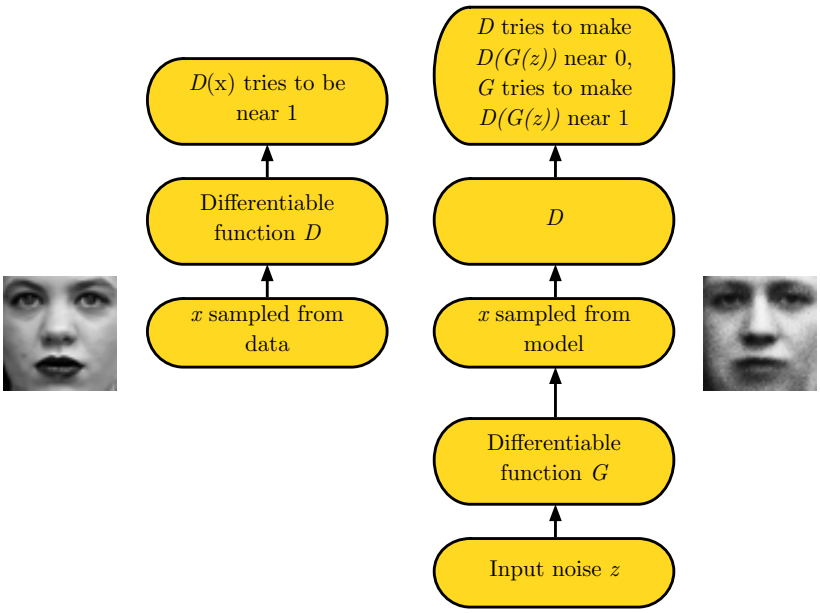


Figure 2.3: GAN framework process. D is the discriminator and G the generator, x are training examples and z is random noise. The goal of the discriminator is to predict zero when the sample has been generated by G ($D(G(z)) \simeq 0$) and one if it is real ($D(x) \simeq 1$). The goal of the generator is to try to make the discriminator predict one when the sample has been generated by G ($D(G(z)) \simeq 1$) [Goodfellow, 2017].

As mentioned, a GAN is simply a MiniMax game between the discriminator and the generator, that type of games are also called **zero-sum game**, where the sum of all player scores should be zero. With this information, we can conclude that the cost function of the generator is simply the opposite of the discriminator (Equation 2.2).

$$J^{(G)} = -J^{(D)} \tag{2.2}$$

About the *Nash equilibria* [Ratliff et al., 2013], in this context and according to [Goodfellow, 2017], a Nash equilibrium is the tuple $(\theta^{(D)}, \theta^{(G)})$ where players are tied, which is a local minimum of J(G) with respect to $\theta^{(G)}$ and a local minimum of J(D) with respect to $\theta^{(D)}$.

Conditional GANs (cGAN)

Presented by [Denton et al., 2015], this model is based on a normal GAN structure but incorporates class labelling of both the generator and the discriminator. More detail about the structure in the Figure 2.4.

AC-GANs

Based on the cGAN architecture, [Odena et al., 2017] proposed a new one with the main difference that the discriminator does not receive the class label, it is trained with an additional classification loss to predict it. More detail in the Figure 2.4.

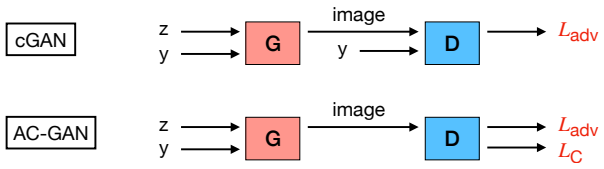


Figure 2.4: cGAN and AC-GAN architectures. [Frolov et al., 2021]

Progressive GANs

One of the main challenges for GANs is the generation of high-resolution images, which is easier to classify between real and generated images using higher resolutions, with a lot of detail [Odena et al., 2017]. To address this problem, [Karras et al., 2018] proposes the concept of Progressive Generative Adversarial Network, a training methodology for GANs based on the idea of progressively increasing the number of layers during the training process. In this way, in the beginning, the network learns simple structures of plain objects, where a smaller layer size is needed, and at the end learns the textures and details, where a bigger layer size is needed. The structure can be seen in the Figure 2.5.

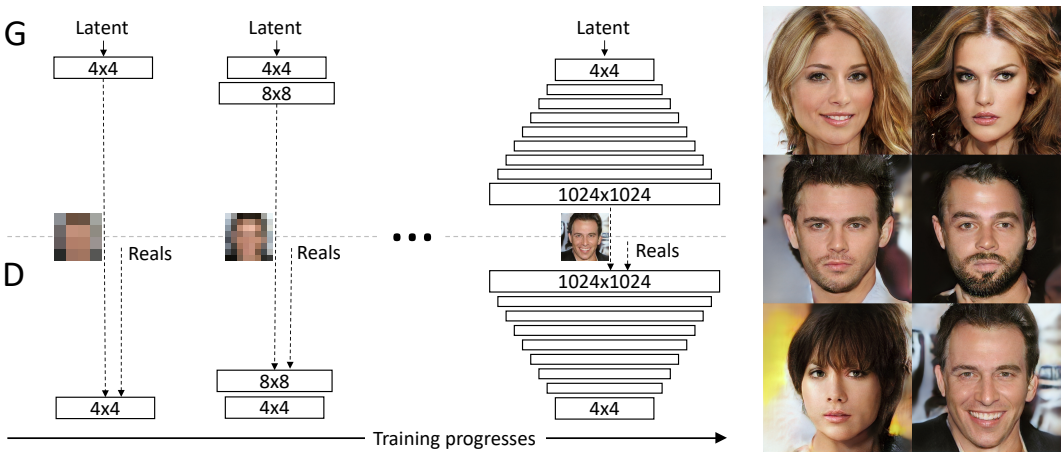


Figure 2.5: Progressive GAN training process. Starting having a low resolution of 4x4 that is being increased during the training process [Karras et al., 2018].

On each step the resolution is increased (doubled) by adding new bigger layers smoothly, fading the existing ones with it. This process is controlled by a parameter α that starts from 0 and linearly increases to 1. More detail about these processes is in the Figure 2.6.

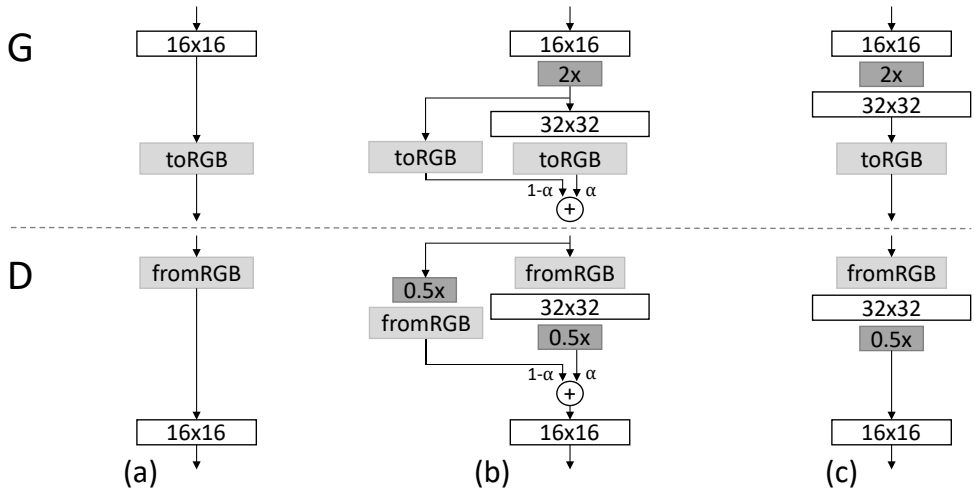


Figure 2.6: ProgGAN resolution increase process [Karras et al., 2018].

cpD-GAN

Focusing on the field of the project, the use of medical images, we have discovered a new structure, **cpD-GAN**. Proposed by [Schütte et al., 2021] it is a model developed just to improve the performance on his benchmark. This model is based on the prog-GAN explained above but with some improvements. The main difference, influenced by StyleGAN [Karras et al., 2019, Karras et al., 2020], is that they dropped the progressive growth, that way it is possible to experiment with different architectures. The best finding was the use of standard residual connections in the discriminator and output skip connections in the discriminator. Also, is important to note that, as a CondictionalGAN, it allows us to generate images "a la carte", selecting the classes as input.

Unfortunately, they don't give much detail about this model in the article, only that it improves performance over ProgGAN, so part of our work will be to evaluate it and try to understand it in more detail.

2.2 Synthetic medical images

According to [Chen et al., 2021]. As the use of AI in the medical field expands, so do regulations and clinical analyses. This type of analysis is essential to look for and eliminate biases and errors in the models. These biases may have been caused by differences between teams from different hospitals, or simply by the class imbalance that is common in this type of dataset. In general, we need large datasets, with more representative data and avoiding ethical biases (gender, ethnicity...).

The use of synthetic medical data is increasing to alleviate the lack of real medical annotated data. In other fields, for example, self-driving cars, simulators are being used to simulate accidents or things that are difficult to capture in real life. However, as with everything else, there are good and bad uses

we can make of this technology, for example, people are trying to make Deep Fakes spread fake news and misinform the population [Chesney and Citron, 2019].

Additionally, if we talk in general about the use of AI in the medical field, we can see that it is growing all the time. For example, the United States Food and Drug Administration (FDA) is close to approving an AI-based software as a medical device (AI-SaMD), used for example to detect atrial fibrillation [Food et al., 2019].

One of the most promising areas of AI for improvement in this field is generative modelling, in particular GANs, as can be seen in [Schütte et al., 2021], the baseline of this research work.

2.2.1 Medical image generation

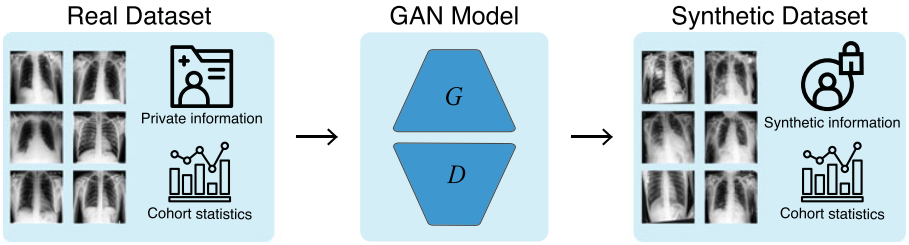


Figure 2.7: General structure where the GAN is used to remove private information [Schütte et al., 2021].

The paper [Schütte et al., 2021] focuses on the idea of removing private information of the patients using synthetic images, the author says that the synthetic images ideally have, in aggregate, similar statistical properties to those of a source dataset but do not contain sensitive personal information (Figure 2.7).

About the models, they use a ProgGAN (section 2.1.1 and [Karras et al., 2018]) and propose cPD-GAN (section 2.1.1), an architecture based on ProgGAN.

Moreover, one of the best contributions of this paper is the benchmark they propose. They provide all the implementation with which you can test the different models they propose and replicate all the experiments. We will talk about that in detail in the section 4.1.

The training of the models can be done using this benchmark, and it is important to note that the training is automatically stopped when the quality improvement has converged, that is measured using the Fréchet Inception Distance (FID) score [Heusel et al., 2017].

2.3 Domain Adaptation

In the context of machine learning, domain adaptation is seen as a special case of transfer learning. Where, transfer learning is the broader research field focused on training a model on a source domain or task in order to extrapolate to a different but related target domain or task, where either the tasks or domains (or both) differ.

