SegSRGAN: Super-resolution and segmentation using generative adversarial networks - Application to neonatal brain MRI

Quentin Delannoy, Chi-Hieu Pham, Clément Cazorla, Carlos Tor-Díez, Guillaume Dollé, Hélène Meunier, Nathalie Bednarek, Ronan Fablet, Nicolas Passat, François Rousseau

SegSRGAN: Super-resolution and segmentation using generative adversarial networks — Application to neonatal brain MRI

Quentin Delannoy[a,b], Chi-Hieu Pham[a,c], Clément Cazorla[b], Carlos Tor-Díez[c], Guillaume Dollé[d], Hélène Meunier[e], Nathalie Bednarek[b,e], Ronan Fablet[f], Nicolas Passat[b], François Rousseau[c]

[a] *Equal first co-authors*
[b] *Université de Reims Champagne Ardenne, CReSTIC EA 3804, 51097 Reims, France*
[c] *IMT Atlantique, LaTIM U1101 INSERM, UBL, Brest, France*
[d] *Université de Reims Champagne Ardenne, LMR FRE 2011, 51097 Reims, France*
[e] *Service de médecine néonatale et réanimation pédiatrique, CHU de Reims, France*
[f] *IMT Atlantique, Lab-STICC UMR CNRS 6285, Brest, France*

Abstract

Background and objective: One of the main issues in the analysis of clinical neonatal brain MRI is the low anisotropic resolution of the data. In most MRI analysis pipelines, data are first re-sampled using interpolation or single image super-resolution techniques and then segmented using (semi-)automated approaches. In other words, image reconstruction and segmentation are then performed separately. In this article, we propose a methodology and a software solution for carrying out simultaneously high-resolution reconstruction and segmentation of brain MRI data.
Methods: Our strategy mainly relies on generative adversarial networks. The network architecture is described in details. We provide information about its implementation, focusing on the most crucial technical points (whereas complementary details are given in a dedicated GitHub repository). We illustrate the behaviour of the proposed method for cortex analysis from neonatal MR images.
Results: The results of the method, evaluated quantitatively (Dice, peak signal-to-noise ratio, structural similarity, number of connected components) and qualitatively on a research dataset (dHCP) and a clinical one (Epirmex), emphasize the relevance of the approach, and its ability to take advantage of data-augmentation strategies.
Conclusions: Results emphasize the potential of our proposed method / software with respect to practical medical applications. The method is provided as a freely available software tool, which allows one to carry out his/her own experiments, and involve the method for the super-resolution reconstruction and segmentation of arbitrary cerebral structures from any MR image dataset.

Keywords: super-resolution, segmentation, 3D generative adversarial networks, neonatal brain MRI, cortex.

1. Introduction

Long-term studies of the outcome of prematurely born infants have clearly documented that the majority of such infants may have significant motor, cognitive, and behavioral deficits [13, 26]. However, there is a limited understanding of the nature of the cerebral abnormality underlying these adverse neurologic outcomes. In this context, Magnetic Resonance Imaging (MRI) provides unique opportunities for in vivo investigation of the early developing human brain. However, the analysis of these clinical neonatal brain MRI data remains challenging. This is mainly due to their low anisotropic resolution. As a consequence, improving morphological data processing such as image resolution enhancement and brain segmentation is the cornerstone for further providing robust morphometry analysis tools to the scientific and clinical communities.

When dealing with anisotropic low resolution images, one of the first key components of the processing pipeline of clinical MRI data is the upsampling image estimation. Super-resolution (SR) is a post-processing technique that aims at enhancing the resolution of an imaging system [10]. However, SR is a challenging inverse problem; in particular the estimation of texture and details remains difficult. Recently, su-

pervised deep learning-based techniques have shown great improvement over model-based approaches, for that purpose. Indeed, applying 3D convolutional neural networks (CNNs) yields promising results for MRI data [31, 6].
However, it is known that in this methodological context, the use of a pixel-wise based loss function may lead to oversmoothing high resolution images [15]. Indeed, a pixel-wise comparison based loss does not take into account how the image aspect seems real (e.g., realistic texture, shape of anatomical details). In order to provide realistic high resolution images, perceptual components will be added to our loss function. As used in [19] to constitute their perceptual loss, the GAN loss will be used to constitute our perceptual loss; in our case, the GAN loss will entirely constitute our considered perceptual loss. Note that, as for the HR image, measuring how the estimated segmentation map is realistic is also relevant. Such GAN networks have already been studied for instance in [28] or [21].
Once a high resolution image reconstruction has been performed, the implementation of an automatic segmentation robust approach is crucial for brain structure analysis. Dedicated (in general, automatic) segmentation methods can be classified into four types (see e.g. [23] for a recent survey): unsupervised [30, 11]; parametric [22]; classification [33, 40, 27, 4, 25] / atlas fusion [34, 7, 16, 35]; and deformable models [37, 38]. In addition, GANs were also recently proposed for brain MRI segmentation purpose [28].
Among the structures of interest, thin ones and in particular the cortical gray matter, remain difficult to analyze. However, the cortex is a region of interest, as emphasized by recent works focusing e.g. on brain folding [9, 20, 29], cortical connectivity [2] and cortical development [3, 41]. Since the cerebral cortex is a thin surface object, it remains difficult to segment in neonatal MRI data. In particular, the segmentation step is generally considered separately from image reconstruction.
Our purpose is to propose an end-to-end methodology dedicated to the analysis of anisotropic low resolution MR images. In particular, we aim to deal with complex anatomical structures, and in particular the cortex. To this end, we propose a GAN-based approach, namely SegSRGAN, that generates both the perceptually super-resolved image and a cortical segmentation map from a single low-resolution (LR) MR image.
This article, which is an extended and improved version of the conference paper [32], is organized as follows. In Section 2, we describe the formulation of the super-resolution and segmentation image problem we aim to tackle. In Section 3, we provide technical details related to our method, first on the methodology and second on the implementation and software. In Section 4, we describe experiments carried out on various kinds of data, and we quantitatively and qualitatively assess the effects of several parameters. Section 5 provides a concluding discussion. For the sake of reproducibility, additional contents dealing with implementation and used resources are available in Appendix A

2. Super-resolution and image segmentation problem formulation

2.1. Formulation of single image super-resolution problem

The goal of single image SR is to estimate a high-resolution (HR) image $\mathbf{X} \in \mathbb{R}^m$ from an observed LR image $\mathbf{Y} \in \mathbb{R}^n$, with $m > n$. Such an SR problem can be formulated using the following linear observation model:

$$\mathbf{Y} = H_{\downarrow} B\mathbf{X} + N = \Theta\mathbf{X} + N \tag{1}$$

where $N \in \mathbb{R}^n$ is the additive noise, $B \in \mathbb{R}^{m\times m}$ is a blur matrix (depending on the point spread function), $H_{\downarrow} \in \mathbb{R}^{n\times m}$ is a downsampling decimation matrix and $\Theta = H_{\downarrow} B \in \mathbb{R}^{n\times m}$.
A popular way of solving this SR problem is to define the matrix Θ^{-1} as the combination of a restoration operator $F \in \mathbb{R}^{m\times m}$ and an upscaling interpolation operator $S^{\uparrow} \in \mathbb{R}^{m\times n}$ that computes the interpolated LR image $\mathbf{Z} \in \mathbb{R}^m$ associated to $\mathbf{Y}$, i.e. $\mathbf{Z} = S^{\uparrow}\mathbf{Y}$. In the context of a supervised learning, given a set of HR images $\mathbf{X}_i$ and their corresponding LR images $\mathbf{Y}_i$, the restoration operator F can be estimated by minimizing a loss of the following form:

$$\widehat{F} = \arg\min_{F} \sum_i d(\mathbf{X}_i - F(\mathbf{Z}_i)) \tag{2}$$

where d can be e.g. a ℓ_1 norm, a ℓ_2 norm or a Charbonnier loss [5] (namely a differentiable variant of ℓ_1 norm). As in [18], our model loss will be based on a Charbonnier loss.
Once $\widehat{F}$ is estimated, given an LR image $\mathbf{Y}$, the computation of the corresponding estimated HR image $\widehat{\mathbf{X}}$ is straightforwardly expressed as $\widehat{\mathbf{X}} = \widehat{F}(S^{\uparrow}\mathbf{Y})$. In [31, 6], it was shown that 3D CNNs could be used to accurately estimate the restoration mapping $\widehat{F}$ for brain MRI.
Hereafter, the estimated HR image $\widehat{\mathbf{X}}$ will be denoted as super-resolution (SR) image.

2.2. Formulation of image segmentation problem

In order to balance the SR and the segmentation contributions to the loss value, image segmentation is indeed viewed as a supervised regression problem:

$$\mathbf{S_X} = R(\mathbf{X}) \tag{3}$$

where R denotes a non-linear mapping from the upscaled image $\mathbf{X}$ to the segmentation map $\mathbf{S_X}$. Similarly to the SR problem, by assuming that we have a set of images $\mathbf{X}_i$ and their corresponding segmentation maps $\mathbf{S_{X_i}}$, a general approach for solving such segmentation problem consists of finding the mapping R by minimizing the following loss function:

$$\widehat{R} = \arg\min_R \sum_i d(\mathbf{X}_i - R(\mathbf{X}_i)) \tag{4}$$

3. Method description

3.1. Network loss function and architecture

We now propose to use a GAN-based approach to estimate jointly an HR image and its corresponding segmentation map from an LR image.
In our case, GAN-based approaches consist of training a generating network G that estimates, for a given interpolated input image, its corresponding HR image and a segmentation map. The discriminator network D is designed to differentiate real HR and segmentation images couples (real) from generated SR and segmentation images couples (fake).

3.1.1. Loss function: joint mapping by generative adversarial networks

Preliminary remark: in order, to make voxel values comparable between SR and segmentation images, the SR and HR images are normalized between 0 and 1.
The considered loss function can be split into two different terms:

- a term $\mathscr{L}_{rec}$ (reconstruction loss) which directly measures the generator output image (voxel-wise) distance from the real one. This first term is directly linked with the regression view of the problem, as explained in Sections 2.1 and 2.2;
- a term $\mathscr{L}_{adv}$ (adversarial loss) which expresses the game between the generator and the discriminator. This term aims to measure how "realistic" is the generated image (i.e., both the SR and segmentation maps).

