SAPIENZA
UNIVERSITÀ DI ROMA

+

Medical Image Synthesis Report

Dipartimento di Ingegneria Informatica, Automatica e Gestionale
Corso di Laurea Magistrale in Engineering in Computer Science

Authors:

Campana Matteo
Taloma Redemptor Jr Laceda

A.Y. 2019-2020

Abstract **3**

Data Pre-Processing **4**
Some definitions 5
Technical Details 5

Architecture **6**
Generator 6
Discriminator 6
GAN 6
Dilated Convolutions 8
The use in our work 9

Loss function and training **9**
Euclidean loss LG (let's call this generator G1) 10
Euclidean loss LG + Adversarial loss LADV (let's call this generator G2) 10
Euclidean loss LG + Adversarial loss LADV + Image Gradient Difference loss LGDL (let's call this generator G3) 11
L1 loss + Adversarial loss LADV + Image Gradient Difference loss LGDL (let's call this generator G4) 11

Volume Reconstruction from patches **12**

T2 -> PD **13**
T2 -> PD: results on the validation set to find the best model for each generator 13
T2 -> PD: plots 14
T2 -> PD: comparison of generated images in the test set 15

Residual learning **17**
PD -> T2: results on the validation set to find the best model for each generator 18
PD -> T2: plots 19
PD -> T2: comparison of generated images in the test set 20

Conclusion **21**
Future Works 21

Abstract

Deep learning has achieved great success in the field of artificial intelligence, and many deep learning models have been developed. Generative Adversarial Network (GAN) is one of the deep learning models that has become a new research hotspot. Introduced by Ian Goodfellow during the work at Google Brain, GANs are a new framework for estimating generative models via an adversarial process, in which we simultaneously train two models: a generative model *G* that captures the data distribution, and a discriminative model *D* that estimates the probability that a sample came from the training data rather than *G*.
The significance of the model is to obtain the data distribution through unsupervised learning and to generate more realistic data.

Medical imaging plays a critical role in various clinical applications. However, due to multiple considerations such as cost and radiation dose, the acquisition of certain image modalities may be limited. Thus, medical image synthesis can be of great benefit by estimating a desired imaging modality without incurring an actual scan. In our study we propose a generative adversarial approach, based on the Pix2Pix GAN. Specifically we train a fully convolutional network (FCN) for the generator model that is provided with a given image as input and generates a translated version of the image. Our work is based on the paper D. Nie et al., "Medical Image Synthesis with Deep Convolutional Adversarial Networks", 2018, which tries to improve the current performance of the state-of-the-art methods for biomedical image synthesis.

The goal of our work is to obtain an accurate translation of images taken in a given modality and translate them in another one. The first experiment will check the effectiveness of the authors' loss function for the synthesis of PD-weighted images from T2-weighted ones; the second experiment is going to assess the contribution of residual learning to improve further the quality of the generation.

Data Pre-Processing

We used the IXI dataset (https://brain-development.org/ixi-dataset/) provided by the brain-development.org foundation (https://brain-development.org/). From this dataset we selected the T2-dataset and the PD-dataset, each of them containing around 500 images related to the same patients.

Some definitions

An **MRI** sequence is a number of radiofrequency pulses and gradients that result in a set of images with a particular appearance. This work presents a simplified approach to recognizing common MRI sequences, but does not concern itself with the particulars of each sequence modality.

T2-weighted image (T2WI) is one of the basic pulse sequences in MRI. The sequence weighting highlights differences in the T2 relaxation time of tissues.

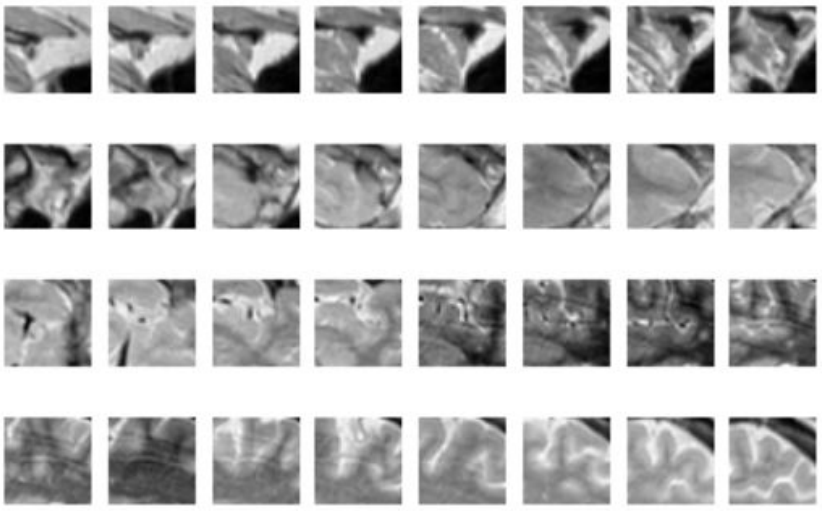
Proton-density weighted images are related to the number of nuclei in the area being imaged (number of hydrogen protons), as opposed to the magnetic characteristics of the hydrogen nuclei. They are produced from the first echo. PD-weighted images result when the contribution of both T1 and T2 contrast is minimized. They have a long TR (2000+ ms) to minimize T1 differences because all tissues exhibit full longitudinal relaxation prior to the next 90 degrees RF pulse. They have a short TE (TE1, 20ms) to minimize T2 differences. High PD tissues appear bright.

Technical Details

All files are in .nii.gz format, a compressed format that contains the MRI details and an array of 3D data representing the image. To process, read, and work with this data we use the python nibabel library (https://nipy.org/nibabel/). Due to the limits of available computing power, we were forced to reduce the original images both in resolution and number of slices.

