Universitat Oberta de Catalunya (UOC)

Màster Universitari en Ciències de Dades (*Data Science*)

TREBALL FINAL DE MÀSTER

Àrea: 3

Generative Adversarial Networks Based Data Augmentation for Ultrasound Fetal Brain Planes Classification

Author: Alberto Montero Agudo

Advisors: Xavier P. Burgos-Artizzu, Elisenda Bonet-Carne

Professor: Ferran Prados Carrasco

Ettlingen, January 26, 2021

Créditos/Copyright

FITXA DEL TREBALL FINAL

Títol del treball:	Generative Adversarial Networks Based Data Augmentation for Ultrasound Fetal Brain Planes Classification
Nom de l'autor:	Alberto Montero Agudo
Nom del consultor/a:	Xavier P. Burgos-Artizzu, Elisenda Bonet-Carne
Nom del PRA:	Ferran Prados Carrasco
Data de lliurament (mm/aaaa):	03/2021
Titulació o programa:	Màster Universitari en Ciències de Dades
Àrea del Treball Final:	Àrea 3
Idioma del treball:	anglès
Paraules clau	Generative Adversarial Networks

Dedication

To my sister Maria, to my father Agustín.

Acknowledgements

I would like to thank my advisors Xavier P. Burgos-Artizzu and Elisenda Bonet-Carne for their valuable guidance and warm support throughout my thesis.

Abstract

Generative adversarial networks have been recently applied to medical imaging on different modalities (MRI, CT, X-ray, etc). However there are not many applications on ultrasound modality as a data augmentation technique applied to downstream classification tasks. This experimental case study aims to explore and evaluate the generation of synthetic ultrasound fetal brain images via generative adversarial networks and apply to ultrasound fetal brain plane classification tasks. State of the art Generative Adversarial Networks *stylegan2-ada* was applied to fetal brain image generation and GAN-based data augmentation classifiers were compared with baseline classifiers. Our experimental results show that GAN-Based data augmentation combined with classical data augmentation outperforms classifiers with only classical data augmentation by 2% in both accuracy and area under the curve score.

Keywords: Generative Adversarial Networks, Data Augmentation, Ultrasound Fetal Brain Images, Image Classification.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Justification and motivation

Deep learning methods, in particular convolutional networks, have become in recent years particularly successful in medical imaging analysis. Among other applications in this field we find image classification, object detection, segmentation and image registration (Litjens et al. (2017)).

One of the main difficulties in medical imaging is the scarcity of labeled data, which is a fundamental piece in supervised learning methods. Additionally, privacy issues related to diagnostic medical image data involves availability constraints. To overcome scarcity and privacy issues, generative adversarial networks (GANs) Goodfellow et al. (2014) have been recently applied to medical image synthesis on different modalities (MRI, CT, X-ray, etc). However there are not many applications of GAN-based data augmentation for classification tasks in ultrasound images.

A positive result of this work might contribute to other classification tasks in ultrasound modality with limited data or unlabeled data.

Personal motivation. From a personal point of view my motivation for this research work is double. From one side, this kind of models belong to a fascinating broader family of models (generative models) which are not covered in an introductory course of deep learning (besides simple autoencoders) with a very beautiful mathematical basis. On the other hand, the opportunity to participate in medical applications based on these models is something very gratifying for me. I also believe, that this kind of methods might contribute in a near future with very valuable medical applications.

1.2 Research objectives

Aim.
The aim of this research is to study the applicability of the generative adversarial networks as a data augmentation method applied to ultrasound fetal brain planes classification tasks.
Objectives.

- Data preparation for classification tasks and GANs training.
- Analyze the performance of *stylegan2-ada* GAN applied to ultrasound fetal image generation.
- Define and implement baseline classification models.
- Compare performance of previous defined classifiers with classifiers using synthetic generated images.

1.3 Methodology

1.3.1 Development methodology

Due to uncertainty of models performance, complexity of hyper-parameters settings and variety of models potentially to be explored, among others, we follow in this work a common machine learning lifecycle project. Planning should be adaptable with very general milestones described below in planing section.

1.3.2 Research methodology

1.3.2.1 Case study: ultrasound fetal brain planes classification

We consider two possible scenarios with respect data availability, which are both common scenarios in medical imaging. In the first scenario we deal with a labeled dataset and in the second scenario we additionally dispose of another larger unlabeled dataset. For the first scenario the data preparation step is straightforward, while for the second one it is a little more complex and we don't know at this planning stage if it will be useful for our project with given time constraints.
First scenario: labeled dataset.
To study the feasibility of GANs as a data augmentation technique we use as a case study the *brainplanes* dataset which is a subset of FETAL_PLANES_DB dataset released by Burgos-Artizzu et al. (2020) containing centered cropped images of fetal brain planes (see figure 1.1).

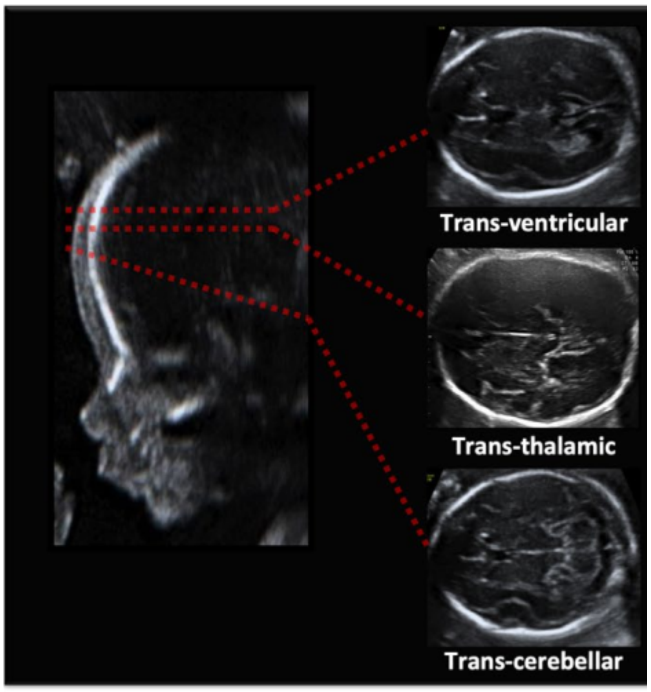


Figure 1.1: Fetal brain planes. Source Burgos-Artizzu et al. (2020)

brainplanes is comprised of about 12K images of ultrasound fetal brain images, annotated with classes (three different planes). It has been previously studied in Burgos-Artizzu et al. (2020) with respect plane classification task and shown that brain plane classification is a challenge classification task, specifically the distinction between Trans-ventricular and Trans-thalamic categories. We will study the feasibility of GANs-based data augmentation methods for fetal brain classification task with this dataset.

Second scenario: labeled dataset plus larger unlabeled dataset.

In the second scenario with will try to extract useful data to study the feasibility of GANs dealing with semi-supervised methods.[1]

1.3.3 Deep learning methodologies

To tackle complex computer vision problems there is no other alternative than use deep convolutional neural based methods. Specifically to our project we will use Generative Adversarial Networks (GANs) and consider several architectures, depending of time availability.

- Unconditional GANs for all classes. These methods generate images of all classes at the same time but we lose control over generation. That is, with this setting is not possible to generate an specific class. Thus, we reject this framework because our goal is to control the

[1]Finally this second scenario has not been possible to execute due time constrains and unavailability of additional dataset.

generation of images by classes.

- One unconditional GAN for each class. In this case we will be able to generate images of all classes separately. These architectures has been used in previous medical imaging works but it has the disadvantage that we need to train so many GANs as classes or categories we have.

- Class conditional GANs. This architecture exploits the labels of the dataset training a single model generating controlled classes.

These are candidates that we consider to be potentially useful for our objectives. Time for this project is scarce and we have to carefully set priorities. GANs development is nowadays a very hot topic in research and every week several papers are published. Other architectures as semi-supervised and self-supervised would be also worth to explore. There are hybrid architectures of above mentioned like self-supervised semi-supervised conditional GAN Sun et al. (2020) that look promising. Even explicit density models like variational autoencoders (which are easier to train) have obtained very promising results in last months (see Pidhorskyi et al. (2020)). Due time constrains we will mainly focus on second architecture mentioned above: one unconditional GAN for each class.

We have to take into account that besides time constrains, we have to find open source code of this models and maybe adapt to our needs, which is another constrain. We don't consider in this work to implement from scratch models due to the training implementation complexity and time disponibility.

1.3.4 Software and hardware resources

We will mainly use PyTorch and Tensorflow libraries for deep learning. PyChram as IDE, git and github for version control, kile as LaTeX editor.

As hardware we will use a laptop with a Nvidia GTX 1060 (6GB) graphic card for data preparation and for prototyping, evaluation and dealing with small models. For training GANs we will use cloud services as Google Colab, Kaggle or Coogle Cloud Platform.

1.4 Planing

1.4.1 Tentative timetable

We estimate 20-22 hours per week of work during the project period.

tasks	deadline	weeks
State of the art Datasets setup	18.10.20	3 weeks
GANs implementation Classifiers setup Analysis and results	20.12.20	9 weeks
Thesis writing	03.01.21	2 weeks
Presentation writing	10.01.21	1 week
Thesis defense	20.01.21	10 days

1.4.2 Implementation tasks

Data preparation. Data extraction and transformation from *brainplanes* dataset. Creation of baseline dataset with train / validation split with no overlapping patients.

For second scenario, we will use a classifier to extract brain planes from US400K dataset with the same distribution of *brainplanes* dataset, obtaining in this way a unlabeled dataset of brain planes. [2]

GANs implementation. We will make use of official implementations of SOTA GANs and variants, adapting the code to our needs when necessary. We will mainly focus on models based on styleGAN2 (Karras et al. (2020b)) and biggan (Brock et al. (2019)) with data augmentation mechanisms as in Karras et al. (2020a) and Zhao et al. (2020) [3] to deal with smalls datasets.

Classifiers setup. Implementation of good ResNet-based baselines classifiers with strong classical data augmentation to assess and compare results.

Analysis and results. We will evaluate GANs performance with GANs based metrics as FID (Fréchet Inception Distance) and GANs-based data augmentation classifiers through comparisons with baseline classifiers.

[2]Finally this second scenario has not been possible to execute due time constrains and unavailability of additional dataset.

[3]https://github.com/mit-han-lab/data-efficient-gans

Chapter 2

State of the Art

2.1 Introduction

Diagnostic ultrasound examination is an essential procedure when clinical complications occurred during pregnancy. The observation that the absence of clear risk factors may also induce complications has promoted routine ultrasound in women pregnancies with the aim of earlier detection of such complications Whitworth et al. (2010).

Ultrasound provides economic and non-invasive examination of fetal and maternal organs development by means of several measures commonly used to monitor fetal growth Hadlock et al. (1985) and fetal abnormalities detection. This procedure consists in ultrasound image acquisition following well defined and repeatable protocols, with extraction of specific image planes relevant for diagnosis. This practice is manually performed by trained technicians which screen more than 20 images and validated by a senior maternal-fetal expert. Thereby, the whole pipeline is time consuming and sensitive to errors.

In order to speed up and improve performance of planes selection an automatic process would be beneficial, reducing costs and errors. Few works addressing this problem can we find in literature with respect 2D still images and to our knowledge the only related work is Burgos-Artizzu et al. (2020), in which several deep convolutional neural networks architectures were evaluated with respect to common maternal-fetal ultrasound plane classification. We find indeed some works related with video and 3D images with rich data that does not correspond to the majority of available data in medical or research centers.

Deep learning methods, in particular deep convolutional neural networks, have become in recent years particularly successful in medical imaging analysis. Among other applications in this field we can find image classification, object detection, segmentation and image registration Litjens et al. (2017). The success of these methods depends to great extend to the amount of data that are trained with. Although pre-trained networks and data augmentation methods can alleviate to some extend the data required, data gathering is still a concern in medical imaging.

One of the main difficulties in medical imaging is the scarcity of labeled data, which is a fundamental piece when training deep convolutional neural networks. In addition, class imbalance is a common issue in medical imaging where anomalies or disease related image samples are much less frequent than normal cases. Privacy issues related to diagnostic medical image data involves also availability constraints.

In this work we study and evaluate the application of generative adversarial networks (GANs) as a data augmentation technique in fetal ultrasound plane classification. Specifically we focus on fetal brain plane classification as a case study, which has been shown by Burgos-Artizzu et al. (2020) to be a hard classification task.

Generative adversarial networks Goodfellow et al. (2014) have obtained a lot of attention in last years due to generation capability of synthetic images. It has been shown to be useful in many medical imaging applications such data augmentation, among many others Yi et al. (2019). To overcome above mentioned issues related to scarcity, imbalance and privacy issues, GANs have been recently applied to medical image synthesis on different modalities (MRI, CT, X-ray, etc).

In the following sections we explore some of the GANs methodologies and architectures related to our work, starting by a brief introduction to GANs.

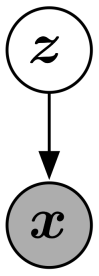
2.2 Generative Adversarial Networks

Generative vs discriminative models

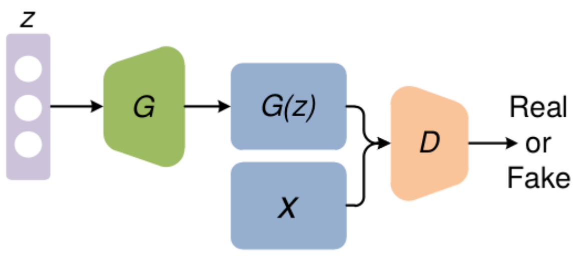
Given a set of observable random variables $\mathbf{X} = \{X_1, X_2, \cdots, X_n\}$, discriminative models are models of the conditional probability of a label or target variable, given an observation or set of observations $\{X_1, X_2, \cdots, X_n\}$, that is, $P(Y|X_1, X_2, \cdots, X_n)$. This is the case, for example, for cat/dog image classifier where a $P(Y = \text{“}cat\text{“}|X)$ estimates the probability of an image X to be a cat. In contrast, a generative model is a probabilistic model of the joint probabilistic distribution $P(X_1, X_2, \cdots, X_n)$ on $X_1 \times, \cdots \times X_n$. Notice that it might be the case that for some $1 \leq j \leq n$, $X_j = Y$ being Y a label. The main difference between both is that generative models are capable to generate new data.

Generative models

Generative models, as discriminative ones, can be trained via maximum likelihood estimation, when they define an explicit density function $p_{\text{model}}(x; \theta)$. These models are called explicit density models. Normally this density function will be much harder to deal with compared to discriminative models, specially when dealing with images. There are two main approaches to



(a) Probabilistic graphical model of GANs. Source Goodfellow (2017)



(b) Simple GAN architecture. Source Pan et al. (2019)

Figure 2.1: Graph and architectural representations of GANs

overcome this complexity: defining tractable density functions (fully visible belief nets, non-linear ICA, etc) or using tractable approximations to the likelihood (variational autoencoders VAE, etc). Although these methods work well in a wide range of applications, they have the inconvenient that either need strong restrictions with respect the family of density functions to be used (tractable), or very good approximations for prior and posterior distributions are required to obtain good quality of samples (approximate methods). For example, in last case, variational autoencoders have been applied to image generation and shown difficulties to generate high realism images (although in last months VAE have achieved very promising results Vahdat and Kautz (2020)).