The convolution-based generator network G takes as input an interpolated LR image $\mathbf{Z}$ and computes an HR image $\widehat{\mathbf{X}}$ and a segmentation map $\widehat{\mathbf{S_X}}$ by minimizing the reconstruction loss:

$$\mathscr{L}_{rec} = \mathbb{E}_{\mathbf{X}\sim\mathbb{P}_\mathbf{X},\mathbf{S_X}\sim\mathbb{P}_{\mathbf{S_X}}}\left[\rho\left((\mathbf{X},\mathbf{S_X}) - G(\mathbf{Z})\right)\right] \tag{5}$$

where the difference is computed voxel-wise, and ρ is the robust Charbonnier loss [5] ($x \in \mathbb{R}^n$):

$$\rho(x_1,..,x_n) = \frac{1}{n}\sum_{i=1}^{n}\sqrt{(x_i^2+\nu^2)} \tag{6}$$

with ν set here to 10^{-3}.
One can note that both segmentation and SR values lie between 0 and 1. This ensures a balanced contribution of the SR and the segmentation maps to the definition of the reconstruction loss $\mathscr{L}_{rec}$.
Traditionally, the adversarial function is modeled as a minimax objective[1]. However this loss can suffer from vanishing gradient, due e.g. to a discriminator saturation. Consequently, we propose to alternatively use Wasserstein GAN loss as described in [12].
Indeed, this type of GAN aims to minimize the Earth-Mover distance between two distributions $\mathbb{P}_r$ and $\mathbb{P}_g$, defined as:

$$W(\mathbb{P}_r,\mathbb{P}_g) = \inf_{\gamma\in\Pi(\mathbb{P}_r,\mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}\left[||x-y||\right] \tag{7}$$

$$= \sup_{||f||_L\leq 1} \mathbb{E}_{x\sim\mathbb{P}_r}[f(x)] - \mathbb{E}_{x\sim\mathbb{P}_g}[f(x)] \tag{8}$$

where, $\Pi(\mathbb{P}_r,\mathbb{P}_g)$ is the set of all the distributions of which marginals are $\mathbb{P}_r$ and $\mathbb{P}_g$, respectively, and the supremum is calculated over all the 1-Lipschitz functions $f : A \to \mathbb{R}$.
In this GAN, the discriminator learns the parametrized function f and the generator aims at minimizing this distance. Hence, in practice, the adversarial part of the loss function is:

$$\begin{aligned}\mathscr{L}_{adv} = {} & \mathbb{E}_{\mathbf{X}\sim\mathbb{P}_\mathbf{X},\mathbf{S_X}\sim\mathbb{P}_{\mathbf{S_X}}}[D((\mathbf{X},\mathbf{S_X}))] \\ & - \mathbb{E}_{\mathbf{Z}\sim\mathbb{P}_\mathbf{Z}}[D(G(\mathbf{Z}))] \\ & - \lambda_{gp}\mathbb{E}_{\widehat{\mathbf{XS}}}[(\| (\nabla_{\widehat{\mathbf{XS}}} D(\widehat{\mathbf{XS}}) \|_2 - 1)^2]\end{aligned} \tag{9}$$

where

$$\widehat{\mathbf{XS}} = (1-\epsilon)(\mathbf{X},\mathbf{S_X}) + \epsilon G(\mathbf{Z}) \tag{10}$$

[1] That is, by solving

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\in p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\in p_z(z)}[\log(1-D(G(z)))]$$

The solution to this minimax problem can be interpreted as a Nash equilibrium, a concept from game theory.

is uniformly sampled between $(\mathbf{X}, \mathbf{S_X})$ and $G(\mathbf{Z})$, whereas $\lambda_{gp} > 0$ and ∇ denotes the gradient penalty coefficient and the gradient operator, respectively.
The images $\mathbf{X}$, $\mathbf{S_X}$ and $\mathbf{Z}$ are randomly extracted from the data distributions of HR images $\mathbb{P}_{\mathbf{X}}$, HR segmentation maps $\mathbb{P}_{\mathbf{S_X}}$ and LR images $\mathbb{P}_{\mathbf{Z}}$, respectively. The terms $D((\mathbf{X}, \mathbf{S_X}))$, $D(G(\mathbf{Z}))$ and $D(\widehat{\mathbf{XS}})$ are the responses of the discriminator with respect to the real data, the generated data and the interpolated data, respectively.
The loss function of the GAN is then:

$$\mathscr{L} = \mathscr{L}_{rec} + \lambda_{adv}\mathscr{L}_{adv} \tag{11}$$

where $\lambda_{adv} > 0$.
Finally the game between the generator and the discriminator is modeled as:

$$\min_G \max_D \mathscr{L} \tag{12}$$

3.1.2. Neural network architecture

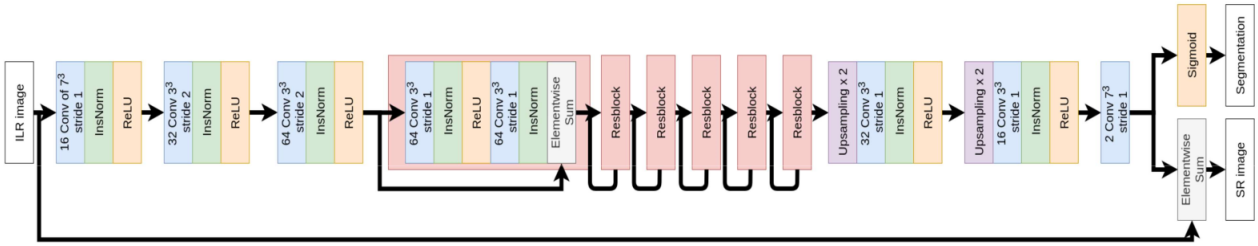
Generator architecture. The generator network is a convolution-based network with residual blocks. It takes as input the interpolated LR image. It is composed of 18 convolutional layers: three for the encoding part, twelve for the residual part and three for the decoding part.
Let C_j^i-S^k be a block consisting of the following layers: a convolution layer of j filters of size i^3 with stride of k, an instance normalization layer (InsNorm) [36] and a rectified linear unit (ReLU). R_k denotes a residual block as Conv-InsNorm-ReLU-Conv-InsNorm that contains 3^3 convolution layers with k filters. U_k denotes layers as Upsampling-Conv-InsNorm-ReLU layer with k filters of 3^3 and stride of 1. The generator architecture is then: C_{16}^7-S^1, C_{32}^3-S^2, C_{64}^3-S^2, R_{64}, R_{64}, R_{64}, R_{64}, R_{64}, U_{32}, U_{16}, C_2^7-S^1 (see Figure 1(a)).
During the encoding, the number of kernels is multiplied by 2 at each convolution, from 16 to 64. The last convolutional layer produces two 3D images: the first will be turned into a class probability map (using a sigmoid activation); the second will be summed with the original interpolated image. In order to improve the training procedure performance, instance normalization layers are used on the result of each convolution (before application of activation function).
Note that summing the prediction of the generator with the interpolated image implies some constraints on the interpolated image size. Indeed, the decoder uses two upsampling layers which multiply the size of each channel by two. Then, the neural network output size is divisible by 4 in each dimension. Finally, to have the same dimension between input and output, we need an input size which is also divisible by 4.

Discriminator architecture. The discriminator network is fully convolutional. It takes as input an HR image and a segmentation map.
The discriminator contains five convolutional layers with an increasing number of filter kernels, increasing by a factor of 2 from 32 to 512 kernels. Let C_k be a block consisting of the following layers: a convolution layer of k filters of size 4^3 with stride of 2 and a Leaky ReLU with a negative slope of 0.01. The last layer C_1^2 is a 2^3 convolution filter with stride of 1. No activation layer is used after the last layer. The discriminator then consists of C_{32}, C_{64}, C_{128}, C_{256}, C_{512}, C_1^2 (see Figure 1(b)).
For the generator as for the discriminator, the number of output channels for each convolutional layer is multiplied by 2 at each layer. The number of output channels of the first layer is a parameter of the algorithm, as further discussed in Section 4.

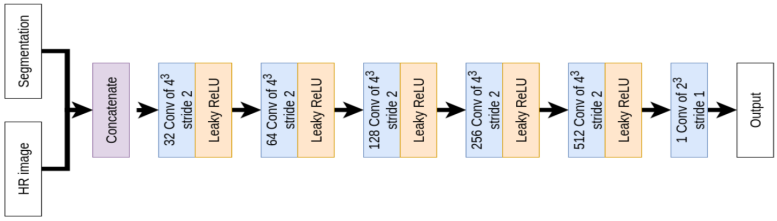
3.1.3. Training and testing

Before each application of the network, the LR images are normalized within $[0,1]$ and interpolated using cubic splines. These two steps are common to both training and testing.
As we will see in the next section, before applying the neural network, the images are also split into patches of a given size.
For training purpose, the data can be augmented with respect to (1) contrast modification (LR and HR images) and (2) Gaussian noise addition (LR images only). In particular, this can be of interest when the data to be processed are distinct from the data used for training.

Training. Figure 2(a) illustrates the pipeline considered for designing the training, at each epoch. Practically, data augmentation is carried out differently for each epoch and for each image. Indeed, for each image and for each epoch, the coefficient used for contrast modification is randomly drawn, such as the additive Gaussian noise added to the image.
From these training data, the discriminator and the generator are trained the following way. For each generator weights update, the discriminator weights are updated five times. The training data for the discriminator are randomly chosen at each iteration,

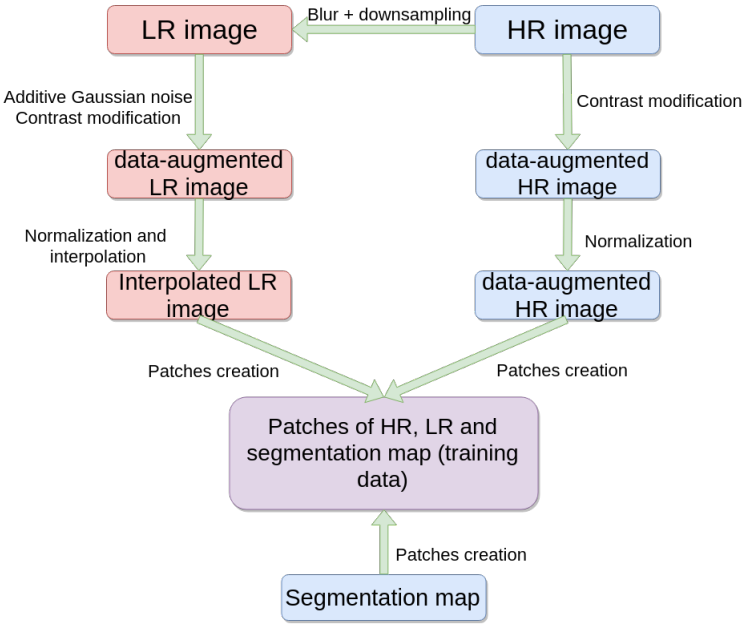


(a) Generator architecture

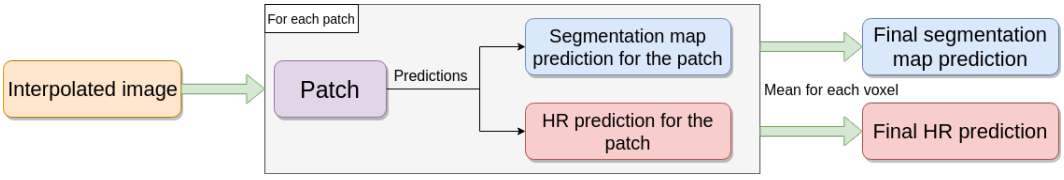


(b) Discriminator architecture

Figure 1: Neural network architecture. (a) Generator architecture. (b) Discriminator architecture. See Section 3.1.2.



(a) Training



(b) Testing

Figure 2: (a) Training data creation for one epoch, and (b) prediction in the testing pipeline. See Section 3.1.3.

whereas the generator training data successively pass over each batch.
The parameters used for the training are the following:

- Adam optimizer parameters similar as in the original Adam paper [17];
- batch size of 32 patches;
- $\lambda_{gp} = 100$ and $\lambda_{adv} = 0.001$;
- patch size of 64^3 with a 20 voxels shifting.

For each training carried out, the maximal number of epochs is set to 200 and the final weights are those which maximize the performance on a testing data set.

Testing. Figure 2(b) illustrates the testing pipeline.