Following the notations of [Pan and Yang, 2010]: Let $\mathbb{D}$ be a domain, composed of a d-dimensional feature space $X \in R^d$ with a marginal probability distribution $P(X)$. Then let $\mathbb{T}$ be a task defined by the ground truth space $\mathbb{Y}$ and the conditional probability distribution $P(Y|X)$, where $X \sim \mathbb{X}$ and $Y \sim \mathbb{Y}$ are random variables. In the context of computer vision, $x \in X$ is typically an RGB image and $y \in Y$ is the ground truth of x. $P(Y|X)$ can be inferred from $X, \{x_1, ..., x_n\}$ of $X \sim \mathbb{X}$, with their corresponding labels $Y, \{y_1, ..., y_n\}$ from $Y \sim \mathbb{Y}$ in a supervised manner.

Transfer Learning is the scenario where two different domains or tasks can be distinguished: $\mathbb{D}^s = \{X^s, P(X^s)\}, \mathbb{T}^s = \{Y^s, P(Y^s|X^s)\}$ the source domain, and $\mathbb{D}^t = \{X^t, P(X^t)\}, \mathbb{T}^t = \{Y^t, P(Y^t|X^t)\}$ the target domain. When $\mathbb{D}^s \neq \mathbb{D}^t$ or $\mathbb{T}^s \neq \mathbb{T}^t$, the models trained on $\mathbb{D}^s$ tend to have a drop in performance when tested on $\mathbb{D}^t$ or are not applicable if $\mathbb{T}^s \neq \mathbb{T}^t$.

Based on these definitions, [Pan and Yang, 2010] categorizes different transfer learning scenarios into: Inductive Transfer Learning, Transductive Transfer Learning and Unsupervised Transfer Learning. Inductive Transfer Learning refers to having different source and target tasks. It requires some labeled target samples for the model to extrapolate to the target domain. This is the most common scenario in computer vision, where models pre-trained for image classification on *Imagenet* [Deng et al., 2009] are used to extrapolate to different tasks such as object detection or semantic segmentation. Transductive Transfer Learning is the scenario where source and target tasks are the same, while the source and the target domains are different. Domain adaptation is an special case of Transductive Transfer Learning where source and target data representations are different but both share the same task. Finally, unsupervised Transfer Learning similar to Transductive Transfer Learning the target task is different but related to the source task. However, unsupervised transfer learning focuses on solving unsupervised tasks, such as clustering and dimensionality reduction.

On this project we will focus on domain adaptation methods, which according to this classification belong to transductive transfer learning solutions, i.e. different target (real) and source (synthetic) data but we will keep the same task (in this case classification), $\mathbb{T}^s = \mathbb{T}^t$.

2.3.1 Unsupervised Domain Adaptation

Domain Adaptation (DA) can be divided into supervised and unsupervised domain adaptation depending on the availability of target labels. Our focus in this work is Unsupervised Domain Adaptation (UDA).

To give a simple example of this UDA concept we can use the Office-31 dataset [Koniusz et al., 2017]. A dataset that provides us with images of products in three different domains, one of them is directly using images from Amazon, another one are photos taken with a professional DSLR camera and the last ones are photos taken with a webcam (Figure 2.8. The objective is to train a model using for example the Amazon images, which are easier to get already labelled and this model works with images taken by a camera without labelling. This is an example where Unsupervised Domain Adaptation is used.

Figure 2.8: Office-31 dataset sample [Koniusz et al., 2017].

DCAN

Typically DA is performed by aligning features between images of both domains. By aligning the final features, weights throughout the network are optimized to produce domain-agnostic features. However, DCAN [Li et al., 2020] argues that alignment of low level features can reduce the network capabilities on the target domain, thus, they propose to employ attention throughout the network to provide generality while maintaining specificity

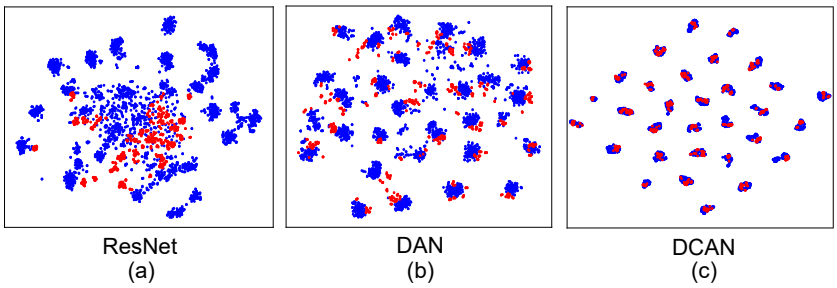


Figure 2.9: Domain Adaptation t-SNE feature representation [Li et al., 2020].

In Figure 2.9 we can see an example of t-SNE representations extracted from different networks, (a) using just ResNet, (b) using just DAN and (c) DCAN, we can observe that ResNet has lost the ability to classify because of the gap between domains, while DCAN produces very compact clusters for each class, with an almost perfect alignment.

Finally, in the paper, they show some results using the dataset, Office-31. As just mentioned, the dataset has 3 domains Amazon (A), DSLR (D) and Webcam (W), that can be combined in 6 possible domain adaptation tasks (A→ D, D→ A...). As we can see in Table 2.1, the results obtained by DCAN are the best in most of the cases (except for two), on average it obtains the best results.

Methods	ResNet	JDDA	DAN	RTN	DANN	ADDA	MADA	GTA	MCD	iCAN	DAAA	CDAN	DSBN	TADA	SymNets	MDD	**DCAN**
A→W	68.4	82.6	80.5	84.5	82.0	86.2	90.0	89.5	88.6	92.5	86.8	94.1	92.7	94.3	90.8	94.5	**95.0**
D→W	96.7	95.2	97.1	96.8	96.9	96.2	97.4	97.9	98.5	98.8	**99.3**	98.6	99.0	98.7	98.8	98.4	97.5
W→D	99.3	99.7	99.6	99.4	99.1	98.4	99.6	99.8	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	99.8	**100.0**	**100.0**	**100.0**
A→D	68.9	79.8	78.6	77.5	79.7	77.8	87.8	87.7	92.2	90.1	88.8	92.9	92.2	91.6	**93.9**	93.5	92.6
D→A	62.5	57.4	63.6	66.2	68.2	69.5	70.3	72.8	69.5	72.1	74.3	71.0	71.7	72.9	74.6	74.6	**77.2**
W→A	60.7	66.7	62.8	64.8	67.4	68.9	66.4	71.4	69.7	69.9	73.9	69.3	74.4	73.0	72.5	72.2	**74.9**
Avg	76.1	80.2	80.4	81.6	82.2	82.9	85.2	86.5	86.5	87.2	87.2	87.7	88.3	88.4	88.4	88.9	**89.5**

Table 2.1: DCAN Results in Office-31 Dataset [Li et al., 2020].

3

Datasets

As is common in this type of project, a considerable amount of time has been spent on research around datasets. We have therefore decided to dedicate a chapter just to analysing the datasets we are going to use in detail and how we are going to do it.

3.1 Chexpert

CheXpert [Irvin et al., 2019] is the dataset used in the paper [Schütte et al., 2021], and the dataset that we have used to train the GANs. The dataset has 224,316 chest radiographs to which 14 binary labels are associated according to the diseases detected.

These labels have been extracted from doctors' reports (plain text), which have been passed through a natural language processing algorithm specialised in detecting whether each of the diseases in the labels had been diagnosed or not. Table 3.1 shows an example of how this labeller works. This is a very good way to extract labels from plain text, but it has some loopholes since each doctor writes the diagnoses differently. For example, some doctors may not mention that the patient is healthy and at the same time not mention any disease, in this case when we extract the labels we will find an array filled with only zeros (even a zero in No Finding), which we do not know what it means. As we will see in the next section this is a real example that happens in this dataset, but there are many others, such as a sub-label being tagged but not the parent label, and surely other problems that we have not yet detected.

As is common in medical datasets, where some diseases may have more representation than others, we find a very unbalanced distribution of classes, as can be seen in Table 3.2. The problem is even more serious than what the Table shows, because as we will see in the analysis section, in our case we will use different combinations of labels, and that is where huge imbalances occur.

It is important to note that, as we can see in Figure 3.1, the same sample can have more than one label this is the most common. Figure 3.1 shows the organisation of the labels in this dataset, in which we can observe superclasses (Lung Opacity) and subclasses within them (Lung Lesion).

	Observation	Labeler Output
1. *unremarkable* cardiomediastinal silhouette	No Finding	
	Enlarged Cardiom.	0
	Cardiomegaly	
2. diffuse reticular pattern, which can be seen with an atypical infection **or** chronic fibrotic change. *no* focal consolidation.	Lung Opacity	1
	Lung Lesion	
	Edema	
	Consolidation	0
3. *no* pleural effusion or pneumothorax	Pneumonia	u
	Atelectasis	
	Pneumothorax	0
	Pleural Effusion	0
4. mild degenerative changes in the lumbar spine and old right rib fractures.	Pleural Other	
	Fracture	1
	Support Devices	

Table 3.1: Output of labeller example in CheXpert dataset [Irvin et al., 2019].

Pathology	Positive (%)	Uncertain (%)	Negative (%)
No Finding	16627 (8.86)	0 (0.0)	171014 (91.14)
Enlarged Cardiom.	9020 (4.81)	10148 (5.41)	168473 (89.78)
Cardiomegaly	23002 (12.26)	6597 (3.52)	158042 (84.23)
Lung Lesion	6856 (3.65)	1071 (0.57)	179714 (95.78)
Lung Opacity	92669 (49.39)	4341 (2.31)	90631 (48.3)
Edema	48905 (26.06)	11571 (6.17)	127165 (67.77)
Consolidation	12730 (6.78)	23976 (12.78)	150935 (80.44)
Pneumonia	4576 (2.44)	15658 (8.34)	167407 (89.22)
Atelectasis	29333 (15.63)	29377 (15.66)	128931 (68.71)
Pneumothorax	17313 (9.23)	2663 (1.42)	167665 (89.35)
Pleural Effusion	75696 (40.34)	9419 (5.02)	102526 (54.64)
Pleural Other	2441 (1.3)	1771 (0.94)	183429 (97.76)
Fracture	7270 (3.87)	484 (0.26)	179887 (95.87)
Support Devices	105831 (56.4)	898 (0.48)	80912 (43.12)

Table 3.2: Number of samples per label in CheXpert dataset [Irvin et al., 2019].

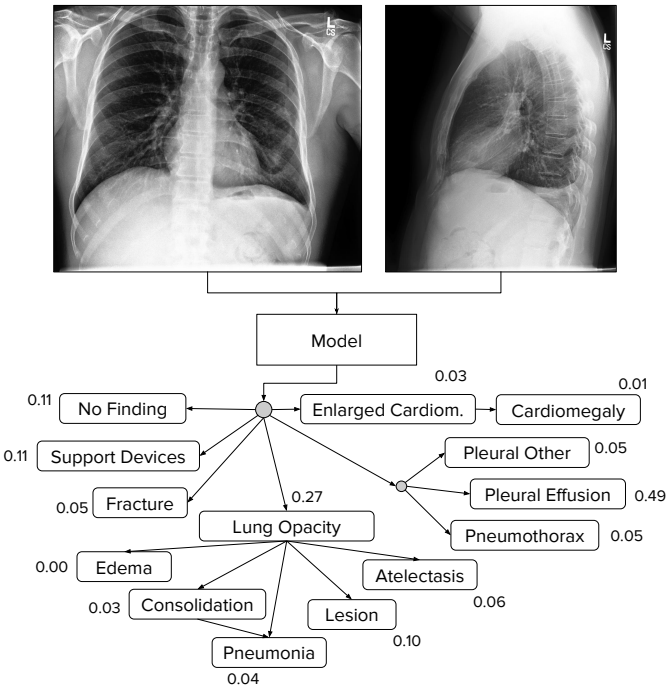


Figure 3.1: CheXpert classes and subclasses [Irvin et al., 2019].

