Generative Adversarial Networks for creating synthetic images of COVID-19 positive patients

Final project report for the course of Medical Imaging Diagnostic

Matteo Guglielmi (232088)

University of Trento
ULTRa Lab
matteo.guglielmi@studenti.unitn.it

Abstract

This work focuses on discussing the implementation of custom GAN models for the generation of synthetic images of COVID-19 positive patients affected by different severity conditions. The training dataset consists in 11 SARS-COV-2 tested positive patients from the San Matteo Hospital, Pavia, Italy. For each of patient, a variable number of medical exams have been carried out and the majority of those presents 14 videos imaging different areas around the body accordingly to the standardized approach to be adopted when performing COVID-19 related measurements.
In this work, two GAN architectures have been proposed and different loss functions are available(i.e. RSGAN, RaSGAN, RaLSGAN, RaHingeGAN). The algorithm has been intended to be plug-and-play with the dataset being the only necessary component. Once everything is setup, from CLI it is possible to choose different combinations of model, loss function and hyper-parameters which impact the GAN model behaviour and performances. Finally, an evaluation phase is performed exploiting some known similarity indexes (e.g. MSSSIM, RASE, SCC, PSNR) with promising outcoming results (see Section 5 for more details).
The source code and obtained results are available at `https://github.com/MatteoGuglielmi-tech/CovidGAN`

Index Terms — COVID-19, Generative Adversarial Networks, similarity score, normalization and standardization.

1. Introduction

The idea of this project took place from the nature of medical data themselves. In fact, it is very common to either have a *small number of available samples and/or a very unbalanced dataset*. These limitations are mainly due to:

- an expensive and time consuming data collection process;
- data is very sensitive and it is not always possible to share it;
- it's impossible to control the amount of collected data per label (e.g. medics cannot decide how many score zero COVID-19 positive patients to record data from).

These aspects may lead to a problematic situation when trying to develop an automatic deep learning based algorithm for the detection or diagnosis of a certain disease. In fact, dealing with **unbalanced data isn't an ideal condition for classification models to train at best** as they would tend to overfit the majority class and it will be very hard to generalize the results on the minority classes.
In light of this, generating synthetic data may be beneficial in overcoming the aforementioned limitations since it is possible to:

- generate as many samples as needed to balance the dataset;
- control the amount of generated samples per label;
- generate samples with a specific distribution.

Another possible benefit generated samples can bring is to introduce some "benign noise" in the dataset. This may be useful to obtain a model that is more robust to noise and that can generalize better.

1.1. COVID-19 related research

With the advent of the COVID-19 pandemic, the scientific community has worked hard to develop automatic algorithms to ease medical diagnosis and to address the health condition of a positive patient. In particular, the research has been focused on:

- the definition of imaging protocols and scoring systems to achieve standardized lung ultrasounds (LUS) images and to assess the severity of the disease;
- the development of automatic algorithms to detect COVID-19 positive patients with a corresponding score based on different characteristic patterns;
- studies about the specificity of COVID-19 patterns with respect to other diseases (e.g. pneumonia);
- the effects of COVID-19 in post-COVID-19 patients.

In this context, having a tool that allows to generate synthetic data could help especially in the first stages of an emergency scenario when the amount of available data is very limited.

1.2. Proposed model

The proposed model has been structured to give the possibility to the final user to decide and experiment different combinations of hyper-parameters. In particular, the user can choose, among all the others , the following parameters:

- the model to use. This impacts the loss function definition;
- the dimension of the latent space;
- the size of the input images the model will be trained with and, as a consequence, the size of the output images;

Generator
$z \in \mathbb{R}^{128} \sim N(0, I)$
ConvTranspose2d 4x4, stride 1, pad 0, no bias, 128->1024
BN and ReLU
ConvTranspose2d 4x4, stride 2, pad 1, no bias, 1024->512
BN and ReLU
ConvTranspose2d 4x4, stride 2, pad 1, no bias, 512->256
BN and ReLU
ConvTranspose2d 4x4, stride 2, pad 1, no bias, 256->128
BN and ReLU
ConvTranspose2d 4x4, stride 2, pad 1, no bias, 128->64
BN and ReLU
ConvTranspose2d 4x4, stride 2, pad 1, no bias, 64->32
BN and ReLU
ConvTranspose2d 4x4, stride 2, pad 1, no bias, 64->3
Tanh

Discriminator (PACGAN2 [Lin et al., 2017])
$x_1 \in \mathbb{R}^{3x256x256}, x_2 \in \mathbb{R}^{3x256x256}$
Concatenate $[x_1, x_2] \in \mathbb{R}^{6x256x256}$
Conv2d 4x4, stride 2, pad 1, no bias, 6->32
LeakyReLU 0.2
Conv2d 4x4, stride 2, pad 1, no bias, 32->64
LeakyReLU 0.2
Conv2d 4x4, stride 2, pad 1, no bias, 64->128
BN and LeakyReLU 0.2
Conv2d 4x4, stride 2, pad 1, no bias, 128->256
BN and LeakyReLU 0.2
Conv2d 4x4, stride 2, pad 1, no bias, 256->512
BN and LeakyReLU 0.2
Conv2d 4x4, stride 2, pad 1, no bias, 512->1024
BN and LeakyReLU 0.2
Conv2d 4x4, stride 2, pad 1, no bias, 1024->1

Figure **(1)** – GENERATOR AND DISCRIMINATOR ARCHITECTURE USED IN THE EXPERIMENTS.

- the number of synthetic samples to generate;
- the number of iterations to perform:
- the batch size;
- the learning rate for both the optimizers:
- either to treat the input data as RGB or greyscale images;

The deployed loss functions are inspired by [1] where a discussion about modifying the standard GAN loss function to improve the performance of the model is presented. In particular, the author proposes to add a *relativistic twist* to improve performances. Briefly, this tweak consists in not just training the generator to increase the probability that fake data are real but also to simultaneously decrease the probability that the original images are real. This approach can be applied to any possible GAN architecture and, although maybe counter-intuitive, leads to better results.

The `tested model` is heavily inspired by Appendix D.3 in [1] (Figure 1) with the addition of small GAN tricks such as the addition of a Dropout after each block of the Generator. The mentioned GAN tricks are discussed in [2].

2. Task formalization

As briefly discussed in `Section 1`, the aim of this work is to devise a Generative Adversarial Network that produces synthetic lung ultrasound images of comparable quality with the real ones.
Moreover, as additional goal, the *source algorithm* should be as much **intuitive and flexible as possible**. To achieve this, all the necessary hyper-parameters are passed through CLI and a parser takes care to interpret them. This approach has been chosen to ensure re-usability and results reproducibility (using the same seed should sample the noise vector, given as input to the generator, from the same pseudo-random distribution leading to similar results).
In addition to that, the training process is monitored through logs and a user-friendly interface [3] to ease loss trends analysis.
Eventually, the results are saved in a folder named accordingly to the training hyper-parameters used. This allows to:

- compare different training sessions;
- compare the final results with different combinations model-loss function;
- have a better understanding of possible correlations between the hyper-parameters and the results.
- identify possible notorious problems (e.g. mode collapse or vanishing gradients) and devise possible solutions to solve them.

The following sections will discuss the implementation details and the achieved results.

3. Data description and analysis

3.1. Dataset composition

The data used in this project are the result of the work from the San Matteo hospital doctors.
The dataset is structured as follows :

- as first level, 11 folders are present, each one containing the data of a specific patient with a relative code to preserve privacy;
- as second level, each folder contains $\mathcal{N}$ subfolders where $\mathcal{N}$ is the number of exams the patient has undergone;
- as third level, each subfolder contains up to 14 MATLAB files where each of them corresponds to a single video for a specific area (each single frame is expressed as a matrix of intensity values).

It is worth mentioning that, as the majority of medical datasets, the systems used to acquire each single video vary across the different exams both in terms of scanner and probe devices. Anyhow, all footages have been recorder meeting the standardized LUS imaging protocols and with a single kind of sensors: the **convex probe**.
Additional criticalities have been observed during the dataset analysis:

- all convex regions aren't centered in the image frame;
- convex regions throughout the different videos are not equivalent in terms of size;
- the pixel intensity across videos varies a lot across different videos;
- the images are acquired as RGB images, but the three channels are the same.

3.2. Preprocessing

The first necessary step was to pre-process the dataset to make it as suitable as possible for the training phase.

The preparation phase consists in:

- read each MATLAB file and create the corresponding frames;
- organize in separate folders the video frames according to their severity score;
- reshape to an equal size each frame. In particular, the biggest dimensions, both in width and height, have been identified across the different images and used as samples target dimensions;
- apply a centering procedure to impose frames uniformity. In light of Section 3.2, this procedure doesn't enforce a perfect frame alignment but it may help in learning a more consistent final product.

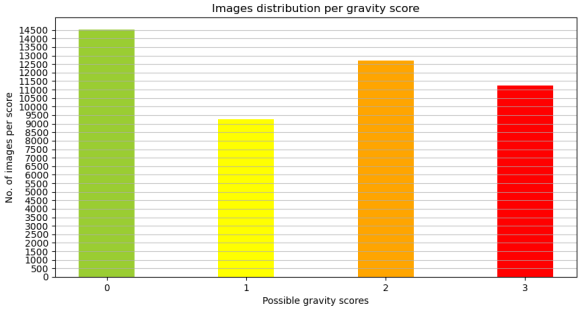


Figure **(2)** – HISTOGRAM REPRESENTING THE SCORE PRESENCE IN THE DATASET.

3.2.1. Score distribution after pre-processing

After the frames arrangement, a score distribution analysis has been conducted. From this, the results reported in Figure 2 have been obtained which highlight the unbalanced genesis of the dataset.

In light of the just obtained results, it is intuitive as a classifier network would encounter a very hard time correctly predicting the minority class (1 in this case) and it would very likely overfit the majority class.

4. Model

As anticipated in Section 1.2, the final user has the chance to choose between two different generator and discriminator backbones independently from the selected loss functions. In greater details, the default model presents a convolutional oriented structure with batch normalization layers and RELU activation functions in between each convolutional block whilst the second model, which requires a lot of VRAM and for computational limitations has not been tested but simply proposed, involves some common GAN improvements such as SELU activations, UpSample layers and spectral normalization.

The training is carried out on the different scores folder and, eventually, four models have been obtained.

4.1. Optimizers

In this work, two different Adam optimizers [1] have been used to update the weights of generator and discriminator networks. This particular type of optimizer has been chosen for its abilities to converge quickly and because it's been demonstrated in literature it leads to better results.
In addition to that, to have a greater ability to control the learning rates trends, an exponential scheduler [2] has been used to strengthen or weaken either the generator or discriminator if one gets too successful over the other.

4.2. Data preparation

At this point, the pre-processed training data need to be prepared to be fed to the network. In particular, the preparation consists in:

- resize the input image to the target dimension (i.e. the dimensions we want the synthetic images to be);
- squash the input image pixel values in the $[0, 1]$ range;
- whether to treat the input image as RGB or greyscale;
- whether standardize the input image;

As common practice, to alleviate computational loads the input data has been divided into min-batches.

4.3. Experimental settings

Several experiments have been carried out trying to find the optimal set of hyper-parameters to guarantee a satisfactory quality level in synthetic images.
Initially, the different frames were treated as RGB images and each input image normalized and standardized (i.e. pixel values in range $[-1, 1]$ and $\mu = 0, \sigma = 1$). Under these conditions, the model seems to understand the overall characteristic patterns of a specific score but the final product appears to be blurred (refers to Figure 3a). Subsequently, an additional trial was to use tailored channel statistics (i.e. mean and standard deviation computed channel-wise) to be used in the standardization process. Unfortunately, in the latter case the training process failed. At this point, a further experiment consisted in removing both normalization and standardization. Results were comparable to Figure 3b but the training process was much more slower due to the more deeper pixel intensity scale. This trial was useful to understand that the major problem in the first runs was treating the input data as RGB images.
Finally, the last experiment was about treating the input data as greyscale images, apply normalization and giving up on standardization. The results (Figure 3b) turned out to be decent and the training process was faster, more predictable and controllable.