3.2. Implementation and software

The method is implemented in Python 3.6 but the code can also be used in Python 2.7. It was packaged in Pipy[2] and can be easily installed using the following command:

```
pip install SegSRGAN
```

However, if GPUs are available, one should install tensorflow-gpu (see installation tutorial[3]) manually.
This implementation is also available on a GitHub repository[4] which also contains all the weights that we have trained.
The proposed implementation has been developed in order to allow one to train and test arbitrarily any derived model from the one initially proposed. This implies that the executable Python file (described in Section 3.2.2), and especially the training one, authorizes the tuning of various parameters (see the readme file of the GitHub repository[4] for more details). Nevertheless, for the sake of reproducibility, one can find in Appendix A the parameters that will allow to reproduce the results presented in Section 4 (training and testing).
Hereafter, two main aspects of the implementation will be investigated :

- How one can make quick test of the method.
- How one can improve the code.

[2] https://pypi.org/project/SegSRGAN
[3] https://www.tensorflow.org/install/gpu
[4] https://github.com/koopa31/SegSRGAN

3.2.1. The makefile: a solution to quickly run the provided code

One can find in the folder "Test_package_and_local" of the GitHub repository[4], a Makefile allowing one to perform quick tests of the method. The Python version with which one can make the test can be modified in the provided Makefile by changing the value of the $python variable (default value: 3.7).
The only needed external dependency is therefore the Python package "virtualenv" which can be installed using the following command:

```
pip install virtualenv
```

All the implemented commands can be executed in command line, once the working directory of the terminal is set to the Test folder:

```
cd /path_to_SegSRGAN_folder/Test_package_and_local
```

Then, one can install the package dependencies, as follows:

```
make create_venv_and_install_SegSRGAN
```

This command creates a virtual environment and installs all the necessary dependencies. All commands, explained below, use the Python interpreter of this virtual environment ("venv_x" folder where x is the Python version).
Finally, the makefile allows one to test several applications of the provided implementation:

1. testing task (prediction) using the provided trained weights on command line;
2. training task from scratch.

The last requirement before carrying out tests is to add images in the corresponding folders: "Image_for_testing" for testing tasks and "Image_for_training" for training ones.

Testing. For testing task, one need to place one or many images in the "Image_for_testing folder". Each image need to be placed it in own folder. The computed result will be saved in a specific folder, created during the script execution.
Then, one can use the following command to test on the images in the "Image_for_testing" folder as follows:

```
make test_job_model_on_image_absolute
```

Training. For training task, the corresponding folder "Image_for_training" contains two folders named "Label" and "HR". The training test provided has been designed for training on a sample size of two images (two HR and segmentation maps). The two HR images need to be placed in the "HR" folder whereas the two segmentation maps need to be placed in the "Label" folder.
Then, one can use the following command to test on the images in the "Image_for_testing" folder as follows:

```
make test_training
```

Note that the default parameters for these tests have been chosen in order to ensure acceptable computation times (even with CPUs), but not to obtain the best results.

3.2.2. Scalability and flexibility

In order to make the implementation scalable, the code is organized in multiple files, classes and functions. A summary of the code organization is presented below.
In the first level of the package folder (namely "SegSRGAN/SegSRGAN") are located the files which can be directly applied to perform a defined task:

- Function_for_application_test_python3.py: provides a Python function for predicting an image;
- job_model.py: provides a command line executable file for predicting multiple images. This file provides an overlay of Function_for_application_test_python3.py;
- SegSRGAN_training.py: provides a command line executable file for training a SegSRGAN model.

In the second level of the package folder (namely "SegSRGAN/SegSRGAN/utils") are located all the functions used by the three functions presented above.
In the package, one also finds some files / classes which provide a certain scalability to the proposed implementation. The SegSRGAN, ImageReader, normalization and interpolation Python files have been designed for such purpose. We now give a short description of these files.

utils/SegSRGAN.py. Used in Function_for_application_test_python3.py (for testing task) and SegSRGAN_training.py (for training task), this file contains a class containing all the information about the generator and discriminator architecture. All the functions with a name containing "generator_block" or "discriminator_block" correspond to the implementation of the architecture of the generator and the discriminator, respectively. Thanks to the use of functions, one can easily implement a new architecture (for the discriminator and/or generator) to replace the provided ones.

utils/ImageReader.py. Used in "utils/-patches.py" (for training task) and "Function_for_application_test_python3.py" (for testing task), this file contains all the image readers implemented. Each image reader corresponds to a class, whereas an abstract class exemplifies how to implement a new one.

utils/interpolation.py. Used in "utils/-patches.py" (for training task) and "Function_for_application_test_python3.py" (for testing task), this file contains a class which has been designed to contain the interpolation methods. Each image interpolation method corresponds to a specific function.

utils/normalization.py. Used in "utils/-patc'hes.py" (for training task) and 'Function_for_application_test_python3.py" (for testing task), this file contains a class which has been designed to contain the normalization and inverse normalization methods (to put the result image in the same range as the LR image in test).

4. Results

4.1. Data

4.1.1. Datasets

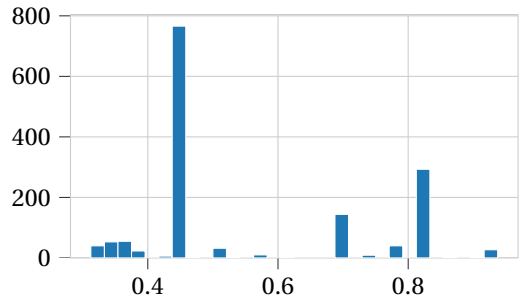
We work on two MRI datasets, namely dHCP[5] [14], and the French Epirmex[6] dataset whose specificities are detailed in Table 1.
In particular, the main differences between Epirmex and dHCP datasets are the following:

- the LR images of dHCP are generated;
- dHCP has real segmentation maps and HR images (both with axial resolution of 0.5 mm) whereas Epirmex does not;

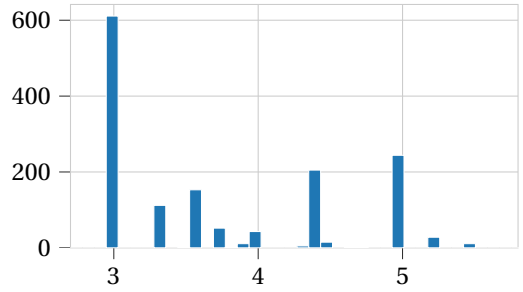
[5] http://www.developingconnectome.org
[6] Epirmex is part of the French epidemiologic study Epipage 2 [1], http://epipage2.inserm.fr.

	dHCP	Epirmex
Number of images	40	1500
Coronal resolution (= Sagittal resolution)	0.5 mm	heterogeneous, mainly 0.45 mm. Histogram of distribution in Fig. 3(a)
Axial resolution	0.5 mm (which implies the term of HR image)	heterogeneous, mainly 3 mm (which implies the term of LR image). Histogram of distribution in Fig. 3(b)
HR	Yes	No
LR	No, generated downsampling HR from 0.5 mm to 3 mm of axial resolution	Yes
Label (segmentation ground-truth)	Yes	No
Acquisition machine	3T Achieva scanner	Various machines from different French hospitals
TR/TE	1200/156 ms	Various, sometimes very different from dHCP (great heterogeneity in contrast) see Fig. 4

Table 1: Comparison of dHCP and Epirmex datasets.



(a) Coronal and sagittal (isotropic) resolution (mm)



(b) Axial resolution (mm)

Figure 3: Histograms of the image resolution for the MR images from Epirmex: (a) coronal and sagittal; (b) axial.

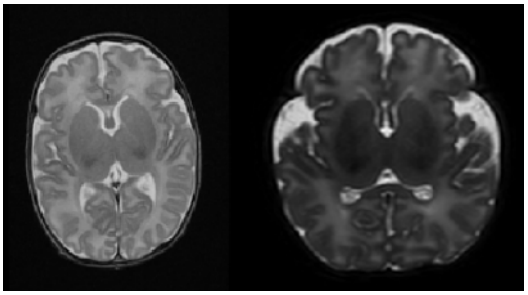


Figure 4: Two examples of MR images (axial slices) from the Epirmex dataset. One can observe the differences in terms of contrast.

- the TR and TE with which the images have been acquired are fixed on dHCP but strongly vary on Epirmex (which induces the motivation to study the training on augmented data modifying the contrast);

- some Epirmex images are noisy, whereas dHCP are not (which induces the motivation to study the training on augmented data by adding random noise).

Therefore, in addition to using dHCP for training the model, using it also for testing is relevant since it allows us to quantitatively evaluate the result of the method. However, applying the method on Epirmex also makes sense since the LR images are directly acquired. In particular, applying the method on these two datasets can provide us with complementary information.

4.1.2. Preprocessing: LR image generation (dHCP)

The dHCP dataset is composed of HR images equipped with binary segmentation (ground-truth), in particular for the cerebral cortex at the same high resolution. As a consequence, in order to train a super-resolution model, we need to determine corresponding LR images for these couples HR images / segmentation maps.

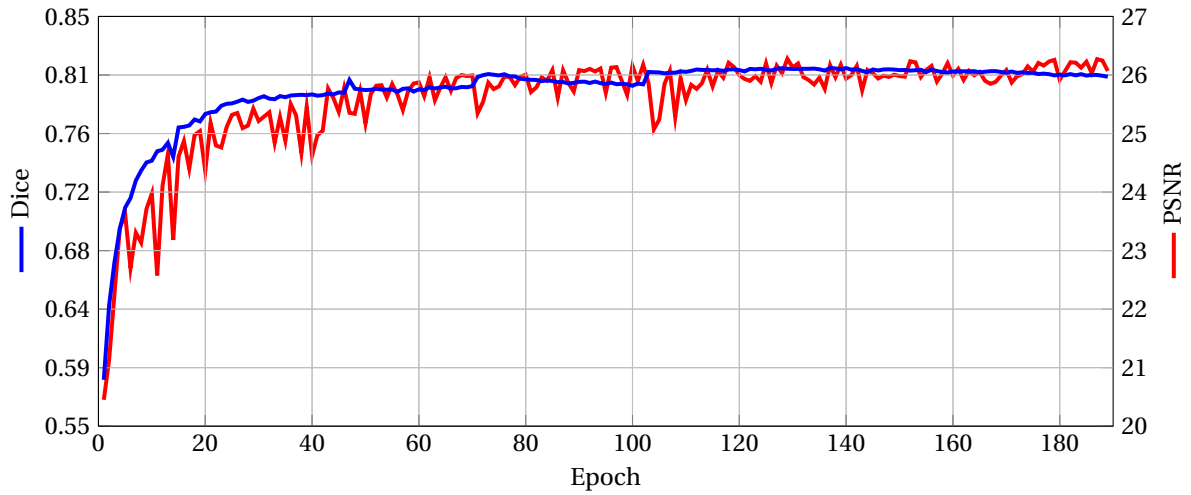


Figure 5: Dice and PSNR evolution during the training on the dHCP dataset.

From a given HR image X, the associated LR image X_{LR} is generated using the following model, as proposed in [10]:

$$X_{LR} = H_{\downarrow} B X \tag{13}$$

where B is a blur matrix and $H_{\downarrow}$ is a downsampling decimation. In particular, we consider a Gaussian filter B with a standard deviation:

$$\sigma = \frac{\text{res}}{2\sqrt{2\log 2}} \tag{14}$$

where res is the resolution of the HR image.
The generated LR images considered in our experimental study are of resolution $0.5 \times 0.5 \times 3$ mm^3, which is compliant with true clinical data, which are usually strongly anisotropic, as emphasized by the above Epirmex data description.