Another way to define generative models is through implicit densities. That is, the model does not define a density function but a mechanism for which to learn to generate new samples by sampling from real data. This models are called implicit density models.

Generative Adversarial Networks

Generative Adversarial Networks (GANs, Goodfellow et al. (2014)) are implicit density models in which a kind of game between two players is performed. These two players are called generator and discriminator. Given a set of real samples, the generator creates samples that simulate the real samples without looking at them, and the discriminator tries to determine whether these samples created by the generator are real or not.

GANs can be represented by probabilistic graphical model containing latent variables z and observed variables x (Goodfellow (2017)). Figure 2.1(a) shows its graphical representation.The generator samples z from some prior distribution and $G(z)$ produce a sample x from p_{model}, where G is a differentiable function. The discriminator D is a differentiable function too and classifies samples in real and fakes. 2.1(b) shows the basic architecture of a simple GAN.

A possibility to describe differentiable functions G and D is by means of deep neural networks. Originally, GANs were defined in Goodfellow et al. (2014) by multilayer perceptrons,

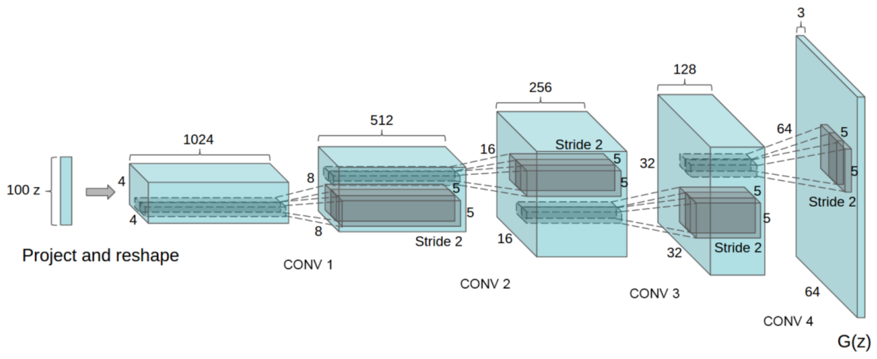


Figure 2.2: DCGAN: deep convolutional GAN. Source Radford et al. (2016)

but after Radford et al. (2016), deep convolutional networks have been used for G and D. In figure 2.2 a deep convolutional network for the generator function G is shown.

Besides differentiable functions G and D, a training process is needed. And a training process needs a cost function. Implicit models like GANs do not use maximum likelihood based cost function for training. Instead, original GANs define its cost function under a game theory framework in which D will minimize errors when classifying real data and fakes, and on the other hand, G will try to maximize to fool D.

Thus, the discriminator cost is defined by

$$J^{(D)} = -\frac{1}{2}\mathbb{E}_{x\sim p_{\text{data}}}\log D(x) - \frac{1}{2}\mathbb{E}_z \log(1 - D(G(z))) \tag{2.1}$$

where $x \sim p_{\text{data}}$ are samples from real data distribution. Actually, $J^{(D)}$ is just a common binary cross-entropy cost function defined by a binary classifier with sigmoid output. The only difference is that this classifier is training by two different types of minibatches of data, one from the training set with labels set to 1 and another coming from the generator with labels set to 0.

The generator cost function $J^{(G)}$ originally was defined as $-J^{(D)}$ as a part of the game theory setting defining in this way a zero-sum game (or minimax). The problem with this cost function for G is that saturates when the D is very confident when rejecting samples and the generator's gradients vanishes. So in practice, minimax framework is left and more computational stable generator cost function was applied:

$$J^{(G)} = -\frac{1}{2}\mathbb{E}_z \log(D(G(z))) \tag{2.2}$$

Mainly, a GAN design is based on the selection of D and G architectures (mainly G) and the loss function. Descriptions above show the most basic ideas about GANs. From here, in the last few years hundred of new architectures and loss functions have been proposed making hard keep updated of all novelties in this field.

2.3 Generative Adversarial Networks in Medical Imaging

We mainly consider two kind of GANs methodologies suitable for applications of GANs in classification tasks: unconditional GANs and semi-supervised GANs. Moreover, we are interested in architectures capable to deal with 128x128 resolution images, which is a good enough resolution for image classification while having at the same time an affordable computation requirements with respect to GANs training.

2.3.1 Unconditional GANs

Unconditional image synthesis with GANs refers to the use of unsupervised GANs for image generation from random noise without additional information from classes or any kind of conditional information (unlike conditional GANs). First GAN model introduced by Goodfellow et al. (2014) was an unsupervised GAN defined by multilayer perceptrons, and Radford et al. (2016) was the first unsupervised GAN defined by CNNs, which exploits the successful CNNs in a unsupervised setting.

First DCGANs and variants were mainly applied by some authors to generate realistically looking low resolution synthetic images, with resolutions ranging from 16x16 to 64x64. Examples can we find in Kitchen and Seah (2017) where 16x16 patches of prostate lesions are generated, or in Chuquicusma et al. (2018) where 56x56 patches of lung cancer nodules are generated. The main objective of those works was to generate fine quality of medical images and any further downstream application was explored.

One of the first successful works applying unconditional GANs to downstream tasks as classification was Frid-Adar et al. (2018). In that work, due data scarcity issues, they applied a two stage pipeline where in the first stage they generated synthetic images using classical data augmentation methods that later in a second stage was used for training a DCGAN. With this approach they were able to increase sensitivity and specificity of liver lesion classification from 78.6% and 88.4% with classical data augmentation methods to 85.7% and 92.4% with additional GANs generated images, on a limited dataset of 64x64 computed tomography (CT) images.

Beyond 128x128 resolution images is quite difficult to obtain good quality of synthetic images with DCGANs, and progressive growing GANs based methods Karras et al. (2018) have been applied in Baur et al. (2018a) on 256x256 skin lesion images, or Korkinof et al.

(2018) where 1280x1024 images of mammograms where synthesized. Moreover, Baur et al. (2018b) shown that when dealing with limited and high variance data, DCGANs performance decreases notably from 64x64 to 128x128 up to 256x256 resolution images.

Unsupervised (or unconditional) image synthesis methods mentioned above can be applied to downstream image classification tasks. The main idea is to train a separated GAN for each class and use in a second stage, the generated images as data augmentation in the classification task. In a very limited data regime, as we saw in Frid-Adar et al. (2018), an intermediate step with classical data augmentation methods can be applied. The downside of these two stage methodology is that for a N classes classification task, N GANs models must be trained. Such methods discard the discriminative network once the training stage is completed and make use of the new generated images as data augmentation method.

We will see in the next section another approach to apply GANs models in classification tasks, mainly by exploiting class labels.

2.3.2 Semi-supervised GANs

Original unsupervised GANs applications for classification tasks make not use of annotated data as classes or categories. Another approach that has been used in classification is to exploit non-labeled data with a semi-supervised GAN architecture Sricharan et al. (2017). In this scheme, the generator remains the same as in unsupervised GANs, but the discriminator is fed also by labeled data. So instead learning to distinguishing between only two classes (real and fake), it learns to classify among N+1 classes, where N is the number of classes plus one class for real/fake. The final trained discriminator can be used as a classifier. The main idea is to exploit both limited labeled data and non-labeled data in a unified architecture. Notice that in this case, both source of data, labeled and non-labeled must come from the same distribution.

This kind of schema has been applied successfully in medical imaging classification in Madani et al. (2018) for chest X-ray image classification and Lecouat et al. (2018) for for abnormality classification in retinal images.

2.4 StyleGAN based architectures

As we saw in previous section DCGANs have been used for unconditional image generation and applied to classification tasks as a data augmentation method. Most of works based on DCGANs and variants have been successfully applied to low resolution images and have been shown less effective for medium/large resolution. 128x128 resolution is just in the limit of the capabilities of DCGANs and we will explore more advanced and recent architectures. StyleGAN family of architectures as StyleGAN Karras et al. (2019) and StyleGAN2 Karras et al.

(2020b) have been state of the art in the last year with respect to high realistic and high resolution image generation. However, applications of StyleGAN based models are scare in medical imaging, mainly because this kind of architectures need tens of thousands of examples. To our knowledge, there is no previous work applying StyleGANs to ultrasound images. And example in whole-body magnetic resonance imaging (wbMRI) image generation can we find in Chang et al. (2020) in which DCGAN and StyleGAN family of GANs architectures are compared. They demonstrated that StyleGAN2 provides best performance with respect radiologist visual inspection, although the Fréchet Inception Distance (FID) obtained by StyleGAN2 was not the best, proposing a novel metric for MRI.

Recently, GANs have incorporated in-built data augmentation mechanisms to face data scarcity. Works by Zhao et al. (2020) and Karras et al. (2020a) show good performance with one order of magnitude less amount of data. Our main objective is to study and evaluate these recent techniques and their applicability to fetal brain ultrasound images.

Chapter 3

Methodology

In this chapter we describe the methodology and study design carried out in this project. As we mention previously the main objective is to study the feasibility of Generative Adversarial Networks as a data augmentation technique for downstream classification task of fetal brain planes.

Due to computational resources limitation, we state two main simplifications of the task: we simplify the classification task to two planes instead three. This means that we trained two unconditional GANs for planes trans-ventricular (trv) and trans-thalamic (dbp), which are the most hard to classify. We also perform GANs training over images of resolution 128x128, reducing considerably training time and complexity of GANs. These two simplifications save us computational time giving us at the same time an approximate understanding of the whole problem.

3.1 Experiment design

The experiment design consists in three main stages that we depict in figure 3.1. In a first stage we carry out data preparation, from the original dataset to the final baseline dataset that was used for experiments (section 3.2). In the second stage we perform GANs training in Google Colab (section 3.3). Finally, we train classifiers for both baseline dataset and GAN augmented datasets (section 3.4). In the following sections we describe these stages in more detail.

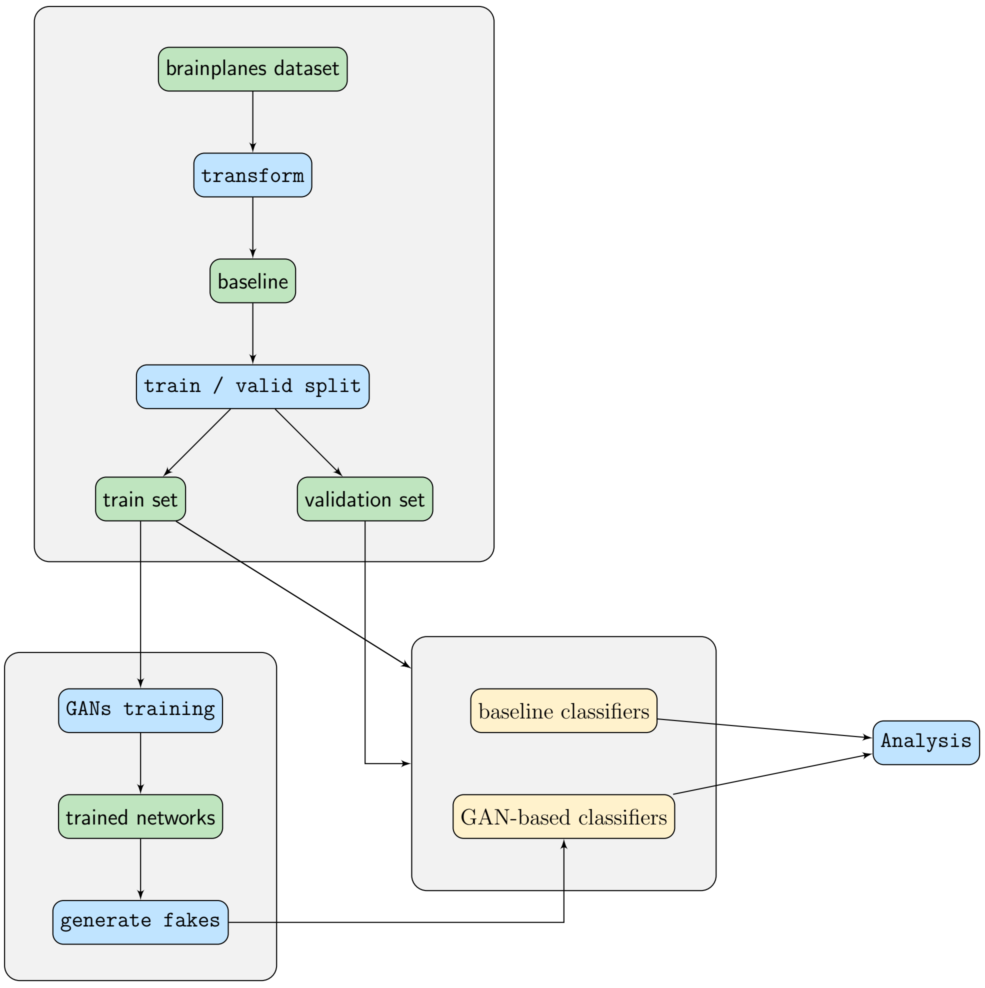


Figure 3.1: *Experiment design*

3.2 Dataset

The original dataset is comprised of 8747 images, 3436 corresponding to trans-ventricular (trv) planes and 5311 corresponding to trans-thalamic (dbp) planes. This dataset comes from a broader dataset, including trans-cerebellar planes, and it was collected by BCNatal, a center with two sites (Hospital Clinic and Hospital Sant Joan de Deu, Barcelona, Spain), with large maternal-fetal experienced practice. Images were acquired from a total of six different US machines by several different operators with similar experience. All images were cropped by means of an automatic brain detector. In image 3.2 a sample of five trans-ventricular and five trans-thalamic are shown.

Along with images, a description file in csv format is given with several fields, as image name, brain plane, patient and others that we handle with pandas *dataframe*. In a first stage we remove all unnecessary columns leaving the image name, patient anonymized identification and brain plane label.

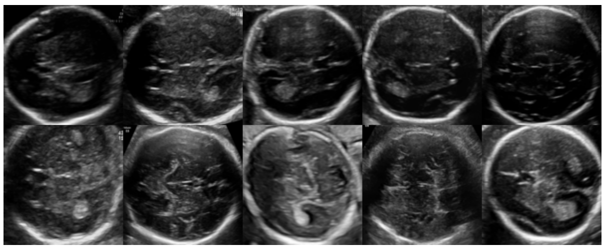


Figure 3.2: *Some examples of the dataset. Top row: trans-ventricular. Bottom row: trans-thalamic*

	training	validation	total
trv	1656	1780	3436
dbp	2620	2691	5311
total	4276	4471	8747

Table 3.1: *Train/validation split with no overlapping patient samples.*

In a second stage a baseline dataset is built with resolution 128x128 and a train-validation split (50%-50%) method by means of *GroupShuffleSplit* sklearn function with seed set to 19, splitting in this way the dataset into training and validation sets with no overlapping patient samples. Once the split is done the patient column is removed. The final *dataframe* consists of 8747 rows and three columns: *image*, *brain_plane* and *Valid*. In table 3.1 the train/split of this baseline is shown. The main goal is to compare performance between GANs-based classifiers and no-augmented classifiers with the validation set. In the following stages, the training set will be used for both, GANs training and baseline classifiers training.