3.2 Chest8

ChestX-ray8 [Wang et al., 2017] is very similar to the previous one it has some labels in common and they have been extracted in the same way, by analysing medical analyses with a computer application. This time we only have 108,948 samples and 9 labels.

As in the previous one, the labels have been extracted using Natural Language Processing algorithm, so it can carry some errors. In this article, they also presented a figure to analyze the co-occurrence of each keyword (Figure 3.2) in which they show which diseases are usually shown together with which diseases because this dataset is also multi-label. In the Figure can be seen how most of the diseases can appear alone, but for example, Pneumonia almost always appears with other diseases. Also can be studied the most common combinations, for example, Mass with Atelectasis, Effusion and Infiltration, and some others like Infiltration with Effusion and Atelectasis. By the way, it is important to remark that by analyzing the images we have found that every disease can be found alone, and we will use that in our project. That figure in general gives us an idea of the difficulty of the challenge that we are facing.

Because it is so similar to CheXpert and has some tags in common (and therefore to the generated synthesised data) it is a good candidate to be used to apply Domain Adaptation, which is why we have chosen this dataset.

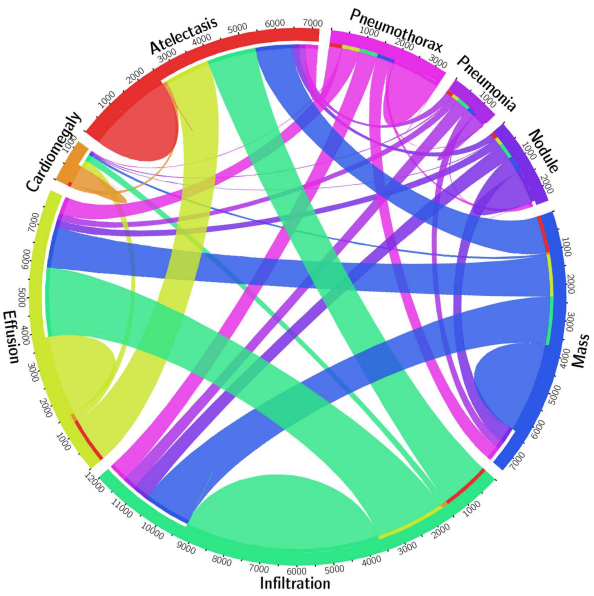


Figure 3.2: Labels' co-occurrence in ChestX-ray8 dataset [Wang et al., 2017].

3.3 Data Analysis

As we have explained in section 2.3, we are going to face the problem as a Domain Adaptation problem, so the tasks must be the same. Unfortunately these datasets do not have exactly the same labels, so we will have to make a selection in order to use them.

Table 3.3 shows the label matches between each of the datasets. There are some that have a 1:1 match, others are not so clear (e.g. *Effusion* vs *Pleural Effusion*) and others have no match at all (e.g. *Support Devices*).

Chexpert	**ChestX-ray8**
No Finding	**Normal**
Enlarged Cardiom.	
Cardiomegaly	**Cardiomegaly**
Lung Opacity	
Lung Lesion	
Edema	
Consolidation	
Pneumonia	**Pneumonia**
Atelectasis	Atelectasis
Pneumothorax	**Pneumothorax**
Pleural Effusion	Effusion*
Pleural Other	
Fracture	
Support Devices	
	Infiltration
	Mass
	Nodule

Table 3.3: Labels comparison between datasets. In bold the labels chosen to be used in the project.

With all this we can make different selections of the data, to keep only the labels we are interested in. In this work we have focused on three selections:

- Version 1: **binary** classification of the two majority classes, *No Finding* vs *Pneumothorax*.
- Version 2: **binary** classification in search of any disease, *No Finding* VS *All*.
- Version 3: A classification of **four** classes, one from each subgroup given by Chexpert (Table 3.3), *No Finding* VS *Cardiomegaly* VS *Pneumonia* VS *Pneumothorax*.

To make each of these selections we can use different techniques, depending on whether we want to take into account the other unselected labels or not. This is an important decision since, as we have seen in the two previous sections, both datasets suffer from the co-occurrence of labels. Taking this into

account, two versions appear for each of the three versions just mentioned. One version is to force all other (unselected) labels to be zero (version with 0s) and another version is to force only the selected labels being zero, without worrying about the others (don't care, version with Xs), more detail about the different versions in the following subsections. As is evident, the first problem is much simpler than the second one, as it focuses on finding only the selected diseases, in the second one you must be able to find a certain disease in a patient who may also suffer from other diseases.

To be able to work with so many versions of each dataset we had two options. The first one is to make a copy modifying the files for each one of the versions, which we discarded because of the disk space that this would suppose. And on the other hand, the second option, which consists in using always the same dataset but creating different CSVs that allow us to read only the images that we are interested in, this is the one we have chosen. To be able to do this we have developed a custom `torch.utils.data.Dataset` [1] that using *pandas library* is reading the data of the requested version.

To conclude this section, we have made an exhaustive analysis of each of the versions, in particular, we have focused on CheXpert as it is the one with which the GAN is trained and it seems that it may be of more importance.

Version 1: Binary - No Finding VS Pneumothorax

This is the simplest version, the one we have experimented with the most and the one that has worked best in general, in particular the version with zeros. The question that could rise is: why Pneumothorax? and the answer is simply that it is the more represented label when we use the version with zeros. In the Figure 3.3 can be seen the labels values for each of the classes in this version.

About the version with zeros there is not much to comment, either it has only the Pneumothorax label or it has the No Finding label. As expected the result is a very unbalanced dataset (there are many more healthy patients than with Pneumothorax).

On the other hand, we have the version with Xs, which, although it may not seem like it, make it a much more difficult problem. The Xs mean that we will have to be able to discern between healthy patients (with or without support devices) and patients with Pneumothorax who may or may not have any other disease. This is very complicated, because, first of all, many doctors only label the main disease, leaving out others, for example they detect a pneumonia and a pacemaker, and in the report they do not mention that there was also a pneumothorax, because it was hardly noticeable and it was not the worst thing that happened to that patient. Moreover, as we will see later, it is quite difficult to detect any of these images, a problem that is aggravated if we add a lot of noise in the form of other diseases.

[1] *Pytorch Datasets*, Pytorch, accessed 2022-05-16. https://pytorch.org/docs/stable/data.html

In the Figures 3.5 and 3.4 we can see the label distribution of CheXpert that we have on each class in the Version with Xs. In the case of the label No Finding we can see that it only can appear with the label Support Devices (SD) or alone. However in the case of the label Pneumothorax, it allows 380 possible combinations, the most common ones are alone, combined with Support Devices or Lung Opacity, but others only appear one time, for example, the combination of Cardiomegaly (C), Lung Opacity (LO), Lung Lesion (LL), Edema (Ed), Consolidation (Co), Atelectasis (A) and Support Devices (SD). Taking all this into account we can deduce the reason why when using the version with Xs we face such a difficult problem, the system has to be able to diagnose Pneumothorax mixed with all the other diseases, and the combinations are endless.

```
+ Version with 0s:
      No Finding      ->  (10000000000000)
      Pneumothorax    ->  (01000000000000)
+ Version with Xs:
      No Finding      ->  (10XXXXXXXXXXXX)
      Pneumothorax    ->  (01XXXXXXXXXXXX)
```

Figure 3.3: Different possibilities of Version 1. Each position in the binary array belongs to a label, a zero means that this disease has not been detected, a one means that it has been detected and an X means that it doesn't care, you choose the sample whether this disease has been detected or not.

	Pneumothorax	No Finding	EC	Ca	LO	LL	Ed	Co	Pn	At	PE	PO	Fr	SD	count
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	9553
1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	7421

Figure 3.4: Label distribution in CheXpert version 1 with Xs and No Finding = 1.

Version 2: Binary - No Finding VS All

At first glance it may seem that this is the simplest version because the problem can be simplified to anomaly detection, but nothing could be further from the truth. We have a problem very similar to the one explained before, the version with Xs, but in this case we have one class alone and another one with even more combination of diseases. As we can see in the Figure 3.6 we have 720 possible combination of diseases, where the most common one is Lung Opacity alone, and the second one, that without any doubt will be omitted because is basically No Finding without the No Finding label, errors of the dataset. In the Figure 3.7 can be seen the labels values for each of the classes in this version.

	Pneumothorax	No Finding	EC	Ca	LO	LL	Ed	Co	Pn	At	PE	PO	Fr	SD	count
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	2274
1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	2213
2	1	0	0	0	1	0	0	0	0	0	0	0	0	1	1938
3	1	0	0	0	1	0	0	0	0	0	1	0	0	1	1645
4	1	0	0	0	1	0	0	0	0	0	0	0	0	0	940
5	1	0	0	0	1	0	0	0	0	0	1	0	0	0	671
6	1	0	0	0	0	0	0	0	0	0	1	0	0	1	647
7	1	0	0	0	0	0	0	0	0	1	0	0	0	1	528
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
372	1	0	0	0	1	1	0	1	0	1	1	0	0	0	1
373	1	0	0	1	0	0	0	0	0	0	1	0	1	0	1
374	1	0	0	1	1	0	0	0	0	0	1	0	1	0	1
375	1	0	0	1	1	0	1	0	1	0	1	0	0	0	1
376	1	0	0	0	1	0	1	0	0	1	0	1	0	1	1
377	1	0	1	1	0	0	0	0	0	0	1	0	1	0	1
378	1	0	0	0	0	0	1	0	0	1	0	1	0	1	1
379	1	0	0	1	1	1	1	1	0	1	0	0	0	1	1

Figure 3.5: Label distribution in CheXpert version 1 with Xs and Pneumothorax = 1.

	Pneumothorax	No Finding	EC	Ca	LO	LL	Ed	Co	Pn	At	PE	PO	Fr	SD	count
0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	8446
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7766
2	0	0	0	0	1	0	0	0	0	0	1	0	0	0	5972
3	0	0	0	0	0	0	0	0	0	0	1	0	0	0	3570
4	0	0	0	0	0	0	1	0	0	0	0	0	0	0	3080
5	0	0	0	0	1	0	1	0	0	0	1	0	0	0	2329
6	1	0	0	0	0	0	0	0	0	0	0	0	0	0	2274
7	0	0	0	0	1	0	1	0	0	0	0	0	0	0	2202
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
713	0	0	1	0	0	1	0	0	1	0	0	0	0	0	1
714	0	0	1	0	1	1	0	0	1	0	0	0	0	0	1
715	0	0	0	1	1	1	1	0	1	0	0	0	0	0	1
716	0	0	0	0	1	1	1	0	1	0	0	0	1	0	1
717	0	0	1	0	1	1	0	0	1	0	1	1	0	0	1
718	1	0	0	0	1	1	0	0	1	0	1	0	0	0	1
719	0	0	0	1	1	1	1	0	1	0	1	0	0	0	1
720	0	0	0	1	1	1	0	1	1	1	1	0	0	0	1

Figure 3.6: Label distribution in CheXpert version 2 with Xs and Finding = 1.

```
+ Only possible version:
      No Finding      ->   (10000000000000)
      All             ->   (0XXXXXXXXXXXXX)
```

Figure 3.7: Only possibility of Version 2. Each position in the binary array belongs to a label, a zero means that this disease has not been detected, a one means that it has been detected and an X means that it doesn't care, you choose the sample whether this disease has been detected or not.