4.2. Quality metrics

Segmentation. In order to assess the quality of segmentation results, we consider the Dice score [8], which is a standard measure for that purpose. In particular, the adequacy of the computed segmentation S with respect to the ground-truth G is then given as:

$$\text{Dice}(S,G) = \frac{2|S \cap G|}{|S| + |G|} \tag{15}$$

and lies in $[0,1]$. The closer the Dice score to 1, the better the correlation between S and G.
In addition to the quantitative information carried by the Dice score, we also consider the number of connected components of the segmented results, noted NCC. By assuming that the cortex is a connected object, NCC provide structural information: the higher this value, the lower the topological quality of the object.

Super-resolution. Measuring the performance of SR algorithms is less straightforward. Indeed, for gauging the visual aspect similarity between two images, a distance between the intensity of the SR and HR voxels may not be sufficient. The performance of SR reconstruction is then measured by two different indices, namely the PSNR and the SSIM [39], defined respectively as:

$$\text{PSNR}(X,Y) = 10\log_{10} \frac{\left(\max_i X_i\right)^2}{\frac{1}{|X|}\sum_i |X_i - Y_i|^2} \tag{16}$$

where X_i and Y_i are the values of X and Y at point i, respectively, and:

$$\text{SSIM}(X,Y) = \frac{(2\mu_X\mu_Y + c_1) + (2\sigma_{XY} + c_2)}{(\mu_X^2 + \mu_Y^2 + c_1)(\sigma_X^2 + \sigma_Y^2 + c_2)} \tag{17}$$

where μ_X (resp. μ_Y) is the mean of X (resp. Y) values, σ_X (resp. σ_Y) is the standard deviation of X (resp. Y) values, σ_{XY} is the covariance between X and Y values, and c_1, c_2 are numerical stabilizers quadratically linked to the dynamics of the image. For both PSNR and SSIM indices, the higher the value, the better the similarity.

4.3. Convergence – Training (dHCP)

First, we observe the evolution the Dice and PSNR scores, along the training. The values presented here

	SegSRGAN	IMAPA	DrawEM
Dice	0.855(±0.014)	0.786(±0.023)	0.730(±0.010)

Table 2: Dice scores mean value (and standard deviation) for segmentation map. Calculated over 8 dHCP images

	SegSRGAN	Cubic spline interpolation
PSNR	26.96	24.22
SSIM	0.73	0.63

Table 3: PSNR and SSIM mean value. Calculated over 8 dHCP images

are calculated as follows. The test images are split into patches of size 64^3 voxels, using a 20 voxel shifting (the same used for the training image set). The PSNR and the Dice scores are then calculated in each patch. The final PSNR and the Dice scores are obtained by averaging the values computed patch-wise.
The results are depicted in Figure 5, that provides the evolution of the Dice and the PSNR scores at the end of each epoch. The initial Dice has a very low value, close to 0.5, and then increases up to 0.8 rather smoothly. It seems to converge from the 100^{th} epoch. The PSNR also converges, but in a more noisy way. However, the size of the peaks progressively decreases whereas the score tends to stabilize, following the same behaviour as the Dice score.

4.4. Results – Testing (dHCP)

The results presented here were computed from the 8 images from dHCP used as testing dataset (see Appendix A.4) . These results were obtained with patches of size 128^3 voxels, with a 30-voxel shifting.
Table 2 summarizes the Dice scores of SegSRGAN, IMAPA [35] and DrawEM [24]. As in a typical clinical settings, all these methods have been applied on interpolated images (using cubic spline). From this table, one can see that, quantitatively, the proposed approach leads to the best cortical segmentation results with significant improvement with respect to the two other methods. Moreover, as mentioned in [35], the use of IMAPA applied on original HR dHCP images leads to a mean Dice score of 0.887 (standard deviation of 0.011). Finally, the result obtained on interpolated images, only decreases by 3% compared to IMAPA applied on HR images.
Table 3 summarizes PSNR and SSIM of SegSRGAN and cubic spline interpolation. One can observe that the two quality scores for the SR image reconstruction exhibit better results with SegSRGAN than with cubic spline interpolation (which constitutes a standard baseline and is the input of the network).

4.4.1. Impact of patch overlapping

Impact on the computation time. The time cost of the algorithm is directly influenced by the choice of the step (that controls patch overlapping). Indeed, the value of step directly implies the number of patches which have to pass through the neural network. Actually, this number of patches depends directly on the value

$$\prod_{\star\in\{x,y,z\}} \frac{n_\star - patch_\star}{step} \tag{18}$$

where (n_x, n_y, n_z) is the size of the smallest image containing the interpolated image and which allows to create an integer number of full patches of size $(patch_x, patch_y, patch_z)$ with $step$ between the successive patches.
From Equation (18), it is plain that the number of patches is calculated from a quantity that evolves like $step^{-3}$. Furthermore, even though (n_x, n_y, n_z) also depends on the step, one can approximate the computation time by an affine function of $step^{-3}$, as confirmed by Figure 6.
The computation times presented in this figure were obtained in the following configuration:

- patch size of 128^3;
- on dHCP, which implies that all the HR images which led to the interpolated images are of size $(290, 290, 198)$;
- computation on Tesla P100 GPU and 2 Intel® Xeon™ Gold "Skylake" 6132 CPU.

Here, the computation time varies from 600 seconds (10 min) for a step value of 10, to 4 seconds for a step value of 100, with a $step^{-3}$ decrease between these two values. It is worth mentioning that the device used here processes the patches quickly (approximately 5 patches per second) which tends to reduce the impact of the variation of (n_x, n_y, n_z); this could not be the same on other kinds of devices.

Impact on algorithm performance. We now investigate the impact of the step value on the algorithm performance.
For segmentation purpose, the Dice scores obtained on dHCP are provided in Figure 7(a). Here, the patches are still of size 128^3 voxels. We observe that in any cases, overlapping between patches (i.e. with steps lower than 128) provides better results than without (i.e. with a step of 128). In overlapping cases,

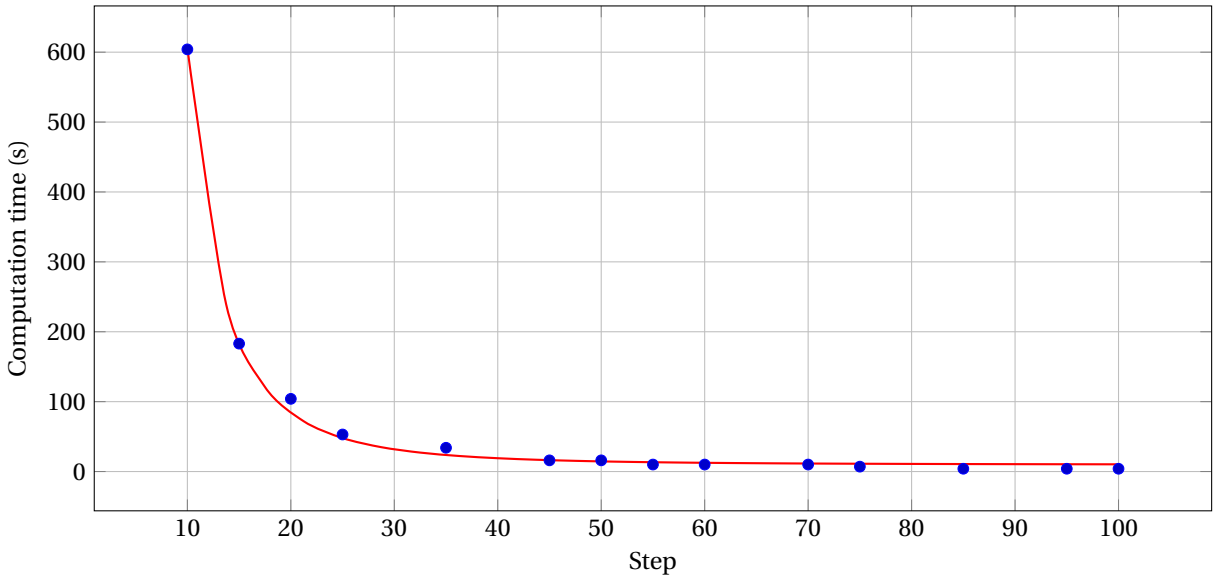


Figure 6: Computation time (in seconds) against the step value.

the Dice score grows roughly linearly with respect to the size of the step, for reaching values around 0.86 with maximal overlaps, i.e. for steps close to 1.
This behaviour for segmentation is confirmed by the behaviour for SR reconstruction, as illustrated by Figure 7(b–c), which provides the PSNR and SSIM scores with respect to the step.
The cortex is a continuous ribbon-like structure. In particular, the connectedness of the segmentation results provided by SegSRGAN is a relevant property, in complement to the three considered quantitative scores. In Figure 8, we show how the value of step between patches influences the number of connected components of the segmented cortex. In theory, only one connected component should be obtained. In other words, obtaining n connected components means that $n-1$ parts of the segmented cortex are erroneously disjoint from the topologically correct structure. Once again, one can observe a linear correlation between the error and the value of the step, with close to 1 values for the higher overlapping / lower steps.

4.4.2. Impact of noise

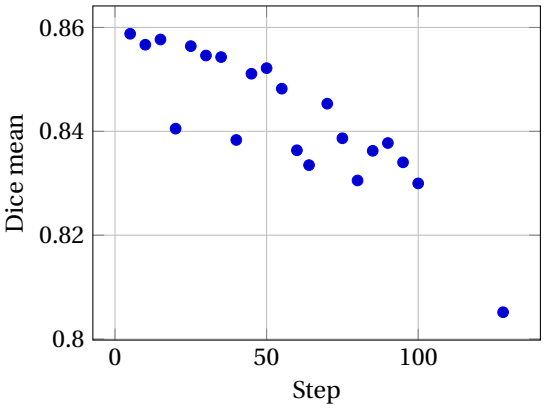
MR images are generally affected by noise. We now investigate in which extent the performance of the segmentation and SR reconstruction is impacted by adding Gaussian noise to the image. In practice, a Gaussian noise with $\sigma = 2$ is added to each voxel of the 8 test MRI images.

Table 4: Mean SSIM scores for SR reconstruction on noisy and non-noisy images (dHCP).

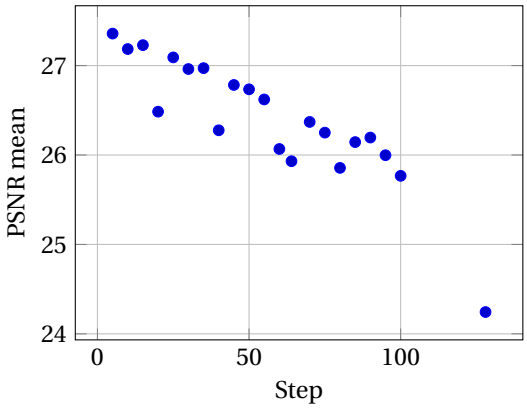
Step	Non-noisy	Noisy
30	0.73	0.49
80	0.69	0.46
128	0.65	0.44

The difference between the Dice scores of the segmentation results with and without noise, respectively, is depicted in Figure 9. As expected, the relative Dice scores are slightly better for non-noisy images than for noisy ones, with a gap within $[0.03, 0.07]$. (Note that from a topological point of view, we also observed that the number of connected components is approximately two times greater in the noisy images compared to non-noisy ones.) We observe that the difference between noisy and non-noisy data increases (i.e. the impact of adding noise increases) when the step grows. One can then conclude that, the larger the step (i.e. the lower the patch overlapping), the lower the robustness of the method versus noise.