The result of the processing is a set of 3D images of size 224x224x96 per subject. Since we need a smaller portion of slices as this file is still too large to

work on it, we have iterated over the slices keeping one slice over three, coming to a final shape of 224x224x32. Then T2 and PD datasets have been standardized to have zero mean and unit standard deviation. Finally, according to the paper, we have divided the processed data in patches of size 32x32x32, thus for each patient we got a set of 49 patches.



Example of a 32x32x32 PD-weighted patch

Architecture

Cfr: *II. Methods - A Supervised Deep Convolutional Adversarial Network - Architecture Details*

Generator

This model is defined as a FCN that takes as input the source image and tries to return a plausible synthetic target image. This network has 9 layers containing convolution, batch normalization, and LeakyRelu operations. The kernel sizes are 9^3, 3^3, 3^3, 3^3, 9^3, 3^3, 3^3, 7^3, 3^3. The numbers of filters are 32, 32, 32, 64, 64, 64, 32, 32, 1. The last layer only includes 1 convolutional filter, and its output is considered as the estimated target image. Regarding the architecture, we avoid Max-Pooling layers since they reduce the spatial resolution of the feature maps.
We checked also in the second experiment a residual generator where, as final operation, the input of the network and the output of the last convolutional layer are element-wise summed to generate the final output of the network.

Discriminator

The discriminator *D* is a typical CNN architecture including 3 stages of convolution, BN, LeakyRelu and Max-Pooling, followed by one convolutional layer and 3 fully connected layers where the first two use LeakyRelu as activation function and the last one uses the Sigmoid. The filter size in each layer is set as 3x3x3, the numbers of the filters are 32, 64, 128 and 256 for the convolutional layers, and the number of outputs nodes in the fully connected layers are 512, 128 and 1.
The input of the network is given as two arrays of size 32x32x32, which represent the source image and the target image.

GAN

The GAN is the merge of the two networks. This network takes two inputs (one source image and the corresponding target image). Connected at the source image we have the generator that takes the input and it generates the relative synthetic target image as output. Attached to the output of the generator there is the discriminator that tries to identify whether the image is real or synthetic. The GAN returns the synthetic target images and the prediction computed by the discriminator, but at testing time only the generator is used.

Model: "Generator"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 32, 32, 32, 1)]	0
conv3d_transpose (Conv3DTran	(None, 32, 32, 32, 32)	23328
batch_normalization (BatchNo	(None, 32, 32, 32, 32)	128
leaky_re_lu (LeakyReLU)	(None, 32, 32, 32, 32)	0
conv3d_transpose_1 (Conv3DTr	(None, 32, 32, 32, 32)	27648
batch_normalization_1 (Batch	(None, 32, 32, 32, 32)	128
leaky_re_lu_1 (LeakyReLU)	(None, 32, 32, 32, 32)	0
conv3d_transpose_2 (Conv3DTr	(None, 32, 32, 32, 32)	27648
batch_normalization_2 (Batch	(None, 32, 32, 32, 32)	128
leaky_re_lu_2 (LeakyReLU)	(None, 32, 32, 32, 32)	0
conv3d_transpose_3 (Conv3DTr	(None, 32, 32, 32, 64)	55296
batch_normalization_3 (Batch	(None, 32, 32, 32, 64)	256
leaky_re_lu_3 (LeakyReLU)	(None, 32, 32, 32, 64)	0
conv3d_transpose_4 (Conv3DTr	(None, 32, 32, 32, 64)	2985984
batch_normalization_4 (Batch	(None, 32, 32, 32, 64)	256
leaky_re_lu_4 (LeakyReLU)	(None, 32, 32, 32, 64)	0
conv3d_transpose_5 (Conv3DTr	(None, 32, 32, 32, 64)	110592
batch_normalization_5 (Batch	(None, 32, 32, 32, 64)	256
leaky_re_lu_5 (LeakyReLU)	(None, 32, 32, 32, 64)	0
conv3d_transpose_6 (Conv3DTr	(None, 32, 32, 32, 32)	55296
batch_normalization_6 (Batch	(None, 32, 32, 32, 32)	128
leaky_re_lu_6 (LeakyReLU)	(None, 32, 32, 32, 32)	0
conv3d_transpose_7 (Conv3DTr	(None, 32, 32, 32, 32)	351232
batch_normalization_7 (Batch	(None, 32, 32, 32, 32)	128
leaky_re_lu_7 (LeakyReLU)	(None, 32, 32, 32, 32)	0
conv3d_transpose_8 (Conv3DTr	(None, 32, 32, 32, 1)	864

Total params: 3,639,296
Trainable params: 3,638,592
Non-trainable params: 704

Model: "Discriminator"

Layer (type)	Output Shape	Param #	Connected to
input_2 (InputLayer)	[(None, 32, 32, 32,	0	
input_3 (InputLayer)	[(None, 32, 32, 32,	0	
concatenate (Concatenate)	(None, 32, 32, 32, 2	0	input_2[0][0] input_3[0][0]
conv3d (Conv3D)	(None, 32, 32, 32, 3	1760	concatenate[0][0]
batch_normalization_8 (BatchNor	(None, 32, 32, 32, 3	128	conv3d[0][0]
leaky_re_lu_8 (LeakyReLU)	(None, 32, 32, 32, 3	0	batch_normalization_8[0][0]
max_pooling3d (MaxPooling3D)	(None, 16, 16, 16, 3	0	leaky_re_lu_8[0][0]
conv3d_1 (Conv3D)	(None, 16, 16, 16, 6	55360	max_pooling3d[0][0]
batch_normalization_9 (BatchNor	(None, 16, 16, 16, 6	256	conv3d_1[0][0]
leaky_re_lu_9 (LeakyReLU)	(None, 16, 16, 16, 6	0	batch_normalization_9[0][0]
max_pooling3d_1 (MaxPooling3D)	(None, 8, 8, 8, 64)	0	leaky_re_lu_9[0][0]
conv3d_2 (Conv3D)	(None, 8, 8, 8, 128)	221312	max_pooling3d_1[0][0]
batch_normalization_10 (BatchNo	(None, 8, 8, 8, 128)	512	conv3d_2[0][0]
leaky_re_lu_10 (LeakyReLU)	(None, 8, 8, 8, 128)	0	batch_normalization_10[0][0]
max_pooling3d_2 (MaxPooling3D)	(None, 4, 4, 4, 128)	0	leaky_re_lu_10[0][0]
conv3d_3 (Conv3D)	(None, 4, 4, 4, 256)	884992	max_pooling3d_2[0][0]
batch_normalization_11 (BatchNo	(None, 4, 4, 4, 256)	1024	conv3d_3[0][0]
leaky_re_lu_11 (LeakyReLU)	(None, 4, 4, 4, 256)	0	batch_normalization_11[0][0]
max_pooling3d_3 (MaxPooling3D)	(None, 2, 2, 2, 256)	0	leaky_re_lu_11[0][0]
flatten (Flatten)	(None, 2048)	0	max_pooling3d_3[0][0]
dense (Dense)	(None, 512)	1049088	flatten[0][0]
dense_1 (Dense)	(None, 128)	65664	dense[0][0]
dense_2 (Dense)	(None, 1)	129	dense_1[0][0]

Total params: 2,280,225
Trainable params: 0
Non-trainable params: 2,280,225

Model: "GAN"

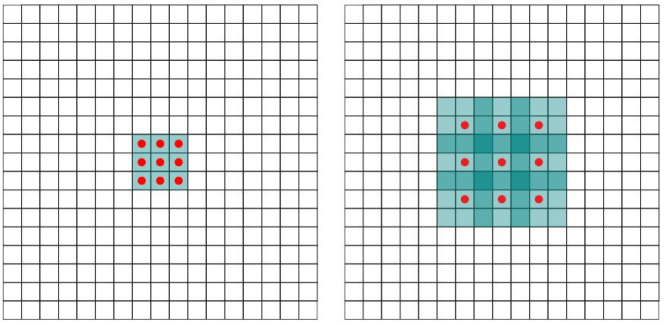
Layer (type)	Output Shape	Param #	Connected to
input_4 (InputLayer)	[(None, 32, 32, 32,	0	
Generator (Model)	(None, 32, 32, 32, 1	3639296	input_4[0][0]
Discriminator (Model)	(None, 1)	2280225	input_4[0][0] Generator[1][0]

Total params: 5,919,521
Trainable params: 3,638,592
Non-trainable params: 2,280,929

Dilated Convolutions

Parts of this paragraph are based on inFERENCe:
https://www.inference.vc/dilated-convolutions-and-kronecker-factorisation/

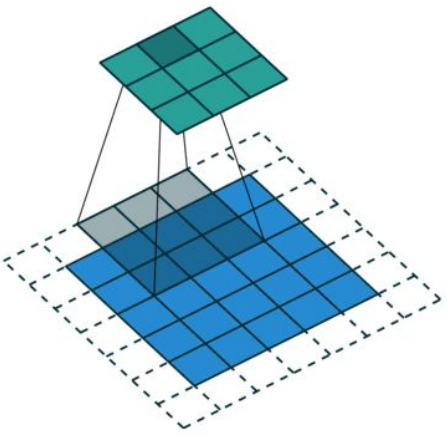
The key application of dilated convolutions is dense prediction, especially vision applications where the predicted object has similar size and structure to the input image (like ours image-to-image translation). In such applications one wants to integrate and balance information from different spatial scales, like pixel-level accuracy (e.g. detection of edges) and the wider global context.

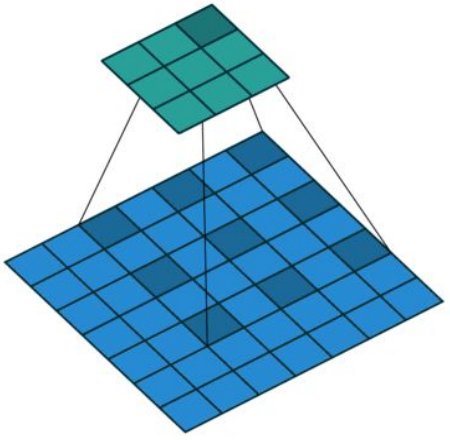


Dilated convolutions support expanding receptive fields by "skipping some points" during convolution according to a dilation rate. This figure shows the expansion of the receptive field with a 3x3 kernel when the dilation rate passes from 1 to 2.

The use in our work

Cfr: *II. Methods - A Supervised Deep Convolutional Adversarial Network - Architecture Details*
Many internal layers in the generator model use the conventional (small) kernel size 3x3x3. The authors (and us with them) adopt the dilated convolution in order to achieve enough receptive field. The dilation rate for the initial and the last convolution layer is 1 (i.e. standard convolution) and 2 for all the internal convolution layers.


Standard Convolution (l=1)


Dilated Convolution (l=2)

Loss function and training

We use a conditional GAN for image-to-image translation, an approach where a target image is generated, conditioned by a given input image. The network includes 1) the generator for estimating the target image and 2) the discriminator for distinguishing the real target from the generated one; in particular, *D* estimates the probability of the input image being part of the distribution of real images (real vs synthetic).

Extract of: *II. Methods - A Supervised Deep Convolutional Adversarial Network - Adversarial Learning*
The loss function for *D* is defined as: $L_D(X,Y) = L_{BCE}(D(Y),1) + L_{BCE}(D(G(X)),0)$, where *X* is the source image, *Y* is the corresponding target image, *G*(*X*) is the estimated image by the generator, and *D*(•) computes the probability of the input to be "real". And, L_{BCE} is the binary cross entropy.

The loss term used to train *G* is defined as:

$$L_{GADV}(X,Y) = \lambda_1 L_{ADV}(X) + \lambda_2 L_G(X,Y) + \lambda_3 L_{GDL}(Y, G(X))$$

Put simply, the training will minimize the overall loss function so that the generated image is both plausible in the content of the target domain (by decreasing the Adversarial loss L_{ADV}), is a plausible translation of the input image (by decreasing the Euclidean loss L_G), and reduces the problem of producing blurry images (by decreasing the Image Gradient Difference loss L_{GDL}).
We are going to check the contribution of all these terms. Regarding their weights, we set $\lambda_2 = \lambda_3 = 1$ and $\lambda_1 = 0.5$ as suggested in the paper.

Euclidean loss L_G (let's call this generator G_1)

The traditional L_2 loss used for image synthesis is defined as $L_G(X,Y) = \|Y - G(X)\|_2^2$. It computes the sum of the pixel-to-pixel squared errors between the ground-truth image *Y* and the generated image *G*(*X*). Using the Euclidean loss only as the whole loss function does not take advantage of the GAN architecture, as the discriminator *D* is not involved.

Euclidean loss L_G + Adversarial loss L_{ADV} (let's call this generator G_2)

The generator does more than minimizing the squared errors between *G*(*X*) and *Y*. Indeed, the network *G* needs to minimize also the Adversarial loss defined as $L_{ADV} = L_{BCE}(D(G(X)), 1)$, that is the binary cross entropy between the decisions by *D* and the wrong labels for the generated images. So, while *D* is learning how to distinguish between the real target data and the synthetic target data generated by *G*, the generator itself aims to produce more realistic target images that confuse *D*.

Inspired to: *II. Methods - A Supervised Deep Convolutional Adversarial Network - Adversarial Learning*
The training of the two networks is performed in an alternating fashion. First, *D* is updated by taking a half mini-batch of real target data and a half mini-batch of generated target data (corresponding to the output of *G*) with their corresponding source samples. Then, *G* is updated by feeding a mini-batch made of the same source samples, plus their corresponding ground-truth target images. We have used 250 subjects for training (which means 12250 patches in the train set), a mini-batch of 7 patches and the Adam version of stochastic gradient descent with small learning rate (1e-4) and modest momentum (0.5).

We realized at validation time that the adversarial approach needs more epochs of training because of the "game" between *D* and *G*. Indeed, the following results will show that the performance of G_2 is worse than G_1 for a relatively modest number of training epochs (32).

Euclidean loss L_G + Adversarial loss L_{ADV} + Image Gradient Difference loss L_{GDL} (let's call this generator G_3)

$$L_{GDL}(Y, G(X)) = \|\nabla Y_X| - |\nabla G(X)_X\|^2 + \|\nabla Y_Y| - |\nabla G(X)_Y\|^2$$

This additional term minimizes the sum of point-to-point squared errors between the magnitudes of the gradients of *Y* and *G*(*X*) along the vertical and the horizontal axis. In this way, the synthetic target image will try to keep the regions with strong gradients (like edges), and will produce possibly sharper images. The performance proved to be better than G_1 and G_2.

Do note that the authors used 3D GDL actually, which means that they had included also the gradient along the *z* axis in the formula (check *II. Methods - A Supervised Deep Convolutional Adversarial Network - Image Gradient Difference Loss*). On the contrary, instead, we were forced to use 2D GDL after discarding ⅔ of the slices in the pre-processing stage, as the slices were no longer continuous. In the end, this means that our experiments are limited to generate and evaluate brain target horizontal sections, whereas the authors consider sagittal and coronal scans too.

L_1 loss + Adversarial loss L_{ADV} + Image Gradient Difference loss L_{GDL} (let's call this generator G_4)

This is a generator that we have built apart from the original paper, which continues the training of the best G_3 model (on the validation set) with the L_2 loss replaced by the L_1 loss. What's the reason? The standardization performed at the pre-processing stage caused the dataset to have zero mean and unit standard deviation. Then we checked that a large part of pixels was actually close to 0 and, as training epochs were passing, the pixel squared errors between ground-truth images and generated ones shrinked even more. The benefit of the L_1 loss on differences < 1 is due to the use of the absolute value rather than the square, thus, it allows to exploit an error margin greater than the one of the L_2 loss for a few more epochs. We observed improvements in terms of MAE and PSNR after just 4 epochs of training.

It is our duty to inform that the current work of the authors (which is still ongoing at present time) allows to specify one of the two losses, and also the power to use in the GDL function. Check https://github.com/ginobilinie/medSynthesis/blob/master/3dganversion/loss_functions.py.

Volume Reconstruction from patches

We remind that all GANs implemented here are actually PatchGANs as they are designed to classify and generate patches of an input image, rather than the entire image. Therefore, a subject's volume (224x224x32) needs to be rebuilt by merging all generated target patches (32x32x32). Indeed, we followed the paper's approach at the testing stage described at *II. Methods*: "an input source image is first partitioned into overlapping patches, and, for each

patch, the corresponding target is estimated by the generator. Then, all generated target patches are merged into a single image to complete the source-target synthesis by averaging the intensities of overlapping regions". In chapter *III. Experiments and Results - A. Experiments on MR-to-CT Synthesis* the authors claim that they set the stride to be 8 along each direction of the image for the overlapping target image regions.
We ended up with 625 overlapping patches per subject after this partition.

T2 -> PD

T2 -> PD: results on the validation set to find the best model for each generator

https://colab.research.google.com/drive/1cR4CY-nSuiFgNCeS38phEiTV8AmkQqEf
The best model of each generator will be used later for a qualitative (visual) comparison of generated images w.r.t. ground-truth images in the test set. Instead, in this chapter, we perform only a quantitative (numerical) assessment of the generators performance in terms of MAE and PSNR on the validation set, which is made of 35 subjects. The URL above points to a notebook whose DataFrames report MAE and PSNR of the 4 models on each patient epoch by epoch, plus the average and the standard deviation of these metrics on the whole validation set. The best model is the one at the epoch with the highest average PSNR, which corresponds often to the epoch with the least average MAE (check the plots of the next notebook).

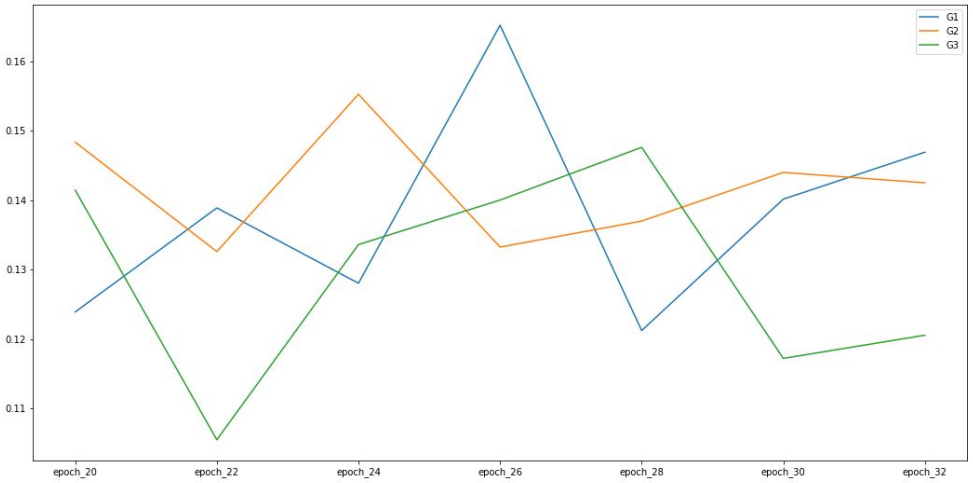
Generator	Best epoch	Average MAE	Average PSNR
G_1	20/34	0.124 (0.017)	26.657 (3.379)
G_2	22/32	0.133 (0.018)	26.638 (3.35)
G_3	22/76	0.106 (0.017)	27.802 (3.294)
G_4	G_3 best epoch + 4	0.089 (0.016)	29.765 (3.225)

G_2 was trained until the epoch 32, almost as G_1 (34 epochs). G_1 reached the maximum average PSNR at epoch 20 (26.657 ±3.379), while G_2 at epoch 22 (26.638 ±3.35). More training epochs would be needed to G_2 in order to leverage the adversarial learning and, just for this reason, G_3 was trained until epoch 76; nevertheless, the maximum average PSNR was recorded at epoch 22 (27.802 ±3.294). Generally, the 3 generators exhibited an increasing of MAE and a decreasing of PSNR after epoch 22. G_4 overcame G_3 definitely with a maximum average PSNR of 29.765 ±3.225.

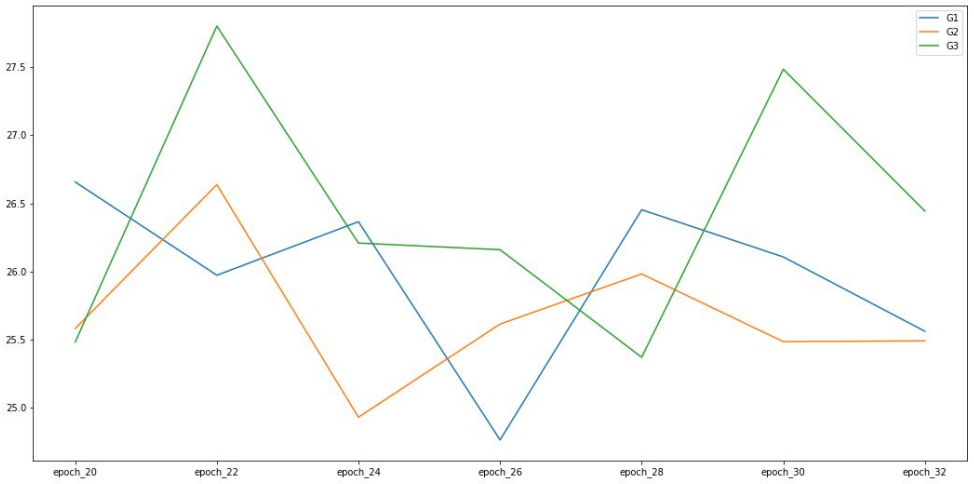
T2 -> PD: plots

https://colab.research.google.com/drive/1AsZ6Gd6wzXuHjyxsHQwZ1VM3njemC2UO
Cartesian plots of the previous DataFrames. You can check the trends of average MAE and average PSNR of each generator on the entire validation set, and also on groups of 5 subjects at a time.

Average MAE on epochs 20-32



Average PSNR on epochs 20-32



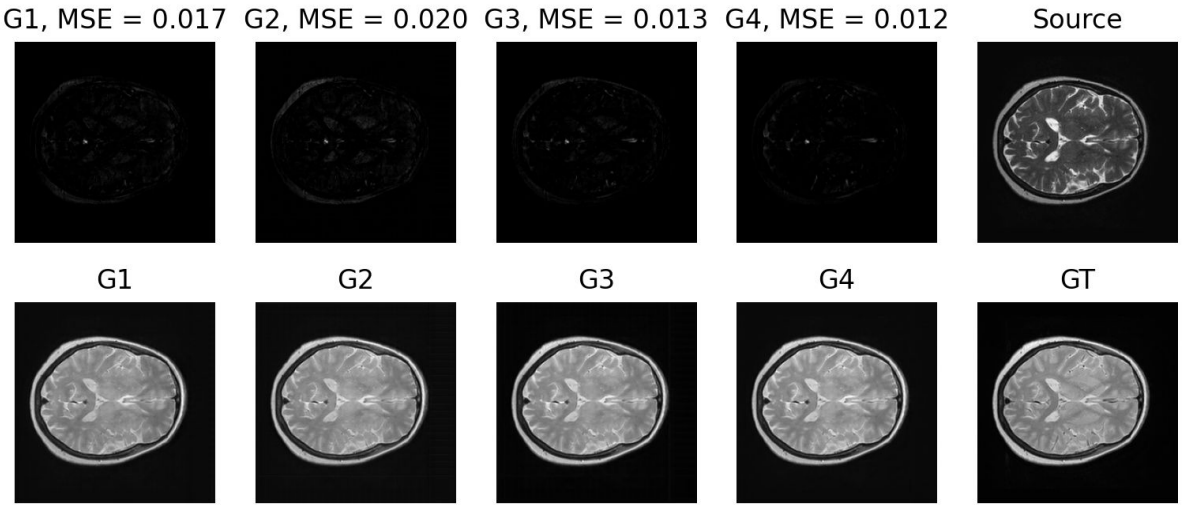
T2 -> PD: comparison of generated images in the test set

https://colab.research.google.com/drive/1nwGQFyoGwgHJfyhZx5XjTKG5NYzSmCHv

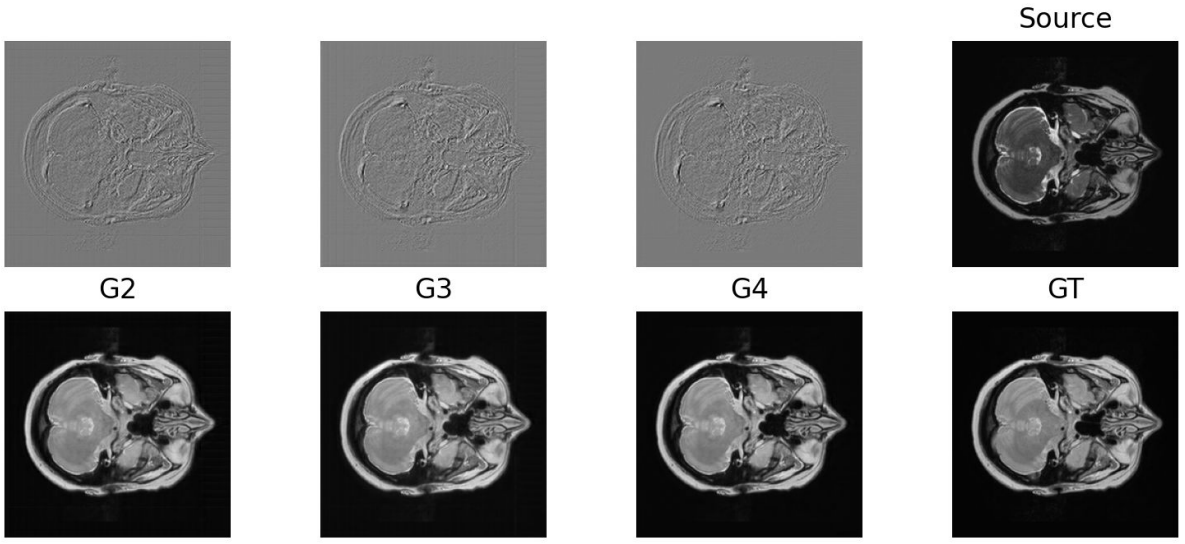
First of all, the scores of MAE and PSNR confirm the performance trends of the 4 generators observed already in the validation set. This table is about the average PSNR on 70 test subjects.

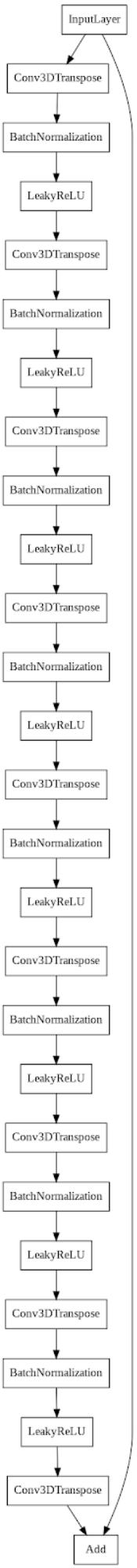
Generator	Average PSNR
G_1	28.054 (3.021)
G_2	27.819 (3.069)
G_3	29.181 (2.884)
G_4	30.199 (2.719)

This is the notebook in which the generated target images are compared visually to the ground-truth target images, first in terms of MSE and then of image gradient difference. G_3 shows a significant drop of MSE: the difference w.r.t. the ground truth is less evident if compared to G_1 and G_2. This proves the effectiveness of the GDL combined with the adversarial learning.



The MSE of G_4 is even lower because the L_1 loss has the double effect of improving the brain appearance and, at the same time, reducing the reconstruction noise of the subject's volume. You can see the benefits very well in this image gradient difference: the horizontal and vertical lines still visible in the background of G_3 are almost disappeared in G_4.





Residual learning

Check: *II. Methods - B. Residual Learning for the Generator*

The effectiveness of residual learning is evaluated on the opposite translation PD -> T2. The reason will be clear at the end of this chapter.
We compare the performance of G_4 with and without the help of residual learning: to be more precise with and without the skip-connection that performs an element-wise addition of the source image to the output of the last convolutional layer. The intuition behind this input-output skip connection is to speed up the training by feeding the starting sample to the bottleneck of the FCN generator. Therefore, the generator will learn epoch by epoch how to correct the output of the last convolutional layer so that, added to the PD source sample, a T2 real target look-alike will be generated. Of course this approach is not viable in case in which PD and T2 modalities were not highly correlated.

The difference between T2 ground-truth target images and PD source images to be exploited in the backpropagation is positive because T2 is darker than PD (dark corresponds to positive values, bright to negative values and gray is around 0 after standardization). Conversely, if the synthesis was performed in the T2 -> PD direction, the LeakyReLu layers would slow down heavily the training because the error to be back-propagated would be negative. Indeed, the saturation of the network occured after very few epochs, thus we stopped the training.

PD -> T2: results on the validation set to find the best model for each generator

https://colab.research.google.com/drive/1FU7Xj5neWYXlmp1VoVrkeKdfud7snEtk

Generator	Best epoch	Average MAE	Average PSNR
G_4	12/24	0.129 (0.031)	29.937 (2.477)
G_4 with RL	24/24	0.101 (0.027)	30.491 (2.336)

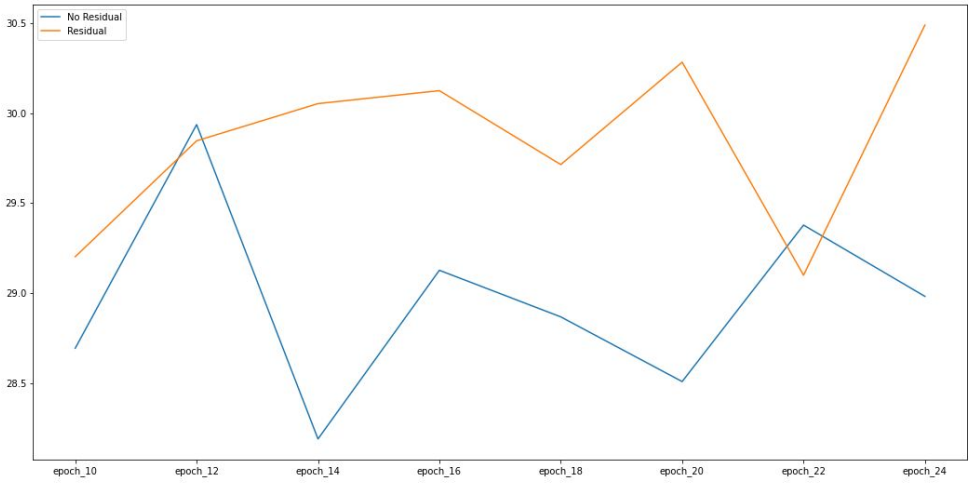
The two generators are trained for 24 epochs with a batch of 49 patches, which is 7 times greater than the batch size used for the T2 -> PD translation, in order to speed up the training. The generator G_4 achieves on the validation set an average PSNR that is similar to the score of the previous experiment. However, the residual learning leads to another significant improvement: +0.554 on average PSNR.

PD -> T2: plots

https://colab.research.google.com/drive/1AsZ6Gd6wzXuHjyxsHQwZ1VM3njemC2UO
Scroll down the notebook to Residual Learning.

Average MAE on epochs 10-24

Average PSNR on epochs 10-24

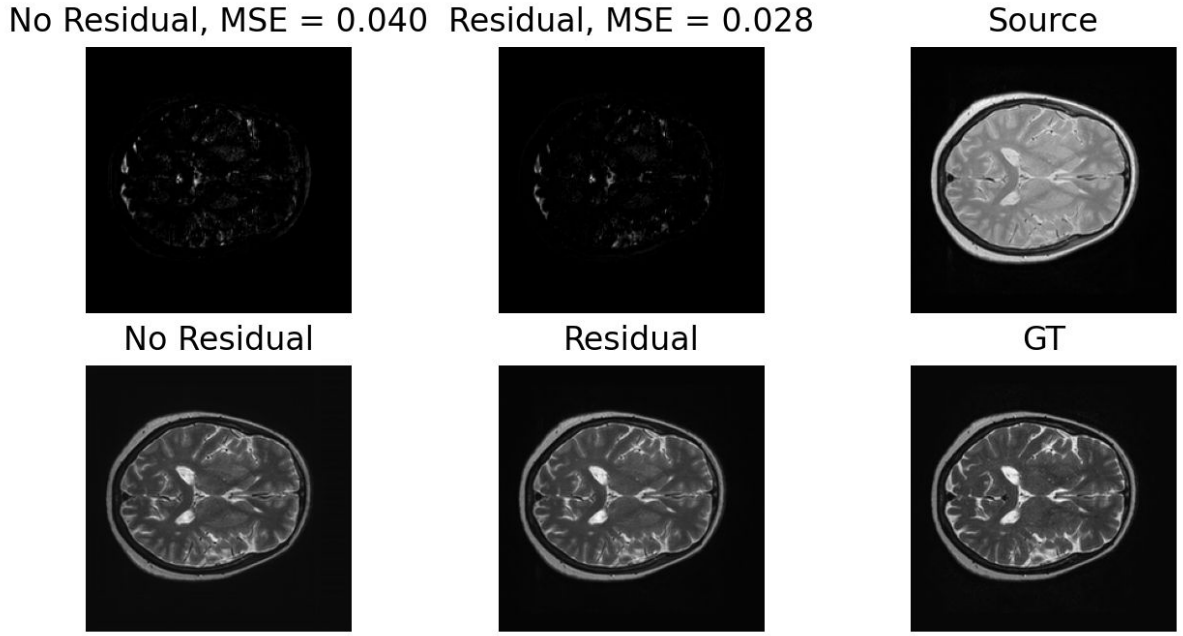


Residual learning lets to achieve lower average MAE and higher average PSNR at training time. Furthermore, the trends of the residual model suggest possible improvements beyond the 24th epoch.

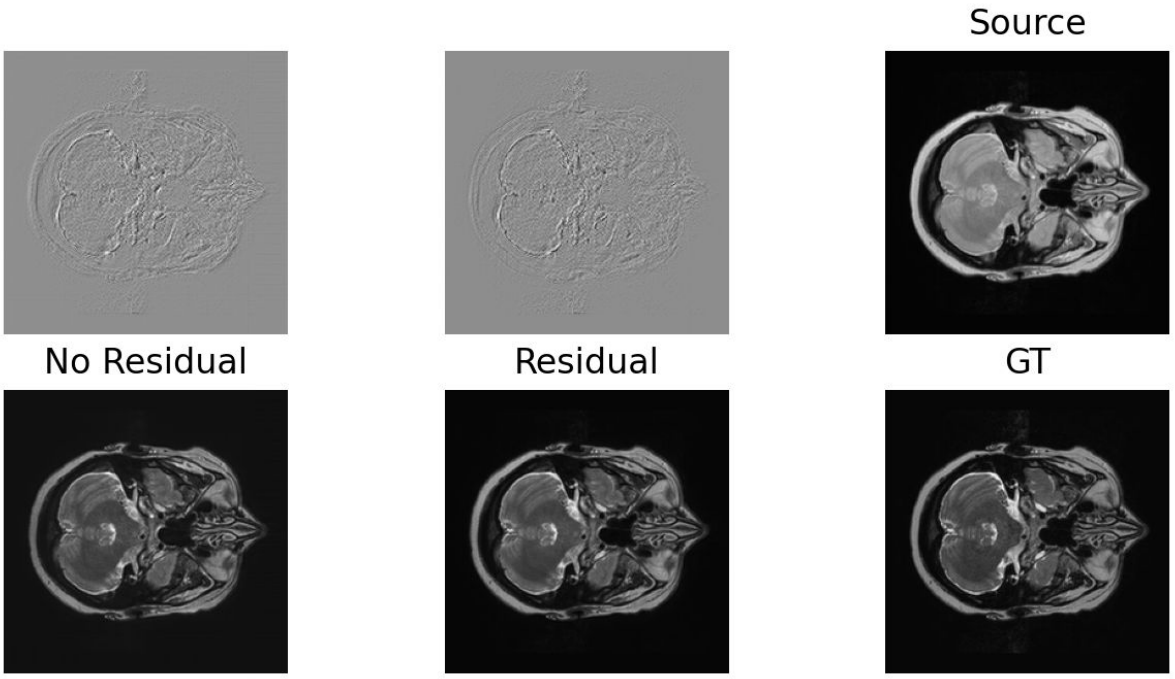
PD -> T2: comparison of generated images in the test set

https://colab.research.google.com/drive/1RTlTg6YlBpg7KkAnCA9e-KQ1xc8b7Vow
The benefit of residual learning is confirmed also by the scores on the test set.

Generator	Average PSNR
G_4	30.015 (1.925)
G_4 with residual learning	30.64 (1.896)



The skip-connection has the advantage of flattening the gradient differences w.r.t. T2 real target image, especially along the edges of the brain, and removing the reconstruction noise almost totally (check the image gradient background carefully).


Source
No Residual
Residual
GT

Conclusion

Cfr: *IV. Conclusion*
We have repeated the authors' experiment for estimating a target image from a source image of different modalities via adversarial learning. Moreover, the quality of the generation is enhanced by image gradient difference loss, especially along the borders of the brain. The high similarity between PD-weighted and T2-weighted images is exploited through a long-term residual connection for accelerating training, improving edges and removing the noise due to patch-based reconstruction.
Performance evaluation is carried out in terms of MAE and PSNR. It is interesting to note that the PSNR at validation and test time is better for models minimizing the L_1 loss rather than the L_2 loss, though the PSNR is based on MSE (a L_2 metric). On the other hand, the PSNR is just a global metric: it is not suited in case of medical setting where the image has to be anatomically correct. Therefore, the authors trained a FCN to segment images into White Matter, Gray Matter and Cerebrospinal Fluid; then they measured the closeness of generated and real segmentation maps (check *III. Experiments and Results - More Evaluation for the Image Reconstruction Quality*). This additional verification is out of our own work.

Future Works

We remind that we worked on horizontal brain sections only because of the limited GPU availability. It would be interesting to assess the image synthesis also on other anatomical planes.
In possible future work we can implement also an Auto-Context Model to compensate the scattering effects generated by the partition of the original source image in patches, and the consequent reconstruction of the synthetic target to obtain the desired final image (cfr. *II. Methods - C. Auto-Context Model (ACM) for Refinement*).