Version 3: Four classes - No Finding VS Pneumothorax VS Pneumonia VS Cardiomegaly

Finally, this third version is very similar to version 1, only it is an even more complicated problem. Not only does the algorithm have to deal with all the difficulties mentioned above, but it also has to be able to distinguish the selected diseases, which is impossible to non-experts. In the Figure 3.12 can be seen the labels values for each of the classes in this version.

About the version with Xs, as can be seen in the Figure 3.8, the No Finding label distribution is similar to the version 1. On the other hand if we observe the Figures 3.9, 3.10 and 3.11, we can see that they are all similar to the previous versions, where, except pneumothorax (whose most repeated label distribution is being alone), the disease are most commonly found in combination with other diseases, rarely appearing alone.

	Fr	LL	Ed	PO	Co	SD	PE	At	No Finding	EC	LO	Cardiomegaly	Pneumonia	Pneumothorax	count
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	9553
1	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	7421

Figure 3.8: Label distribution in CheXpert version 3 with Xs and No Finding = 1.

	Fr	LL	Ed	PO	Co	SD	PE	At	No Finding	EC	LO	Cardiomegaly	Pneumonia	Pneumothorax	count
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	2274
1	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	2213
2	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	1938
3	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	1645
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	940
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
242	0.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	1
243	0.0	0.0	0.0	0.0	1.0	1.0	0.0	1.0	0.0	1.0	1.0	0.0	0.0	1.0	1
244	0.0	1.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	1
245	0.0	1.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	1.0	0.0	0.0	0.0	1.0	1
246	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1

Figure 3.9: Label distribution in CheXpert version 3 with Xs and Pneumothorax = 1.

	Fr	LL	Ed	PO	Co	SD	PE	At	No Finding	EC	LO	Cardiomegaly	Pneumonia	Pneumothorax	count
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	720
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	423
2	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	303
3	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	230
4	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	188
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
182	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	1
183	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	1.0	0.0	1
184	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	1
185	0.0	0.0	1.0	1.0	1.0	0.0	1.0	1.0	0.0	0.0	1.0	0.0	1.0	0.0	1
186	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	1

Figure 3.10: Label distribution in CheXpert version 3 with Xs and Pneumonia = 1.

	Fr	LL	Ed	PO	Co	SD	PE	At	No Finding	EC	LO	Cardiomegaly	Pneumonia	Pneumothorax	count
0	0.0	0.0	1.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	1784
1	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	1579
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1539
3	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1439
4	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1308
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
295	0.0	1.0	1.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	1.0	1.0	0.0	0.0	1
296	0.0	0.0	0.0	1.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1
297	0.0	0.0	0.0	1.0	1.0	0.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	1
298	0.0	0.0	0.0	1.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1
299	1.0	1.0	1.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	1

Figure 3.11: Label distribution in CheXpert version 3 with Xs and Cardiomegaly = 1.

```
+ Version with 0s:
      No Finding      ->  (10000000000000)
      Pneumothorax    ->  (01000000000000)
      Pneumonia       ->  (00100000000000)
      Cardiomegaly    ->  (00010000000000)
+ Version with Xs:
      No Finding      ->  (1000XXXXXXXXX)
      Pneumothorax    ->  (0100XXXXXXXXX)
      Pneumonia       ->  (0010XXXXXXXXX)
      Cardiomegaly    ->  (0001XXXXXXXXX)
```

Figure 3.12: Different possibilities of Version 3. Each position in the binary array belongs to a label, a zero means that this disease has not been detected, a one means that it has been detected and an X means that it doesn't care, you choose the sample whether this disease has been detected or not.

3.3.1 Sample distribution by class

In Table 3.4 can be seen the sample distribution in CheXpert dataset. Can be seen a high imbalance in all the versions except Version 1 with Xs, which is perfectly balanced. It is important to note that the train test split was made manually, not using the dataset one, because when using it we found that there are some classes that in some versions are not represented at all, so we cannot use it.

CheXpert			**No Finding**	**Pneumothorax**	**Pneumonia**	**Cardiomegaly**	**Finding**
Version 1	0s	Training	7.642 (81%)	1.819 (19%)	-	-	-
		Validation	1.911 (81%)	455 (19%)	-	-	-
	Xs	Training	13.535 (49%)	14.198 (51%)	-	-	-
		Validation	3.439 (50%)	3.495 (50%)	-	-	-
Version 2	-	Training	7.630 (11%)	-	-	-	59.455 (89%)
		Validation	1.923 (11%)	-	-	-	14.849 (89%)
Version 3	0s	Training	7.641 (69%)	1.231 (11%)	341 (3%)	1.818 (17%)	-
		Validation	1.912 (69%)	308 (11%)	82 (3%)	456 (17%)	-
	Xs	Training	7.630 (11%)	51.988 (78%)	2.251 (3%)	5.216 (8%)	-
		Validation	1.923 (11%)	13.017 (78%)	534 (3%)	1.298 (8%)	-

Table 3.4: Sample distribution in CheXpert.

On the other hand, in Table 3.5 can be seen that in this dataset the problem with the imbalance is even worse and that we should keep that in mind throughout the project. Also can be seen that in the ChestX-ray8 dataset we don't have all the versions available, that is so because we found that the results are not good as expected so we decided to continue with other versions instead.

ChestX-ray8		**No Finding**	**Pneumothorax**	**Pneumonia**	**Cardiomegaly**
Version 1 with 0s	Training	50.500 (96%)	777 (2%)	-	-
	Validation	9.861 (97 %)	316 (3%)	-	-
Version 3 with 0s	Training	50.500 (96%)	777 (1%)	234 (1%)	1.241 (2%)
	Validation	9.861 (97%)	316 (0%)	88 (0%)	953 (0%)

Table 3.5: Sample distribution in ChestX-ray8.

4

Algorithm development

In this chapter we will go into detail on the algorithms we have used throughout this project, as well as the design decisions we have been making to successfully complete this project. As previously mentioned, the work is divided into two main tasks, on the one hand the generation of synthetic data and on the other hand the study of the use of this synthetic data to learn how to classify real data, domain adaptation, so this will be the organisation of this chapter.

4.1 Data Generation

To generate the synthetic data, as we have already mentioned, we are going to use Generative Adversarial Networks. Specifically, we will rely on the code provided by the authors of the paper [Schütte et al., 2021], in which they provide a complete framework for training and generating synthetic data using GANs.

This took us quite some time, it is not easy to understand and use someone else's code and even less if it uses old versions incompatible with your CUDA installation.

4.1.1 Model Training

Firstly, the training of the model, the most computationally expensive part of the whole project. To train the model we follow the scheme explained in chapter 2.1.1, in which we have two parts, a generator and a discriminator. The **generator**, which is creating samples trying to follow the original distribution of the training date. On the other hand, the **discriminator** is trying to detect if the sample received is real (from the dataset CheXpert dataset) or fake (made by the generator).

The framework we have used is the one provided by paper [Schütte et al., 2021], whose code is available on their Github repository [AugustDS, 2021]:

`https://github.com/AugustDS/synthetic-medical-benchmark`

In this code are provided different possible benchmarks to train models, with different resolutions, labels, number of training images, etc. Also they analyze both chest X-ray images and brain computed tomography images, and we will focus on the first type. For more details on the different benchmarks, see Table 4.1.

We decided to keep only one of the models, the one that seemed to work best according to the paper [Schütte et al., 2021], cpD-GAN. This will be the model we will use to generate our synthetic images.

We used different versions of the models:

- A simplified version of the smaller model that had the benchmark (the 32x32 with 4 labels, in red Table 4.1), where we reduced its resolution to 16x16. Result in Figure 4.1(a)
- The 32x32 with 4 labels model, images generated in Figure 4.1(b).
- The 64x64 model, this time using all the labels. Images generated in Figure 4.1(c).

The training time for these models is too long, the 32x32 takes 6 days and the 64x64 takes around 8 days of training, we don't have enough time to train bigger models. Due to this constrain we asked the authors of article [Schütte et al., 2021] for the weights of some of the trained models. They shared the weights of the two largest models they had trained, the 256x256 and the 512x512 (marked with green in Table 4.1, both using all the labels. As we can see in Figures 4.1(d) and 4.1(e), the results are almost indistinguishable, and it is not easy for the non-expert human eye to tell whether we are looking at real or synthetic images.

4.1.2 Data generation

Once the model has been trained, it is time to generate the synthetic data massively, the generation of our synthetic dataset. We will use part of the cpd-GAN network, obviously, the generator (Figure 4.2, which once trained can produce all the synthetic data we need, in addition, as we explained in section 2.1.1, it offers us the opportunity to generate synthetic data "a la carte", that is to say, with the labels we want, just like a Condictional GAN (section 2.1.1) would do.

To carry out this task we will follow two approaches, the first one is to generate a dataset with a perfect distribution of classes, generating the same samples of each of the classes and the second one is to replicate the distribution of classes of our real dataset, in this case, CheXpert.

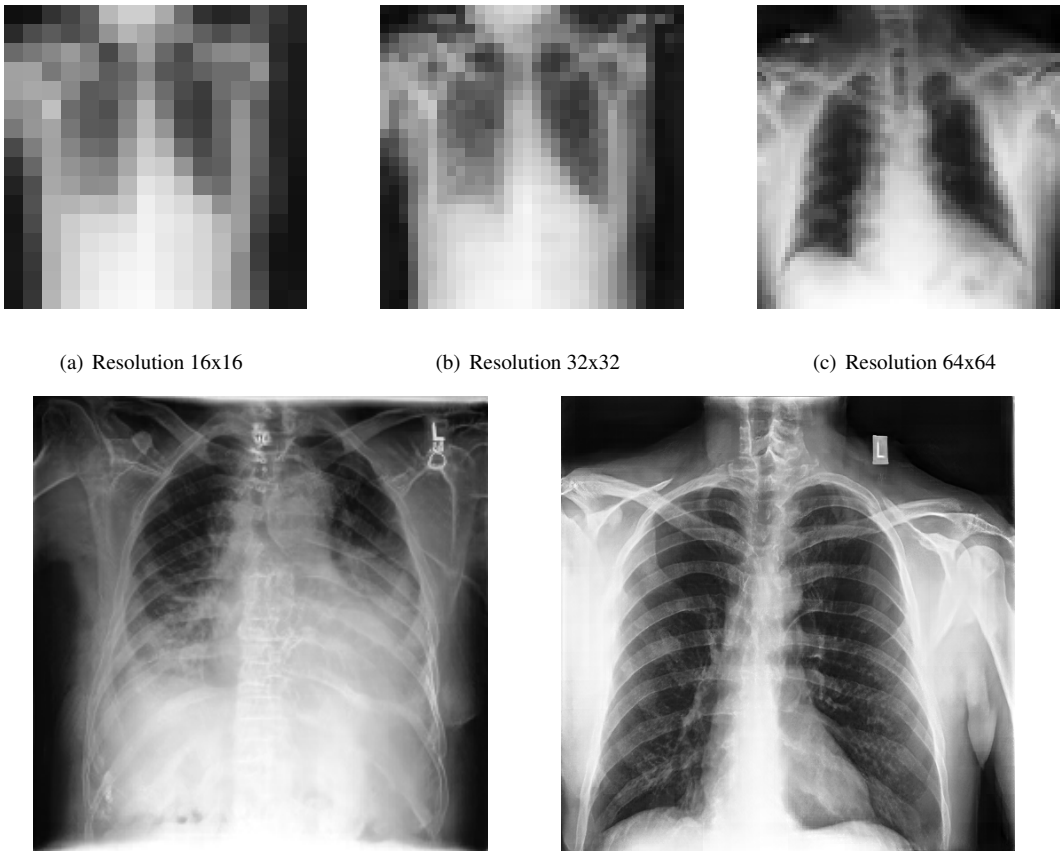
The easiest way is the first one, we only have to ask for a certain amount (in our case we chose 4000 for training and 1000 for validation) of samples of each of the classes and we would have our synthetic dataset ready. The problem with this is that we will be generating data only compatible with the versions of "zeros" (not with "Xs") that we explained in section 3.3, this is why we have created the second approach.