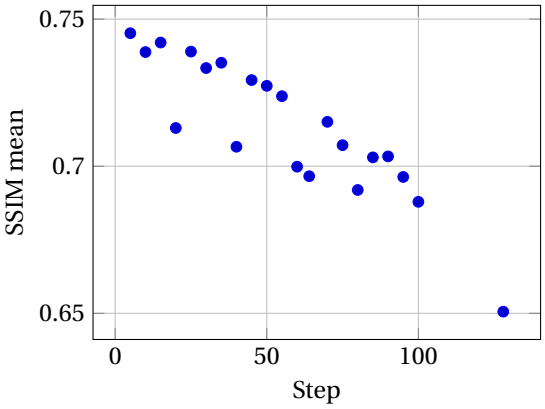
In terms of SR reconstruction, we also compared the evolution of SSIM and PSNR with respect to the step value. The results are given in Tables 4–5. They emphasize a strong degradation of SSIM, and a much lower degragation of PSNR. As already observed for segmentation results, when increasing patch over-



(a) Dice



(b) PSNR



(c) SSIM

Figure 7: (a) Dice scores of segmentation results on dHCP, depending on the step between successive patches of size 128^3 voxels. PSNR (b) and SSIM (c) scores of SR reconstruction results on dHCP, depending on the step between successive patches of size 128^3 voxels.

lapping, both scores are significantly improved. For SSIM, the gain between steps sizes of 128 and 30 is of

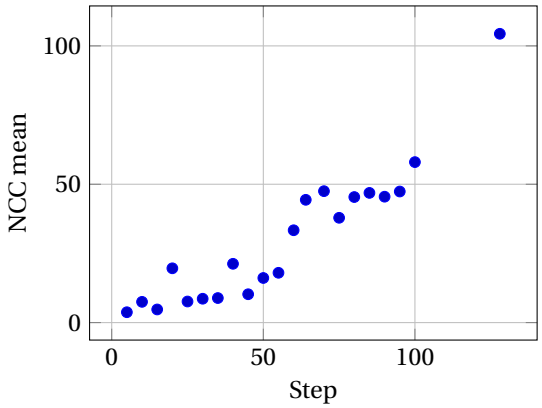


Figure 8: Number of connected components (NCC) of the segmented cortex, depending on the step between successive patches of size 128^3 voxels; calculated on dHCP database.

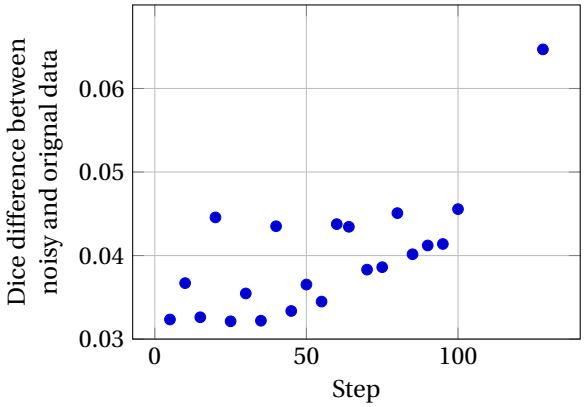


Figure 9: Difference between the Dice scores with non-noisy and noisy images from dHCP image. Positive values mean that the Dice scores are higher without than with Gaussian noise.

Table 5: Mean PSNR scores for SR reconstruction on noisy and non-noisy images (dHCP).

Step	Non-noisy	Noisy
30	26.96	25.40
80	25.85	24.45
128	24.24	23.24

approximately 10% in both noisy and non-noisy cases. For PSNR, it is slightly lower (approximately 8%) with than without noise (approximately 10%).

4.4.3. Impact of data augmentation

Motivation. The experiments described until now have been carried out on dHCP, i.e. an image dataset

designed for research purpose, with good properties in terms of noise and signal homogeneity. In real, clinical cases, the images forming a dataset are generally of lower quality both in terms of signal-to-noise ratio and signal homogeneity. For instance, all the images in dHCP were acquired with the same TR and TE (implying inter-image signal homogeneity); by contrast the TR and TE in Epirmex can strongly vary. Moreover, by contrast with research-oriented datasets, clinical data are generally not equipped with ground-truths; indeed, defining such ground-truths is a complex and time-consuming task that requires heavy work by medical experts. (The same remark also holds for the non-availability of HR images associated to the native LR ones.) Although our final purpose is to be able to process real, clinical data, it is generally not possible to perform the training on comparable images. Such training then need to rely on dHCP-like data. As a consequence, it is indeed relevant to consider data augmentation strategies for increasing the ability of the trained networks to take into account real noise and inhomogeneity properties of usual MR images.

Data augmentation. The considered data augmentation on dHCP images consists of applying contrast modification uniformly drawn in $[0.5, 1.5]$, and adding Gaussian noise with a standard deviation set to 3% of the highest value.
Two questions then arise: (1) Does this training improve the result on noisy / signal-biased data? (2) Does this training make the performance decrease on the original images? In order to answer these questions, three supplementary families of images, created from the original database, are considered during the testing, namely images:

- with additive Gaussian noise ($\sigma = 2$);
- with values increased by a square function;
- with values decreased by a square root function.

Hereafter, these three types of images are denoted as "augmented test dataset".

Segmentation. The results obtained for segmentation purpose are depicted in Figure 10. First, one can observe that in case of large steps, data augmentation improves more significantly the segmentation results (the best example is when the algorithm is applied without overlapping). However, it is important to keep in mind that the Dice score decreases with respect to the step. In other words, data augmentation allows, in a certain extent, to compensate the defects

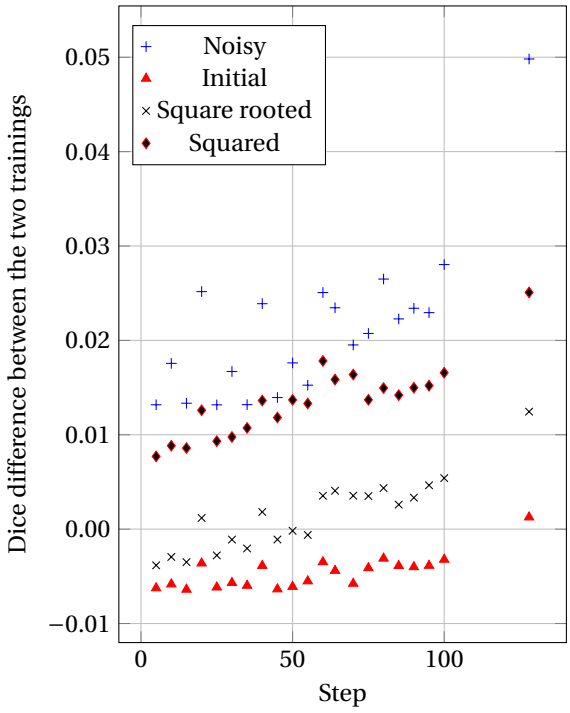


Figure 10: Difference between the mean Dice scores of segmentation results when training with and without data augmentation, depending on the step value. Positive (resp. negative) values mean that the Dice scores are better with (resp. without) data augmentation. Calculated on dHCP database

caused by the choice of a large step. In the case of overlapping, the value of step seems to have a negligible quantitative impact on the improvements. In most cases, data augmentation allows to slightly increase the robustness of segmentation results, with Dice scores increase between $[0.00, 0.03]$, when considering augmented test datasets. This is not the case for the native dHCP dataset, where the Dice scores are slighly lowered by data augmentation training. Indeed, data augmentation allows the network to learn from a wider range of data; making it more robust with respect to data variability. As a counterpart, such a versatile network becomes less robust for handling a homogeneous population of data such as dHCP.
In complement to the relative evaluation provided by Figure 10, we also propose, in Table 6, an absolute comparison of the Dice scores between segmentation performed with and without data augmentation, both on the dHCP images and their noisy versions. One can observe that the segmentation results for original dHCP images are slightly degraded (Dice decrease lower than 0.01) with data augmentation, with reasons similar to those evoked in the analysis of Fig-

Table 6: Mean values of Dice scores for segmentation of dHCP images (noisy and non-noisy) based on standard or data-augmented training.

	Original			Noisy		
Step	30	80	128	30	80	128
Without data augmentation	0.855	0.831	0.805	0.819	0.785	0.740
With data augmentation	0.849	0.827	0.806	0.836	0.812	0.790

Table 7: Mean number of connected components for segmentation of dHCP images (noisy and non-noisy) based on standard or data-augmented training.

	Original			Noisy		
Step	30	80	128	30	80	128
Without data augmentation	8.6	45.3	104.3	26.6	98.5	193.1
With data augmentation	5.7	33.6	61.7	5.8	33.5	61.8

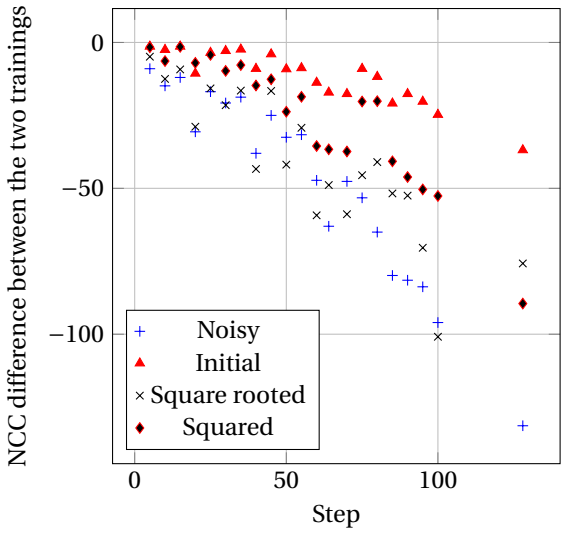


Figure 11: Difference between the number of connected components of segmentation results when training with and without data augmentation, depending on the step value. Negative values mean that the number of connected components is lower with data augmentation. Calculated on dHCP database.

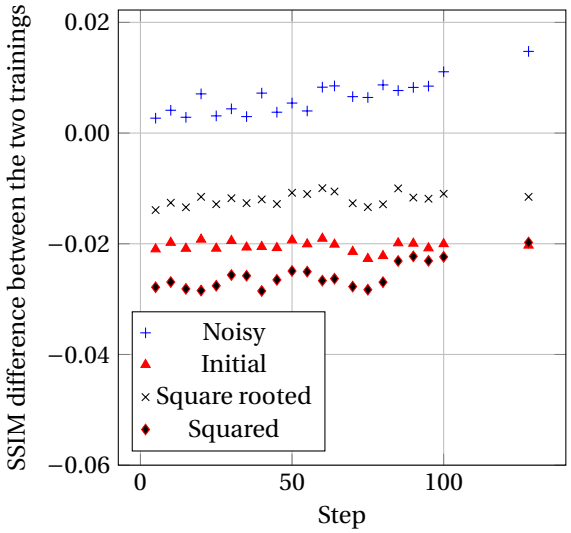
ure 10. In the meantime, the segmentation results for noisy images are improved (Dice increase from 0.015 to 0.050). In particular, in the case of data augmentation, the difference between Dice scores on standard and noisy images becomes low (approximately 0.015), compared to the case without data augmentation (approximately 0.030 to 0.060). In other words, data augmentation makes the segmentation more robust to the tested images, at the price of a loss of specialization with respect to the trained images. However, such trade-off is relevant with respect to real clinical applications.

In order to complete this quantitative segmentation analysis, we also propose, in Figure 11 and Table 7, a topological analysis of the results, by providing the mean number of connected components of the segmented cortex. These topological results strengthen the conclusions obtained for Dice analysis on augmented data. In particular, on original data, an improvement is also observed from this topological point of view. Indeed, the number of connected components is reduced by data augmentation for all types of data studied. In particular, the improvement is important for noisy images. In addition, (see Table 7) as for Dice analysis, it appears that segmentation results present very similar topological properties independently of the noise level of the input images.

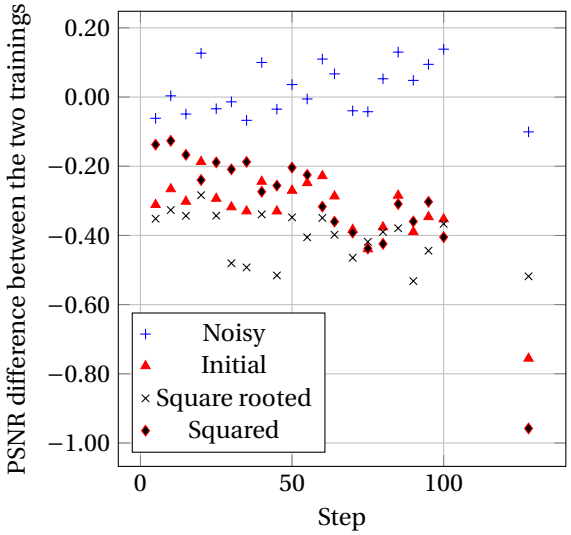
SR reconstruction. Concerning SR reconstruction, when observing Figures 12(a) and 12(b), it appears that for both SSIM and PSNR, the training with data augmentation (noise and contrast variation) slightly decreases the quality of reconstruction on initial images. This behaviour, already observed in the case of segmentation can be explained by the same factors. However, we also observe that the reconstruction is slightly improved for noisy data, whereas it is slightly degraded for contrast-modified data. These results on noisy images corroborate those of segmentation, whereas they are antagonistic in the case of contrast variation. This last point would require to more accurately investigate the specific effects of data augmentation with respect to either noise and contrast variation.

4.5. Results – Testing (Epirmex)

In this last part of the experimental section, we apply our SR reconstruction and segmentation method to the MR images from the Epirmex dataset.



(a) SSIM



(b) PSNR

Figure 12: SSIM (a) and PSNR (b) difference of SR reconstruction between the training with and without data augmentation. Positive (resp. negative) values mean that the metric (PSNR or SSIM) is higher with (resp. without) data augmentation. Calculated on dHCP database

Epirmex contains more than 1500 images, with very different properties, e.g. the TE and TR values, as discussed above (see Figure 4). These MR images are also of low resolution and a strong anisotropy (see Figure 3).

In addition, Epirmex is endowed neither with ground-truth nor with HR images associated to the LR ones, with two main side effects. The first is that the training cannot be carried out on Epirmex images, and it is then mandatory to learn from another dataset, with a data augmentation paradigm; this is what we did with dHCP. Second, there is no objective way of assessing the quality of the results, both in terms of segmentation and SR reconstruction. As a consequence, the results provided hereafter have mainly a qualitative value.

Step	30	50	80	128
NCC with data augmentation	30	35	96	387
NCC without data augmentation	36	53	102	720

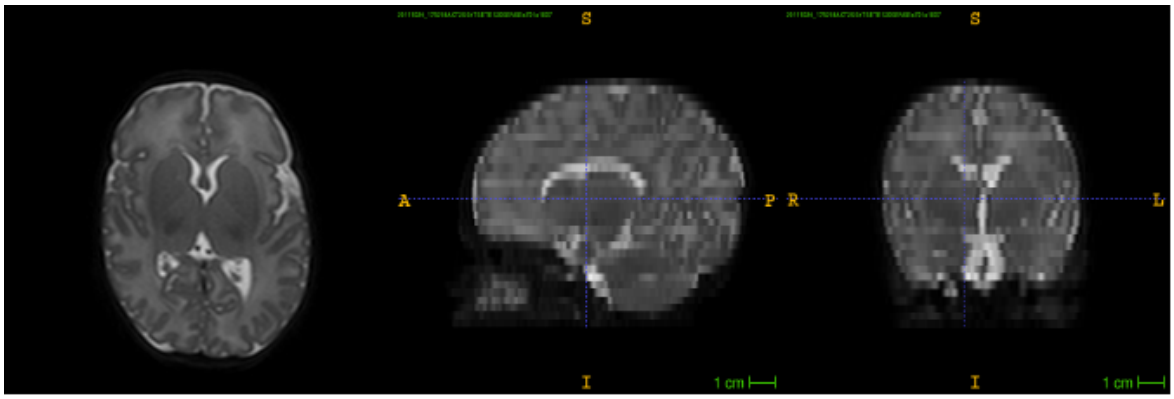
Table 8: Mean number of connected components (NCC) computed from 7 Epirmex images.

In particular we propose, in Figure 13, some SR reconstruction and cortex segmentation results for an MR image representative of the data within Epirmex. Those results were obtained from the parameters learned during the training on dHCP, with data augmentation. A visual analysis of these results leads to satisfactory conclusions. In particular, the segmented cortex is geometrically consistent, with a good visual correlation to the input MR image. Regarding the SR reconstruction, the contrast between the cortex and the surrounding tissues seems higher in the SR image than in the LR one, even in the axial slices, where the resolution is however not modified between LR and SR. Globally, these qualitative experiments on Epirmex tend to corroborate the quantitative ones carried out on dHCP, and to suggest that the proposed method is indeed relevant both for reconstruction and segmentation in the context of real, clinical data analysis.

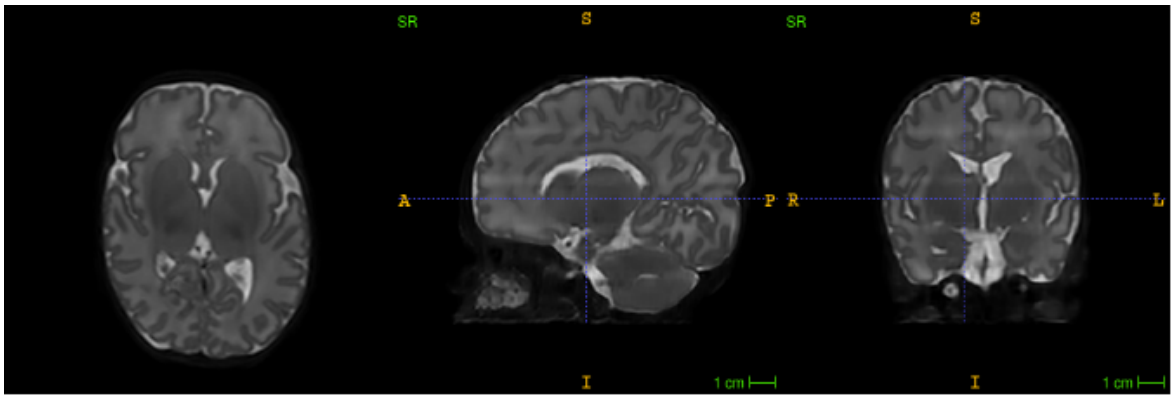
Due to the lack of ground-truth both for segmention and SR reconstruction, we can rely neither on Dice nor on SSIM and PSNR for assessing the quality of the results on Epirmex. From a topological point of view, it is however possible to assess the structural quality of the segmented cortex, that is assumed to be fully connected.

In particular, the results stated in Table 8 confirm those previously obtained on dHCP (see Table 7).

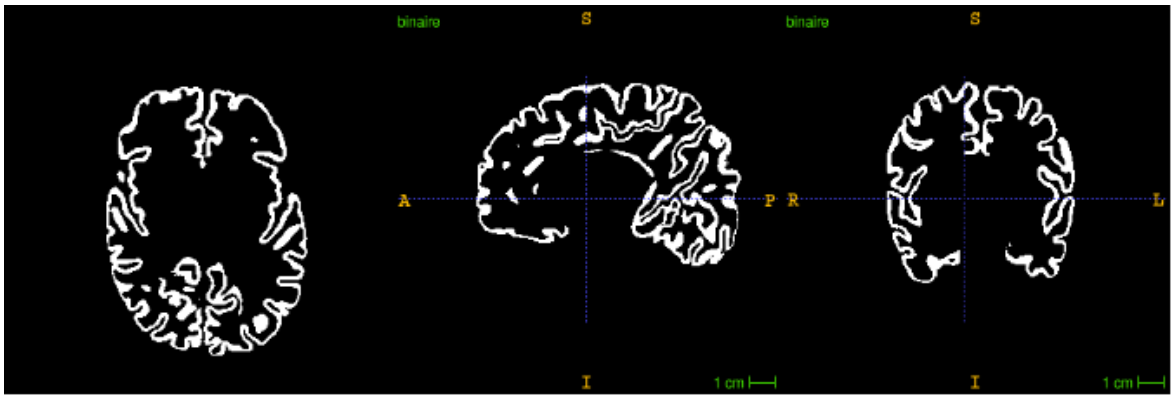
Indeed, one can observe that, in case of patch overlapping, the number of connected components is better when training with data augmentation. However, the highest difference between these two trainings occurs when the algorithm is used without patch overlapping (i.e. with a step of 128).



(a) LR (initial) MR image



(b) SR (reconstructed) MR image



(c) Segmented image (cortex)

Figure 13: (a) LR image from the Epirmex dataset. (b) SR reconstructed image obtained from (a). (c) Segmentation of the cortex obtained from (a).

Similarly to the experimental results obtrained on dHCP, the evolution of the number of connected components with respect to the step value argues in favour of reducing as much as possible the step for improving the topological coherence of the segmented cortex. In particular, on Epirmex, the sensitivity of NCC to the number of steps is higher than with dHCP. This can be easily explained by the lower quality of the data, that tends to induce erroneous disconnections, partially avoided by increasing patch overlapping.