3.3 GANs training and image generation

In this section we describe the training process of GANs. Since we are approaching this problem under an unconditional framework, two GANs were trained, one for each class. The training was done in Google Colab with our own stylegan2-ada[1] fork with minor modifications and custom training configurations. GANs training is entirely based on Karras et al. (2020a) work, which first version was published on June 11th, 2020 and its open implementation that was published in github on October 8th, 2020.

In the following sections we describe the training configuration we applied for GANs training 3.3.1, the evaluation methods used for network snapshot selection 3.3.2 and the procedure for image generation 3.3.3.

[1]https://github.com/albertoMontero/stylegan2-ada

mb	mbstd	fmaps	ema
32	4	0.5	10

Table 3.2: *Fixed parameters for one gpu and 128x128 resolution image.*

3.3.1 Training

stylegan2-ada has several training parameters that are described in Karras et al. (2020a). Selecting the proper parameters is the harder decision due to time spent in training GANs. We made decisions based on experiments and suggestions described in that work (particularly on similar data regime datasets) and defaults defined in the implementation, while evaluation was preformed through metrics defined in next section.

Among the training parameters we have some that we might consider fixed due computational resources (number of gpus available) and image resolution. In our case, since we can only use one gpu and our image resolution is 128x128 we set the following parameters shown in table 3.2, [2] where mb is the minibatch size, mbstd the minibatch standard deviation layer at the end of the discriminator, fmaps is the ratio of feature maps used with respect high resolution settings, and ema is the exponential moving average of generator weights.

On the other hand other parameters are more sensitive to particular datasets, like R_1 regularization γ, learning rate, the target value for r_t overfitting heuristic and augpipe (augmentation pipeline).

We remind the reader that the training dataset is comprised of 2620 trans-thalamic (dbp) images and 1656 trans-ventricular (trv) images. For trans-thalamic (dbp) images we set R_1 regularization γ as 0.16, which is the suggested value (and it is slightly different from the implementation default), the target value to 0.5 since it is shown that it works better than default value of 0.6 for small datasets (see figure 5 (b) in Karras et al. (2020a)), learning rate to 0.002 and augpipe to configuration 'bg' (blit and geometric augmentations) which is shown that performs better than default 'bgc', specially for gray images.

For trans-ventricular (trv) images we had to experiment more with R_1 regularization γ with values 0.08 and 0.24 getting the best results for $\gamma = 0.24$. Also we decrease the learning rate to 0.001 and use data augmentation setting parameter mirror to 1, which it seems to be beneficial for small datasets.

For both image sets we set the mapping network depth (map) to 8 although values between 2 and 8 are very similar and a depth equals to 2 might be slightly better (see appendix D.1 Karras et al. (2020a)). Finally, although there are not pre-trained networks for 128x128 image resolution, we find out that higher resolution trained networks can be used for 128x128 resolu-

[2]We can find these fixed parameters in *auto* configuration of stylegan2-ada as a suggestion for a good starting point.

	mb	mbstd	fmaps	ema	γ	target	lr	augpipe	map	TL
dbp	32	4	0.5	10	0.16	0.5	0.002	bg	8	celebahq256
trv	32	4	0.5	10	0.24	0.5	0.001	bg	8	celebahq256

Table 3.3: *GANs training configuration for dbp and trv images.*

tion for transfer learning. In our case we transferred from CelebAHQ pre-trained on 256x256 (celebahq256) images for both image sets. In table 3.3 we show the final configuration for GANs training.

With above configurations we trained dbp-GAN in five Colab sessions (about 45 hours) and trv-GAN in three sessions (about 27 hours).

3.3.2 GANs evaluation

GANs have proved to be remarkably effective at generating both high-quality and extensive synthetic images in a range of problem domains. However, GANs lack an objective function, which makes it difficult to compare performance of different models.

Several quantitative measures have been proposed to evaluate GANs performance in the last years. Specifically, stylegan2-ada implements the following metrics for small datasets: FID (Fréchet inception distance Heusel et al. (2018)), KID (Kernel inception distance Bińkowski et al. (2018)) and PR (Precision and Recall Kynkäänniemi et al. (2019)). These metrics are calculated over the whole training set, instead of 50K real images as in previous versions.

Fréchet inception distance

This is one of the most popular metrics to evaluate GANs performance. It was design to overcome some issues of its predecessor inception score (IS Salimans et al. (2016) related with the lack of diversity detection. The main idea is to embed a set of generated samples into a feature space given by an specific intermediate layer of Inception network (although other CNNs can be used). Then, the embedding space can be modeled as a continuous multivariate Gaussian and the mean and covariance can be estimated for both fakes images and real data. The Fréchet distance (also known as Wasserstein-2 distance) between these two Gaussians quantifies the quality of generated samples. Equation 3.1 shows FID score for distribution $g = (\mu_g, \Sigma_g)$ of embedded fakes with respect distribution $r = (\mu_r, \Sigma_r)$ of embedded real samples.

$$\text{FID}(r, g) = ||\mu_r - \mu_g||_2^2 + Tr(\Sigma_r + \Sigma_g - 2(\Sigma_r \Sigma_g)^{\frac{1}{2}}) \tag{3.1}$$

FID is more robust to noise and better measure for image diversity than IS. Since we are working in this project under an unconditional framework and training one GAN per class, one might think that we don't deal with classes, but classes and labels are artificial and in this case image diversity is important because as it was described in section 3.2 the dataset contains images acquired from several machines and IS might not be able to capture the lack of diversity with respect machines.

Precision and Recall

Unlike FID, the main idea of PR metric is to form explicit non-parametric representations of real and generated manifolds and estimate from them precision and recall. Similarly to FID, real and generated samples are drawn from $X_r \sim P_r$ and $X_g \sim P_g$ and embedded into a high-dimensional feature space using a pre-trained classifier network. Let ϕ_r and ϕ_g be real and generated feature vectors respectively, and Φ_r and Φ_g the corresponding sets of feature vectors. Then for any ϕ and any Φ a binary function is defined as in equation 3.2.

$$f(\phi, \Phi) = \begin{cases} 1, & \text{if } ||\phi - \phi'||_2 \leq ||\phi - NN_k(\phi', \Phi)||_2 \text{ for at least one } \phi' \in \Phi \\ 0, & \text{otherwise} \end{cases} \tag{3.2}$$

where $NN_k(\phi', \Phi)$ is the kth nearest feature vector of $\phi' \in \Phi$.

This equation defines a way to decide whether a given image looks realistic or might be produced by the generator with $f(\phi, \Phi_r)$ and $f(\phi, \Phi_g)$ respectively.

Finally, precision and recall are defined in equations 3.3 and 3.4

$$\text{precision}(\Phi_r, \Phi_g) = \frac{1}{|\Phi_g|} \sum_{\phi_g \in \Phi_g} f(\phi_g, \Phi_r) \tag{3.3}$$

$$\text{recall}(\Phi_r, \Phi_g) = \frac{1}{|\Phi_r|} \sum_{\phi_r \in \Phi_r} f(\phi_r, \Phi_g) \tag{3.4}$$

In practice Kynkäänniemi et al. (2019) set $k = 3$ and features vectors ϕ are calculated with a pre-trained VGG-16 classifier, extracting the activation vector after the second fully connected layer.

Karras et al. (2020a) notice that FID is not a good metric for small datasets and suggests to use KID instead. We observed in our experiments that both are quite correlated. On the other hand, precision and recall has little bias for small datasets and in Ravuri and Vinyals (2019) observed that precision is better than recall for GANs-based classifiers, although in that work another architecture (BigGAN) and another dataset (ImageNet) was studied. These two observations suggest that precision and recall might be better metric for GANs-based classifiers than other metrics available. With this in mind, the strategy for selecting a network was as following: among the three or four networks with lower FID, those with higher precision were selected for both GANs, although we found experimentally that high precision and low FID are quite correlated. We have to mention at this point that, as we will see in next section, precision

	FID	precision	recall
dbp	13.08	0.6616	0.3336
trv	17.4856	0.6609	0.2850

Table 3.4: *Metrics: FID, precision and recall for dbp and trv trained GANs.*

may be controlled by means of the truncation trick, so maybe a better strategy would be to select those networks with higher recall. In any case, among better networks with respect FID, we didn't observe significant differences between precision and recall.

In table 3.4 the obtained metrics are shown. We can compare these values with results obtained in Karras et al. (2020a) for BRECAHAD dataset (Aksac et al. (2019)) which consists of breast cancer histopathology images with similar number of training images (1994, compared with 1656 for trv and 2620 for dbp), where they obtained a FID value of 15.71 when training from scratch and 16.33 when using transfer learning.

3.3.3 Fakes generation

An open problem in all generative models is the difficulty for the generator to learn from low density areas that are poorly represented. In order to improve quality of samples, a method call truncation trick proposed by Brock et al. (2019) is applied in most of recent architectures. The main idea consists in sampling from a truncated distribution instead from original distribution applied in training. In most of the cases models are trained with $\mathcal{N}(0, I)$ or $\mathcal{U}[-1, 1]$. Sampling from these truncated distributions will generate in most of the models more realistic images, increasing precision at the price of less variety or recall. As the truncated threshold $\psi \to 0$ the generated images tend to some kind of training set mean image. On the contrary, when $\psi \to 1$, generated images show high variety but also might present artifacts and seem unrealistic.

Notice that in stylegan architecture family, the truncation is done over an intermediate latent space $w \in \mathcal{W}$ (Karras et al. (2019)) coming from a mapping network consisting of 8 fully connected layers, instead of traditional latent code $y \in \mathcal{Z}$ defined by the input layer. Although in theory negative values of ψ are possible, we didn't experiment in this work with them, limiting the experiments only to positive values.

Figures 3.3 and 3.4 show five sweeps by rows over ψ for values $\psi = 0.1, 0.2, \ldots, 0.9, 1$ for five random seeds over dbp and trv respectively. We can observe that for $\psi = 0.1$ (first column) images are more similar to each other than for $\psi = 1$ (last column).

Figures 3.5 and 3.6 show 25 random generated images for $\psi = 0.3, 0.5, 0.7, 1$ for both dbp and trv planes, using the same 25 seeds for each grid.

As we will see in chapter 4 the selection of ψ value when generating images is an important parameter to study, giving different results with respect classification tasks.

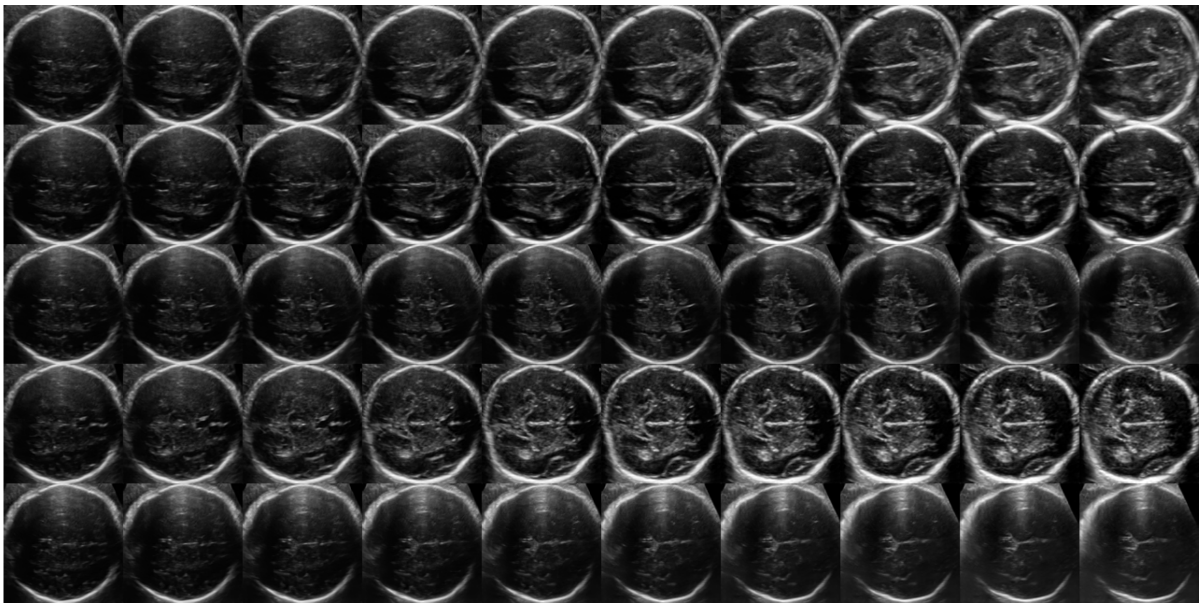


Figure 3.3: *dbp fakes generation:* $\psi = 0.1, 0.2, \ldots, 0.9, 1$ *sweep for five random seeds.*

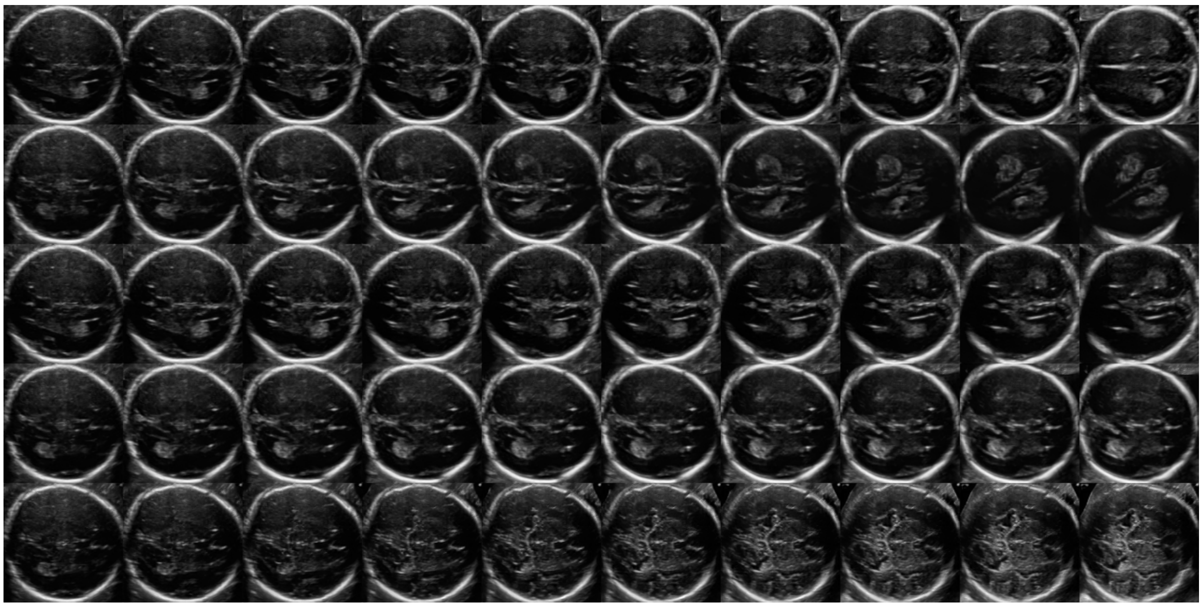
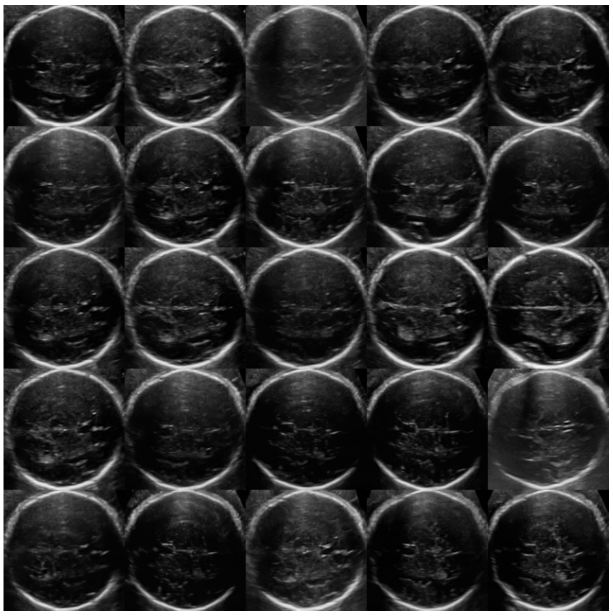
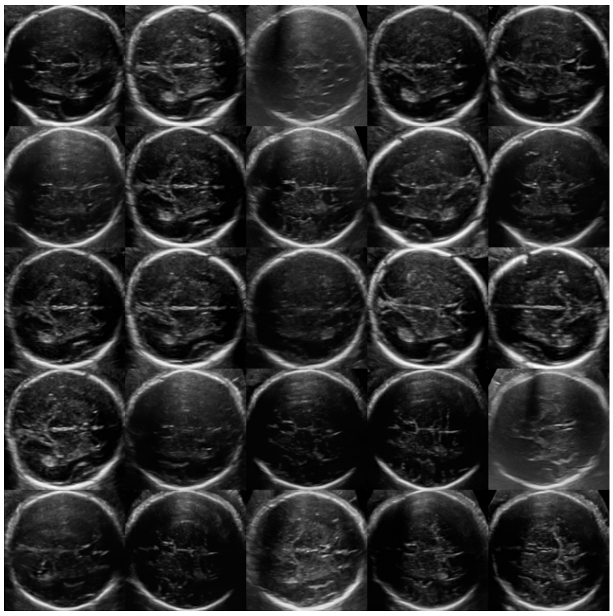
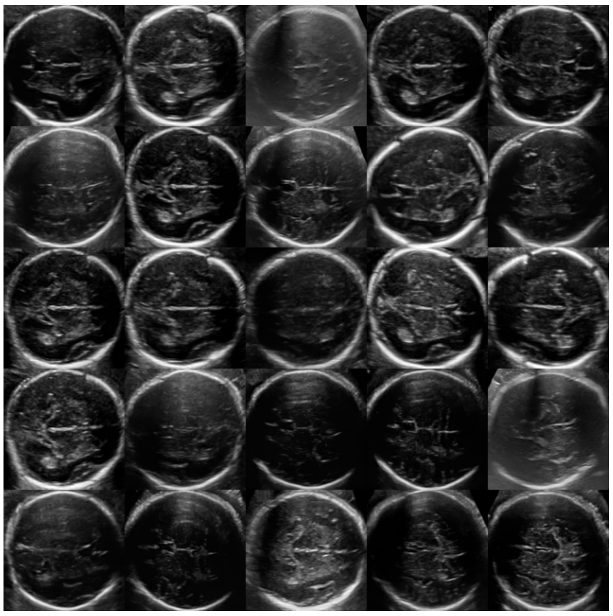


Figure 3.4: *trv fakes generation:* $\psi = 0.1, 0.2, \ldots, 0.9, 1$ *sweep for five random seeds.*


(a) $\psi = 0.3$


(b) $\psi = 0.5$


(c) $\psi = 0.7$

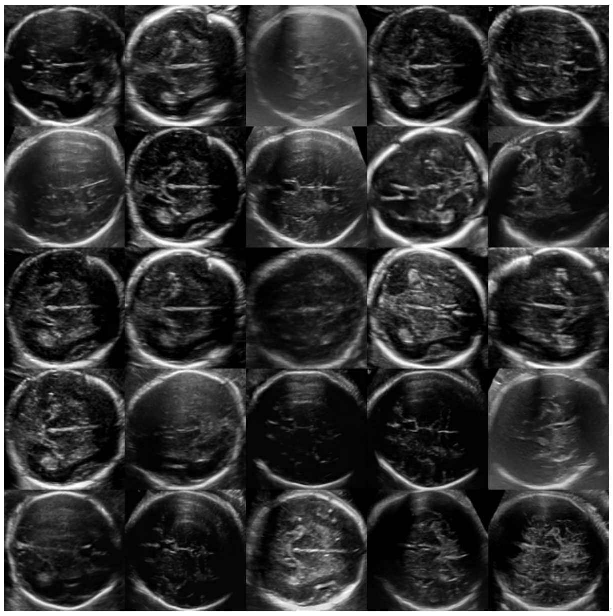
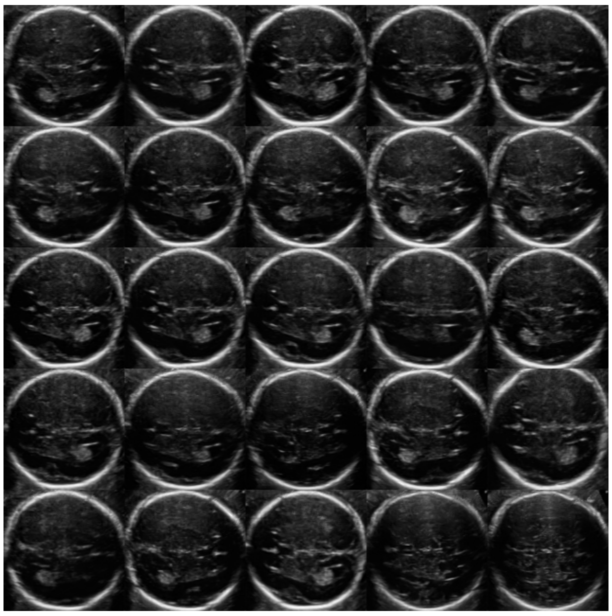
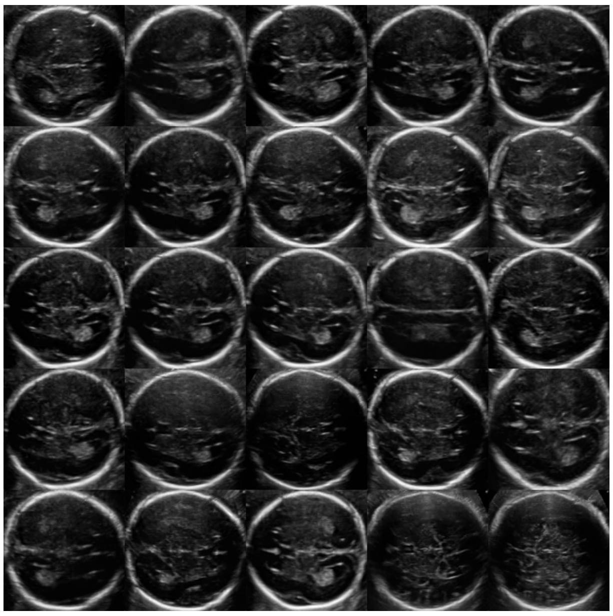
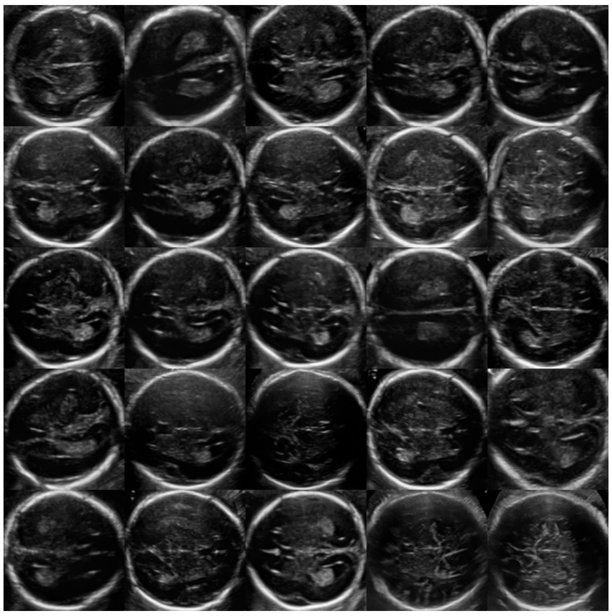

(d) $\psi = 1$

Figure 3.5: Generation of dbp images for some random seeds and different ψ. Same 25 seeds were applied to each grid giving the same 25 brain plane generation for four ψ values.


(a) $\psi = 0.3$


(b) $\psi = 0.5$


(c) $\psi = 0.7$

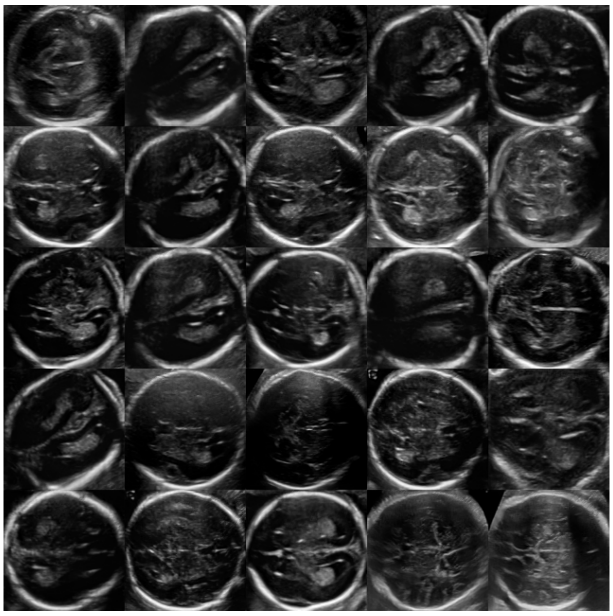

(d) $\psi = 1$

Figure 3.6: Generation of trv images for some random seeds and different ψ. Same 25 seeds were applied to each grid giving the same 25 brain plane generation for four ψ values.

3.4 Classifiers

The main goal of this work is to study the feasibility of GANs-based classifiers. To perform comparisons we implement the same classifier for both GANs-based and classical classification with no fakes images, with default settings based on fastai library, which has a good set of default values for hyper-parameters and data augmentation. These defaults have been shown to work quite good in most of classification tasks Howard and Gugger (2020).

As shown in figure 3.1, training set was used for training both, baseline classifiers and GANs-based classifiers and the validation set used to compare performance. In addition to real images, GANs-based classifiers were trained also with fakes images. In chapter 4 we describe all experiments and results.

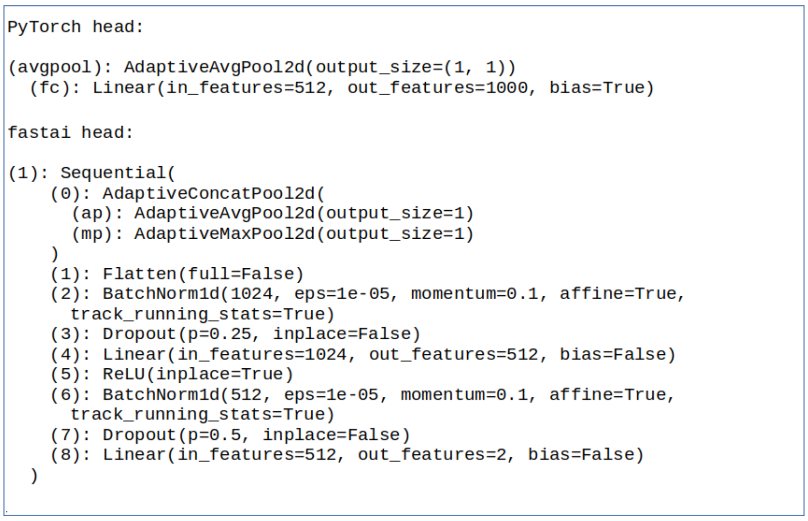
3.4.1 Implementation details

3.4.1.1 Architecture

We describe briefly in this section the deep learning architecture implementation details applied to both baseline and GANs-based classifiers.

The classifier architecture is based on Resnet architecture He et al. (2015) with pre-trained weights on ImageNet dataset, slightly modified by fastai library. The differences between original PyTorch Resnet and fastai version are two. First, the head of original PyTorch Resnet (Paszke et al. (2019)) is modified, adding one more *Linear* layer with dimensions (in_features, out_features) = (1024, 512) before the last layer, and modifying the dimension of the last one to (512, 2), since we are dealing with a binary classification task. Moreover, the next to last layer is followed by *ReLU*, *BatchNorm* and *Dropout* (p=0.5) layers and preceded by *Flatten*, *BatchNorm* and *Dropout* (p=0.25). These modifications applied to the head give most of the time an increase of performance in classification tasks.

The second modification is located in the link between body and head. While original PyTorch Resnet has an *AdaptiveAvgPool2d* layer connecting the last feature map with the head, fastai version has *AdaptiveAvgPool2d* and *AdaptiveMaxPool2d* layers concatenated. Again, fastai team have experimentally observed that by concatening both *AdaptiveAvgPool2d* and *AdaptiveMaxPool2d* layers give in most of the cases better performance that using a unique *AdaptiveAvgPool2d* layer. Notice that in both cases, input image size can be smaller than (224, 224). The body of the Resnet remains the same as the original PyTorch. In figure 3.7 differences between both architectures are shown.



```
PyTorch head:

(avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=512, out_features=1000, bias=True)

fastai head:

(1): Sequential(
    (0): AdaptiveConcatPool2d(
      (ap): AdaptiveAvgPool2d(output_size=1)
      (mp): AdaptiveMaxPool2d(output_size=1)
    )
    (1): Flatten(full=False)
    (2): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True,
      track_running_stats=True)
    (3): Dropout(p=0.25, inplace=False)
    (4): Linear(in_features=1024, out_features=512, bias=False)
    (5): ReLU(inplace=True)
    (6): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True,
      track_running_stats=True)
    (7): Dropout(p=0.5, inplace=False)
    (8): Linear(in_features=512, out_features=2, bias=False)
  )
```

Figure 3.7: *Comparison between original PyTorch and default fastai Resnet head architecture.*

3.4.1.2 Training

The same training procedure was applied to baseline and GANs based classifiers. All classifiers were trained a maximum of 20 epochs with early stopping on validation loss with patience set to 5 and batch size to 64. The network was fine tuned the first epoch, by training only the head, while the remaining epochs all the layers were unfrozen and trained. All hyper-parameters were set using default fastai values with *fine_tune* method. Moreover, loss function and optimizer were set to default values of *create_lerner* method. Finally, also default data augmentation was applied with method *aug_transforms* with image resolution of 128x128. As we mention previously, all these default values define good baselines.

Chapter 4

Experiments and Results

In this chapter we describe the experiments performed in this work.

Generation of fakes is done off-line in order to speed up the experiments. That is, the generated images are limited to subsets of the first N generated images for each plane defined by seeds $1 - N$, where $N = 30000$ for each plane. All experiments are performed over 5 runs with pre-defined seeds. In the case of experiments involving fakes, each run samples a different set of fakes with limitations mentioned above. In table 4.1 results for the baseline are shown. In all experiments, accuracy (acc), area under the curve score (auc), as well as maximum and minimum for both, and validation loss are reported.

In the next two sections two kind of experiments are described: augmentation experiments, where the training dataset is augmented with fake images, and replacement experiments, where some real training samples are replaced by fakes.

4.1 Augmentation experiments

The first question we ask ourselves is whether augmenting the training set with GANs generated images improve the performance of brain plane classification task. As we mentioned in previous chapter the quality of images can be controlled by the truncation parameter ψ. We performed several experiments for values $\psi = 0.3, 0.5, 0.7, 1$, with increasing amount of fakes defined by ratios $r_a = \frac{\#\text{fakes}}{\#\text{train set}}$ with respect the training set. Tables from 4.2 to 4.5 show results obtained and figure 4.1 shows graphs for all four experiments.

	acc	$\max_{acc}$	$\min_{acc}$	auc	$\max_{auc}$	$\min_{auc}$	loss_{avg}
resnet18	0.799 ± 0.004	0.805	0.792	0.850 ± 0.003	0.855	0.844	0.460

Table 4.1: *Baseline results for 5 runs.*

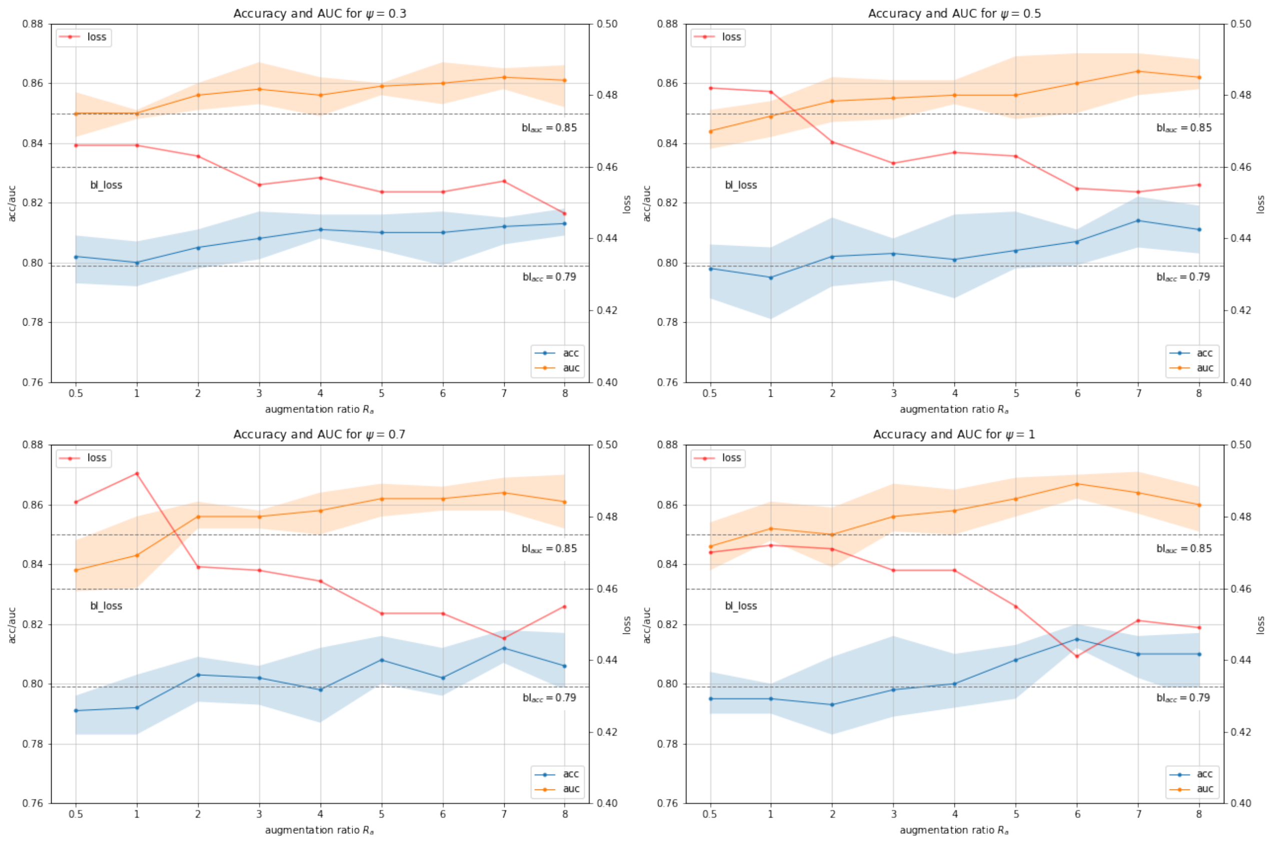


Figure 4.1: Accuracy (blue, with max and min), auc (orange, with max and min) and validation loss (red) for experiments with $\psi = 0.3$, $\psi = 0.5$, $\psi = 0.7$ and $\psi = 1$. Vertical gray lines are baseline auc, validation loss and accuracy.

R_a	fakes$_{trv}$	fakes$_{dbp}$	acc	max$_{acc}$	min$_{acc}$	auc	max$_{auc}$	min$_{auc}$	loss$_{avg}$
bl	0	0	0.799 ± 0.004	0.805	0.792	0.850 ± 0.003	0.855	0.844	0.460
0.5	828	1310	0.802 ± 0.006	0.809	0.793	0.85 ± 0.006	0.857	0.842	0.466
1	1656	2620	0.8 ± 0.005	0.807	0.792	0.85 ± 0.001	0.851	0.848	0.466
2	3312	5240	0.805 ± 0.004	0.811	0.798	0.856 ± 0.003	0.86	0.851	0.463
3	4968	7860	0.808 ± 0.006	0.817	0.801	0.858 ± 0.005	0.867	0.853	0.455
4	6624	10480	0.811 ± 0.003	0.816	0.808	0.856 ± 0.005	0.862	0.849	0.457
5	8280	13100	0.81 ± 0.004	0.816	0.804	0.859 ± 0.002	0.86	0.856	0.453
6	9936	15720	0.81 ± 0.006	0.817	0.799	0.86 ± 0.005	0.867	0.853	0.453
7	11592	18340	0.812 ± 0.003	0.815	0.806	$\mathbf{0.862} \pm 0.003$	0.865	0.858	0.456
8	13248	20960	$\mathbf{0.813} \pm 0.003$	0.818	0.809	0.861 ± 0.005	0.866	0.852	**0.447**

Table 4.2: *Augmentation experiment for $\psi = 0.3$ (5 runs). In left column augmentation ratios with respect training set size are shown. Baseline metrics in first row.*

R_a	fakes$_{trv}$	fakes$_{dbp}$	acc	max$_{acc}$	min$_{acc}$	auc	max$_{auc}$	min$_{auc}$	loss$_{avg}$
bl	0	0	0.799 ± 0.004	0.805	0.792	0.850 ± 0.003	0.855	0.844	0.460
0.5	828	1310	0.798 ± 0.007	0.806	0.788	0.844 ± 0.005	0.851	0.838	0.482
1	1656	2620	0.795 ± 0.008	0.805	0.781	0.849 ± 0.004	0.854	0.842	0.481
2	3312	5240	0.802 ± 0.008	0.815	0.792	0.854 ± 0.006	0.862	0.847	0.467
3	4968	7860	0.803 ± 0.005	0.808	0.794	0.855 ± 0.005	0.861	0.848	0.461
4	6624	10480	0.801 ± 0.009	0.816	0.788	0.856 ± 0.003	0.861	0.853	0.464
5	8280	13100	0.804 ± 0.007	0.817	0.798	0.856 ± 0.007	0.869	0.848	0.463
6	9936	15720	0.807 ± 0.004	0.811	0.799	0.86 ± 0.006	0.87	0.85	0.454
7	11592	18340	$\mathbf{0.814} \pm 0.006$	0.822	0.805	$\mathbf{0.864} \pm 0.004$	0.87	0.856	**0.453**
8	13248	20960	0.811 ± 0.006	0.819	0.803	0.862 ± 0.004	0.868	0.858	0.455

Table 4.3: *Augmentation experiment for $\psi = 0.5$ (5 runs). In left column augmentation ratios with respect training set size are shown. Baseline metrics in first row.*

R_a	fakes$_{trv}$	fakes$_{dbp}$	acc	max$_{acc}$	min$_{acc}$	auc	max$_{auc}$	min$_{auc}$	loss$_{avg}$
bl	0	0	0.799 ± 0.004	0.805	0.792	0.850 ± 0.003	0.855	0.844	0.460
0.5	828	1310	0.791 ± 0.004	0.796	0.783	0.838 ± 0.006	0.848	0.831	0.484
1	1656	2620	0.792 ± 0.008	0.803	0.783	0.843 ± 0.009	0.856	0.832	0.492
2	3312	5240	0.803 ± 0.005	0.809	0.794	0.856 ± 0.004	0.861	0.852	0.466
3	4968	7860	0.802 ± 0.005	0.806	0.793	0.856 ± 0.003	0.858	0.852	0.465
4	6624	10480	0.798 ± 0.008	0.812	0.787	0.858 ± 0.005	0.864	0.85	0.462
5	8280	13100	0.808 ± 0.006	0.816	0.8	0.862 ± 0.005	0.867	0.856	0.453
6	9936	15720	0.802 ± 0.006	0.812	0.796	0.862 ± 0.003	0.866	0.858	0.453
7	11592	18340	$\mathbf{0.812} \pm 0.004$	0.818	0.807	$\mathbf{0.864} \pm 0.003$	0.869	0.858	**0.446**
8	13248	20960	0.806 ± 0.008	0.817	0.798	0.861 ± 0.007	0.87	0.852	0.455

Table 4.4: *Augmentation experiment for $\psi = 0.7$ (5 runs). In left column augmentation ratios with respect training set size are shown. Baseline metrics in first row.*

R_a	fakes$_{trv}$	fakes$_{dbp}$	acc	max$_{acc}$	min$_{acc}$	auc	max$_{auc}$	min$_{auc}$	loss$_{avg}$
bl	0	0	0.799 ± 0.004	0.805	0.792	0.850 ± 0.003	0.855	0.844	0.460
0.5	828	1310	0.795 ± 0.005	0.804	0.79	0.846 ± 0.006	0.854	0.838	0.47
1	1656	2620	0.795 ± 0.004	0.8	0.79	0.852 ± 0.005	0.861	0.848	0.472
2	3312	5240	0.793 ± 0.009	0.809	0.783	0.85 ± 0.007	0.859	0.839	0.471
3	4968	7860	0.798 ± 0.009	0.816	0.789	0.856 ± 0.005	0.867	0.851	0.465
4	6624	10480	0.8 ± 0.006	0.81	0.792	0.858 ± 0.005	0.865	0.85	0.465
5	8280	13100	0.808 ± 0.007	0.813	0.795	0.862 ± 0.006	0.869	0.856	0.455
6	9936	15720	$\mathbf{0.815} \pm 0.003$	0.82	0.812	$\mathbf{0.867} \pm 0.003$	0.87	0.862	**0.441**
7	11592	18340	0.81 ± 0.006	0.816	0.802	0.864 ± 0.005	0.871	0.857	0.451
8	13248	20960	0.81 ± 0.008	0.817	0.795	0.86 ± 0.006	0.866	0.851	0.449

Table 4.5: *Augmentation experiment for $\psi = 1$ (5 runs). In left column augmentation ratios with respect training set size are shown. Baseline metrics in first row.*

4.1.1 Analysis

As we can observe in tables and graphs, we found a little improvement with respect baseline metrics. Best results were for $\psi = 1$ for which we obtained a maximum accuracy of 0.815, which represents a 2% of improvement, and a maximum auc of 0.867, which means an improvement of 2% in auc. Finally, we observe an improvement of 0.019 in validation loss also for $\psi = 1$.

When comparing results above it doesn't seem to be a clear winner. Performances are very similar to each other and best are found for $R_a \geq 6$ and for $R_a = 8$ seem start decreasing. However we notice some differences with respect the quality of fakes. For fakes obtained by $\psi = 0.3$ (which have higher precision and lower recall than others), we see that classifiers are the first to cross the barrier of baseline validation loss at ratio $R_a = 3$ while the rest need more samples to reach similar results. That is, when we add less fakes ($R_a \leq 3$) the quality (precision) matters, but as we add more fakes, variety (recall) seems to compensate quality.

There is not much previous work related to GANs-based data augmentation for classification tasks in real applications to compare. In Frid-Adar et al. (2018) they perform GANs-based synthetic computed tomography (CT) images for downstream liver lesion classification task and they were able to increase sensitivity and specificity of liver lesion classification from 78.6% and 88.4% with classical data augmentation methods to 85.7% and 92.4%. Although in that work the dataset belongs to low-regime data with only 182 examples, and they applied a previous classical data augmentation method to increase the training set. Another related work is Ravuri and Vinyals (2019) where the experiment is performed with ImageNet dataset and BigGAN. They observed an increase in top-1 accuracy between 1% and 3% for a very few classes.

We have to mention here some important circumstances about this dataset that might affect the obtained performance to some extend. First, our training set has about 4.2K images, along with strong classical augmentation methods makes difficult to overcome baseline metrics.

Second, trv/dbp classification task is hard and there is little room for improvement. State of the art algorithms reach about 85% of accuracy (obviously with higher image resolution and better architectures). Actually, agreement between two specialists is 89.3% and 80% for trv and dbp respectively, and therefore, both the training and validation set may contain label errors. Although deep learning algorithms for classification are to some extend robust to noise, GANs training might be affected by mixing sample labels.

Last observation suggest a first task for future improvement: a sample filtering module to select good samples for GANs training. Similarly, a module for filtering good fakes candidates might be worth exploring. We discuss these points in more detail in chapter 7.

4.2 Replacement experiments

Augmentation experiments suggest that we might obtain similar performance replacing some real images by fakes. If we create a new baseline extending the training set with half of the validation set samples, we obtain similar performance to previous GANs-based experiments. We formalize this idea creating a new baseline setting and comparing with the same validation set as we describe below.

Replacement experiments are those where real images are replaced by fakes. With this kind of experiment we aim to explore a new point of view. Here the question to answer is: can we reach similar performance by replacing some real images by GANs generated images?

In order to be fair and not replace real samples that have been also get involved in GANs training, we implement a new baseline dataset from the previous one, in which the validation set is split again into train and validation with no overlapping patients, obtaining in this way a train/valid split of 75%-25%. However, in this case we differentiate training samples involved in GANs training and not. In table 4.6 the new train/valid split is shown. Notice that column gan shows the same amount of samples as the original baseline and we extend the training with about half of the original validation set with samples not involved in GANs training. In the following replacement experiments, all replacements will take place over training samples not involved in GANs training (no gan column). Also notice that the validation set is now half of the original which it means that the confidence of results are slightly lower than with initial baseline.

We show in tables 4.7 and 4.8 replacement experiments for $\psi = 0.3$ with augmentation ratios $R_a = 5$ and $R_a = 6$, and $\psi = 0.5$ with augmentation ratios $R_a = 6$ and $R_a = 7$. In both experiments all samples not involved in GANs training (854 and 1368 for trv and dbp respectively) are replaced by fakes. Metrics for new baseline are shown in first row.

planes	train		validation	total
	gan	**no gan**		
trv	1656	854	926	3436
dbp	2620	1368	1323	5311
total	6498		2249	8747

Table 4.6: *New baseline setting distinguishing between samples involved in GANs training (gan column) and not involved (no gan column).*

R_a	fakes$_{trv}$	fakes$_{dbp}$	acc	max$_{acc}$	min$_{acc}$	auc	max$_{auc}$	min$_{auc}$	loss$_{avg}$
bl	0	0	$\mathbf{0.808} \pm 0.005$	0.815	0.803	$\mathbf{0.861} \pm 0.004$	0.867	0.854	**0.444**
5	8280	13100	0.797 ± 0.009	0.808	0.783	0.853 ± 0.009	0.868	0.84	0.46
6	9936	15720	0.807 ± 0.002	0.81	0.805	$\mathbf{0.861} \pm 0.002$	0.864	0.858	0.457

Table 4.7: *Replacement experiment* $\psi = 0.3$. *In left column augmentation ratios with respect training set size are shown.*

R_a	fakes$_{trv}$	fakes$_{dbp}$	acc	max$_{acc}$	min$_{acc}$	auc	max$_{auc}$	min$_{auc}$	loss$_{avg}$
bl	0	0	$\mathbf{0.808} \pm 0.005$	0.815	0.803	0.861 ± 0.004	0.867	0.854	**0.444**
6	9936	15720	0.804 ± 0.005	0.81	0.795	0.86 ± 0.003	0.863	0.857	0.463
7	11592	18340	0.806 ± 0.006	0.813	0.796	$\mathbf{0.862} \pm 0.002$	0.865	0.859	0.452

Table 4.8: *Replacement experiment* $\psi = 0.5$. *In left column augmentation ratios with respect training set size are shown.*

4.2.1 Analysis

Results show that auc obtained are similar to baseline, although validation loss is slightly higher. For example, for $\psi = 0.5$ and $R_a = 7$ (table 4.8) we obtained an $auc = 0.862 \sim 0.861$. In this specific case we are replacing 2222 (854 + 1368) real images by 29932 (11592+18340) fakes and obtaining similar performance. This suggests that for this specific case with a set of 4276 (2620+1656) real training images, the waiting for the acquisition of about 2000 real images might be saved and replaced by images generated by a trained GAN.

By construction of train/validation split, these 2200 images are from different patients than the training set and taking into account that there are in average 3.5 images per patient, we conclude that the waiting for about 570 patient visits and examinations might be saved with a similar performance. Moreover, the acquisition, collection and selection of images containing planes of interest is realized by trained research technicians followed by a validation from a senior expert, which is a slow process and sensitive to mistakes. Thereby, this contribution might save time, resources and money in the whole process towards automatic classification of fetal brain planes.

4.3 Discussion

We enumerate and discuss briefly our findings and limitations in this work.

Findings

- The results of replacement experiments indicate that by applying GAN-based data augmentation to this specific use case with 4276 real training images, the waiting for 2200 images and about 570 patient examinations might be saved obtaining similar performance in binary classification of trans-ventricular and trans-thalamic fetal brain planes with respect baseline, with significant savings in time, resources and money.
- The results of augmentation experiments show an improvement of 2% in both accuracy and auc, and an improvement of 0.019 in validation loss when GAN-based data augmentation is applied to binary classification of trans-ventricular and trans-thalamic fetal brain planes with 4276 real training images, showing that when combined with classical data augmentation methods, GAN-based augmentation improves performance in this use case. We observed this improvement for all ψ values explored from a certain amount of fakes.
- We performed an exploratory grid search over several truncation values ψ and amount of fakes and obtained insights about about quality (precision) and variety (recall) trade-off when generating GAN-based synthetic images.
- To our knowledge, this is one the first works applying GAN-based data augmentation for classification tasks in ultrasound images and specifically in ultrasound fetal brain planes.
- To our knowledge this is the first project in medical imaging using recent work Karras et al. (2020a) and its implementation of the state of the art Generative Adversarial Network *stylegan2-ada* released on October 2020.

Limitations

- The main limitation of this work is that images used are clearly centered with almost no background, which it is highly beneficial when training these kind of GANs. We don't know to what extend these results might be extensible to not centered images containing more background.
- Another important limitation is that the image set used in GANs training contains about 4K images and we didn't explore performance for smaller training sets, which it is a common situation in medical imaging. Further experiments would be necessary to study the applicability to smaller datasets.

Chapter 5

Project structure, design and implementation

In this chapter we describe briefly the structure, software design and implementation used for this project. In appendix A we show the code of most important modules of experiment package which is the central package.

5.1 Project structure

The structure of the project has been organised as a typical data science project with the following directory structure:

```
usbrains
├── data
│   ├── interim
│   ├── processed
│   └── raw
├── experiments➦
├── notebooks
└── usbrains
    ├── classifiers
    ├── data
    ├── experiments
    ├── gans
    └── tests
```

- Data folder contains datasets in several stages, from raw to processed, with an interim stage containing a pre-processed snapshot of raw data.

- Experiments folder is a symbolic link to another machine partition and contains all fake-extended datasets used in different experiments.
- Notebooks folder with all notebooks implemented for data preparation, experiments, development, tests, visualizations, etc.
- Usbrains. Python source code of the project. We describe it briefly in next section.

5.2 Design

The software design is strongly coupled with the experimental design (see ????). The project is defined by for packages: *classifiers*, *data*, *experiments* and *gans*. Each package handles a specific group of tasks and it is composed of several modules. Most of the modules implement functions for specific sub-tasks. The *experiments* package is defined also by several classes, being Experiment an abstract class which implements a common run method for all derived classes (BaselineExperiment, AugmentationExperiment and ReplacementExperiment) and abstracts the method *create_train_df()* to be implemented for derived classes. Details in appendix A.

The source code has the following directory structure.

```
usbrains
 ├── classifiers
 │    ├── __init__.py
 │    └── baseline_classifier.py
 ├── data
 │    ├── __init__.py
 │    ├── basic_transforms.py
 │    ├── data.py
 │    ├── datasets.py
 │    ├── utils.py
 │    └── visualization.py
 ├── experiments
 │    ├── __init__.py
 │    ├── augmentation.py
 │    ├── baseline.py
 │    ├── datasets.py
 │    ├── experiment.py
 │    ├── replacement.py
 │    ├── results.py
 │    └── utils.py
 ├── gans
 │    ├── __init__.py
 │    └── generate.py
 └── tests
```

```
│
├── graphs.py
├── images.py
└── utils.py
```

Most relevant modules has documentation, and package documentation are in *__init__.py* files. We describe briefly package the dependencies and functionalities. In figure 5.1 the package dependencies are shown.

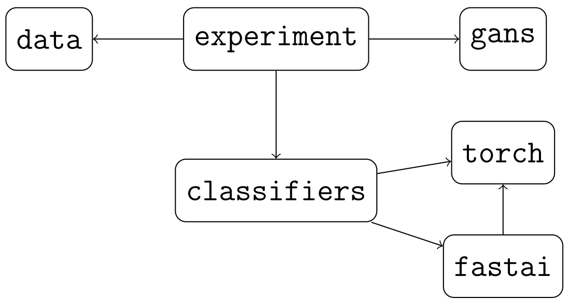


Figure 5.1: Package dependencies

5.2.1 Packages description

5.2.1.1 data package

The data package handles all processes related with data extraction, cleaning, transformation and baseline datasets creation.
Modules:
data: basic read, save, clean functionalities, and train_test_split() function with patient based train / test split.
datasets: contains create_dataset() method for baseline dataset creation.
basic_transforms: contains TransformAndSave callable method used by create_dataset() to perform parallel transforms.
visualization: basic functionalities for image visualization in jupyter notebooks.
utils: basic util functionalities.

5.2.1.2 experiments package

The experiments package handles the whole experimental process definition. Experiment class in experiment module is the core of the package. It is an abstract class with abstract method create_train_df() which is implemented by derived classes, and implements the run() method which it performs the experiment process. Also, it handles all settings related with training

parameters and paths configuration, experiment and report names etc, defining some default values that can be overridden.
Experiment pipeline (given GANs trained networks snapshots):
1. datasets.prepare_train_set(args)
2. e = ReplacementExperiment(args) [or any Experiment derived class]
3. e.run(args)
Modules:
experiment: Experiment class
baseline: BaselineExperiment class
augmentation: AugmentationExperiment class
replacement: ReplacementExperiment class
datasets: prepare_train_set() → prepare fakes offline, before running experiments
results: Result and Report classes.
utils: utils. Contains AttributeDict which is an attribute access style dictionary (some kind of EasyDict).

5.2.1.3 classifiers package

The classifiers package handles image classifier definition.
Modules:
baseline_classifier: run_classifier() → default classifier defined to perform classifications tasks. It defines a set of default settings and it is used to compare baseline with gan-augmented classifiers.

5.2.1.4 gans package

The gans package handles the generation of fake images given network snapshots.
Modules:
generate: generate_images() → generate images from given GAN network, trained with styleGAN2-ada.

5.2.2 Examples

We show in this section some examples of how to run experiments (see figure 5.2 for a visual description).

5.2.2.1 Fakes generation

In order to speed up the experimental stage, we perform offline fakes generation for several values of tpsi's. The following snippet creates 10K of class trv and 10K of class dbp, and save the images in ROOT_PATH/images.

```python
import experiments.datasets as ds
import data.data as data

# path folder where training images will be saved, fakes and reals
ROOT_PATH = Path("/usbrains/experiments/example/tpsi03")
# path folder where trained gan network snapshots are located
GANS_NETWORKS_PATH = ROOT_PATH / "ns"
# folder containing baseline dataframe
BASELINE_PATH = Path("/usbrains/data/processed/baseline_sz128")

# parameters
trv_ns = "network-snapshot-000512_s2.pkl"
dbp_ns = "network-snapshot-000065_s5.pkl"
tpsi = 0.3

gan_seeds = {
    "trv": "1-10000",
    "dbp": "1-10000"
}

# generate fakes and copy baseline dataset into ROOT_PATH / images
path = ds.prepare_train_set(trv_ns, dbp_ns, tpsi, gan_seeds, ROOT_PATH,
    ↪ BASELINE_PATH, GANS_NETWORKS_PATH)
```

5.2.2.2 Augmentation experiment

We show in this example a complete experiment for a specific value of $\psi = 0.7$, including the graph obtained in image 4.1 of chapter 4 corresponding to $\psi = 0.7$ and its table shown in 4.4

```python
from experiments.experiment import DEFAULT_SETTINGS
from experiments.augmentation import AugmentationExperiment
from graphs import plot_tpsi_experiment
from utils import create_tex_table
```

```
random.seed(19)
seeds = random.sample(range(1000), 5)

tpsi = "tpsi07"
experiment_path = Path("/usbrains/experiments/gan2/trv_512s2_dbp_65s5") / tpsi

name = "augmentation_grid"
settings = {"model_name": name, "experiment_name": name,
           "log_path": DEFAULT_SETTINGS.log_path / "augmentation" / tpsi}

e = AugmentationExperiment("baseline_sz128", experiment_path, setting_params=
    ↪ settings)

rs = [0.5, 1, 2, 3, 4, 5, 6, 7, 8] # augmentation ratios

for r in rs:
    trv_fakes = int(n_trv * r)
    dbp_fakes = int(n_dbp * r)
    e.setting_params.report_name = f"aug_ar_{r}"
    print("#" * 40)
    print(f"Running experiment: {e.setting_params.report_name}")
    print("#" * 40)
    res = e.run(0, 0, trv_fakes, dbp_fakes, seeds=seeds)


# creates plot automatically from all reports saved in e.results_path folder
plot_tpsi_experiment(e.results_path, 0.7, "augmentation ratio")

# creates latex table automatically from all reports saved in e.results_path folder
fname = f"latex_table_{tpsi}.txt"
create_tex_table(e.results_path, fname, "Augmentation experiment for $\psi=0.7$ (5
    ↪ runs)", "table:exp_aug_07")
```

5.2.2.3 Replacement experiment

Now we perform a replacement experiment where we replace some real images by fakes. Specifically we replace 500 trv and 500 dbp images by 3K trv fakes and 3K dbp fakes. We run the experiment 3 times with given seeds. All Experiment derived classes as ReplacementExperiment need two positional arguments, baseline name to be used in the experiment and a path where training images for experiment were previously created. Also, it has two optional arguments that allow to set the configuration related to training and settings. Defaults for both arguments are defined in experiment.py (see appendix A). Most relevant training events are logged to a file in log_path folder.

```
from experiments.experiment import DEFAULT_SETTINGS
from experiments.replacement import ReplacementExperiment

# generate seeds
random.seed(19)
seeds = random.sample(range(1000), 3)

# parameters
tpsi = "tpsi03"
# path folder with all training images, previously created by prepare_train_set()
experiment_path = Path("/usbrains/experiments/example") / tpsi
log_path = DEFAULT_SETTINGS.log_path / "example" / "replacement" / tpsi
name = "replacement_experiment_example"

settings_params = {"model_name": name, "experiment_name": name, "log_path": log_path
    ↪ }

e = ReplacementExperiment("baseline_sz128_2", experiment_path, setting_params=
    ↪ settings)

trv_reals = dbp_reals = 500
trv_fakes = dbp_fakes = 3000

result = e.run(trv, dbp, trv_fakes, dbp_fakes, seeds=seeds)
```

5.3 Software and Hardware

5.3.1 Software

- **Deep Learning**. For image generation we use the official Nvidia stylegan2-ada implementation which is supported for Tensorflow 1.14 and 1.15, and Python 3.6 and 3.7. We

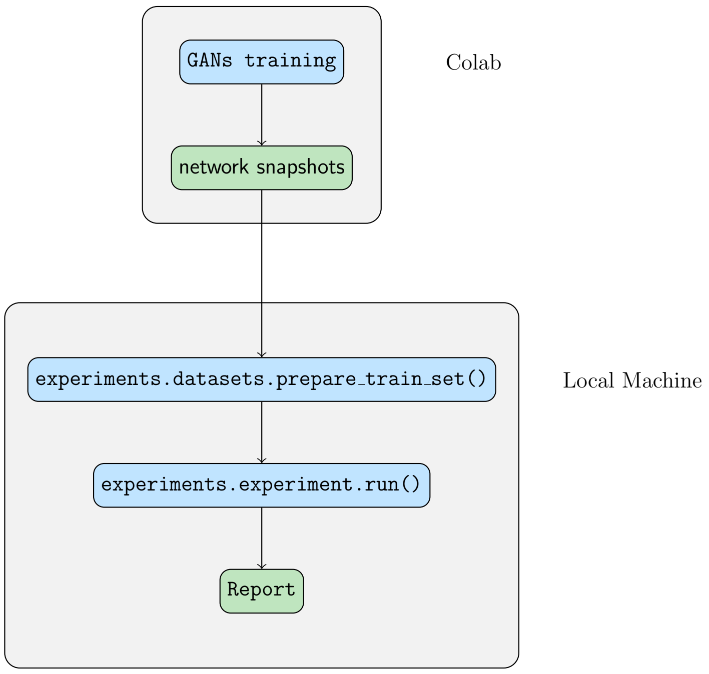

GANs training
network snapshots
Colab
experiments.datasets.prepare_train_set()
experiments.experiment.run()
Report
Local Machine

Figure 5.2: High level description of an experiment pipeline

forked and maintain our own repository in GitHub and adapt to our necessities with minor modifications [1].

The whole training process of GANs is executed on Google Colab with Tensorflow 1.15, where we clone our fork of stylegan2-ada. Once the training stage is finished we download the networks snapshots and the generation of fakes is executed in the local machine where we have cloned our fork of stylegan2-ada and installed its dependencies. We apply locally the same Tensorflow and Python versions as in Colab, that is, Tensorflow 1.15 and Python 3.6.

For our experimental project, we implemented a python project with several packages and modules, with Pytorch 1.8 and fastai 2.0.16. We decide developing in Pytorch instead Tensorflow, because we are more familiar and also because fastai (which is a wrapper for Pytorch) has a good default settings for classifiers that we apply for baselines.

- **Software development tools.** The project was implemented with PyCharm Community 2020.2.4, which is one of the best IDE's for Python development. For running experiments we exploit jupyter notebooks, which are friendly to use and at the same time serve as reporting. For version control we use git along with GitHub, where we maintain both stylegan2-ada fork and a private repository for our project called usbrains.

[1] https://github.com/albertoMontero/stylegan2-ada

5.3.2 Hardware

For GANs training we use Google Colab. Only P100 GPUs are able to run properly stylegan2-ada for training, with the limitation that we cannot calculate metrics at the same time. For metrics calculation, another session must be run with P100 or T4 GPU. For image generation and classification tasks we utilise a local machine with Ubuntu 18.04, 16 GB RAM and Nvidia GTX 1060 with 6 GB.

Chapter 6

Conclusions

This work aimed to study the performance of GAN-based image data augmentation applied to ultrasound fetal brain plane classification task. Based on the results obtained it can be concluded that *stylegan2-ada* GAN-based data augmentation applied to this dataset achieves an improvement in both accuracy and AUC of 2%. Additionally, results shown that data collection can be replaced to some extend by GANs generated images. In this experimental case study, results shown that with about 4K real training images the waiting for the collection of about 2K real images might be saved and replace by fakes. This is especially important when data acquisition is expensive and time consuming as in medical imaging.

Taking into account simplifications made in this project and considerations described in chapter 7 better performance is expected with further exploration. That said, GANs have their limitations and the efficiency of fakes generated by current methods is very far from real samples when dealing with classification tasks, although their impressive realism.

Chapter 7

Future work

The experimental design shown in 3.1 lacks two important modules that we think might affect the performance obtained in our experiments. The first module is a sort of quality control for GANs training set. In our current experimental pipeline we don't filter samples that are trained with GANs. We just split the dataset into train/validation sets based on patients but there is no a quality analysis of images we are using for training. This is an important aspect in most of the cases but it is even more relevant with this dataset because as we already mention in chapter 4 the dataset might contain errors in labels. The ground truth comes from an specialist and a previous work shows that the agreement between two specialists is 89.3% and 80% for trv and dbp respectively. Having bad samples in GANs training set might confuse models and we should avoid them. This bad samples filtering module might be implemented by filtering out samples with low agreement by specialists or by means of good classifiers and confidence scores. Another possibility that might be combined with above ones would be a recent work by DeVries et al. (2020) where they improve the quality of training samples by selecting those from high-density regions.

The second module missing in our experimental design is a fake quality control method. Precision and recall metrics of trained GANs give us some kind of measure of the quality of generated images. When generating fakes, we can control the precision and recall to some extend with truncation trick. But in any case this quality control by means of precision is with respect the whole set and not respect individual sample. A sample-centered precision metric as in Kynkäänniemi et al. (2019) might improve the quality of fakes and therefore improve the performance of GANs-based classifiers.

Another important point we skipped in our study is related to mode collapse. Although both GANs have been trained under an unconditional framework without classes, they have indeed classes. At least we might consider the ultrasound machine with which images where acquired as a class in both datasets. This means that at this point of the study we don't know how good is the ultrasound machine distribution generated by these two GANs. We would need to investigate this distribution and compare with real data because if distributions are not similar this might impact on performance.

When training stylegan2-ada based GANs there are some parameters sensitive to datasets that we would need to explore deeper. We found that for trv dataset (which contains 1656 training images compared to 2620 for dbp), decreasing the learning rate and increasing the regularization γ helped, but extensive experiments would be necessary, especially when dealing with small datasets.

After above considerations, we aim to apply this GANs-based augmented technique to a real scenario. This means remove all simplifications we made in our work concerning image resolution. Thus, with proper computational resources, the next natural step would be to train GANs with higher resolution and investigate if better performance for ultrasound fetal brain plane classification is possible. Moreover, with higher resolution generated images we would be able to realize visual experiments with domain experts that we skipped in this work due to low resolution of generated images.

Bibliography

Aksac, A., Demetrick, D. J., Ozyer, T., and Alhajj, R. (2019). Brecahad: a dataset for breast cancer histopathological annotation and diagnosis. *BMC Research Notes*, 12(1):82.

Baur, C., Albarqouni, S., and Navab, N. (2018a). Generating highly realistic images of skin lesions with gans. In *OR 2.0/CARE/CLIP/ISIC@MICCAI*.

Baur, C., Albarqouni, S., and Navab, N. (2018b). Melanogans: High resolution skin lesion synthesis with gans.

Bińkowski, M., Sutherland, D. J., Arbel, M., and Gretton, A. (2018). Demystifying mmd gans.

Brock, A., Donahue, J., and Simonyan, K. (2019). Large scale gan training for high fidelity natural image synthesis. *ArXiv*, abs/1809.11096.

Burgos-Artizzu, X. P., Coronado-Gutiérrez, D., Valenzuela-Alcaraz, B., Bonet-Carne, E., Eixarch, E., Crispi, F., and Gratacós, E. (2020). Evaluation of deep convolutional neural networks for automatic classification of common maternal fetal ultrasound planes. *Scientific Reports*, 10(1):10200.

Chang, A., Suriyakumar, V. M., Moturu, A., Tewattanarat, N., Doria, A., and Goldenberg, A. (2020). Using generative models for pediatric wbmri.

Chuquicusma, M. J. M., Hussein, S., Burt, J., and Bagci, U. (2018). How to fool radiologists with generative adversarial networks? a visual turing test for lung cancer diagnosis. *2018 IEEE 15th International Symposium on Biomedical Imaging (ISBI 2018)*, pages 240–244.

DeVries, T., Drozdzal, M., and Taylor, G. W. (2020). Instance selection for gans.

Frid-Adar, M., Klang, E., Amitai, M., Goldberger, J., and Greenspan, H. (2018). Synthetic data augmentation using GAN for improved liver lesion classification. *CoRR*, abs/1801.02385.

Goodfellow, I. (2017). Nips 2016 tutorial: Generative adversarial networks.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. C., and Bengio, Y. (2014). Generative adversarial networks. *ArXiv*, abs/1406.2661.

Hadlock, F. P., Harrist, R. B., Sharman, R. S., Deter, R. L., and Park, S. K. (1985). Estimation of fetal weight with the use of head, body, and femur measurements–a prospective study. *Am J Obstet Gynecol*, 151(3):333–337.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition.

Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S. (2018). Gans trained by a two time-scale update rule converge to a local nash equilibrium.

Howard, J. and Gugger, S. (2020). Fastai: A layered api for deep learning. *Information*, 11(2):108.

Karras, T., Aila, T., Laine, S., and Lehtinen, J. (2018). Progressive growing of gans for improved quality, stability, and variation. *ArXiv*, abs/1710.10196.

Karras, T., Aittala, M., Hellsten, J., Laine, S., Lehtinen, J., and Aila, T. (2020a). Training generative adversarial networks with limited data. *ArXiv*, abs/2006.06676.

Karras, T., Laine, S., and Aila, T. (2019). A style-based generator architecture for generative adversarial networks.

Karras, T., Laine, S., Aittala, M., Hellsten, J., Lehtinen, J., and Aila, T. (2020b). Analyzing and improving the image quality of stylegan. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8107–8116.

Kitchen, A. and Seah, J. (2017). Deep generative adversarial neural networks for realistic prostate lesion mri synthesis. *ArXiv*, abs/1708.00129.

Korkinof, D., Rijken, T., O'Neill, M., Yearsley, J., Harvey, H., and Glocker, B. (2018). High-resolution mammogram synthesis using progressive generative adversarial networks. *ArXiv*, abs/1807.03401.

Kynkäänniemi, T., Karras, T., Laine, S., Lehtinen, J., and Aila, T. (2019). Improved precision and recall metric for assessing generative models.

Lecouat, B., Chang, K., Foo, C.-S., Unnikrishnan, B., Brown, J. M., Zenati, H., Beers, A., Chandrasekhar, V., Kalpathy-Cramer, J., and Krishnaswamy, P. (2018). Semi-supervised deep learning for abnormality classification in retinal images.

Litjens, G., Kooi, T., Bejnordi, B. E., Setio, A. A. A., Ciompi, F., Ghafoorian, M., Laak, J. V. D., Ginneken, B., and Sánchez, C. (2017). A survey on deep learning in medical image analysis. *Medical image analysis*, 42:60–88.

Madani, A., Moradi, M., Karargyris, A., and Syeda-Mahmood, T. (2018). Semi-supervised learning with generative adversarial networks for chest x-ray classification with ability of data domain adaptation. *2018 IEEE 15th International Symposium on Biomedical Imaging (ISBI 2018)*, pages 1038–1042.

Pan, Z., Yu, W., Yi, X., Khan, A., Yuan, F., and Zheng, Y. (2019). Recent progress on generative adversarial networks (gans): A survey. *IEEE Access*, 7:36322–36333.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Pidhorskyi, S., Adjeroh, D., and Doretto, G. (2020). Adversarial latent autoencoders. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14092–14101.

Radford, A., Metz, L., and Chintala, S. (2016). Unsupervised representation learning with deep convolutional generative adversarial networks.

Ravuri, S. V. and Vinyals, O. (2019). Seeing is not necessarily believing: Limitations of biggans for data augmentation.

Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., and Chen, X. (2016). Improved techniques for training gans.

Sricharan, K., Bala, R., Shreve, M., Ding, H., Saketh, K., and Sun, J. (2017). Semi-supervised conditional gans. *ArXiv*, abs/1708.05789.

Sun, J.-Z., Bhattarai, B., and Kim, T.-K. (2020). Matchgan: A self-supervised semi-supervised conditional generative adversarial network. *ArXiv*, abs/2006.06614.

Vahdat, A. and Kautz, J. (2020). Nvae: A deep hierarchical variational autoencoder.

Whitworth, M., Bricker, L., Neilson, J. P., and Dowswell, T. (2010). Ultrasound for fetal assessment in early pregnancy. *The Cochrane database of systematic reviews*, (4):CD007058–CD007058. 20393955[pmid].

Yi, X., Walia, E., and Babyn, P. (2019). Generative adversarial network in medical imaging: A review. *Medical Image Analysis*, 58:101552.

Zhao, S., Liu, Z., Lin, J., Zhu, J.-Y., and Han, S. (2020). Differentiable augmentation for data-efficient gan training. *ArXiv*, abs/2006.10738.

Appendix A

Code

In this section we show the major s in experiments package, which is the main package for experimental tasks, along with baseline_classifier.py and generate.py s:

- **experiment.py** : contains Experiment abstract class, with abstract method create_train_df() and implemented method run(). The following classes derived from it.
- **baseline.py** : containing BaselineExperiment class. Performs experiments for baselines.
- **augmentation.py** : contains AugmentationExperiment. Performs experiments related with data augmentation via GANs.
- **replacement.py** : contains ReplacementExperiment class. Performs experiments in which real images are replaced by fakes.
- **baseline_classifier.py**: implements *run_classifier* which runs the classifier with given parameters and default data augmentation transforms.
- **generate.py**: generate fakes given the network snapshots. System call to stylegan2-ada generate algorithm with python interpreter defined in a separated conda enviorment for stylegan2-ada.