Table 1. All benchmark settings.

Type	Benchmark	Resolution	0/1 labels	Classes	Train set	Test/val set	Per class
Chest	Classes	32 × 32	9	20	29,000	3800	1450
		32 × 32	8	15	24,000	2850	1600
		32 × 32	5	10	20,000	1900	2000
		32 × 32	5	6	13,800	1140	2300
		32 × 32	5	4	15,600	760	3900
		32 × 32	4	2	12,600	380	6300
	Samples	32 × 32	4	3	17,850	2250	5950
		32 × 32	4	3	13,500	2250	4500
		32 × 32	4	3	9000	2250	3000
		32 × 32	4	3	4500	2250	1500
		32 × 32	4	3	3000	2250	1000
		32 × 32	4	3	1500	2250	500
		32 × 32	4	3	1200	2250	400
		32 × 32	4	3	600	2250	200
	Resolution	32 × 32	14	138	117,168	4000	256–7586
		64 × 64	14	138	117,168	4000	256–7586
		128 × 128	14	138	117,168	4000	256–7586
		256 × 256	14	138	117,168	4000	256–7586
		512 × 512	14	138	117,168	4000	256–7586
Brain	Classes	32 × 32	5	10	25,000	3000	2500
		32 × 32	5	8	24,960	2400	3120
		32 × 32	5	6	25,020	1800	4170
		32 × 32	4	4	25,000	1200	6250
		32 × 32	2	2	25,000	600	12,500
	Samples	32 × 32	5	6	32,400	3000	5400
		32 × 32	5	6	27,000	3000	4500
		32 × 32	5	6	18,000	3000	3000
		32 × 32	5	6	9000	3000	1500
		32 × 32	5	6	6000	3000	1000
		32 × 32	5	6	3000	3000	500
		32 × 32	5	6	1800	3000	300
		32 × 32	5	6	600	3000	100
	Resolution	32 × 32	6	20	117,168	4000	155–85,876
		64 × 64	6	20	117,168	4000	155–85,876
		128 × 128	6	20	117,168	4000	155–85,876
		256 × 256	6	20	117,168	4000	155–85,876
		512 × 512	6	20	117,168	4000	155–85,876

Each row defines the composition of a specific benchmark setting. After GAN training, the synthetic datasets are generated by conditioning on the real label sets, resulting in equivalent data folds. Our chest radigraph data pool consists of 117,168 (44,153) training, 15,418 (5519) validation and 14,687 (5520) test samples (patients), respectively. Our brain computed tomography scan data pool consists of 173,271 (15,133) training, 22,095 (1892) validation and 20,500 (1892) test samples (patients), respectively. The 14 binary chest X-ray labels are enlarged cardiomediastinum, cardiomegaly, lung opacity, lung lesion, oedema, consolidation, pneumonia, atelectasis, pneumothorax, pleural effusion, pleural other, fracture and support device, and no finding. The six binary brain CT scan labels are epidural, subarachnoid, subdural, intraparenchymal and intraventricular haemorrhage, and no finding. 0/1 Labels: number of binary labels. Classes: number of classes. Note: the number of classes refers to the number of unique binary label combinations. If different binary labels co-occur, we can have fewer classes than 0/1 labels. Train set: number of samples in training set. Test/val set: number of samples in each the test and validation set. Per class: number of training samples per class.

Table 4.1: Data generation benchmarks. [Schütte et al., 2021]



(a) Resolution 16x16 (b) Resolution 32x32 (c) Resolution 64x64

(d) Resolution 256x256 (e) Resolution 512x512

Figure 4.1: Data generation examples.

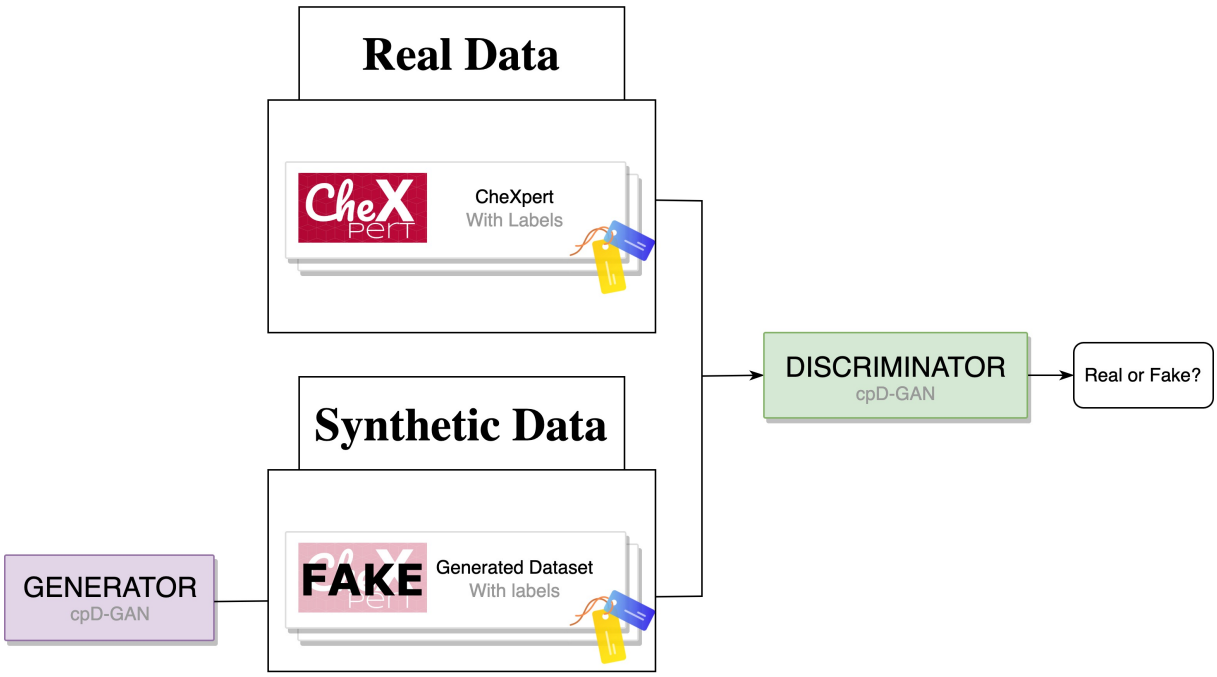
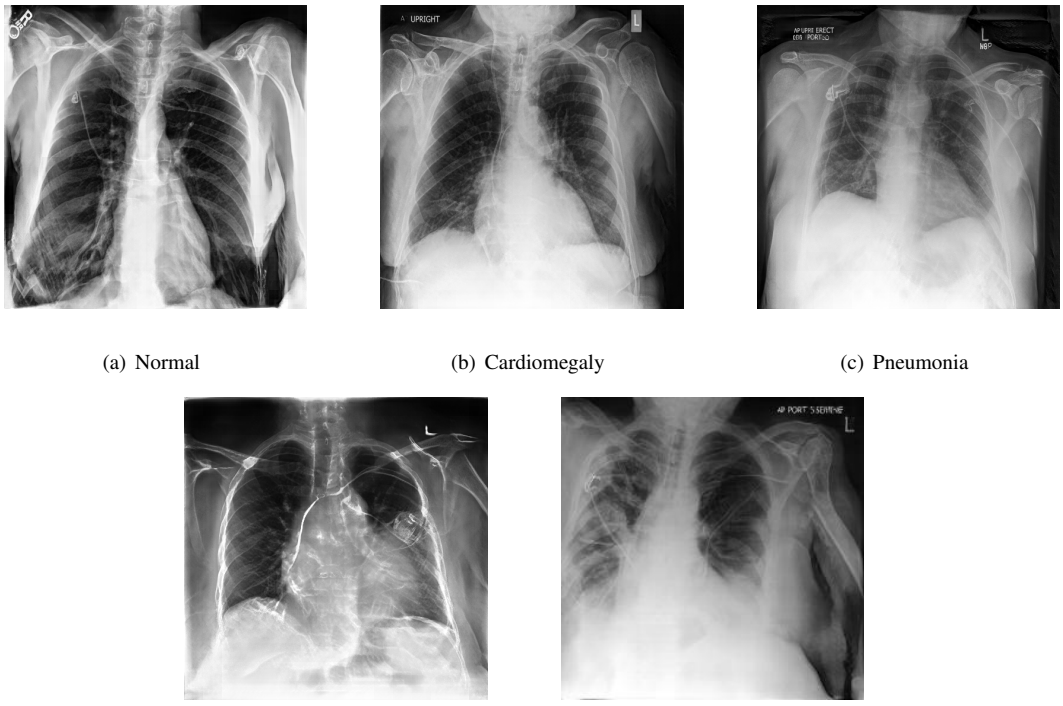


Figure 4.2: Data generation framework.

The second version is more difficult without any doubt, it is not easy to generate a dataset with the same combinations that CheXpert has, especially because some of them do not make much sense. The process was, first pre-processing the dataset, then keeping the most representative tag combinations, and finally using those generate a synthetic dataset as close to a real one as possible, which allow us to work with both the versions with "zeros" and the versions with "Xs".

We have generated these synthetic datasets for each of the resolutions for which we have trained models, 32x32, 64x64, 256x256 and 512x512. In the following experiments we will use the 256x256 images, since as we will see in the next chapter these are more representative and had statistically higher representativeness, it seems that by focusing on improving the detail by raising the resolution so much, the synthetic images provided less information. In Figure 4.3 you can see some examples of the four classes that we have used in the experiments (generated with "zeros"), as we can see with the naked eye of a non-expert it is very difficult to distinguish the classes of each of the images. And in the Figure 4.3(e) we have an example of a image diagnosed as Cardiomegaly, but as it was generated with "Xs" (following the first approach) so it also has Lung Opacity and Edema. In the Table 4.2 can be seen that we have generated 4000 samples per class in training and 1000 samples in validation, in all the versions of the dataset.



(a) Normal (b) Cardiomegaly (c) Pneumonia

(d) Pneumothorax (e) **Cardiomegaly**, Lung Opacity and Edema

Figure 4.3: Data generation examples per class.

Synthetic			**No Finding**	**Pneumothorax**	**Pneumonia**	**Cardiomegaly**	**Finding**
Version 1	0s	Training	4000 (50%)	4000 (50%)	-	-	-
		Validation	1000 (50%)	1000 (50%)	-	-	-
	Xs	Training	4000 (50%)	4000 (50%)	-	-	-
		Validation	1000 (50%)	1000 (50%)	-	-	-
Version 2	-	Training	4000 (50%)	-	-	-	4000 (50%)
		Validation	1000 (50%)	-	-	-	1000 (50%)
Version 3	0s	Training	4000 (25%)	4000 (25%)	4000 (25%)	4000 (25%)	-
		Validation	1000 (25%)	1000 (25%)	1000 (25%)	1000 (25%)	-
	Xs	Training	4000 (25%)	4000 (25%)	4000 (25%)	4000 (25%)	-
		Validation	1000 (25%)	1000 (25%)	1000 (25%)	1000 (25%)	-

Table 4.2: Sample distribution in synthetic dataset.

4.2 Domain Adaptation

This section aims to detail which algorithms we used to study the usefulness of the synthetic data we used. Firstly we will present the direct approach without applying any alignment technique and secondly the use Unsupervised Domain Adaptation techniques.

As explained in the State of the Art (section 2.3) domain adaptation is an special case of Transfer Learning, where source and target data representations are different but both share the same task. In this case, the objective is using real images as target data, synthetic images as source data, and both sharing the task of classifying x-ray images.