5. Discussion

In this article, we have proposed a new methodological and software solution for performing SR reconstruction and segmentation from complex 3D MR images. Our framework, based on generative adversarial networks has been described in details, both from theoretical and technical points of view. In particular, a free, documented software version is available, for dissemination to the scientific and clinical communities and for the sake of reproducibility of the results.
Although our purpose was not to carry out a comparative work with other methods (we do not claim the superiority of our method, but its usefulness in certain contexts), we have proposed a consequent experimental analysis in the case of cortex investigation in the neonate. This experimental analysis, carried out on both research and clinical datasets, and from qualitative and quantitative points of view, tend to prove the relevance of our approach, with satisfactory results both in terms of SR reconstruction and segmentation.
In particular, the ability of the method to take advantage of data-augmentation strategies allows one to involve it for SR reconstruction and segmentation of clinical datasets, which are generally not endowed with ground-truth and/or examples of high-resolution MR images associated to LR ones. In this context, it was observed that carrying out a learning procedure on a research dataset, whereas degrading the data (e.g. in terms of noise and intensity bias) constitutes a tractable approach.
Of course, the proposed method is not specific to the cortex, and it could also be used for other kinds of cerebral structures in MRI. The segmentation results obtained on the cortex are indeed satisfactory. However, since the segmentation and SR reconstruction are handled in a common way, it is possible that the quality of SR reconstruction may be impacted by the structure of interest targeted for segmentation. In other words, it is possible that the SR reconstruction may be more efficient in the neighbourhood of the cortical ribbon, compared to other loci in the brain. A way of tackling this issue would be to consider a more global segmentation purpose, involving the main cerebral tissues in parallel, leading to a more global guidance of SR reconstruction by the information carried by segmentation. In theory, this is indeed possible, since the method architecture can allow multi-segmentation, whereas research datasets are indeed equipped with complete ground-truth defined as atlases.
Regarding perspective works, from a methodological point of view, we will more deeply investigate the trade-off between computational cost and result quality, in particular with respect to data variability within MR image cohorts. In this context, we will also experimentally assess the various consequences on data augmentation with respect to usual features likely to degrade MR image quality, namely noise, signal bias, contrast heterogeneity, or movement artifacts. From a more applicative point of view, our next purpose will consist of processing large image cohorts for automatically extracting regions of interest, in a clinical context.

Acknowledgements

The research leading to these results has been supported by the ANR MAIA project, grant ANR-15-CE23-0009 of the French National Research Agency (http://recherche.imt-atlantique.fr/maia); INSERM and Institut Mines Télécom Atlantique (*Chaire "Imagerie médicale en thérapie interventionnelle"*); the *Fondation pour la Recherche Médicale* (grant DIC20161236453); and the American Memorial Hospital Foundation. We also gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan Xp GPU used for this research.

References

[1] P.-Y. Ancel, F. Goffinet, and EPIPAGE 2 Writing Group. EPIPAGE 2: A preterm birth cohort in France in 2011. *BMC Pediatrics*, 14:97, 2014.

[2] G. Ball, J. P. Boardman, P. Aljabar, A. Pandit, T. Arichi, N. Merchant, D. Rueckert, A. D. Edwards, and S. J. Counsell. The influence of preterm birth on the developing thalamocortical connectome. *Cortex*, 49(6):1711–1721, 2013.

[3] G. Ball, L. Srinivasan, P. Aljabar, S. J. Counsell, G. Durighel, J. V. Hajnal, M. A. Rutherford, and A. D. Edwards. Development of cortical microstructure in the preterm human brain. *Proceedings of the National Academy of Sciences of the United States of America*, 110(23):9541–9546, 2013.

[4] M. J. Cardoso, A. Melbourne, G. S. Kendall, M. Modat, N. J. Robertson, N. Marlow, and S. Ourselin. AdaPT: An adaptive preterm segmentation algorithm for neonatal brain MRI. *NeuroImage*, 65:97–108, 2013.

[5] P. Charbonnier, L. Blanc-Féraud, G. Aubert, and M. Barlaud. Deterministic edge-preserving regularization in computed imaging. *IEEE Transactions on Image Processing*, 6(2):298–311, 1997.

[6] Y. Chen, Y. Xie, Z. Zhou, F. Shi, A. G. Christodoulou, and D. Li. Brain MRI super resolution using 3D deep densely connected neural networks. In *ISBI, Proceedings*, pages 739–742, 2018.

[7] P. Coupé, J. V. Manjón, V. Fonov, J. Pruessner, M. Robles, and D. L. Collins. Patch-based segmentation using expert priors: Application to hippocampus and ventricle segmentation. *NeuroImage*, 54:940–954, 2011.

[8] L.R. Dice. Measures of the amount of ecologic association between species. *Ecology*, 26:297–302, 1945.

[9] J. Dubois, M. Benders, A. Cachia, F. Lazeyras, R. Ha-Vinh Leuchter, S. V. Sizonenko, C. Borradori-Tolsa, J. F. Mangin, and P. S. Hüppi. Mapping the early cortical folding process in the preterm newborn brain. *Cerebral Cortex*, 18(6):1444–1454, 2008.

[10] H. Greenspan. Super-resolution in medical imaging. *The Computer Journal*, 52(1):43–63, 2008.

[11] L. Gui, R. Lisowski, T. Faundez, P. Hüppi, F. Lazeyras, and M. Kocher. Morphology-based segmentation of newborn brain MR images. In *MICCAI NeoBrainS12, Proceedings*, pages 1–8, 2012.

[12] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville. Improved training of Wasserstein GANs. In *NIPS, Proceedings*, pages 5769–5779, 2017.

[13] M. Hack and A. A. Fanaroff. Outcomes of children of extremely low birthweight and gestational age in the 1990s. *Seminars in Neonatology*, 5:89–106, 2000.

[14] E. Hughes, L. Cordero-Grande, M. Murgasova, J. Hutter, A. Price, A. Santos Gomes, J. Allsop, J. Steinweg, N. Tusor, J. Wurie, J. Bueno-Conde, J.-D. Tournier, M. Abaei, S. Counsell, M. Rutherford, M. Pietsch, D. Edwards, J. Hajnal, S. Fitzgibbon, E. Duff, M. Bastiani, J. Andersson, S. Jbabdi, S. Sotiropoulos, M. Jenkinson, S. Smith, S. Harrison, L. Griffanti, R. Wright, J. Bozek, C. Beckmann, A. Makropoulos, E. Robinson, A. Schuh, J. Passerat Palmbach, G. Lenz, F. Mortari, T. Tenev, and D. Rueckert. The Developing Human Connectome: Announcing the first release of open access neonatal brain imaging. In *OHBM, Proceedings*, 2017.

[15] J. Johnson, A. Alahi, and L. Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *ECCV, Proceedings*, pages 694–711, 2016.

[16] H. Kim, C. Lepage, A. C. Evans, J. Barkovich, and D. Xu. NEOCIVET: Extraction of cortical surface and analysis of neonatal gyrification using a modified CIVET pipeline. In *MICCAI, Proceedings*, pages 571–579, 2015.

[17] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[18] W.-S. Lai, J.-B. Huang, N. Ahuja, and M.-H. Yang. Deep Laplacian pyramid networks for fast and accurate super-resolution. *CoRR*, abs/1704.03915, 2017.

[19] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi. Photo-realistic single image super-resolution using a generative adversarial network. In *CVPR, Proceedings*, pages 105–114, 2017.

[20] J. Lefèvre, D. Germanaud, J. Dubois, F. Rousseau, I. de Macedo Santos, H. Angleys, J.-F. Mangin, P. S. Hüppi, N. Girard, and F. De Guio. Are developmental trajectories of cortical folding comparable between cross-sectional datasets of fetuses and preterm newborns? *Cerebral Cortex*, 26(7):3023–3035, 2016.

[21] P. Luc, C. Couprie, S. Chintala, and J. Verbeek. Semantic segmentation using adversarial networks. *CoRR*, abs/1611.08408, 2016.

[22] D. Mahapatra. Skull stripping of neonatal brain MRI: Using prior shape information with graph cuts. *Journal of Digital Imaging*, 25:802–814, 2012.

[23] A. Makropoulos, S. J. Counsell, and D. Rueckert. A review on automatic fetal and neonatal brain MRI segmentation. *NeuroImage*, 170:231–248, 2017.

[24] A. Makropoulos, I. S. Gousias, C. Ledig, P. Aljabar, A. Serag, J. V. Hajnal, A. D. Edwards, S. J. Counsell, and D. Rueckert. Automatic whole brain MRI segmentation of the developing neonatal brain. *IEEE Transactions on Medical Imaging*, 33(9):1818–1831, 2014.

[25] A. Makropoulos, C. Ledig, P. Aljabar, A. Serag, J. V. Hajnal, A. D. Edwards, S. J. Counsell, and D. Rueckert. Automatic tissue and structural segmentation of neonatal brain MRI using Expectation-Maximization. In *MICCAI NeoBrainS12, Proceedings*, pages 9–15, 2012.

[26] N. Marlow, D. Wolke, M. A. Bracewell, M. Samara, and EPICure Study Group. Neurologic and developmental disability at six years of age after extremely preterm birth. *The New England Journal of Medicine*, 352:9–19, 2005.

[27] A. Melbourne, M. J. Cardoso, G. S. Kendall, N. J. Robertson, N. Marlow, and S. Ourselin. NeoBrainS12 Challenge: Adaptive neonatal MRI brain segmentation with myelinated white matter class and automated extraction of ventricles I-IV. In *MICCAI NeoBrainS12, Proceedings*, pages 16–21, 2012.

[28] P. Moeskops, M. Veta, M. W. Lafarge, K. A. J. Eppenhof, and J. P. W. Pluim. Adversarial training and dilated convolutions for brain MRI segmentation. In *DLMIA and ML-CDS, Proceedings*, pages 56–64. 2017.

[29] E. Orasanu, A. Melbourne, M. J. Cardoso, H. Lomabert, G. S. Kendall, N. J. Robertson, N. Marlow, and S. Ourselin. Cortical folding of the preterm brain: A longitudinal analysis of extremely preterm born neonates using spectral matching. *Brain and Behavior*, 6(8):e00488, 2016.

[30] M. Péporté, D. E. I. Ghita, E. Twomey, and P. F. Whelan. A hybrid approach to brain extraction from premature infant MRI. In *SCIA, Proceedings*, pages 719–730, 2011.

[31] C.-H. Pham, A. Ducournau, R. Fablet, and F. Rousseau. Brain MRI super-resolution using deep 3D convolutional networks. In *ISBI, Proceedings*, pages 197–200, 2017.

[32] C.-H. Pham, C. Tor-Díez, H. Meunier, N. Bednarek, R. Fablet, N. Passat, and F. Rousseau. Simultaneous super-resolution and segmentation using a generative adversarial network: Application to neonatal brain MRI. In *ISBI, Proceedings*, pages 991–994, 2019.

[33] M. Prastawa, J. H. Gilmore, W. Lin, and G. Gerig. Automatic segmentation of MR images of the developing newborn brain. *Medical Image Analysis*, 9:457–466, 2005.

[34] A. Serag, M. Blesa, E. J. Moore, R. Pataky, S. A. Sparrow, A. G. Wilkinson, G. Macnaught, S. I. Semple, and J. P. Boardman. Accurate learning with few atlases (ALFA): An algorithm for MRI neonatal brain extraction and comparison with 11 publicly available methods. *Scientific Reports*, 6:23470, 2016.

[35] C. Tor-Díez, N. Passat, I. Bloch, S. Faisan, N. Bednarek, and F. Rousseau. An iterative multi-atlas patch-based approach for cortex segmentation from neonatal MRI. *Computerized Medical Imaging and Graphics*, 70:73–82, 2018.

[36] D. Ulyanov, A. Vedaldi, and V. S. Lempitsky. Improved texture networks: Maximizing quality and diversity in feed-forward stylization and texture synthesis. In *CVPR, Proceedings*, pages 4105–4113, 2017.

[37] L. Wang, F. Shi, W. Lin, J. H. Gilmore, and D. Shen. Automatic segmentation of neonatal images using convex optimization and coupled level sets. *NeuroImage*, 58:805–817, 2011.

[38] L. Wang, F. Shi, P.-T. Yap, J. H. Gilmore, W. Lin, and D. Shen. 4D multi-modality tissue segmentation of serial infant images. *PLoS One*, 7:e44596, 2012.

[39] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004.

[40] H. Xue, L. Srinivasan, S. Jiang, M. Rutherford, A. D. Edwards, D. Rueckert, and J. V. Hajnal. Automatic segmentation and reconstruction of the cortex from neonatal MRI. *NeuroImage*,

38:461–477, 2007.
[41] Q. Yu, A. Ouyang, L. Chalak, T. Jeon, J. Chia, V. Mishra, M. Sivarajan, G. Jackson, N. Rollins, S. Liu, and H. Huang. Structural development of Human fetal and preterm brain cortical plate based on population-averaged templates. *Cerebral Cortex*, 26(11):4381–4391, 2016.

Appendix A. Technical description

Appendix A.1. General statements

The proposed code is written in Python 3.6 (but it also works in Python 2.7). It was packaged in Pipy[7] and can be easily installed using:

```
pip install SegSRGAN
```

It depends on several packages, all of which are automatically installed when installing the SegSRGAN pip package. The dependencies are mainstream, widely used libraries such as Numpy[8], Tensorflow[9], Keras[10], Pandas[11], or SimpleITK[12].
Since we use tensorflow-gpu dependency, the program can be computed on several GPUs. It is indeed recommended to run the code on GPUs instead of CPUs whenever possible, for the sake of computational efficiency. However, if GPUs are available, one should install tensorflow-gpu (see installation tutorial[13]) manually. Trained weights for the SegSRGAN networks are available in a GitHub repository[14]. When installing the pip package, these weights are locally downloaded while importing the SegSRGAN package in Python. (The weights folder is contained in the SegSRGAN package directory.) In particular, all the test results presented in Section 4 were obtained with these weights, for the sake of reproducibility.
All the Python files and the weights considered hereafter can be found in the Python package directory. The path of this folder is obtained by executing the following Python command:

```
import importlib
importlib.util.find_spec("SegSRGAN").
    submodule_search_locations[0]
```

and then entering the subdirectory "SegSRGAN" in the path returned by the result of this command.

[7] https://pypi.org/project/SegSRGAN
[8] https://www.numpy.org
[9] https://www.tensorflow.org
[10] https://keras.io
[11] https://pandas.pydata.org
[12] http://www.simpleitk.org
[13] https://www.tensorflow.org/install/gpu
[14] https://github.com/koopa31/SegSRGAN

Appendix A.2. Training

The code runs on multiple GPUs thanks to the Keras library (a parameter allows to use all available GPUs). In our models of generator and discriminator, we used the multi_gpu_model function[15]. It is optimized for the merging of the results when the GPUs are connected via NVLinks (wire-based communications protocol serial multi-lane near-range communication link developed by NVIDIA).

An example of basic command line. The package contains a file name "SegSRGAN_training.py" that allows one to train a SegSRGAN model. This file is located in the package folder which can be found as explained in Appendix A.1. The basic command for performing the training from an image training database is for instance:

```
python SegSRGAN_training.py --new_low_res 0.5 0.5 3
    --csv /home/user/data.csv --snapshot_folder /
    home/user/training_weights --dice_file /home/
    user/dice.csv --mse_file /home/user/
    mse_example_for_article.csv --
    folder_training_data /home/user/
    temporary_file_for_training
```

The only requirement to be fulfilled beforehand is the definition of an appropriate csv file. One can find an example of such file in Table A.9. Here, the "HR_image" column must contain all the paths to the HR images; the "Label_image" column must contain all the paths to the segmentation maps; and finally, the "Base" column must contain either the "Train" or "Test" label.
The learned weights are saved into the folder "/home-/user/training_weights" that contains one dedicated file per epoch. At the end of each epoch, the dice and mse file (containing the Dice and the MSE scores, respectively) are also saved.
Warning – The argument "folder_training_data" corresponds to a folder, created during the script execution, in which all the training data are split into patches and organized by batch. This folder, which can be large (20 to 60 GB, depending on the chosen arguments) is deleted at the end of each epoch. In case of unexpected interruption of the script, it may be required to delete it manually.

Options. The parameters that control the training from scratch are the following (all the given values are the values to use in order to reproduce our results) :

[15] https://keras.io/utils

Table A.9: Example of a csv file used for training.

HR_image	Label_image	Base
/home/user/data/HR/1.nii.gz	/home/user/data/Label/1.nii.gz	Train
/home/user/data/HR/2.nii.gz	/home/user/data/Label/2.nii.gz	Test
/home/user/data/HR/3.nii.gz	/home/user/data/Label/3.nii.gz	Test
/home/user/data/HR/4.nii.gz	/home/user/data/Label/4.nii.gz	Train
/home/user/data/HR/5.nii.gz	/home/user/data/Label/5.nii.gz	Test

- **new_low_res**: resolution of the LR image generated during the training. One value is given per dimension, for fixed resolution (e.g. "−−new_low_res 0.5 0.5 3"). Test value: 0.5 0.5 3.

- **csv** (string): file that contains the paths to the files used for training. These files are divided into two categories: train and test. Consequently, it must contain 3 columns, called: HR_image, Label_image and Base (which is equal to either Train or Test), respectively. In our case, the test database was composed of 8 images and the train one was composed of 32 images all from the first release of the dHCP database. The list of the file names is detailed at the end of the Appendix, for the sake of reproducibility.

- **snapshot_folder** (string): path of the folder in which the weights will be regularly saved after a given number of epochs (this number is given by the **snapshot** (integer) argument).

- **dice_file** (string): csv file where the Dice scores are stored at each epoch.

- **mse_file** (string): MSE file where the Dice scores are stored at each epoch.

- **folder_training_data** (string): folder where temporary files are written during the training (created at the begining of each epoch and deleted at the end of it).

Finally, the package allows one to either train from an augmented dataset (as described in section 3.1.3) or not, by using the following parameters:

- **percent_val_max**: multiplicative value that gives the ratio of the maximal value of the image, for defining the standard deviation of the additive Gaussian noise. For instance, a value of 0.03 means that $\sigma = 0.03 \max(X)$ where $\max(X)$ is the maximal value of the image X. Test value: 0 (without data augmentation); 0.03 (with data augmentation).

- **contrast_max**: controls the modification of contrast of each image. For instance, a value of 0.5 means that at each epoch, each image will be set to a power uniformly drawn between 0.5 and 1.5. Test value: 0 (without data augmentation); 0.5 (with data augmentation).

Complementary information about all the parameters and their default values can be obtained by:

```
python SegSRGAN_train_avec_base_test.py --help
```

(In order to know all the possible parameters, see the readme file.)
Warning – The patch size is set to 64 (argument **patch_size**). It is possible to change it; however this requires that the discriminator architecture be modified too, as it computes successive convolutions until the output is of size one (see Section 3.1.2).

Appendix A.3. Testing

Appendix A.3.1. Testing on one or more images (command line)

We now describe the way of performing segmentation on a set of many LR images (instead of segmenting one image many times). This command line works with NIfTI and DICOM images but the results are returned as NIfTI images (located in the package folder which can be found as explained in Appendix A.1). An example of command for such a group processing is:

```
python job_model.py --path /home/data.csv --patch "
    64,128" --step "32 64,64 128" --
    result_folder_name "
    weights_without_augmentation" --weights_path "
    weights/Perso_without_data_augmentation"
```

csv path parameter. A csv file, as the one mentioned in the above example, is used to get the paths of all the images to be processed. Only the first column of each entry will be used, and the file must contain only paths (i.e. no header).

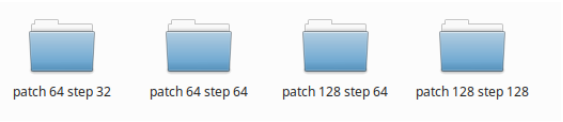


Figure A.14: Data organization for the output images of a same input image.

Step and patch parameters. In this example, we run steps 32 and 64 for patch 64 and steps 64 and 128 for patch 128. The list of the paths of the images to be processed must be stored in a csv file.
Warning – It is mandatory to respect exactly the same shape for the given step and patch.

Weights parameter. In this article, two types of weights provided in the GitHub repository have been used. These weights correspond to the following parameters:

- **weights/Perso_without_data_augmentation:** corresponding to the weights without data augmentation.
- **weights/Perso_with_constrast_0.5_and_noise_0.03_val_max:** corresponding to the weights with data augmentation as described in Section 4.

Others weights not presented in this article are available (the help of SegSRGAN provides the list of all these available weights). (A more detailed description of all the available weights is given in the readme file.)

Organizing the output storage. Each image to be processed has to be stored in its own folder. When processing a given input image (which can be either a NIfTI image or a DICOM folder), a dedicated folder is created for each output. This folder will be located in the folder of the input image which has been processed and will be named with respect to the value of the parameter −−result_folder_name (in our example the folder will be named "result_with_Weights_without_augmentation"). In Figure A.14, one can see how the data are organized in the folder "result_with_Weights_without_augmentation". Finally, each folder presented in Figure A.14 contains two NIfTI files, namely the SR and the segmentation.

Appendix A.3.2. Testing code on one image in Python
The command line presented below allows one to segment one or many images. Practically, it relies on a Python function that segments one image and writes the result on the disk.

This function, called "segmentation", is located in the "Function_for_application_test_python3.py" file and can be applied as follows:

```
from SegSRGAN.Function_for_application_test_python3
    import segmentation

segmentation(input_file_path, step, new_resolution,
    patch, path_output_cortex, path_output_hr,
    weights_path)
```

The main two differences between the function and a command line invoking it are the following:

- only one image can be segmented by the function, whereas many can be via a command line;
- the result folders need to be created before applying the Python function.

Appendix A.4. List of the database files

Test database:
sub-CC00122XX07.nii.gz, sub-CC00363XX09.nii.gz,
sub-CC00480XX11.nii.gz, sub-CC00201XX03.nii.gz,
sub-CC00172BN08.nii.gz, sub-CC00237XX15.nii.gz,
sub-CC00162XX06.nii.gz, sub-CC00268XX13.nii.gz

Train database:
sub-CC00367XX13.nii.gz sub-CC00281BN10.nii.gz
sub-CC00170XX06.nii.gz sub-CC00250XX03.nii.gz
sub-CC00338BN17.nii.gz sub-CC00415XX11.nii.gz
sub-CC00099AN18.nii.gz sub-CC00216AN10.nii.gz
sub-CC00421AN09.nii.gz sub-CC00117XX10.nii.gz
sub-CC00205XX07.nii.gz sub-CC00379XX17.nii.gz
sub-CC00347XX18.nii.gz sub-CC00357XX11.nii.gz
sub-CC00126XX11.nii.gz sub-CC00418BN14.nii.gz
sub-CC00217XX11.nii.gz sub-CC00421BN09.nii.gz
sub-CC00422XX10.nii.gz sub-CC00303XX06.nii.gz
sub-CC00164XX08.nii.gz sub-CC00209XX11.nii.gz
sub-CC00206XX08.nii.gz sub-CC00172AN08.nii.gz
sub-CC00138XX15.nii.gz sub-CC00221XX07.nii.gz
sub-CC00413XX09.nii.gz sub-CC00176XX12.nii.gz
sub-CC00069XX12.nii.gz sub-CC00168XX12.nii.gz
sub-CC00313XX08.nii.gz sub-CC00334XX13.nii.gz

These files were downloaded from the dCHP first release website[16].

[16] http://www.developingconnectome.org/project/data-release-user-guide/