A.1 experiment.py

```python
import numpy as np
import copy
from abc import ABC, abstractmethod
import os
import pathlib
```

```
from datetime import datetime
import gc

import torchvision
from fastai.callback.tracker import EarlyStoppingCallback, SaveModelCallback

from data.data import read_df
from classifiers.baseline_classifier import run_classifier
from experiments.results import Result, Report
from experiments.utils import AttributeDict

DEFAULT_SETTINGS = AttributeDict({
    "baseline_path": pathlib.Path("/home/alberto/Data/master/TFM/usbrains/data/
        ↪ processed/baseline_sz128"),
    "model_name": "model",
    "log_path": pathlib.Path("/home/alberto/Data/master/TFM/usbrains/experiments/
        ↪ results"),
    "experiment_name": "experiment",
    "report_name": "report",
    "save_best_model": False
})

DEFAULT_TRAIN = AttributeDict({
    "classifier": torchvision.models.resnet18,
    "freeze_epochs": 1,
    "epochs": 20,
    "bs": 64,
    "data_augmentation": True,
    "cbs": [EarlyStoppingCallback(monitor='valid_loss', patience=5),
           SaveModelCallback(fname=DEFAULT_SETTINGS.model_name)],
    "img_size": 128,
    "label_col": 1,
    "all_planes": False,
})


class Experiment(ABC):
    """
    Abstract class to perform experiments with abstract method create_train_df and
        ↪ implemented method run.
    """

    def __init__(self, bl_name, experiment_path, train_params=None, setting_params=
        ↪ None):
        """

        Parameters
        ----------
        bl_name: str
            name of baseline used
        experiment_path: str or Path
            path to folder containing a sub-folder called images with all images used
                ↪ in training
        train_params: AttributeDict
            training parameters. If None, DEFAULT_TRAIN are used. Parameters can be
                ↪ overridden.
        setting_params: AttributeDict
            setting parameters. If None, DEFAULT_SETTINGS are used. Parameters can be
                ↪ overridden.
        """
        assert os.path.exists(experiment_path)
        self.bl_name = bl_name
        self.experiment_path = pathlib.Path(experiment_path)
        self.imgs_path = self.experiment_path / "images"

        self.train_params = copy.deepcopy(DEFAULT_TRAIN)
        self.setting_params = copy.deepcopy(DEFAULT_SETTINGS)

        if setting_params:
            self.setting_params.update(setting_params)
        if train_params:
            self.train_params.update(train_params)
            self.train_params.cbs[1].fname = self.setting_params.model_name

        self.train_params.patience = self.train_params.cbs[0].patience
        self.bl = read_df(bl_name, self.setting_params.baseline_path)
        self.model_name = self.setting_params.model_name
        date = datetime.today().strftime('%d_%m_%Y__%H_%M_%S')
        self.results_path = self.setting_params.log_path / f"{self.setting_params.
            ↪ experiment_name}_{date}"

        self.best_val = [np.inf, 0.0, 0.0, 0.0]
        self.best_learn = None
        self.best_cr = None
        self.report = None

        os.makedirs(self.results_path)

    @abstractmethod
    def create_train_df(self, *args):
        pass

    def run(self, *args, seeds=19):
        """ run method for all experiments. create_train_df must be overridden by
            ↪ all derived classes.
            performs #seeds runs with *args and class defined arguments.
        """

        self.report = Report(name=self.setting_params.report_name, path=self.
            ↪ results_path)
        self.report.write(f"algorithm: {self.__class__.__name__}")
        self.report.write(f"run args: {args}\n")
        self.print_params()

        if not isinstance(seeds, list):
            seeds = [seeds]

        result = Result(name=self.model_name)
        n_experiments = len(seeds)

        for i, seed in enumerate(seeds):
            self.report.write("=" * 50)
            self.report.write(f"Running experiment {i + 1} of {n_experiments} (with
                ↪ seed {seed})")
            self.report.write("=" * 50)
            train_df = self.create_train_df(*args, seed)

            learn, cr = run_classifier(train_df, self.experiment_path, self.
                ↪ train_params.classifier,
                     epochs=self.train_params.epochs,
                     freeze_epochs=self.train_params.freeze_epochs,
                     bs=self.train_params.bs, da=self.train_params.
                         ↪ data_augmentation,
                     cbs=self.train_params.cbs,
                     label_col=self.train_params.label_col,
                     size=self.train_params.img_size,
                     all_planes=self.train_params.all_planes,
                     save_model_path=self.results_path)

            self.report.write("validating best model...")
            validation = learn.validate().items
            self.report.write(f"best model: {validation}")
            if validation[0] < self.best_val[0]:
                self.best_val = validation
                self.best_learn = learn
                self.best_cr = cr
            result.valdidations.append(validation)
            result.crs.append(cr.classification_report(True))
            result.cms.append(cr.confusion_matrix())
            self.report.write("\nconfusion matrix\n")
            self.report.write(str(cr.confusion_matrix()))
            self.report.write("\nclassification report\n")
            self.report.write(cr.classification_report())
            cr.plot_confusion_matrix(title=f"Confusion matrix: run {i + 1} of {
                ↪ n_experiments}")

        self.report.write("=" * 50)
        self.report.write("Metrics average: ")
        experiment_avg = np.mean(np.array(result.valdidations), axis=0)
        experiment_std = np.std(np.array(result.valdidations), axis=0)
        self.report.write(f"accuracy. avg.: {experiment_avg[1]:.5f}, std.: {
            ↪ experiment_std[1]:.5f}")
        self.report.write(f"f1-score macro. avg.: {experiment_avg[2]:.5f}, std.: {
            ↪ experiment_std[2]:.5f}")
        self.report.write(f"roc_auc. avg.: {experiment_avg[3]:.5f}, std.: {
            ↪ experiment_std[3]:.5f}")
        self.report.write(f"validation avg.: {experiment_avg}")
        self.report.write("=" * 50)

        self.report.write(f"Metrics: ")
        self.report.write(result.get_metrics())
        self.report.write("=" * 50)
        self.report.write(f"Stats")
        self.report.write(result.stats())
        self.report.write("=" * 50)

        if self.setting_params.save_best_model:
            self.best_learn.save(self.model_name)
            self.best_learn.export()

        return result

    def print_params(self):
        self.report.write("Training parameters:")
        self.report.write(self.train_params)
        self.report.write("Setting parameters:")
        self.report.write(self.setting_params)
        self.report.write(f"Baseline: {self.bl_name}")
        self.report.write(f"Images path: {self.imgs_path}")
        self.report.write(f"Model name: {self.model_name}")

    def clear(self):
        self.best_learn = None
        self.best_cr = None
        gc.collect()

    def __repr__(self):
        return f"{self.bl.shape}, {self.experiment_path}\n{self.train_params}\n" \
               f"{self.setting_params}"
```

A.2 baseline.py

```
import pathlib

from .experiment import Experiment

DEFAULT_BASELINE = pathlib.Path("/home/alberto/Data/master/TFM/usbrains/data/
    ↪ processed/baseline_sz128")


class ExperimentBaseline(Experiment):
    """
    Class for baseline experiments. create_train_df() is identity, returning the
        ↪ baseline dataframe unmodified.
    """

    def __init__(self, bl_name, experiment_path=DEFAULT_BASELINE, train_params=None,
        ↪  setting_params=None):
        super().__init__(bl_name, experiment_path, train_params, setting_params)

    # dummy seed
    def create_train_df(self, seed=19):
        return self.bl
```

A.3 augmentation.py

```
import pathlib
import pandas as pd

from experiments.experiment import Experiment
from experiments.utils import create_fakes_df, sample_by_condition


class AugmentationExperiment(Experiment):
    """
    Class for augmentation experiments. Overrides create_train_df method. The main
        ↪ purpose is data augmentation via
```

```
fakes. Although it allows also replacement experiments, notice that replacement
    ↪ is done over images
involved in GANs training. For replacement over images no GAN involved use
    ↪ ExperimentReplacement.
"""
def __init__(self, bl_name, experiment_path, train_params=None, setting_params=
    ↪ None):
    super().__init__(bl_name, experiment_path, train_params, setting_params)

def create_train_df(self, trv_real, dbp_real, trv_fakes, dbp_fakes, seed=19):
    """

    Parameters
    ----------
    trv_real: int
        number of trv reals to remove
    dbp_real: int
        number of dbp reals to remove
    trv_fakes: int
        number of trv fakes to be included in training
    dbp_fakes: int
        number of dbp fakes to be included in training
    seed: int

    Returns
    -------
    pandas.Dataframe
        Dataframe containing specified reals and fakes
    """
    df = self.bl.copy()
    df["fake"] = False
    fakes_df = create_fakes_df(self.imgs_path, trv_fakes, dbp_fakes, seed=seed)

    n_trv = df[(df.Brain_plane == "TRV") & ~df.Test].shape[0]
    assert n_trv - trv_real >= 0
    n_dbp = df[(df.Brain_plane == "DBP") & ~df.Test].shape[0]
    assert n_dbp - dbp_real >= 0

    trv_cond = {"column": "Brain_plane", "value": "TRV"}
```

```
        dbp_cond = {"column": "Brain_plane", "value": "DBP"}

        df = sample_by_condition(df, n_trv - trv_real, trv_cond, random_state=seed)
        df = sample_by_condition(df, n_dbp - dbp_real, dbp_cond, random_state=seed)

        return pd.concat([df, fakes_df], ignore_index=True)
```

A.4 replacement.py

```
import pandas as pd

from .experiment import Experiment
from .utils import create_fakes_df, sample_by_condition


class ReplacementExperiment(Experiment):
    """
    Class for replacement experiments. Overrides create_train_df method. The main
        ↪ purpose is to perform experiments
    replacing real images by fakes. All replacements are done over images not
        ↪ involved in GANs training.
    """

    def __init__(self, bl_name, experiment_path, train_params=None, setting_params=
        ↪ None):
        super().__init__(bl_name, experiment_path, train_params, setting_params)

    def create_train_df(self, trv_real_rep, dbp_real_rep, trv_fakes, dbp_fakes,
                        seed=19):
        """

        Parameters
        ----------
        trv_real_rep: int
            number of trv reals to replace
        dbp_real_rep: int
            number of dbp reals to replace
    trv_fakes: int
        number of trv fakes to be included in training
    dbp_fakes: int
        number of dbp fakes to be included in training
    seed: int

    Returns
    -------
    pandas.Dataframe
        Dataframe containing specified reals and fakes
    """
    bl = self.bl.copy()

    if trv_real_rep == dbp_real_rep == trv_fakes == dbp_fakes == 0:
        return bl

    n_trv = bl[(bl.Brain_plane == "TRV") & ~bl.gan & ~bl.Test].shape[0]
    assert n_trv - trv_real_rep >= 0
    n_dbp = bl[(bl.Brain_plane == "DBP") & ~bl.gan & ~bl.Test].shape[0]
    assert n_dbp - dbp_real_rep >= 0

    fakes_df = create_fakes_df(self.imgs_path, trv_fakes, dbp_fakes, seed=seed)
    if not fakes_df.empty:
        fakes_df["gan"] = False

    trv = {"column": "Brain_plane", "value": "TRV"}
    dbp = {"column": "Brain_plane", "value": "DBP"}
    no_gan = {"column": "gan", "value": False}

    bl_df = sample_by_condition(bl, n_trv - trv_real_rep, trv, no_gan,
        ↪ random_state=seed)
    bl_df = sample_by_condition(bl_df, n_dbp - dbp_real_rep, dbp, no_gan,
        ↪ random_state=seed)
    bl_df["fake"] = False

    df = pd.concat([bl_df, fakes_df], ignore_index=True)

    assert df.shape[0] == bl.shape[0] - trv_real_rep - dbp_real_rep + trv_fakes
        ↪ + dbp_fakes, \
        f"{df.shape[0]} != {bl.shape[0] - trv_real_rep - dbp_real_rep + trv_fakes
            ↪  + dbp_fakes}"

    return df
```

A.5 results.py

```
import copy
import pathlib

import numpy

from experiments.utils import AttributeDict



class Result:

    """
    Class for results storage.
    """
    def __init__(self, name=None):
        self.name = name if name else "result"
        self.valdidations = []
        self.crs = []
        self.cms = []

    def get_metrics(self):
        s = AttributeDict({
            "precision": [],
            "recall": [],
            "f1-score": [],
            "support": []
        })
        stats = AttributeDict({v: copy.deepcopy(s) for v in ['DBP', 'TRV', 'macro
            ↪ avg', 'weighted avg']})
        stats["accuracy"] = []
        for r in self.crs:
            for k, v in r.items():
                if isinstance(v, dict):
                    for kk, vv in v.items():
                        stats[k][kk].append(vv)
                else:
                    stats[k].append(v)

        return stats

    def print_stats(self):
        print(self.stats())

    def stats(self):
        stats = self.get_metrics()
        r = f""
        for k, v in stats.items():
            if isinstance(v, dict):
                r += f"{k}"
                r += "\n"
                for kk, vv in v.items():
                    r += f"\t{kk}: {numpy.mean(vv):.4f} (std: {numpy.std(vv):.4f})"
                    r += "\n"
            else:
                r += f"{k}: {numpy.mean(v):.4f} (std: {numpy.std(v):.4f})"
                r += "\n"

        return r


class Report:
    """
    Simple class for reporting. Prints and write messages in given file.
    """
    def __init__(self, name=None, path="./"):
        self.name = name if name else "experiment"
        self.path = pathlib.Path(path)
        self.fname = self.path / f"{self.name}.txt"
        open(self.fname, 'a').close()
    def write(self, msg, print_console=True):
        if print_console:
            print(msg)
        with open(self.fname, "a") as f:
            f.write(str(msg) + "\n")
```

A.6 baseline_classifier.py

```
import torchvision
from fastai.vision.all import *

import sklearn.metrics as skm



class ClassificationReport(ClassificationInterpretation):
    def __init__(self, dl, inputs, preds, targs, decoded, losses):
        super().__init__(dl, inputs, preds, targs, decoded, losses)
        self.vocab = self.dl.vocab
        if is_listy(self.vocab): self.vocab = self.vocab[-1]

    def classification_report(self, output_dict=False):
        d, t = flatten_check(self.decoded, self.targs)
        return skm.classification_report(t, d, labels=list(self.vocab.o2i.values()),
                                         target_names=[str(v) for v in self.vocab],
                                            ↪ output_dict=output_dict)



def run_classifier(train_df: pd.DataFrame, train_path: Union[str, Path], classifier:
    ↪  torchvision.models, epochs,
                 freeze_epochs=1, bs=64,
                 da=True, cbs=None, label_col=1, size=224, all_planes=False,
                    ↪ save_model_path=None):
    """

    Run resnetN classifier with default parameters and data augmentation, given a
        ↪ dataframe and a path containing the
```

```
training images. train_df must contain at least three columns: Image,
    ↪ Brain_plane, Test.

Parameters
----------
train_df: pandas.Dataframe
    train Dataframe with at least columns: Image, Brain_plane, Test
train_path: str or Path
    path where training images are located
classifier: torchvision.models
epochs: int
freeze_epochs: int
bs: int
da: bool
    data augmentation
cbs: fastai.callback.tracker.TrackerCallback
    callbacks
label_col: int
    dataframe col where it is the label. By default 1
size: int
    image size (square)
all_planes: bool
    whether or not include all planes. By default only trv and dbp are included
save_model_path: str or Path

Returns
-------
 Tuple[Learner, ClassificationReport]
"""

if not save_model_path:
    save_model_path = train_path
batch_tfms = None
if da:
    batch_tfms = aug_transforms(size=size) # default data augmentation
dls = ImageDataLoaders.from_df(train_df, train_path / "images", label_col=
    ↪ label_col, valid_col="Test",
                                batch_tfms=batch_tfms, bs=bs)
    metrics = [accuracy, F1Score(average='macro')]
    if all_planes:
        metrics.append(RocAuc())
    else:
        metrics.append(RocAucBinary())

    learn = cnn_learner(dls, classifier, path=save_model_path, metrics=metrics)

    print("training...")
    learn.fine_tune(epochs, freeze_epochs=freeze_epochs, cbs=cbs)
    print("training done.")

    cr = ClassificationReport.from_learner(learn)

    return learn, cr


def run_classifier_random_split(train_df, train_path, classifier, epochs,
    ↪ freeze_epochs=1, bs=64, da=True, cbs=None,
                                label_col=1, size=224):
    batch_tfms = None
    if da:
        batch_tfms = aug_transforms(size=size)
    dls = ImageDataLoaders.from_df(train_df, train_path / "images", label_col=
        ↪ label_col,
                                   batch_tfms=batch_tfms, bs=bs)

    print("training...")
    learn = cnn_learner(dls, classifier, path=train_path, metrics=[accuracy, F1Score
        ↪ (average='macro'), RocAucBinary()])
    learn.fine_tune(epochs, freeze_epochs=freeze_epochs, cbs=cbs)
    cr = ClassificationReport.from_learner(learn)

    return learn, dls, cr
```

A.7 generate.py

```python
import os
from data.utils import cd

PYTHON = "/home/alberto/anaconda3/envs/sg2/bin/python"
SG_ADA_PATH = "/home/alberto/Data/github/stylegan2-ada"

def generate_images(network: str, outdir: str, trunc: float, seeds: str, prefix=None
    ↪ ):

    """
    Generate images from given GAN network, trained with styleGAN2-ada

    Parameters
    ----------
    network: str
        network snapshot used to generate images
    outdir: str
        folder where images are generated
    trunc: float
        truncation tpsi
    seeds: str
        seeds string, eg, 1-100
    prefix:str
        prefix to add to images name. By default they are named seed_n.png

    """
    with cd(SG_ADA_PATH):
        desc = f"{PYTHON} generate.py --network {network} --outdir {outdir} --seeds
            ↪ {seeds}"
        if prefix:
            desc += f" --prefix {prefix}"
        if trunc:
            desc += f" --trunc {trunc}"
        os.system(desc)
```