4.2.1 Baseline - Without alignment

First of all we are going to use a classical neural network, without any add on. To perform that test we have used a ResNet-50 [He et al., 2016] pre-trained with ImageNet [Deng et al., 2009]. Since the learning was somewhat worse than expected, we decided to vitaminise our algorithm by using some techniques such as using different learning rates in each layer or adding a scheduler.

In the Figure 4.4 can be seen a diagram of the framework. Where we only test the behaviour of the algorithm only by training this model on the labelled source data (generated), and when the model is trained we use it to classify target data (real).

4.2.2 Unsupervised Domain Adaptation

To carry out Unsupervised Domain Adaptation (UDA) we will use the network described in the State of the Art (section), DCAN [Li et al., 2020]. At a low level DCAN uses a Resnet-50 backbone with an attention module that is trained using numerous training losses to achieve the desired feature alignment.

Due to the reduced number of classes, and the lack of reliability of the ground truth we decided to remove from the training loss the regularization loss of the l−th feature L^l_{reg} which aims at solving the over-correction problems caused by the added feature correction blocks with the guide of source data [Li et al., 2020]. This what we called "Our Alignment".

In the Figure 4.5 can be seen a diagram of the framework where we train DCAN using source data with labels (generated) and using target data without labels (real), and when the model is trained we use it to classify the target data (real).

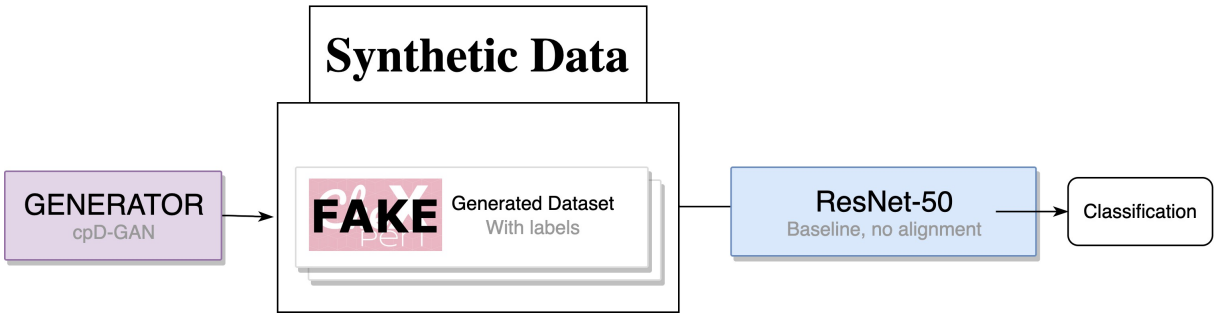


Figure 4.4: Baseline framework.

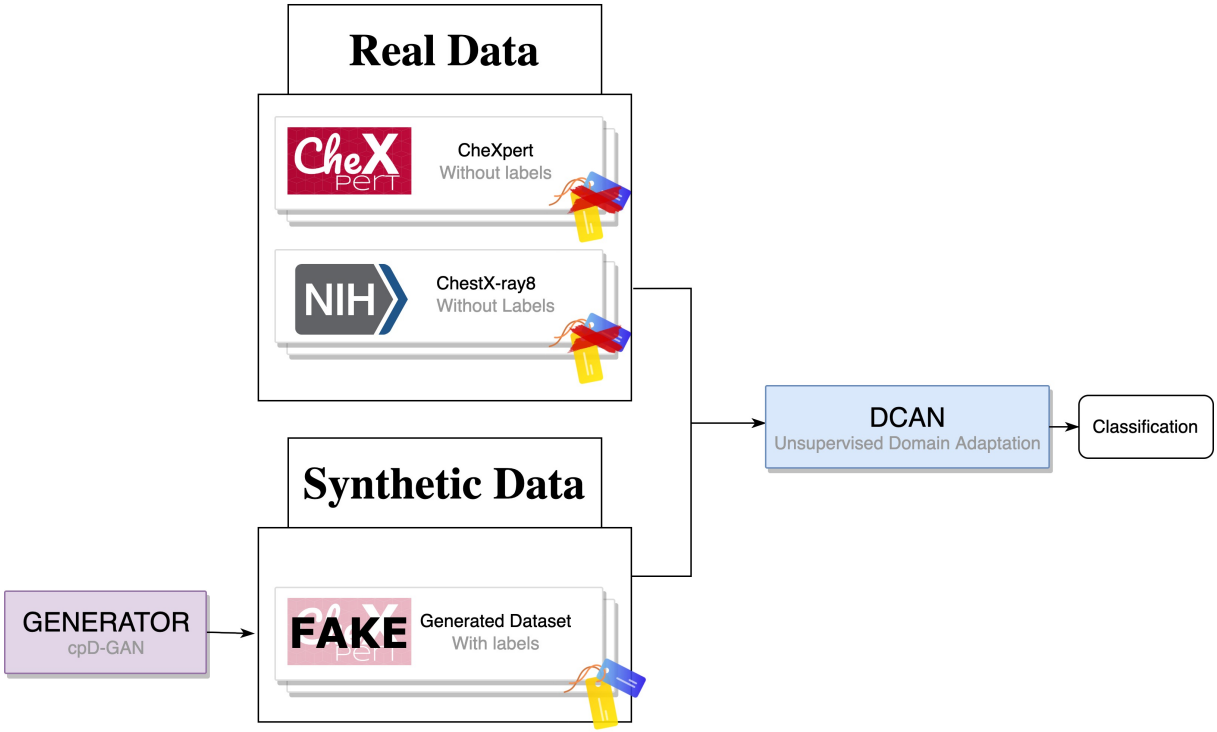


Figure 4.5: Unsupervised Domain Adaptation framework.

5 Evaluation

In this chapter, we will analyze the evaluation results obtained in this project. It is divided into two parts. Firstly we will analyze the methodology followed, analysing the framework, the metrics and the environment used and secondly we will detail the results obtained in the different experiments.

5.1 Methodology

5.1.1 Framework

In this chapter, we will use the frameworks and algorithms detailed in the previous one. Once the synthetic data has been generated following the diagram in Figure 4.2, we will evaluate the usefulness of the synthetic data using a simple neural network (diagram in Figure 4.4) and Unsupervised Domain Adaptation techniques (diagram in Figure 4.5).

Network configuration and parameters

We have decided to used the following network configurations and parameters, these parameters have been chosen on the basis of other projects using similar architectures [1] [2].

Baseline network

	lr	momentum	weight_decay	nesterov
First 2 epochs (only class layer)	0.01	0.9	1e-04	False
Next 10 epochs (all the network)	0.001	0.9	0	False

Table 5.1: Baseline parameters SGD optimizer.

[1] *Synthetic Medical Benchmark*, Github, accessed 2022-05-11.
https://github.com/AugustDS/synthetic-medical-benchmark

[2] *Domain Conditioned Adaptation Network*, Github, accessed 2022-06-04.
https://github.com/BIT-DA/DCAN

As we have explained in the previous chapter (Section 4.2.1) we have used a simple ResNet-50 as baseline classifier, but as we will see in the result section we have several versions of the network. Here we are going to detail the parameters of the simple version and the improvement one (with or without schedule). The configuration with using the scheduler used is the same but adding the scheduler with the default parameters, also is important to not that the scheduler used is: *torch.optim.lr_scheduler.CosineAnnealingLR* [3].

As can be seen in the Table 5.1 the parameters of the SGD optimizer of the simple network are divided in two times. First the parameters used in the first two epochs of the training, where only the classifier layer is being trained. Secondly the parameters used during the next ten epochs, where all the network is being trained.

Then, in the Table 5.2 and 5.3 we have the parameters of the improvement version. The improvement version is exactly the same network but modifying the optimizer parameters and adding the multipliers to the learning rate and to the decay on each layer separately. More specifically in the Table 5.2 we can see the value of the multipliers, divided in the base network (all the network) and the classifier head (only the last layers). And in the Table 5.3 can be seen the parameters of the optimizer divided as the simple network ones.

	lr_mult	**decay_mult**
base network	1	2
classifier layer	10	2

Table 5.2: ”Baseline improvement” parameters per layer.

	lr	**momentum**	**weight_decay**	**nesterov**
First 2 epochs (only class layer)	0.01	0.9	1e-04	False
Next 10 epochs (all the network)	1e-04	0.9	5e-04	True

Table 5.3: ”Baseline improvement” parameters SGD optimizer.

Lastly, we have the network configuration of the DCAN network used, both with the class alingment and without it uses the same parameters. As before in the Table 5.5 we have the parameter multipliers per layer and in the Table 5.4 we have the parameters of the optimizer, which are the same in all the epochs.

lr	**momentum**	**weight_decay**	**nesterov**
1e-04	0.9	5e-04	True

Table 5.4: UDA parameters SGD optimizer.

[3] *CosineAnnealingLR*, Pytorch, accessed 2022-06-19. torch.optim.lr_scheduler.CosineAnnealingLR.html

	lr_mult	decay_mult
base network	1	2
classifier layer	10	2
class residual layer	0.01	2
feature residual layer	0.01	2

Table 5.5: UDA parameters per layer.

5.1.2 Metrics

In order to evaluate the performance of our models we will use the following metrics. Because the datasets are highly unbalanced we have chosen metrics that are not susceptible to imbalances.

- **Recall**: This metric measure the percentage of positives samples in total that we are classifying correctly.

$$Recall = \frac{TP}{TP + FN} \tag{5.1}$$

- **Balanced Accuracy**: This metric allows us to calculate an estimate of how much data we are classifying correctly, without being biased by class imbalance. It is defined as the average of the Recall per class. Equation 5.2. This time we have used the version implemented by the Sklearn library [4].

$$BalancedAccuracy = \frac{\sum_{i=0}^{\#classes} Recall(class_i)}{\#classes} \tag{5.2}$$

5.1.3 Environment

The final hardware used is summarised in Table 5.6. But is important to remark that due to the enormous computational cost of training the GANs, sometimes we used another machine with a larger GPU, 24GB RTX Titan, as the 11GB RTX 2080 ti could not fit the model in memory. I would like to take this opportunity to thank the EPS and VPULab for the availability of these machines.

Componente	Características
CPU	Intel Core i9 10900k
GPU	Nvidia RTX 2080 11 GB VRAM
RAM	32 GB
S.O.	Ubuntu 20.04 LTS

Table 5.6: Table of technical details of the training and evaluation computer.

About the software we have used Python programming language with some of the most common open source data science and deep learning libraries, the main ones are detailed in Table 5.7.

[4] *Balanced Accuracy*, Sklearn, accessed 2022-05-12. sklearn.metrics.balanced_accuracy_score.html

Package	Version
Pytorch	1.6.0
Tensorflow GPU	1.14.0
Numpy	1.19.1
Pandas	1.3.5
Sklearn	0.23.2

Table 5.7: Table of technical details of the training and evaluation computer.

5.2 Results of v1: No Finding VS Pneumothorax (binary)

In this section we will follow the same structure as in section 4.2, first we will detail the results obtained without applying alignment and then we will show the results obtained using different types of alignments in Unsupervised Domain Adaptation. It is important to note that we will start by **using only the datasets version 1 with 0s**, which is explained in detail in section 3.3, but after that, in the next section we will show some results using other versions of the datasets.

