Capstone Project Report
Machine Learning Engineer Nanodegree
Udacity

Raul Pino
October, 2022

Ensemble of Generative Adversarial Networks as a Data Augmentation Technique for Alzheimer research

I. Definition

I.I. Project Overview

Data scarcity is a regular problem in research, and in medicine it's especially difficult to find datasets publicly available. The main reason is its rarity, by definition images of anomalies are scattered or not common, and also there are a lot of legal issues that prevent sharing personal information about patients to the rest of the world (Hello Future [1]).

The area I'm interested in is Alzheimer and Dementia research through Magnetic Resonance Imaging (MRI) on brains, where there have been some interesting initiatives using Deep Neural Networks (DNN) for classification (NVIDIA [10] [11]) and Generative Adversarial Networks (GAN) for data augmentation (NVIDIA [12], Filippos Konidaris et al. [7]).

From a high level perspective, the main goal of this project is to learn about GAN Ensembles [16], while putting into practice most of the knowledge I've learned during Deep Learning Nanodegree and Machine Learning Engineer Nanodegree. The idea is to implement multiple deep neural networks ensembling techniques to train deep convolutional generators, capable of creating MRIs of demented brains, those being the most scarce.

I.II. Problem Statement

The problem is based on the Kaggle challenge"I'm Something of a Painter Myself" [5], where the idea is to train GANs, and compare it with the real images to verify how accurate and performant models are. In this capstone project, I trained a Deep Convolutional Generative Adversarial Network (Alec Radford et al. [15]), for generating MRIs of brains with Alzheimer and/or dementia, and calling that model CM (CM: Control Model). Also, I trained three different models, as results of ensembles of GANs, using three different techniques: Standard Ensemble of GANs, Self-ensemble of GANs, and Cascade of GANs (Yaxing Wang et al. [16]); using the same MRI dataset, and calling those models EM1, EM2, and EM3 (EM: Ensemble Model).

The idea is to measure how performant are all these models by comparing real images with generated images, calculating how different those are using Fréchet Inception Distance or FID (Martin Heusel [8]) as the standard for evaluating GANs.

I.III. Metrics

Although we trained multiple GANs, we used the same network architecture for all GANs, where we controlled the performance of these models depending on the generator loss and discriminator loss, to decide in which epoch to stop the training phase. These losses were calculated using a Sigmoid layer and the Binary Cross Entropy between the target and the input probabilities, in one single class (BCEWithLogitsLoss [14]).

In terms of performance evaluation, and comparison between the four models we trained, we used Fréchet Inception Distance (FID) computed for each model in the benchmarking process, to calculate the difference between generated images and real images. In FID, the Inception network is used to extract features from an intermediate layer. Then the process models the data distribution for these features using a multivariate Gaussian distribution with mean μ and covariance Σ. The FID between the real images r and generated images g is computed as:

$FID = ||\mu r - \mu g||\ 2 + Tr(\Sigma r + \Sigma g - 2(\Sigma r \Sigma g)1/2)$

Where Tr sums up all the diagonal elements. FID is calculated by computing the Fréchet distance between two Gaussians fitted to feature representations of the Inception network. In short, as smaller the distance, the similar are the analysed images [8].

II. Analysis

II.I. Data Exploration

The chosen dataset used for training is “Alzheimer's Dataset (4 class of Images) - Images of MRI Segementation” [4]. The data is hand collected from various websites with each and every label verified. Consists of MRIs, having four classes of images, both in training as well as a testing set:

1. Mild Demented
2. Moderate Demented
3. Non Demented
4. Very Mild Demented

The dataset has around 51Mb in images. In total 6400 MRIs, 1279 in the test set, and 5121 in the train set.

Each image is in grayscale, with size 176x208:

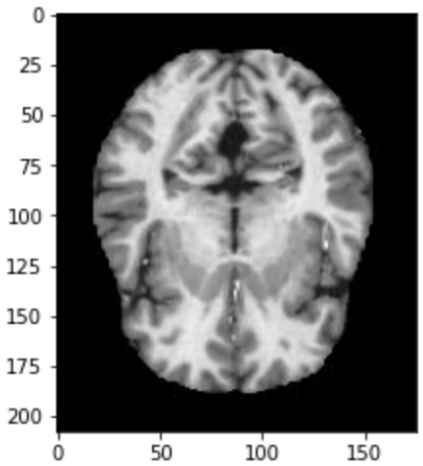

Figure 1: MRI sample of a non-demented brain.

This is a MRI of a non demented brain (Figure 1), we see that it's more or less centered, borders of gray matter are kind of cropped, there is a good separationg between gray matter and non gray matter (black background), there is no bluriness or gradual transition between brain and background.

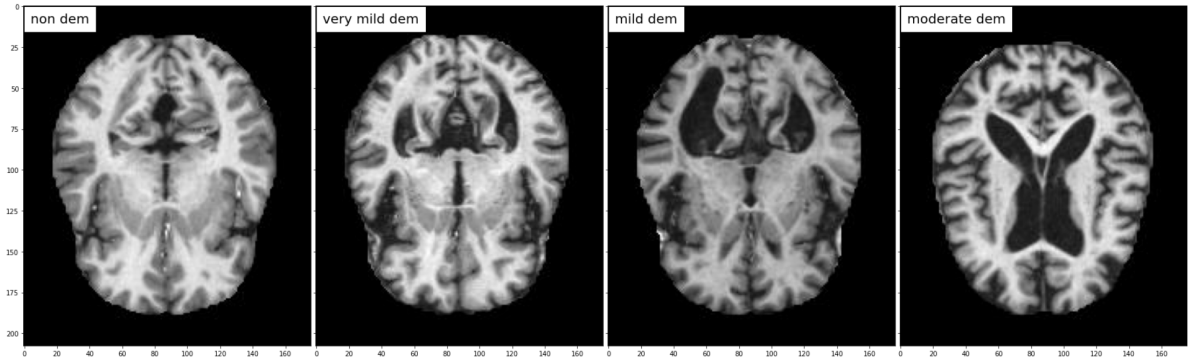

Figure 2: MRI samples of brains in all categories.

We can see that the rest of the image classes (Figure 2) have the same resolution as the first image, 176x208. And we can see some of the same characteristics we saw in the first one. Good separation between brain and background, and the brain is centred in the image. We also see something interesting, differences at naked eye between each class, and of course, a lot of differences between a non demented brain and a moderate demented brain.

II.II. Exploratory Visualisation

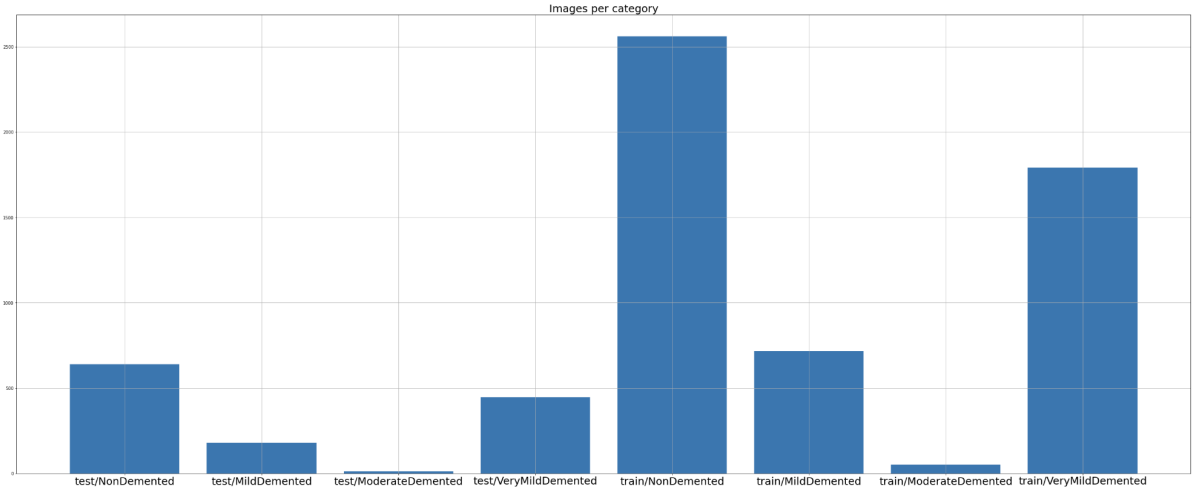

Figure 3: Distribution of categories in train and test sets.

In this chart (Figure 3) we can see the distribution of images per category, the idea is to have a better sense of category distribution in each dataset.

* In test set:
  * NonDemented: 640
  * VeryMildDemented: 448
  * MildDemented: 179
  * ModerateDemented: 12

* In train set:
    * NonDemented: 2560
    * VeryMildDemented: 1792
    * MildDemented: 717
    * ModerateDemented: 52

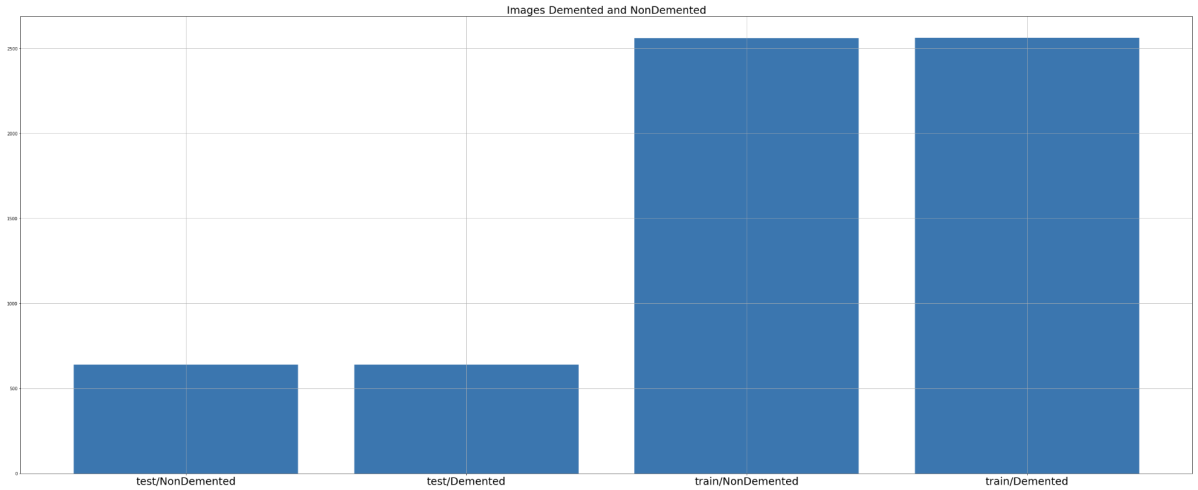

Figure 4: Distribution of demented and non-demented categories in train and test sets.

We can see the distribution between demented and non-demented (Figure 4). Because the goal is to generate MRIs of demented brains ti's important to know the distribution of categories in train and test datasets.

Defining Demented as any image labeled: VeryMildDemented, MildDemented, ModerateDemented, images in each of these categories are:

* NonDemented: 3200, Demented: 3200.
* With something like a disparity between Demented and NonDemented inside train and test sets:
    * test: NonDemented: 640, Demented: 639.
    * train: NonDemented: 2560, Demented: 2561.

In test set, one more NonDemented image than a Demented image. And in train set, one more Demented than NonDemented. Maybe some coinsidence there but balanced in terms of demented and non-demented. Also, it is important to notice that definitely the data is not balanced in terms of categories, in the train and test sets there is a big difference between the MorerateDemented category and the rest, very few MorerateDemented images.

II.III. Algorithms and Techniques

GANs Architecture

As mentioned before, the solution consisted of training 4 models, one for control, and one per each type of GAN ensemble. The basic idea of a GAN ensemble is that a group or GANs might capture many axis of variations that one individual model wouldn't (Ian Goodfellow at Lex Fridman Podcast [2]). In particular, I used the three GAN ensemble designs described by Yaxing Wang et al. [16]: Standard Ensemble of GANs (eGANs), Self-ensemble of GANs (seGANs), and Cascade of GANs (cGANs).

In order to train 4 different models, first we needed a base architecture, accompanied with different training techniques. This base was an implementation of DCGAN architecture following the original paper by Alec Radford et al. [15], in page 3, we can find some useful guidelines for defining it:

Architecture guidelines for stable Deep Convolutional GANs:

- *Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).*
- *Use batchnorm in both the generator and the discriminator.*
- *Remove fully connected hidden layers for deeper architectures.*
- *Use ReLU activation in generator for all layers except for the output, which uses Tanh.*
- *Use LeakyReLU activation in the discriminator for all layers.*

From a high level perspective, the architecture would look something like this for all models:

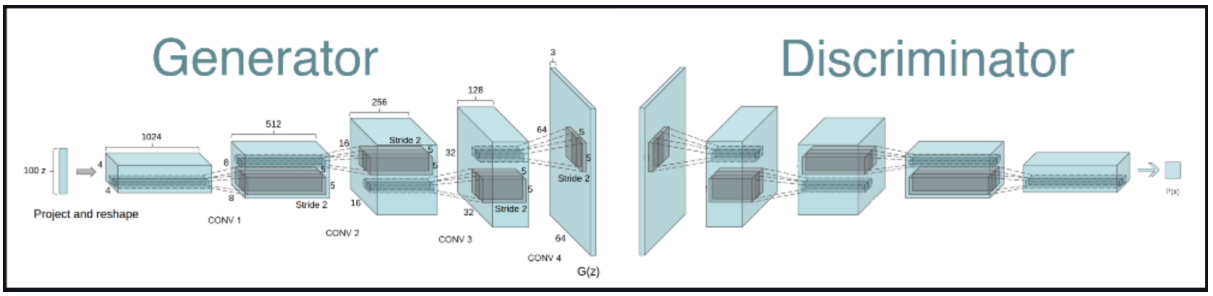

Figure 5: Illustration of DCGAN implementation [6].

What was different was the technique to train each of those four models. The control model was trained once, and used a fixed epoch when to stop. The ensembles were trained depending on the type established in the paper referenced before.

This capstone project was built using two main frameworks/packages in python: pytorch [13] as the machine learning framework, and pytorch-fid [8] as the tool for calculating performance metrics when comparing each trained model.

The DCGAN architecture implemented in this capstone project is based on what I delivered for the "Project: Generate Faces", the final project of "Generative Adversarial Networks" chapter in the "Udacity Deep Learning Nanodegree".

Ensemble techniques

Training and generation processes was implemented following the three techniques detailed by Yaxing Wang et al. [16]:

- ***Standard Ensemble of GANs (eGANs):*** *We first consider a straightforward extension of the usage of ensembles to GANs. This is similar to ensembles used for discriminative CNNs which have shown to result in significant performance gains. Instead of training a single GAN model on the data, one trains a set of GAN models from scratch from a random initialization of the parameters. When generating data one randomly chooses one of the GAN models and then generates the data according to that model.*
- ***Self-ensemble of GANs (seGANs):*** *Other than discriminative networks which minimize an objective function, in a GAN the min/max game results in a continuing shifting of the generative and discriminative network. An seGAN exploits this fact by combining models which are based on the same initialization of the parameters but only differ in the number of*

training iterations. This would have the advantage over eGANs that it is not necessary to train each GAN in the ensemble from scratch. As a consequence it is much faster to train seGANs than eGANs.

- ***Cascade of GANs (cGANs):*** *The cGANs is designed to address the problem described in observation 2; part of the data distribution might be ignored by the GAN. The cGANs framework as illustrated in Figure 2 is designed to train GANs to effectively push the generator to capture the whole distribution of the data instead of focusing on the main mode of the density distribution. It consists of multiple GANs and gates. Each of the GAN trains a generator to capture the current input data distribution which was badly modeled by previous GANs. To select the data which is re-directed to the next GAN we use the fact that for badly modeled data x, the discriminator value D(x) is expected to be high. When D(x) is high this means that the discriminator is confident this is real data, which most probably is caused by the fact that there are few generated examples G(z) nearby.*

In short, for each of these techniques, an ensemble was trained following the recipe, and then when images need to be generated, for each new image one of the GANs in the ensemble is chosen to execute this phase.

II.IV. Benchmark

The hypothesis was that ensemble models perform better than one single model, because they capture many axes of variations that one individual model wouldn't [2]. So for a benchmark model, in order to measure the performance of all trained GANs, I used Fréchet Inception Distance (FID) [8], comparing the control model (which is a single GAN trained until a certain epoch we know that using DCGAN implementation is going to learn to generate images from the training set fairly good), with the other three ensemble (which each of them are a set of GANs, trained on the same dataset, but with different techniques for training and generation phase). At the end, after generating all FIDs comparing generated with real images, we tested our hypothesis, and chose which model performed better.

III. Methodology

III.I. Data Preprocessing

The first thing I did was to remove images we don't need in the dataset. The goal of the project is to generate images of demented brains, so the folder containing non-demented brain images were removed from train and test datasets.

```
!rm -rf data/alzheimers_mri_dataset/train/NonDemented/
!rm -rf data/alzheimers_mri_dataset/test/NonDemented/
```

After cleaning the dataset, I used the complete train set for training control and ensemble models, exactly the same images. Test set was used only for the benchmark model, to generate performance metrics of control and ensemble models.

In the Exploratory Visualisation section, we saw that MRIs images are rectangular, having a resolution of 176x208 pixels, so in order to make calculations easier we needed to make it a square. But also, we want a resolution that is easy to think of while downsampling the data (which is going to make training more predictable when defining discriminator and generator layers outputs), so instead of just cropping the image to the smallest axes, we should also resize the image. The selected resolution

was 128x128x1, where we only want the grayscale channels (black and white images needed). For doing all of this I applied these transformers:

```
transform = transforms.Compose([
    transforms.Grayscale(num_output_channels=1),
    transforms.CenterCrop(176),
    transforms.Resize(size=(128, 128)),
    transforms.ToTensor()
])
```

From the code, we can see transformers defined to:

- Preserve grayscale channels only.
- Define a centred square of 176 pixels and crop it.
- Resize the resulting 176x176 image to 128x128 image.

Comparing with the original image we can see the differences:

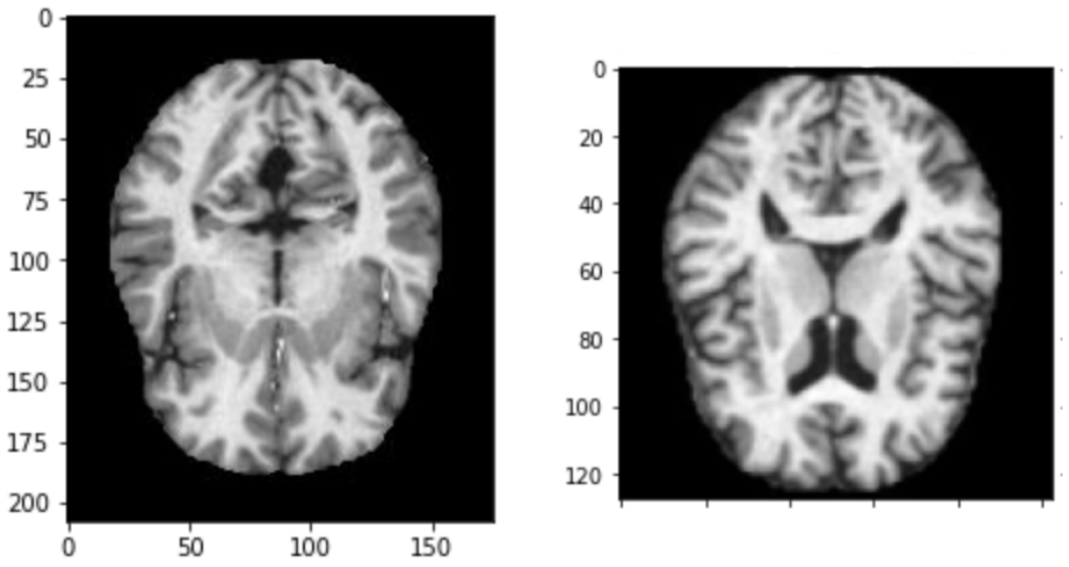

Figure 6: Original MRI left, transformed MRI right.

After transforming these images I showed a set of them, just to make sure that we won't have any image of a brain cutted in a border or an edge case like that.

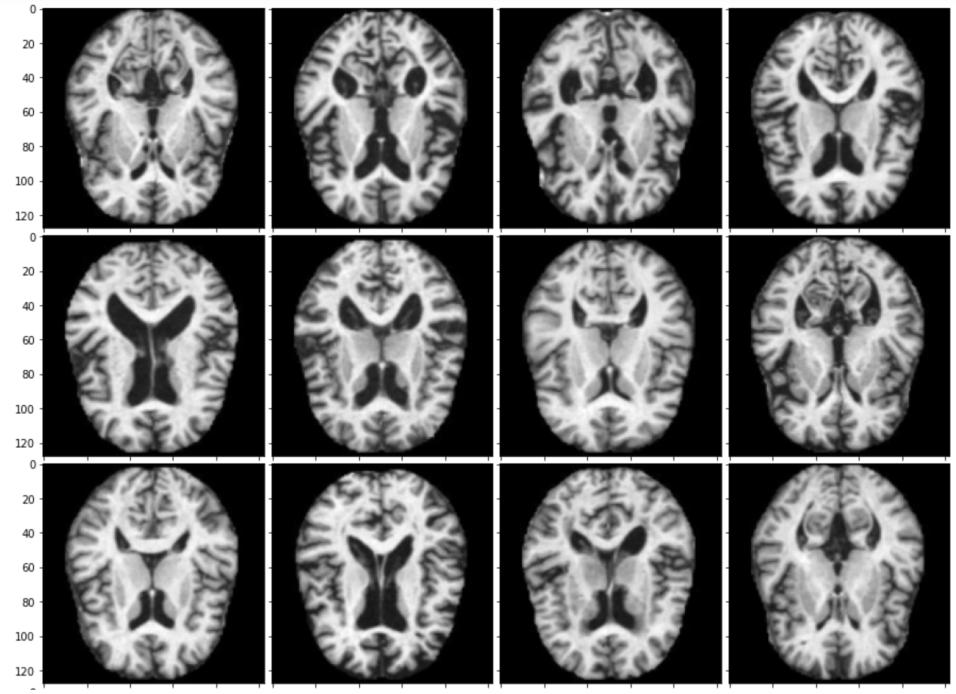

Figure 7: Transformed MRI samples.

These transformers were applied the same way while training control and ensemble models.

III.II. Implementation

GAN implementation

As mentioned before the chosen implementation was DCGAN architecture, following the original paper by Alec Radford et al. [15]. As I needed to define 3 different models, I decided to first implement the control model in detail in one long notebook for showing how DCGAN was defined, and then for the ensembles, I created helper functions that I could use based on the base logic created in the notebook.

In this paper, they mention that Hyperbolic Tangent (Tanh) is used in the output layer of the generator, and the output of Tanh will contain pixel values in a range from -1 to 1, so we need to preprocess pixel values and scale it to be between -1 to 1, instead of 0 to 1. For doing this I implemented this function to scale image values:

```
def scale(x, feature_range=(-1, 1)):
    min_val, max_val = feature_range
    x = x * (max_val - min_val) + min_val
    return x
```

A GAN is actually composed of two networks, a Discriminator and a Generator, so in order to define these two networks I defined these two functions for creating each convolutional layer:

```
def conv(in_channels, out_channels, kernel_size, stride=2, padding=1,
batch_norm=True)
```

```
def deconv(in_channels, out_channels, kernel_size, stride=2, padding=1,
batch_norm=True)
```

Both take as input the size of the previous layer and the size of the current layer, also taking some parameters like stride and batch_norm, which are important for DCGAN definition.

Below we can see how the convolutional classifier, the Discriminator, is receiving an input of a 128x128x1 image, and returning and outputting a single value to determine if the image is real or fake.

```
# 128x128x1
self.conv1 = conv(1, self.conv_dim, kernel_size=4, stride=2, padding=1,
batch_norm=False)
# 64x64x128
self.conv2 = conv(self.conv_dim, self.conv_dim*2, kernel_size=4, stride=2,
padding=1, batch_norm=True)
# 32x32x256
self.conv3 = conv(self.conv_dim*2, self.conv_dim*4, kernel_size=4, stride=2,
padding=1, batch_norm=True)
# 16x16x512
self.conv4 = conv(self.conv_dim*4, self.conv_dim*8, kernel_size=4, stride=2,
padding=1, batch_norm=True)
# 8x8x1024
self.fc = nn.Linear(8*8*8*self.conv_dim, 1)
```

No need for batchnorm in the first layer according to DCGAN paper [15], also we can see that the downsampling of the image is down using stride of 2.

And here we can see how all these convolutional layers are applied in sequence in the discriminator forward path:

```
x = F.leaky_relu(self.conv1(x), 0.2)
x = F.leaky_relu(self.conv2(x), 0.2)
x = F.leaky_relu(self.conv3(x), 0.2)
x = F.leaky_relu(self.conv4(x), 0.2)

# fully connected layer
x = x.view(-1, 8*8*8*self.conv_dim)
output_logits = self.fc(x)
```

One thing to notice here is that we output the logits, so we can apply the loss function later.

Next, we define the Generator, which is going to upsample a noise vector z_size, to an image of 128x128x1.

```
self.fc = nn.Linear(self.z_size, 8*8*8*self.conv_dim)
# 4x4x1024
self.t_conv1 = deconv(self.conv_dim*8, self.conv_dim*4, kernel_size=4, stride=2,
padding=1, batch_norm=True)
```

```
# 16x16x512
self.t_conv2 = deconv(self.conv_dim*4, self.conv_dim*2, kernel_size=4, stride=2,
padding=1, batch_norm=True)
# 32x32x256
self.t_conv3 = deconv(self.conv_dim*2, self.conv_dim, kernel_size=4, stride=2,
padding=1, batch_norm=True)
# 64x64x128
self.t_conv4 = deconv(self.conv_dim, 1, kernel_size=4, stride=2, padding=1,
batch_norm=False)
```

No need for batchnorm in last layer according to DCGAN paper [15], and we can also see that the upsampling in this case is done by applying a stride of 2.

And here we can see how all these convolutional layers are applied in sequence in the generator forward path:

```
x = self.fc(x)
# reshape to target tensor dimensionality
x = x.view(-1, self.conv_dim*8, 8, 8)
# transpose convolutional layers
x = F.relu(self.t_conv1(x))
x = F.relu(self.t_conv2(x))
x = F.relu(self.t_conv3(x))
# finally apply hiperbolic tangent
x = torch.tanh(self.t_conv4(x))
```

For training:

- First I choose a high number of epochs, 65, to train the control model. The idea was to check how performant the model was in terms of generator and discriminator loss, and also the quality of the images it produced, by showing at the end of the training samples of images at all epochs.
- After checking the quality of those images, I was able to see that it was not changing much after epoc 35, so the model was getting stable between epoch 35 and the last epoch 65.
- Epoch 35 was chosen as the epoch control for all models, control model and ensemble models. For the control model I retrained the DCGAN, and saved the state of the model at epoch 35, to be used later when generating metrics comparing with real images.

Ensemble implementation

For each ensemble, a folder was defined to hold all models composing the ensemble, so it was easier to instantiate later when generating images. Folders em1/models, em2/models, em3/models. Implementation of all ensembles was divided into two phases, training and generation. Each ensemble has its own way to train, but for generation I created two heuristics, one that generates a set of images by randomly using a model in the ensemble, and another that generates a set of images by distributing uniformly that set into all models in the ensemble. As mentioned before, epoch 35 was established as a base for all models. Yaxing Wang et al. [16] established that a good number of GANs to compose the ensemble is 8, so I decided to use 8 GANs for the three ensembles.

For training each ensemble:

- **Ensemble Model 1 (EM1) or Standard Ensemble of GANs (eGANs):** trained multiple models from scratch until epoch 35, and saved each Generator state to defined folder. In this case it was easy, because I just needed to add the code for saving the model state after each complete training cycle was finished. This way I was able to create 8 different standard ensemble GANs.
- **Ensemble Model 2 (EM2) or Self-ensemble of GANs (seGANs):** first I defined a list of random 8 numbers between 35 and 65 (the epochs I established when training the control model), so these numbers will be the epoch used to create a snapshot of the state of the generator. So after reaching epoch 35, I was checking if the current epoch was in the list of random numbers, if it was included, then I saved the state of the generator. This way I generated 8 self-ensemble GANs.
- **Ensemble Model 3 (EM3) or Cascade of GANs (cGANs)**: this was the most completed ensemble to generate. In order to create the cascade of GANs, I needed to first define a function that selects all images from the training set that the trained Discriminator thinks are real images, depending on its prediction value, meaning that the output of Discriminator for is 1 or near to 1, this is what is named in the paper as the “Gated Function Q”. After having this function, every time a new GAN is trained, I needed to get all those images that the Q Function selects, and use it as training dataset for the next GAN training. The number of images as a subset of the initial training set was given by content from the paper, they established that using a ratio r of 0.8 it was the optimal value for the cascade, so after each GAN training I was getting 80% of images with the highest prediction value of that GAN trained Discriminator. This was how I was able to generate 8 GANs for the cascade.

Benchmark implementation

For a benchmark model in order to measure the performance of all trained GANs, I did the following:

- Instantiate each model snapshot saved after training.
- Generate the same number of MRIs in the test set, 639 images:
    - For the control model this generation was straightforward, just a regular cycle to generate one image at a time.
    - For the ensembles, according to the paper, the idea is to use each model in the ensemble randomly in the generation phase. I created two ways to generate images, randomly choosing which models, and also distributing uniformly all images to be generated equally by all models.
- After generating all images using GANs, I also preprocessed real images to have the same aspect ratio as the generated images (128x128) from the test set. And from the train set, I randomly chose 639 images and exported them using the same ratio.
- For each images folder I executed pytorch-fid comparing in groups:
    - Real images from test set with train set, and vice versa, so we have a control metric between different sets of real images. A distance, showing how different, how apart are these two sets.
    - Real images from test set with control model and ensemble generated images.
    - Real images from train set with control model and ensemble generated images.
- Having the metrics of all these groups I proceeded to compare, and graph these results.

Refinement

A couple of improvements were made while training control model and ensembles:

- In the DCGAN paper, the default value they recommended for hyperparam learning rate was 0.0002, but I noticed if I changed the learning rate to be 0.0001, generator and discriminator losses were stabilising a bit more quickly.

- Yaxing Wang et al. [16] when defining how the cascade of GANs uses the Q function to select which images are going to be feeded into the next GAN training, is not clear if the images dataset is going to be always reducing in each cycle. I tried to do it like that, in each cycle using the previous training dataset and selecting only 80% of it, but maybe because the dataset I'm working on is too small, the quality of generated images was degrading in each subsequent cycle. I ended up selecting 80% of the original dataset in each cycle, so the first GAN was trained using 2561 images, and the rest of GANs in the ensemble were trained using 2049 images. After doing this, the quality of the images improved, and generator and discriminator losses were stabilising in a better manner.

IV. Results

IV.I. Model Evaluation and Validation

After generating images using control model and ensembles, reformatting real images from test and train datasets into the same as generated images, and then generating Fréchet Inception Distance [8] metrics comparing all these groups, we are able to create charts below:

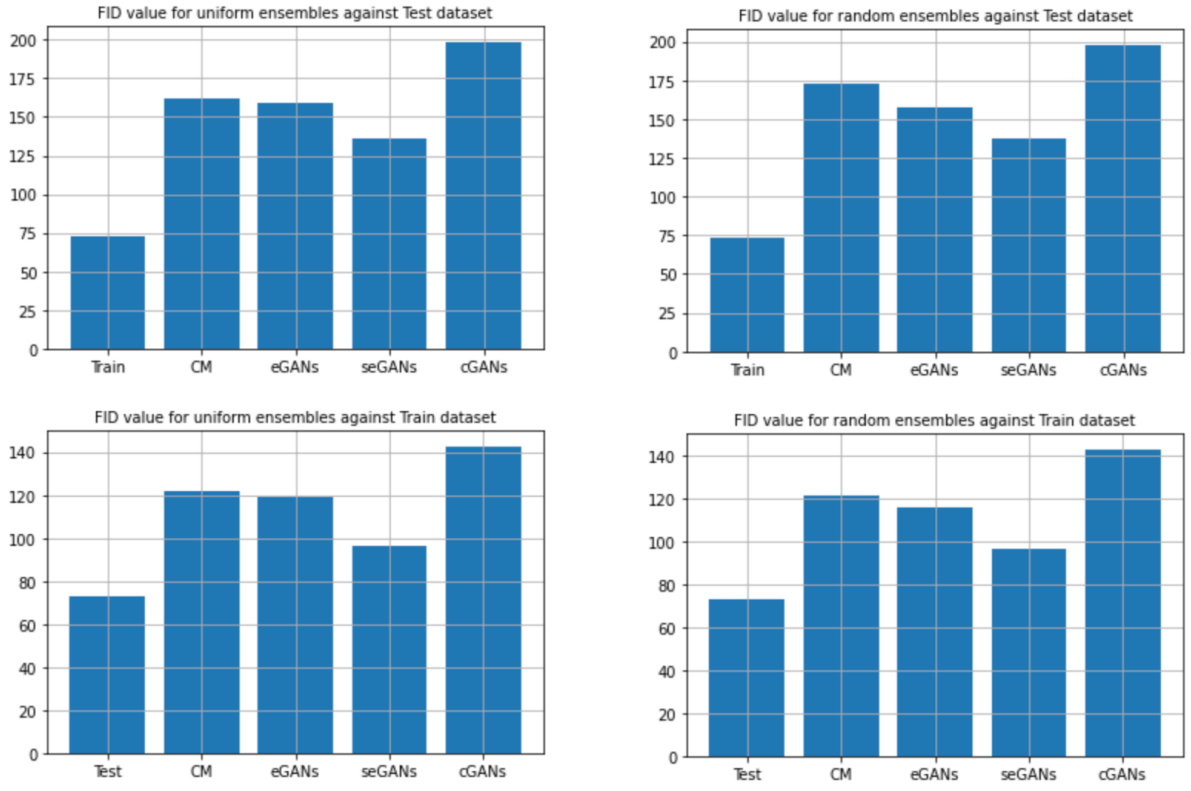

Figure 8: FID value comparing against test and train sets, in uniform and random way.

We can see in these charts (Figure 8) that results in the paper by Yaxing Wang et al. [16] were replicated, where the best ensemble was seGANs, followed up by eGANs, which performed better than the single DCGAN from the control model, in terms of less distance, less difference between generated images from those models than real images, from both test and train set.

IV.II. Justification

In the previous graph (Figure 8), the first bar that we see in each chart, is comparing training with test datasets, and vice versa, this is because it's good to have a metric that would tell us the best result possible that we could get. In general, comparing generated images with train dataset tells us about how well the model learned to represent the dataset used to train it, and comparing with the test dataset tells us how good the model learned to generalise the images.

From previous charts (Figure 8), we see there is a big difference between control model and ensembles performance. But where we see a difference is comparing the control model with the ensembles, where we can see seGANs and eGANs were better than the single GAN control model, and even the same behaviour seen with cascade of GANs, that was outperformed by these other two ensembles.

Furthermore, in the next graph we can see how well each ensemble performed compared to the single DCGAN control model:

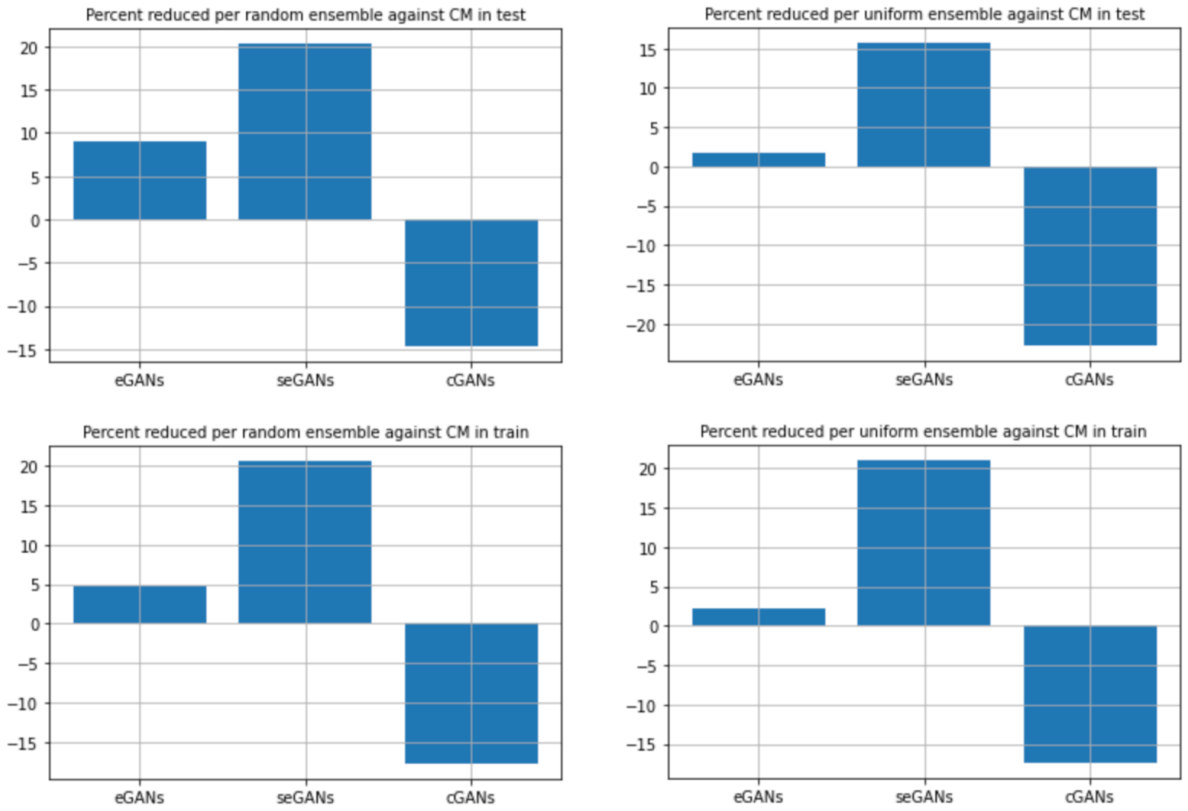

Figure 9: Percent of FID reduced comparing ensembles with control model, in train and test images.

EM1 or eGANs: Generating images with eGANs by randomly choosing a model in the ensemble, performed better than the control model and even better than itself when generating images uniformly choosing GANs in the ensemble, between 5% and 9% better (Figure 9 top and bottom left).

EM2 or seGANs: Generating images with seGANs by randomly choosing a model in the ensemble, performed better than control model and even better than itself when generating images uniformly choosing GANs in the ensemble, the same that happened with eGANS, but the improvements was even higher, around 20% better than single DCGAN control model (Figure 9 all charts).

EM3 or Cascade of GANs: was outperformed even by the single DCGAN control model, probably because of the issue of having too small dataset, in which only one GAN was trained using the whole set of images and the rest was having a reduced space for training (Figure 9 all charts).

V. Conclusion

V.I. Free-Form Visualisation

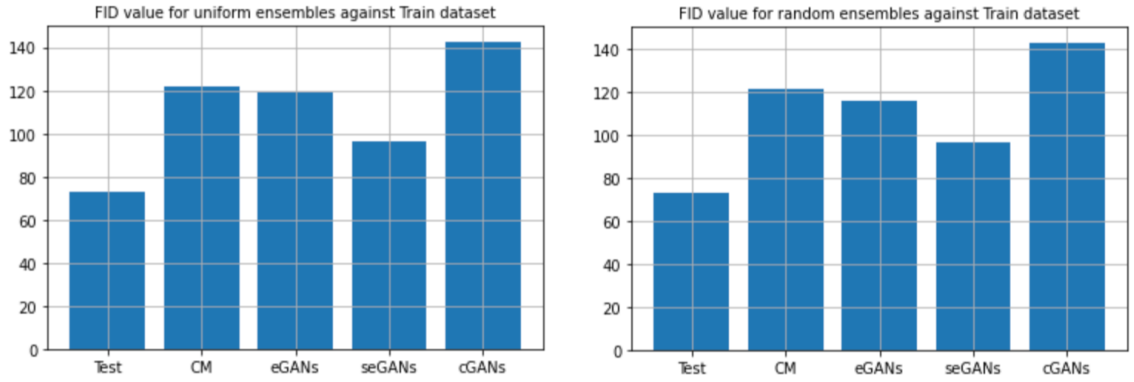

Figure 10: FID value compared against Train dataset.

In between FID charts there are two that look more interesting than the rest (Figure 10), probably more evidence proving results from Yaxing Wang et al. [16], but still good to remark. seGANs FID value is more similar to train and test images FID value, when comparing it to images in train set, meaning that seGANS were able to learn well to represent the dataset used to train it. In these cases, seGANs compared with train images was 96, and test images compared with train images was 72. Any other model was even above 110.

V.II. Reflection

Researching for this project I had the opportunity to mix interesting topics, Alzheimer studies, MRI datasets, different neural networks architecture designs, and use a kaggle challenge related to art for getting a benchmark to measure the results from the project itself. Relating art and science in some way using FID.

The idea was simple, to generate synthetic MRI of brains, to help with data augmentation. I found different alternatives that were already doing that, but I listened to Lex Fridman podcast having Ian Goodfellow as a guest [2], and he mentioned the usage of ensembles and I liked the idea. Then finding the paper from Yaxing Wang et al. [16], where multiple techniques are explained, I needed a benchmark model, which was the most difficult thing to find. After a while I found how GANs are tested, which type of metrics are used to measure the performance of a generator, and I choose FID, which was the easiest way to do all of this, and even more convenient having a pytorch cli [8] tool ready to just compare generated images with real images.

The final product of this project was sufficient to me, fit my expectations because I was able to gather all these interesting papers, articles, and tools, into a document that I think explains in a simple way how to use DGAN ensembles to generate MRI of brains for research, or even use the same architecture and techniques for a different purpose or goal.

V.III. Improvement

In general there are two ways we could improve GAN ensembles performance:

- In terms of the data we can get more and/or better MRIs. Probably a better and bigger dataset can be found by requesting access to The Open Access Series of Imaging Studies (OASIS) [9].
- In terms of improving the architecture of the deep neural network, we can manually try to create a more complex DCGAN, adding more layers. Or we can implement Filippos Konidaris et al. [7] generator and discriminator architecture, and then prepare a seGANs ensemble using that as an individual base mode.

References

[1] Hello Future (2022) Generative AI: a new approach to overcome data scarcity. https://hellofuture.orange.com/en/generative-ai-a-new-approach-to-overcome-data-scarcity/

[2] Ian Good fellow at Lex Fridman Podcast (2019) Ian Goodfellow: Generative Adversarial Networks (GANs) | Lex Fridman Podcast #19. https://www.youtube.com/watch?v=Z6rxFNMGdn0&t=3013s

[3] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter (2018) GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium. https://arxiv.org/pdf/1706.08500.pdf

[4] Kaggle (2019) Alzheimer's Dataset (4 class of Images) - Images of MRI Segementation. https://www.kaggle.com/datasets/tourist55/alzheimers-dataset-4-class-of-images

[5] Kaggle (2020) I'm Something of a Painter Myself: Use GANs to create art - will you be the next Monet?. https://www.kaggle.com/competitions/gan-getting-started/overview/evaluation

[6] Can Kocagil (2021) DCGAN paper implementation using PyTorch to generate faces. https://github.com/cankocagil/DCGAN

[7] Filippos Konidaris, Thanos Tagaris, Maria Sdraka and Andreas Stafylopatis (2019) Generative Adversarial Networks as an Advanced Data Augmentation Technique for MRI Data. https://pdfs.semanticscholar.org/56fc/4ac13c24776692e9f130197e7f55ed29408a.pdf

[8] Steven Lang (2021) FID score for PyTorch. https://github.com/mseitzer/pytorch-fid

[9] LaMontagne P et al (2019), Koenig L et al (2020) Open Access Series of Imaging Studies - OASIS 3 and OASIS 4. https://www.oasis-brains.org

[10] NVIDIA (2018) Stanford Researchers Develop AI that Can Help Diagnose Alzheimer's Disease. https://developer.nvidia.com/blog/stanford-researchers-develop-ai-that-can-help-diagnose-alzheimers-disease

[11] NVIDIA (2019) AI Study Predicts Alzheimer's Six Years Before Diagnosis. https://developer.nvidia.com/blog/ai-study-predicts-alzheimers-six-years-before-diagnosis

[12] NVIDIA (2018) AI Can Generate Synthetic MRIs to Advance Medical Research. https://developer.nvidia.com/blog/ai-can-generate-synthetic-mris-to-advance-medical-research

[13] Paszke, A. et al., 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In Advances in Neural Information Processing Systems 32. Curran Associates, Inc. .http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[14] PyTorch Foundation (2022) BCEWithLogitsLoss - PyTorch. https://pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html

[15] Alec Radford, Luke Metz, Soumith Chintala (2016) UNSUPERVISED REPRESENTATION LEARNING WITH DEEP CONVOLUTIONAL GENERATIVE ADVERSARIAL NETWORKS. https://arxiv.org/pdf/1511.06434.pdf

[16] Yaxing Wang, Lichao Zhang, Joost van de Weijer (2016) Ensembles of Generative Adversarial Networks. https://arxiv.org/pdf/1612.00991.pdf