More tweaks adopted to avoid possible well-known GAN problems such as overfitting and mode collapse are:

- double-sided **label smoothing**:
  - instead of using 1 as positive labels for real samples, a random value in the range $[0.7, 1.2]$ is assigned;
  - instead of using 0 as negative labels, a random value in the range $[0, 0.3]$ is assigned;

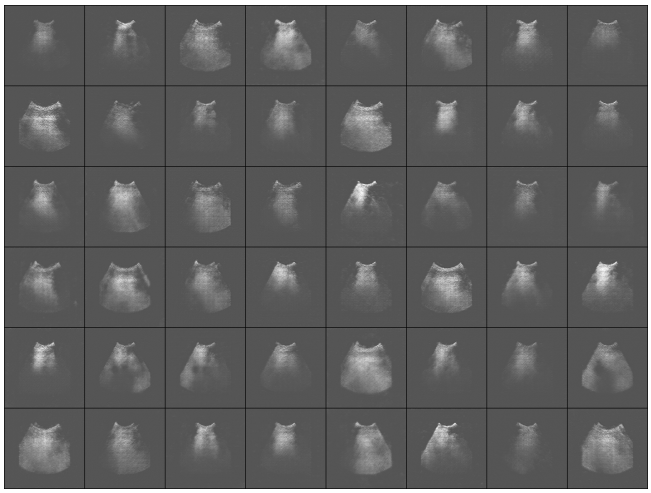
  This trick is useful to avoid the discriminator to be too confident about its predictions;
- **instance noise**: randomly flip labels, with a certain probability p, when computing the discriminator loss. This technique is useful to avoid the discriminator to overfit (i.e. getting too strong) especially in the first training stages when the generator output is essentially noise.

4.3.1. Further trials

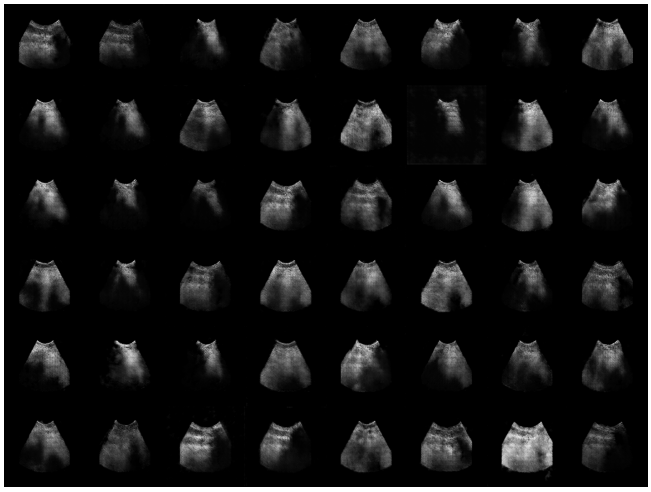
Other tweaks have been tried to improve the model performance but they did not lead to better results. In particular:

- adding white noise to the input of the discriminator;
- completely flip batch labels when training discriminator;
- add noise layers to the generator structure.

[1] https://pytorch.org/docs/stable/generated/torch.optim.Adam.html
[2] https://pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.ExponentialLR.html



(a) RGB SYNTHETIC IMAGES FOR SCORE 0.



(b) GENERATED BATCH OF GREYSCALE SYNTHETIC IMAGES FOR SCORE 0.

5. Evaluation

This section will showcase results from different runs. In particular, the following hyper-parameters have been used:

- `RaSGAN` [3]: adopted model;
- `latent space dimension`: set to 128;
- `batch size`: set to 48 examples randomly sampled at each iteration (without repetition);
- `image size`: set to 256×256 pixels. This format has been chosen since the deep classifiers majority are trained on images of similar size (e.g. ResNet accepts as input 224× 224 images);
- `channels`: set to 1 to work with greylevel images;
- `iterations`: set to 4 000 iterations;
- `no normalization and standardization`: the images are not normalized nor standardized;
- `learning rate`:
  - `generator`: set to 0.000 4 and left it as is;
  - `discriminator`: set to 0.000 2 with a decay gamma coefficient of 0.000 5 (i.e. after 1 550 iterations the learning rate has decreased to $4.594\,6 \cdot 10^{-05}$);
- `generator iterations`: to one discriminator update corresponds 3 generator iterations;
- for reproducibility purposes, the random seed has been kept equal to 999.

5.1. Evaluation metric

The generated images realness has been visually assessed while a more scrupulous analysis has been performed to address whether the trained model suffers from mode collapse. In particular, the following metrics are available:

- `MSSSIM` (**M**ultiscale **S**tructural **SI**milarity **I**ndex) [4]: it measures the similarity between two images. It is based on the computation of the `SSIM` [5] index on different scales which allows for a better incorporation of image details at different scales. The latter is based on the computation of three different values: `luminance`, `contrast` and `structure`.
  The final value is in the range $[0, 1]$ where 1 means that the two images are identical;
- `RMSE` (**R**oot **M**ean **S**quare **E**rror): it measures the difference between two images. It is based on the computation of the mean of the squared differences between the two images.
  The final value is in the range $[0, 1]$ where 0 means that the two images are identical;
- `RASE` (**R**elative **A**bsolute **S**pectral **E**rror) [6] : it quantifies the difference between two images. It is based on the computation of the mean of the absolute differences between the two images.
  The final value is in the range $[0, 1]$ where 0 means that the two images are identical;

[3] Side note: RaSGAN has been used since in [1] is quoted, as well as RaLSGAN, that it tends to learn better and in shorter times with respect to standard GANs

Table **(1)** – MSSSIM SCORE AMONG THE 1ST SCORE 0 SYNTHETIC IMAGE AND THE FIRST 10S.

Index	MSSSIM
1	1.0
2	0.72
3	0.7155
4	0.997
5	0.8657
6	0.4697
7	0.4886
8	0.4632
9	0.6306
10	0.5216
Avg. Match	0.6524

- `SAM` (**S**pectral **A**ngle **M**apper) [7]: it is based on the computation of the angle between the two images in the spectral domain.
  The final value is in the range $[0, 1]$ where 0 means that the two images are identical;
- `PSNR` (**P**eak **S**ignal to **N**oise **R**atio): it is a difference metric between two images. It is based on the computation of the ratio between the maximum possible power of a signal and the power of the noise that affects the fidelity of its representation.
  The final value is in the range $[0, 1]$ where 1 means that the two images are identical;
- `PSNR-B` (**P**eak **S**ignal to **N**oise **R**atio - **B**lock-sensitive) [8]: it essentially consists in a redefinition of the PSNR index considering a blocking factor.
  The final value is in the range $[0, 1]$ where 1 means that the two images are identical;

Due to the very high time requirements, only for score 0 all similarity scores have been computed among all the generated and real images (this computation required more than a day). The complete log shows a perfect match just in case the comparison operator is computed on the same image and in all other cases, despite some frame pairs are very close to a complete correspondence, the similarity scores highlight a mismatch. This likeness of samples can be justified considering that two consecutive frames are very likely to resemble each other except when abrupt changes take place (that are very rare in scenarios like this one).
A further verification to address whether the model has experienced mode collapse during training is the computation of such similarity scores among generated images. In particular, the `MSSSIM` score has been exploited. A showcasing subset of obtained results from the comparison between the 1st synthetic image and all the others is reported in Table 1 which highlights an average matching coefficient of 0.652 4[4].

The same procedure has been applied for all the other scores. Despite this is not an absolute verification that the model has not experienced either mode collapse, vanishing/exploding gradients or other issues, it is a good indicator that the model has been trained properly.

[4]The average has been computed excluding the perfect match

5.2. Results

Important note is that the **reported results have been obtained with the sames set hyper-parameters for all scores**. This might not be the best approach since the hyper-parameters should be tuned for each study case. However, with different trials has been verified that trying to optimally tune each parameter specifically for each score doesn't lead to significant improvements as well as being a very expensive procedure.

In Figure 4, a test batch per score, obtained at the final iteration of the training procedure, is reported. It is possible to notice that the model has been able to generate images with good quality and details.

5.3. Error and possible improvements analysis

In this section, errors and possible further developments will be analysed.

5.3.1. Error analysis

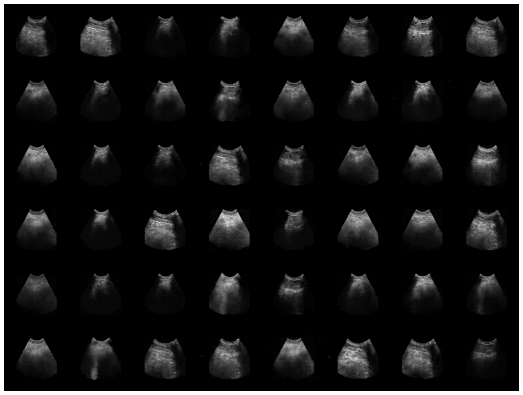
Although precisely identifying the source of possible limitations of a Generative Adversarial Network is not a trivial task, the following points represent the most likely sources of error:

- as already mentioned, the dataset is highly non-normalized and this may have influenced the training procedure. In fact, dealing with such great variable dataset, in terms of pixel values, may discourage the model in being confident on associating a specific pattern to a certain score leading to learn a very generic pattern representation.
  Unfortunately, a simple standardization is not sufficient to solve this issue since data are not only non-normalized but also highly non-linear. Moreover, a standardization procedure would lead to images falling in an unfavourable value range since it's been demonstrated that generative models learn better when the value range of the input image is $[-1, 1]$;
- in addition to the previous point, it is necessary to bare in mind that there is no perfect alignment among the different convex regions and I consider it as an auxiliary source of noise in the training procedure;
- if observed closely, it is possible to identify a regular Gaussian pattern within the generated images. This periodic texture are residuals of the Gaussian vector noise used to generate the artificial samples. This kind of artifact may be caused by the limited number of training iterations the model has been trained for;
- during the experiment, sub-optimal hyper-parameters have been used. In particular, the learning rates and the batch size, the most impactful parameters in the training procedure, have been chosen based on the results obtained from the training of score 0 and then used for all the other scores.

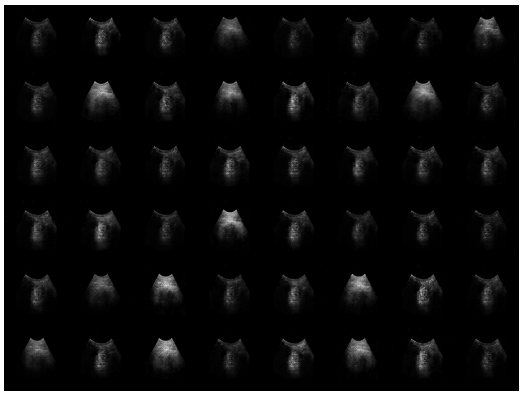
5.3.2. Possible improvements

The following points represent possible improvements that can be applied to the proposed model:

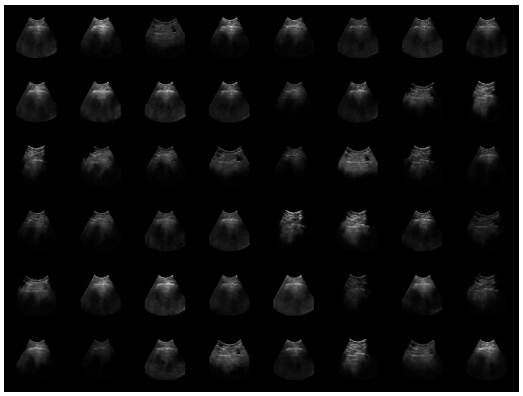
- the most straightforward improvement would be to try training the model for a greater number of epochs. This



(a) GENERATED BATCH OF SYNTHETIC IMAGES FOR SCORE 0.



(b) GENERATED BATCH OF SYNTHETIC IMAGES FOR SCORE 1.



(c) GENERATED BATCH OF SYNTHETIC IMAGES FOR SCORE 2.

(d) GENERATED BATCH OF SYNTHETIC IMAGES FOR SCORE 3.

Figure **(4)** – GENERATED BATCHES OF SYNTHETIC IMAGES FOR EACH SCORE.

might allow the model to learn more complex patterns and to better fit the data (being careful about overfitting). To achieve this, with great probability, it will be necessary to find the right trade-off between `itersD` (i.e. number of discriminator updates in a single training iteration) and `iterG`. If an optimal balance between these two parameters is found, the training would improve significantly;

- consider to introduce more sophisticated mechanisms, such as spectral normalization [9], SELU activation function [10] and, eventually, gradient penalty, to regularize and promote a smoother training process. These aspects have been deployed in the "improved" version of both generator and discriminator network;
- further tune learning rates and scheduler to achieve better convergence trends;
- further address the obtained results: compute the FID score [11] to have a more reliable metric to compare the different models (although even in this case overfitting cannot be detected). To do so, a fine-tuning of InceptionV3 network [12] on a LUS dataset would be necessary;
- introduce data augmentation in the training procedure to increase the variability of the dataset. This would be helpful to avoid overfitting;

5.4. Experimental settings

The results have been obtained using the following tools:

- Python 3.11.3;
- PyTorch 2.0.1;
- Torchvision 0.15.2;
- CUDA toolkit 12.1;
- Conda 23.3.1;
- NVIDIA GeForce RTX 3060, 12GB of VRAM;

6. Conclusion

To summarise, in this work I've proposed an highly flexible method for generating synthetic medical data.

As final remarks, I would like to point out that the results obtained are not intended as optimal, but they might be an interesting starting point for further research and they may lead to performance enhancement when evaluating classifier networks. In particular, for future improvements I would focus on dealing with the irregularity of the training procedure and normalising the data in a more efficient way.

7. References

[1] A. Jolicoeur-Martineau, "The relativistic discriminator: a key element missing from standard gan," https://arxiv.org/pdf/1807.00734.pdf, September 2018.

[2] S. Chintala, E. Denton, M. Arjovsky, and M. Mathieu. How to train a gan? tips and tricks to make gans work. [Online]. Available: https://github.com/soumith/ganhacks

[3] PyTorch, "Torch.utils.tensorboard," https://pytorch.org/docs/stable/tensorboard.html, 2023, accessed: 2023-07-20.

[4] Z. Wang, E. P. Simoncelli, and A. C. Bovik, "Multiscale structural similarity for image quality assessment," *Proceedings of the 37th Asilomar Conference on Signals, Systems and Computers*, pp. 1398–1402, November 2003.

[5] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.

[6] M. Gonzalez-Audicana, J. Saleta, R. Catalan, and R. Garcia, "Fusion of multispectral and panchromatic images using improved ihs and pca mergers based on wavelet decomposition," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 42, no. 6, pp. 1291–1299, 2004.

[7] B. J. Yuhas Roberta H., Goetz Alexander F. H., "Discrimination among semi-arid landscape endmembers using the spectral angle mapper (sam) algorithm," *JPL, Summaries of the 3rd Annual JPL Airborne Geoscience Workshop*, vol. 1:AVIRIS, June 1992.

[8] C. Yim and A. C. Bovik, "Quality assessment of deblocked images," *IEEE Transactions on Image Processing*, vol. 20, no. 1, pp. 88–98, 2011.

[9] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida, "Spectral normalization for generative adversarial networks," *arXiv preprint arXiv:1802.05957*, February 2018. [Online]. Available: https://arxiv.org/pdf/1802.05957.pdf

[10] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter, "Self-normalizing neural networks," *arXiv preprint arXiv:1706.02515*, September 2017. [Online]. Available: https://arxiv.org/pdf/1706.02515v5.pdf

[11] Y. Yu, W. Zhang, and Y. Deng, "Frechet inception distance (fid) for evaluating gans," 09 2021.

[12] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," *arXiv preprint arXiv:1512.00567v3*, December 2015. [Online]. Available: https://arxiv.org/pdf/1512.00567.pdf