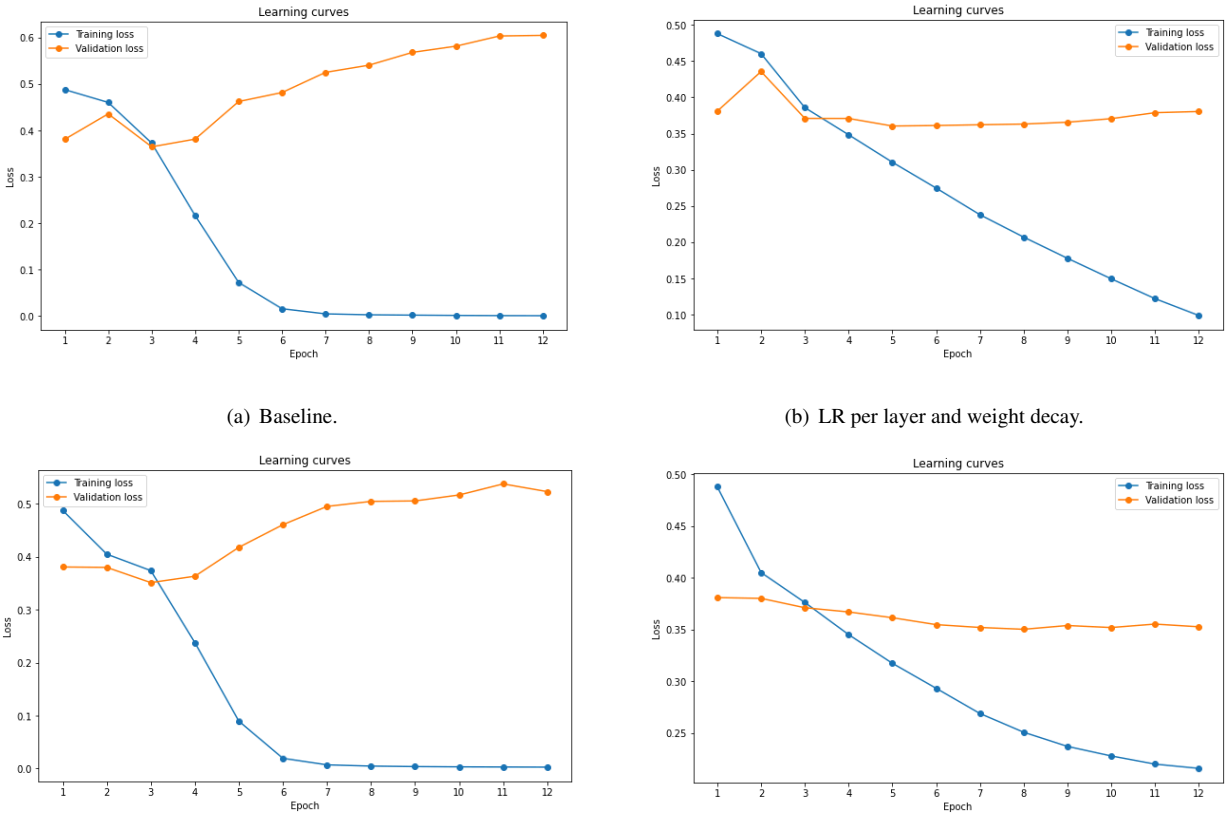
5.2.1 Baseline - No Alignment

First, without alignment, as explained above, it is simply a matter of training a pre-trained ResNet50 in Imagenet on the labelled synthetic data and then evaluating the performance of this algorithm on the real CheXpert and ChestXray8 data.

We tested different versions of this network because as we can see in Figure 5.1(a) the training curve without any improvement was very bad, we can observe a great overfitting in which we can hardly appreciate an improvement in the validation data as the training progresses.

In the first improved version, we made some changes to the network, specifically we modified the Learning Rate individually in each layer, lowering the Learning Rate of the backbone, which was trained in Imagenet, and raising the Learning Rate of the classification head so that it would learn to classify the X-rays better. We also added a scheduler to modify the Learning Rate of the network as it learned. As we can see in Figure 5.1(d), the results improved considerably, although they were still not perfect, now the network was at least learning little by little.

To analyse the contribution of each improvement we decided to carry out a small ablation study, testing each contribution separately, as we can see in Figures 5.1(b) and 5.1(c), the most beneficial modification was undoubtedly the Learning Rate per layer. And to finalize with the analysis of the training process we have also keep the best validation result obtained in each version (Table 5.8), with the epoch when it occurs and the training loss of that moment.



(a) Baseline.

(b) LR per layer and weight decay.

(c) Scheduler

(d) Scheduler, LR per layer and weight decay

Figure 5.1: Training curves baseline algorithm, numerical results in Table 5.8.

When evaluating the model over other datasets, as we can see in Table 5.8, when we **train these networks using the synthetic data** (remember that they have been generated based on CheXpert), without any improvement we seemed to obtain the best results for ChestX-ray8, even better than CheXpert, thanks to the training curve we can see that this is a local minimum at the beginning of the network training, which by chance gives us some good results, but they are not representative at all. The results that start to make sense are once we have added the learning rate per layer improvement, where we can see that the algorithm behaves better in CheXpert than in ChestX-ray8, a behaviour that is maintained with the rest of the versions. Also, as we can see in the final version with both improvements, the algorithm generalises very well, even loses some performance if we look at the test data of the synthetic dataset, but it is not representative. Finally, the results are better in the scheduler-only version than in the different learning rate version, even though the curves of the latter looked better, this makes it clear that we should certainly look at various evaluation metrics, in this case, the curves did not have information on how the network would generalise on other data, they only looked at the behaviour on synthetic data.

Different LR between layers	**Scheduler**	Trained with Synthetic evaluated with:			Synthetic Dataset Loss:	
		Synthetic	**Chesxpert**	**ChestX-ray8**	**Validation**	**Training**
-	-	100.00 %	55.71 %	**64.15 %**	0.374	0.382
✓	-	100.00 %	57.36 %	54.28 %	0.371	0.322
-	✓	**100.00 %**	61.24 %	57.35 %	0.359	0.378
✓	✓	99.15 %	**63.31 %**	56.42 %	**0.353**	**0.253**

Table 5.8: Results supervised. Trained with Synthetic data using version 1 with 0s. Including training loss values.

However, if we look at the results obtained when training with the CheXpert data and the ChestX-ray8 data (Table 5.9) we realise that these ”improvements” were not that much, and that the best versions are only adding the scheduler in the case of CheXpert and adding nothing in the case of ChestX-ray8.

Different LR between layers	**Scheduler**	Trained with CheXpert evaluated with:			Trained with ChestX-ray8 evaluated with:		
		Synthetic	**CheXpert**	**ChestX-ray8**	**Synthetic**	**CheXpert**	**ChestX-ray8**
-	-	72.90 %	**75.32 %**	62.15 %	**62.75 %**	**65.04 %**	**66.58 %**
✓	-	67.85 %	71.41 %	55.74 %	62.20 %	60.05 %	64.03 %
-	✓	**72.95 %**	74.84 %	**63.83 %**	61.90 %	61.69 %	65.52 %
✓	✓	65.55 %	69.04 %	56.44 %	60.34 %	58.93 %	63.32 %

Table 5.9: Results supervised. Trained with CheXpert and ChestX-ray8 datasets version 1 with 0s.

5.2.2 Using UDA

In this section we study the results obtained applying Unsupervised Domain Adaptation and we compare them with the results of the algorithm presented in the previous section (without alignment). Also, as introduced in the section 4.2.2 we have two versions of UDA algorithms, DCAN [Li et al., 2020] (DCAN Alignment) and our modification of DCAN removing the class alignment (Our alignment).

We are more interested in evaluating how the algorithms behave on the real data, so for simplicity we will only use the real data as target data, and we will also omit the results of using the same dataset as target and source.

Taking all this into account, in Table 5.10 we can see the results obtained in each of the combinations between source and target. As we can see the best option if we use synthetic images as source data is Our UDA approach, while if we use real images as source data the best option is not to apply any alignment. Also, as expected, when using CheXpert as target data, the best option is to use synthetic images as source data, with quite a difference ($\approx$ 8%). This is probably due to the fact that these images have been generated with the GAN that was trained using CheXpert. On the contrary, when using ChestX-ray8, the best option, although with very little difference (<2%) is to use the CheXpert data, this makes sense as it is a larger dataset and probably contains more information. Finally, it should be noted that we have not been able to run UDA using ChestX-ray8 as source and CheXpert as target, the model always predicted the same class, thus achieving a 50% Balanced accuracy.

Lastly, looking again at Table 5.9 we can see the results obtained when training and evaluating with the same dataset, with CheXpert and with ChestX-ray8. This helps us to know what would be a good goal to achieve, to evaluate how good UDA is performing. As we can see, the best result we obtain in the case of CheXpert is 75.32 %, while applying UDA we have managed to reach 73.98 % using synthetic data and without using any real label, we have come quite close (-1.34%). In the case of ChestX-ray8, training with the same dataset we reached 66.58 %, while applying UDA with synthetic data we were left with 61.88 %, this time we were a little further away (-4.7%). We would get closer by applying UDA using CheXpert, reaching 63.83% (-2.75%). This is a good way to check how the experiments have turned out, and it seems that in this case it has been a success.

Target	*Source*	*Proposal*	*Balanced Acc*
CheXpert	**Synthetic**	*No Alignment*	63.31 %
		DCAN Alignment	73.03 %
		Our Alignment	**73.98 %**
	ChestX-ray8	*No Alignment*	**65.04** %
		DCAN Alignment	50.00 %
		Our Alignment	50.00 %
ChestX-ray8	**Synthetic**	*No Alignment*	56.42 %
		DCAN Alignment	61.13 %
		Our Alignment	**61.88** %
	ChestXpert	*No Alignment*	**63.83 %**
		DCAN Alignment	62.04 %
		Our Alignment	61.84 %

Table 5.10: Summary of all results using version 1 with 0s.

5.3 Results other experiments

In addition to the experiments shown using dataset version 1 (binary with 0s), we have tested many other versions of the data, the results of which are detailed in this section. It's important to note that not all the versions have the same experiments, that is so because we increase the number of experiments depending on the results obtained in the previous ones, and if we detect that the results are not good enough we continue with more promising versions. More detail about the different versions available in the section 3.3.

5.3.1 Version 1: binary with Xs

In this case, we only have results with the CheXpert dataset. As we explained in the section on dataset analysis, this is a much more complicated problem than the version with 0s, and this probably makes the synthetic data not representative enought. As we can see in Table 5.11, the best result when training with synthetic data is obtained without any type of alignment, reaching 54.96 %. This value is far from the result obtained by training and evaluating on CheXpert, which reaches 80.38 %. Due to these results, we abandoned this version and stopped doing experiments using it.

Target	*Source*	*Proposal*	*Balanced Acc*
CheXpert	**Synthetic**	*No Alignment*	**54.96 %**
		DCAN Alignment	52.04 %
		Our Alignment	53.46 %
CheXpert	**CheXpert**	*No Alignment*	**80.37 %**

Table 5.11: Summary of results using version 1 (binary with Xs).

5.3.2 Version 2: No Finding vs All

Here something similar to the previous section happens, but the problem is even more difficult, so the results are even worse. As we can see (Table 5.12) this time the best result using synthetic data is using our alignment, reaching 52.94 %, far away from the 74.78 % obtained from training and evaluating on CheXpert. Due to these results, we do not continue experimenting with other datasets here either.

Target	*Source*	*Proposal*	*Balanced Acc*
CheXpert	**Synthetic**	*No Alignment*	51.17 %
		DCAN Alignment	52.61 %
		Our Alignment	**52.94 %**
CheXpert	**CheXpert**	*No Alignment*	**74.78 %**

Table 5.12: Summary of results using version 2 (No Finding vs All).

5.3.3 Version 3: four classes

With 0s

This version has more tests than the previous ones because as it is a version with zeros we expected it to do well. In Table 5.13 we have results with both CheXpert and ChestX-ray8.

As expected, the results are not bad, but they are not too good either. In the case of CheXpert, UDA works very badly, with DCAN failing to learn anything and always predicting the same class (that's why it gets 25 % correct), but without using UDA we managed to improve that 25 % to 30.13 % (+5.13%). Despite this improvement, we are still a bit far from the 41.5 % (-11.37%) obtained using CheXpert over CheXpert. In the case of ChestX-ray8, the results are surprisingly better, using our UDA alignment we achieved 30.34 %, which is a little closer to the ChestX-ray8 result of 36.74 % (-6.4%).

Target	*Source*	*Proposal*	*Balanced Acc*
CheXpert	**Synthetic**	*No Alignment*	**30.13 %**
		DCAN Alignment	25.00 %
		Our Alignment	28.07 %
CheXpert	**CheXpert**	*No Alignment*	**41.50 %**
ChestX-ray8	**Synthetic**	*No Alignment*	28.54 %
		DCAN Alignment	27.73 %
		Our Alignment	**30.34 %**
ChestX-ray8	**ChestX-ray8**	*No Alignment*	**36.74 %**

Table 5.13: Summary of results using version 3 with 0s (four classes).

With Xs

Finally, the version that we thought as the most difficult one, but in the end, we have discovered that it is not so difficult. As we can see in Table 5.14 we obtain very similar results to those obtained in the version 3 with 0s, staying even closer to the result obtained by training and evaluating with the same dataset. In CheXpert, using DCAN we reached 29.28 %, which is a little closer than before to the supervised result of 35.84 % (-6.57 %).

After obtaining these results we have meditated on why this version has a similar complexity to the previous one and we believe it is because, as could be seen in Figure 3.12, when generating the dataset we set the selected label to 1 and all other labels to X except the labels used by other classes we are using, which we set to 0. This slightly eliminates the ambiguity of the dataset and probably that is why this version is simpler than version 1 with Xs.

Target	*Source*	*Proposal*	*Balanced Acc*
CheXpert	**Synthetic**	*No Alignment*	27.27 %
		DCAN Alignment	**29.28 %**
		Our Alignment	27.33 %
CheXpert	**CheXpert**	*No Alignment*	**35.84 %**

Table 5.14: Summary of results using version 3 with Xs (four classes).

6

Conclusions and future work

6.1 Conclusions

As presented in the introduction, according to [Chen et al., 2021], a key challenge for applying AI in the medical field is the representativeness of the data employed for training AI models and in this project we have proved it. The results obtained in the binary with zeros dataset are good, so it seems like the idea works, but as soon as the classification task start getting harder the problems begin.

This behaviour is probably because the synthesised data is very good visually but it is not able to generate new information. It only compresses what we have and gives us different mixes. To the human eye it is more than enough, but for a neural network to learn it is only useful for a simple problem, but not enough for a more complex one. During this project, we have experimented using different versions of the data, ranging from the simplest version to the most complex one and thus studying the degree of representativeness of the data.

Regarding the use of Domain Adaptation algorithms, these seem to improve the results, but like classical algorithms, they also require sufficiently representative data. Fortunately, the design of the project allows the different parts to be modified as a block, so it opens the door to replace the data generation model with a better performing one in the future, and to evaluate the Domain Adaptation algorithms with these new data.

6.2 Future work

As mentioned in the conclusions, the main advantage of this work is that it is divisible into two parts that can be interchanged as a block, one is the generation of data and the other is to study its usefulness for training machine learning models. So it opens the door to experimentation as better approaches emerging in the state of the art.

Based on the above, the first task suggested by the project would be to test with other data generation techniques that generate data with stronger statistical power, can be tested approaches using another type of generative model such as transformers [Ramesh et al., 2021], autoencoders [Wan et al., 2017] or even extracting images using medical simulators [Sújar et al., 2019]. Once we have sufficiently representative synthetic data, the next natural step would be to test more complex versions of the data, the ones presented here or even others using a larger number of classes.

On the other hand, in the classification models, Unsupervised Domain Adaptation is interesting, but perhaps this approach could also use Inductive Transfer Learning, i.e. pre-training using a lot of labelled synthetic source data and finishing the training using a few labelled real target data. But of course, in order to apply this, the few labelled real data is needed. Finally, in case we have a few labelled real data, can be also experimented with classical self-supervised algorithms, which we tried to do in this work, but it was too time consuming and we decided to prioritise other experiments.

Bibliography

[AugustDS, 2021] AugustDS (2021). Synthetic medical benchmark.

[Çalli et al., 2021] Çalli, E., Sogancioglu, E., van Ginneken, B., van Leeuwen, K. G., and Murphy, K. (2021). Deep learning for chest x-ray analysis: A survey. *Medical Image Anal.*, 72:102125.

[Chen et al., 2021] Chen, R. J., Lu, M. Y., Chen, T. Y., Williamson, D. F., and Mahmood, F. (2021). Synthetic data in machine learning for medicine and healthcare.

[Chesney and Citron, 2019] Chesney, B. and Citron, D. (2019). Deep fakes: A looming challenge for privacy, democracy, and national security. *Calif. L. Rev.*, 107:1753.

[Csurka, 2017] Csurka, G. (2017). A comprehensive survey on domain adaptation for visual applications. In Csurka, G., editor, *Domain Adaptation in Computer Vision Applications*, Advances in Computer Vision and Pattern Recognition, pages 1–35. Springer.

[Deng et al., 2010] Deng, J., Berg, A. C., Li, K., and Fei-Fei, L. (2010). What does classifying more than 10, 000 image categories tell us? In Daniilidis, K., Maragos, P., and Paragios, N., editors, *Computer Vision - ECCV 2010 - 11th European Conference on Computer Vision, Heraklion, Crete, Greece, September 5-11, 2010, Proceedings, Part V*, volume 6315 of *Lecture Notes in Computer Science*, pages 71–84. Springer.

[Deng et al., 2009] Deng, J., Dong, W., Socher, R., Li, L., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*, pages 248–255. IEEE Computer Society.

[Denton et al., 2015] Denton, E. L., Chintala, S., Szlam, A., and Fergus, R. (2015). Deep generative image models using a laplacian pyramid of adversarial networks. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 1486–1494.

[Food et al., 2019] Food, Administration, D., et al. (2019). Proposed regulatory framework for modifications to artificial intelligence/machine learning (ai/ml)-based software as a medical device (samd). *Department of Health and Human Services (United States)*.

[Frolov et al., 2021] Frolov, S., Hinz, T., Raue, F., Hees, J., and Dengel, A. (2021). Adversarial text-to-image synthesis: A review. *Neural Networks*, 144:187–209.

[Gazda et al., 2021] Gazda, M., Plavka, J., Gazda, J., and Drotar, P. (2021). Self-supervised deep convolutional neural network for chest x-ray classification. *IEEE Access*, 9:151972–151982.

[Goodfellow, 2017] Goodfellow, I. J. (2017). NIPS 2016 tutorial: Generative adversarial networks. *CoRR*, abs/1701.00160.

[Goodfellow et al., 2014] Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. C., and Bengio, Y. (2014). Generative adversarial nets. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 2672–2680.

[He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778. IEEE Computer Society.

[Heusel et al., 2017] Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S. (2017). Gans trained by a two time-scale update rule converge to a local nash equilibrium. In Guyon, I., von Luxburg, U., Bengio, S., Wallach, H. M., Fergus, R., Vishwanathan, S. V. N., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 6626–6637.

[Irvin et al., 2019] Irvin, J., Rajpurkar, P., Ko, M., Yu, Y., Ciurea-Ilcus, S., Chute, C., Marklund, H., Haghgoo, B., Ball, R. L., Shpanskaya, K. S., Seekins, J., Mong, D. A., Halabi, S. S., Sandberg, J. K., Jones, R., Larson, D. B., Langlotz, C. P., Patel, B. N., Lungren, M. P., and Ng, A. Y. (2019). Chexpert: A large chest radiograph dataset with uncertainty labels and expert comparison. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 590–597. AAAI Press.

[Karras et al., 2018] Karras, T., Aila, T., Laine, S., and Lehtinen, J. (2018). Progressive growing of gans for improved quality, stability, and variation. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

[Karras et al., 2019] Karras, T., Laine, S., and Aila, T. (2019). A style-based generator architecture for generative adversarial networks. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 4401–4410. Computer Vision Foundation / IEEE.

[Karras et al., 2020] Karras, T., Laine, S., Aittala, M., Hellsten, J., Lehtinen, J., and Aila, T. (2020). Analyzing and improving the image quality of stylegan. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, pages 8107–8116. Computer Vision Foundation / IEEE.

[Koniusz et al., 2017] Koniusz, P., Tas, Y., and Porikli, F. (2017). Domain adaptation by mixture of alignments of second-or higher-order scatter tensors. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 7139–7148. IEEE Computer Society.

[Li et al., 2020] Li, S., Liu, C. H., Lin, Q., Xie, B., Ding, Z., Huang, G., and Tang, J. (2020). Domain

conditioned adaptation network. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 11386–11393. AAAI Press.

[Odena et al., 2017] Odena, A., Olah, C., and Shlens, J. (2017). Conditional image synthesis with auxiliary classifier gans. In Precup, D. and Teh, Y. W., editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 2642–2651. PMLR.

[Pan and Yang, 2010] Pan, S. J. and Yang, Q. (2010). A survey on transfer learning. *IEEE Trans. Knowl. Data Eng.*, 22(10):1345–1359.

[Ramesh et al., 2021] Ramesh, A., Pavlov, M., Goh, G., Gray, S., Voss, C., Radford, A., Chen, M., and Sutskever, I. (2021). Zero-shot text-to-image generation. In Meila, M. and Zhang, T., editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 8821–8831. PMLR.

[Ratliff et al., 2013] Ratliff, L. J., Burden, S., and Sastry, S. S. (2013). Characterization and computation of local nash equilibria in continuous games. In *51st Annual Allerton Conference on Communication, Control, and Computing, Allerton 2013, Allerton Park & Retreat Center, Monticello, IL, USA, October 2-4, 2013*, pages 917–924. IEEE.

[Rivest, 1987] Rivest, R. L. (1987). Game tree searching by min/max approximation. *Artif. Intell.*, 34(1):77–96.

[Russakovsky et al., 2015] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M. S., Berg, A. C., and Fei-Fei, L. (2015). Imagenet large scale visual recognition challenge. *Int. J. Comput. Vis.*, 115(3):211–252.

[Schütte et al., 2021] Schütte, A. D., Hetzel, J., Gatidis, S., Hepp, T., Dietz, B., Bauer, S., and Schwab, P. (2021). Overcoming barriers to data sharing with medical image generation: a comprehensive evaluation. *npj Digital Medicine*, 4.

[Sújar et al., 2019] Sújar, A., Kelly, G., García, M., and Vidal, F. P. (2019). Projectional radiography simulator: an interactive teaching tool. In Vidal, F. P., Tam, G. K. L., and Roberts, J. C., editors, *Computer Graphics & Visual Computing, CGVC 2019, Bangor, UK, September 12-13, 2019*, pages 125–128. Eurographics Association.

[Wan et al., 2017] Wan, Z., Zhang, Y., and He, H. (2017). Variational autoencoder based synthetic data generation for imbalanced learning. In *2017 IEEE Symposium Series on Computational Intelligence, SSCI 2017, Honolulu, HI, USA, November 27 - Dec. 1, 2017*, pages 1–7. IEEE.

[Wang et al., 2017] Wang, X., Peng, Y., Lu, L., Lu, Z., Bagheri, M., and Summers, R. M. (2017). Chestx-ray8: Hospital-scale chest x-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases.