Convolutional Autoencoder for MRI modality synthesis

Felix KLINGELHOEFER, François ROUSSEAU

Abstract—In Magnetic Resonance Imaging, various pulse sequences are used to produce image contrasts that give different information on tissues. Having access to more modalities can therefore be useful for subsequent processing by a multi-modal algorithm, or one optimized for a particular contrast. We propose a Deep Learning based method for modality synthesis. More specifically, we use a Convolutional Auto-Encoder that we train on input patches extracted from T1-w images and the corresponding T2-w images. This neural network is then compared to another state-of-the-art algorithm called Magnetic Resonance Image Example-Based Contrast Synthesis (MIMECS) [16], and is shown to outperform it on our data both in precision and running time.

Index Terms—Deep Learning, Convolutional Auto-Encoder, Image synthesis, MRI contrast.

1 Introduction

Magnetic Resonance Imaging (MRI) is a medical imaging technique that uses strong magnetic fields and radio waves in order to produce 3-dimensional images of organs. Contrary to Computed Tomography, it does not utilize ionizing radiation, which often makes it the preferred option for visualizing tissues.

It is commonly used in neuroimaging to investigate conditions such as dementia, epilepsy or cerebro-vascular diseases. There are several different modalities that are adapted to visualize different tissues (Figure 1):

T1-w In T1 weighted images, fat (and therefore white matter) appears bright and fluid (in particular the cerebro-spinal fluid) dark. It is good for visualizing the brain anatomy.

T2-w T2 weighting makes fluid bright and fat somewhat less bright, making it useful for visualizing some types of pathology.

FLAIR Fluid-attenuated inversion recovery weighting nullifies the signal of fluid, and is used for disorders such as Multiple Sclerosis lesion.

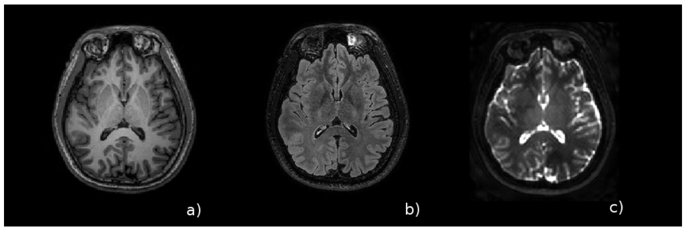

Fig. 1. Three different MRI contrasts. a) T1, b) FLAIR, c) T2.

These contrasts capture different and complementary information on the tissues, so many algorithms use multi-modal processing, where several modalities are used simultaneously, whereas others are optimized for a particular contrast which best captures the relevant information. However, sometimes some image sequences are not available, either because they were not taken (due to limited scan time), or are of poor quality (as the result of movement artifacts for example).

In these cases, contrast synthesis can provide a replacement for the missing sequences, allowing the use of the desired algorithms. Because some information is unique to a given modality, it is not possible to replicate the missing contrast exactly, but the approximation can be of sufficient quality to analyze the data better

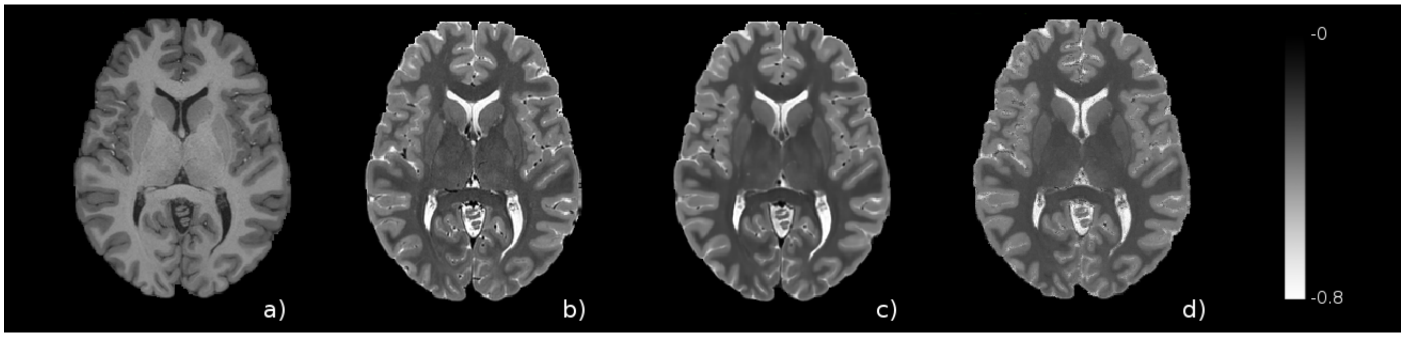

Fig. 2. MRI modalities and synthetic results from our neural network and the example based method MIMECS. a) T1-w input, b) T2-w ground truth, c) Our method, d) MIMECS

than with the original contrast (for example to segment brain lesions).

State-of-the-art methods for modality synthesis are mostly example based: several studies [10], [16], [15] and [17] all use sparse coding approaches on patches to build a new sequence from a different one, and an atlas (that contains both the initial and target contrasts). Similar techniques have been used for other computer vision problems such as super-resolution or image denoising, for which recent research has shown that deep learning, and in particular convolutional neural networks, can provide better results [4], [1].

Motivated by these successes, we consider a deep convolutional auto-encoder for modality synthesis. In order to improve our results, numerous recently-developed techniques of deep learning are applied to the neural network, and variations to its architecture are experimented. Visual and numerical results are presented in Figures 2 and 3.

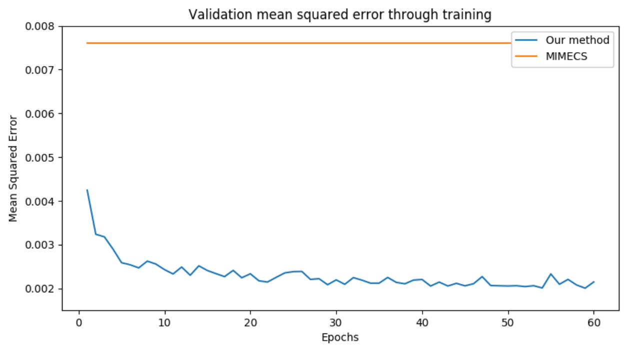

Fig. 3. Mean squared error of our neural network through training, and of MIMECS for reference

Finally, we show that our method performs better than a recent example based approach called Magnetic Resonance Image Example-Based Contrast Synthesis (MIMECS) [16] even when training for a single epoch (Figure 3).

Contents

2 Overview of Deep Learning

Deep learning is the application of neural networks to learning tasks. Even though the theory behind it dates back several decades, its use has seen a surge in the past few years, thanks to the increase in computing power that came from GPU processing. It is now applied to many fields including bioinformatics, speech recognition, machine translation and computer vision. Specialized architectures have been developed to adapt it to the specific requirements of these different fields. We will start by explaining what an artificial neural network is, then see how it can be trained automatically, and finally present two classes of nets that are adapted to image synthesis: Convolutional Neural Nets and Auto-Encoders.

2.1 Artificial Neural Networks

Artificial neural networks are inspired by biological neural nets (animal brains). The human central nervous system for example is made up over 86 billion neurons, with over 7000 synapses each, connecting them to the other neurons. In comparison, most artificial neural networks only have between several thousand and a few million units, which are very simplified versions of neurons.

2.1.1 Neurons and computational equivalent

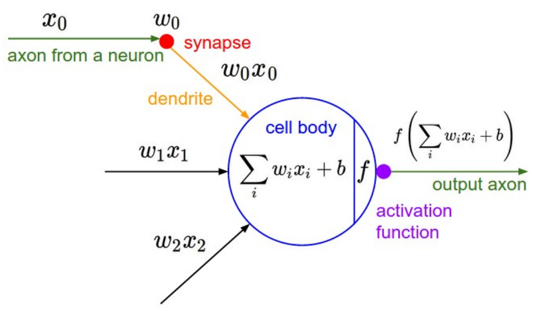

Fig. 4. Simplified mathematical model of a neuron from http://cs231n.github.io/neural-networks-1/

Figure 4 shows a coarse mathematical approximation of a single neuron that will be used for computation. In the mathematical model, the neuron described by the function g performs the composition of a linear function f of its inputs and an activation function a:

$$g = a \circ f \tag{1}$$

The linear function f can be described with the following formula: for an input vector $X \in \mathbb{R}^n$, a weight vector $W \in \mathbb{R}^n$ and a bias b the output is

$$f(X, W, b) = b + \sum_{i=1}^{n} W_i X_i \tag{2}$$

A neuron that has an input of length n is described by $n + 1$ parameters: n weights and one bias. These parameters will be learned throughout training, In order to best fit the output to the expected results.

The signal then goes through an activation function that is non-linear. We will use two of them: the sigmoid function $\sigma(x) = 1/(1 + e{-}x)$ and the Rectified Linear Unit (ReLU) which computes $f(x) = max(0, x)$. It has been shown that the ReLU significantly improves performance compared to the sigmoid function, or the hyperbolic tangent function [13]. For this reason the network will mainly use ReLUs for non-linearity, and the sigmoid function only to normalize the output, on the last layer.

2.1.2 Layered architecture

A neural network is a collection of neurons, and can be represented as an acyclic graph where a single neuron is a node and the connections between neurons are the vertices. The neurons are then organized in layers. The basic layer is a fully-connected (or dense) layer, in which every neuron is connected pairwise to every neuron of the previous layer. This can be described as a matrix product and addition: given an input vector $X \in \mathbb{R}^n$, a fully connected layer of d neurons, described by the weight matrix $W \in M_{n,d}(\mathbb{R})$ and bias vector $B \in \mathbb{R}^d$, and that uses the activation function a, will compute

$$f(X, W, B) = a(W.X + B) \tag{3}$$

In this case, the hidden layer has $(n + 1) * d$ parameters: $n * d$ weights and d biases. These layers can then be stacked one after the other to produce a neural network, as seen in figure 5.

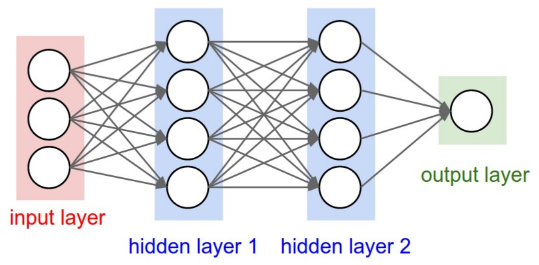

Fig. 5. Neural Network with two dense hidden layers from http://cs231n.github.io/neural-networks-1/

2.2 Training

The idea behind machine learning for neural nets is that all its parameters can be learned automatically through training. It is most often done through supervised training where the neural net is fed with input and the desired corresponding output. It is then necessary to assess the quality of the output of the net for a given input, compared to the desired one: that is done by a loss function. In order to progressively improve its results, the algorithm attempts to minimize the loss function through stochastic gradient descent and backpropagation: it determines what changes in parameters could have provided better results on an input batch and then updates them, and repeats this thousands of times.

2.2.1 *Loss function*

The loss function measures the quality of the output of the neural network. As training will consist in trying to minimize it, it must be lower on results that are closer to the desired output. In our case, we will be working with images, and will call these desired output images the ground truth.

To compare the neural net's output images X to the ground truth T (as flattened pixel intensity vectors), there are two functions, the first of which is the mean absolute error (mae), and the second the mean squared error (mse).

$$mae(X,T) = \frac{1}{n}\sum_{i=1}^{n}|X_i - T_i| \tag{4}$$

$$mse(X,T) = \frac{1}{n}\sum_{i=1}^{n}(X_i - T_i)^2 \tag{5}$$

Both these functions are sensitive to intensity scaling: if the intensity of two images are multiplied by two, their mse is multiplied by four and their mae by two. This means all the input and ground truth images need to be normalized to a given intensity before being passed to the neural network.

2.2.2 *Stochastic Gradient Descent*

This method is a stochastic approximation of the gradient descent optimization method. The goal of training is to minimize the loss function over the entire training dataset. In order to achieve this, the stochastic gradient descent method consists in iteratively computing the approximate gradient at a data point of the input, by using the backpropagation method, and then updating the weights proportionally to the gradient.

Backpropagation uses the chain rule to calculate the partial derivative with respect to every parameter of a previous layer, and can be used successively for every layer until the input.

The parameter vector (weights and biases) w is then updated proportionally to the gradient of the loss function L:

$$w := w - \eta \nabla L(f(I,w),T) \tag{6}$$

where f is the neural network function, I is the input, T the ground truth and η is the learning rate.

In order to get a better approximation of the gradient, it is in practice calculated on batches of input. The size of the batches is a hyperparameter (non trainable parameter of the neural network) that is often simply determined by memory constraints. For batches of size n, the weight update formula becomes

$$w := w - \frac{\eta}{n}\sum_{i=1}^{n}\nabla L(f(I,w),T) \tag{7}$$

Many improvements upon the basic stochastic gradient descent have been proposed. We will use Adam (short for Adaptive Moment Estimation) [12], which uses the running average of gradients and the second moment of gradients to smooth out parameter updates, given by the following formulas.

$$\begin{aligned}
m_w^{(t+1)} &:= \beta_1 m_w^{(t)} + (1-\beta_1)\nabla_w L^{(t)} \\
v_w^{(t+1)} &:= \beta_2 v_w^{(t)} + (1-\beta_2)(\nabla_w L^{(t)})^2 \\
\hat{m}_w &= \frac{m_w^{(t+1)}}{1-\beta_1^t} \\
\hat{v}_w &= \frac{v_w^{(t+1)}}{1-\beta_2^t} \\
w^{(t+1)} &:= w^{(t)} - \eta \frac{\hat{m}_w}{\sqrt{\hat{v}_w}+\epsilon}
\end{aligned} \tag{8}$$

Using the same notations as before, and indexing the training iteration with t. β_1 and β_2 are the forgetting factors for the gradients and their moments, and ϵ is simply a small number to avoid division by zero. m indicates the gradients, and v their second moment, while $\hat{m}$ and $\hat{v}$ are their respective running averages.

The training algorithm makes multiple passes on the input dataset in order to converge towards a satisfying local minimum in the high-dimensional parameter space. IT can be stopped either after an arbitrary number of passes, also called epochs, or when a condition on the loss is satisfied (for example, when the loss on some validation data falls below a given threshold). To avoid getting stuck in a cycle, the data is shuffled each epoch.

2.3 Convolutional Neural Networks

Many computer vision problems are translation invariant: for example in image classification, the result should not be changed by a translation (as long as the object of interest stays in the frame). In these tasks, neural nets often work by trying to detect features (for a face, the darker areas of the eyes and mouth, and their shape, for example) everywhere in the picture. This motivated the use of convolutional layers instead of dense layers, which have shown much better empirical results [14].

In a convolutional layer, the neurons are arranged along the same dimensions as the input (2D for an image, 3D for a volume), to which one dimension is added. A kernel is then convolved along the input: a given neuron is connected to every neuron that is within its window in the previous layer (Figure 6). Neurons are always connected along the last dimension. Intuitively, a convolutional layer can be seen as a collection of filters that are applied to an image (or volume), and produce a collection of images arranged along a third dimension. Every neuron then has a limited spatial receptive field, but is connected to surrounding neurons of a previous layer, no matter from which filter.

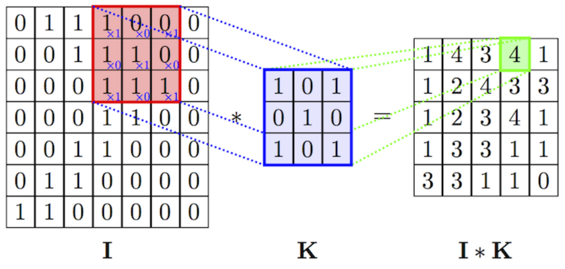

Fig. 6. Diagram of a 3x3 convolution kernel (K) applied to a 7x7 image (I) from https://cambridgespark.com/content/tutorials/convolutional-neural-networks-with-keras/index.html

The goal of these restriction is to be able to have more neurons in a network without the number of learnable parameters exploding. Instead of having $(x_{i-1} * y_{i-1} * z_{i-1} * n_{filters,i-1} + 1) * x_i * y_i * z_i * n_{filters,i}$ learnable parameters, where x, y, and z are the dimensions of the previous layer, it has only $(k_x * k_y * k_z * n_{filters,i-1} + 1) * n_{filters,i}$, with k_i being the size of the kernel along the i axis. To give an idea of the magnitude of the reduction, for a $300 \times 300 \times 3$ image (3 color channels) connected to a $300 \times 300 \times 20$ dense layer, there would be over 486 billion learnable parameters. Using a convolutional layer with a 5×5 window, this number is reduced to 1520.

One side-effect that needs to be pointed out is that the output of a convolutional layer will be slightly smaller than its input, when no padding is added to the input, as illustrated in Figure 6. This is because a convolutional kernel cannot be applied to a pixel if it would go over the border of the image. This means that for a kernel of size $K_x * K_y$, and an input of size $I_x * I_y$, the output O would have dimensions

$$\begin{aligned} O_x &= I_x - \frac{K_x - 1}{2} \\ O_y &= I_y - \frac{K_y - 1}{2} \end{aligned} \tag{9}$$

Thanks to this limitation of parameters, convolutional neural networks can stack many layers (often ten or more). It has been found empirically that adding depth to the networks is usually preferable to making the kernels larger, and proven for some particular problems [5]. One reason for this is that the receptive field of a neuron increases by stacking layers: for example, a neuron after two 3×3 convolutions has an effective 5×5 receptive field, the same as after one 5×5 convolution. It also adds more non-linearities, which have a positive effect on results.

For this reason, and because 3D convolutions have more learnable parameters and are more computationally expensive than 2D ones, we will use only $3 \times 3 \times 3$ kernels, in order to make deeper networks.

2.4 Auto-Encoders

Auto-Encoders are a class of Artificial Neural Networks. Their goal is to encode an input by reducing its dimensions, and then to reconstruct the original input from its reduced representation (Figure 7). They are not particularly good at this task however, since handcrafted compression algorithms perform better than them in most cases, as for example jpeg for image compression. Luckily, they do have other uses. Indeed, dimensionality reduction makes them extract robust features from images, which enables them to perform well on tasks such as denoising [20] and in our case contrast synthesis.

An autoencoder that uses convolutional layers is called a convolutional autoencoder. Its architecture follows the pattern of autoencoders: first the dimensions of the input image or volume are reduced, through either strided convolutions or pooling layers (usually maximum pooling), and then are increased through upsampling layers. This is the type of neural network that will be used for contrast synthesis.

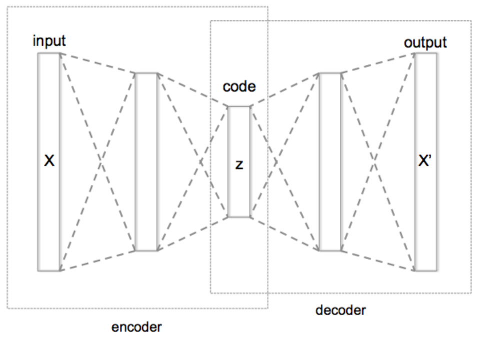

Fig. 7. Schematic of an Autoencoder with 3 hidden fully-connected layers from https://en.wikipedia.org/wiki/Autoencoder

3 Architecture of the Neural Network

In this section we will start by presenting the baseline architecture and training dataset and will then test variations to the architecture, and dataset.

3.1 Baseline architecture

As stated previously, a convolutional autoencoder will be used for the contrast synthesis, since this problem involves volumes, for which convolutions are particularly adapted, and in order to transform one contrast to another the neural net needs to extract features from the base image, hence the use of an autoencoder. Figure 8 presents the architecture that will serve as baseline.

This neural net is sequential: three convolutional layers are followed by one max pooling layer, followed by another three convolutions, one upscaling layer, and two more convolutions. All convolutions have a kernel of size $3 \times 3 \times 3$, and 32 filters. It is noteworthy that this network doesn't actually compress the data, since the intermediate representation is actually four times bigger than the input size, because of the number of filters. It does however reduce the dimensions of the input, which is the basis for auto-encoders.

3.2 Training dataset and preprocessing

The dataset that was used for training consists of pairs of MRIs, with T1 and T2 weighting,

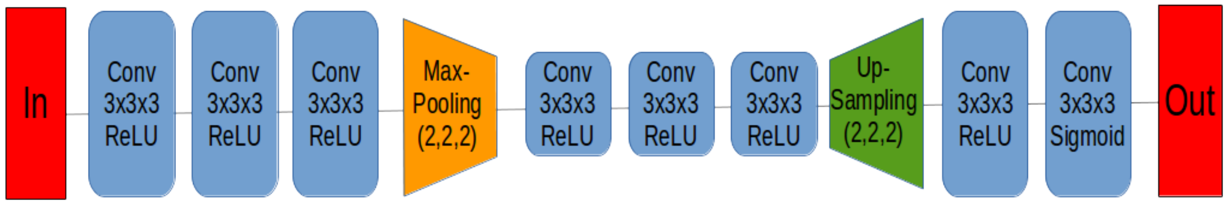

Fig. 8. Diagram of our base architecture. Each convolutional layer has 32 filters.

for one hundred different patients. Each image has voxels of size $0.7 \times 0.7 \times 0.7\ mm^3$. Their dimensions where $260 \times 311 \times 260$. In the network, each dimension is halved through a pooling layer, and then doubled, which means that the output was of size $260 \times 310 \times 260$, effectively cropping the image by one on the y axis, which wasn't a problem because the brains where surrounded by a fair amount of background voxels (with zero intensity). In the base experiment, the network is trained on the ninety first patients, and tested on the ten last. We also explored training it only on the ten first patients, while validating on the same ten last, and will discuss those results in section 3.4. This work focuses on the problem of synthesizing a T2 weighted contrast from the T1, but the same network could be used for the opposite transformation, or any other contrast synthesis, if provided with the appropriate data. The MRI acquired T2-w image will be called ground truth T2 (as opposed to synthetic T2 for those produced by an algorithm).

Before training the network, both modalities for a same patient need to be registered. This means an algorithm has to be used in order to find the coordinate system that offers the best anatomical juxtaposition between both contrasts, but this had already been done on the dataset.

Brain masks provided with the dataset were then applied to the images in order to strip the skulls from the brains. This allows the network to focus only on learning the contrasts transformations of interest, those for the brain.

As intensities can vary significantly between two MRI images, the voxel (equivalent of pixels for volumes) values need to be normalized before the image is given to the neural network. The first attempt to do so was by dividing all intensity values by the maximal value, to get voxel intensities in the [0,1] range, but we noticed that the output of the neural net often had an intensity shift compared to the ground truth. This issue was solved by dividing all intensity values by three times the mean intensity (calculated only on the non-zero intensity voxels), because the mean intensity was less subject to variation than the maximum, so the normalized ground truth would have less unpredictable intensity shifts. This second technique therefore gave better results.

Finally, to obtain a large number of training inputs, smaller patches are extracted from the base images. Due to memory limitations, patches of size $38 \times 38 \times 38$ voxels were extracted from the T1 images, with a step of 16. To avoid getting patches that contain mainly background voxels with null intensities, patches whose central voxel (16,16,16) are not within the brain mask are discarded. Because no padding was added to the convolutional layers, they crop the input, and the output size of the neural network for these patches is only $16 \times 16 \times 16$ voxels, so patches of this size (centred at the same point) were extracted from the corresponding ground truth T2 images. In total, 89 018 patches were obtained from the first 90 patients, and an additional 9 692 patches for validation, from the ten last patients.

The implementation was done using the Keras library [2], running on the Theano backend [19]. This software was running on a NVidia GeForce GTX 1080 Ti graphics card with 11 GB of video ram, and 64 GB of normal ram. A batch size of 32 was used for training, because of video memory limitations.

3.3 Tweaking for deeper networks

When this deep neural net was first trained, it did not converge at all, and was stuck producing black output no matter the length of prior training.

Layer saturation may account for this problem: the signal may explode or vanish in the network, in turn blocking any further training. This phenomenon is explored more deeply by existing studies [7], and is caused by the variance of the signal being successively multiplied at every layer. In order to prevent this problem, many elaborate initialization schemes have been developed. We used an existing one [9], wherein the initial weights for the convolutional layers are sampled from the normal distribution $\mathcal{N}(0, \sqrt{2/n_f^{in}})$, with n_f^{in} being the number of filters of the previous layer.

Another phenomenon that can prevent successful training in deeper networks is called internal covariate shift: throughout training, the statistical distribution of the output of a neural layer is ever changing, which means the next layer must adapt to the changing form of its input, thereby slowing down the learning process. One solution to this problem is using batch normalization [11], which normalizes the mean and variance of the output of a layer for every batch. It also has the added benefit of reducing the risk of signal saturation. For these reasons, a Batch Normalization layer is added before every hidden convolutional layer.

After these changes, it was possible to train the network successfully, as shown in Figure 9.

3.4 Architecture cross-validation

In this section, we will discuss experiments with small changes to the baseline architecture in order to justify the choices that are made. The training time was a limiting factor to the number of experiments made. Indeed, it took over 800 seconds per epoch to train the base model, and all experiments involved 60 epochs of training. This choice seemed like a reasonable compromise between training time and

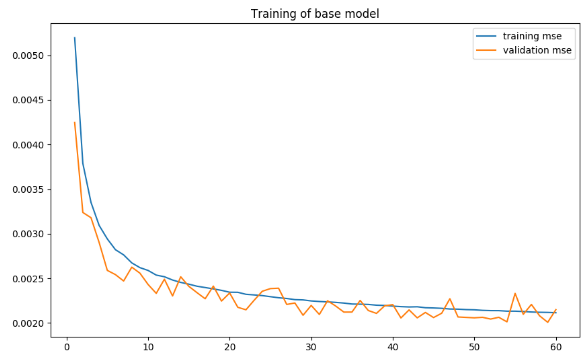

Fig. 9. Test (90 first patients) and validation (10 last) loss through training for 60 epochs of the baseline network

performance: as can be seen in Figure 9, the convergence slows down significantly before the 60 epoch mark.

3.4.1 *Network width*

Two new networks were trained to determine the influence of the width of the network on the results. The first, which will be called shallow, only has 16 filters on every convolutional layer, and the second, called wide, has 48. As a reminder, the base network has 32 filters per layer.

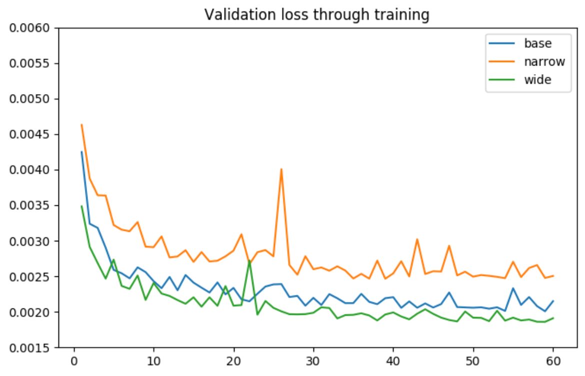

Fig. 10. Results of training for 60 epochs on the base model (32 filters), a wider (48 filters) and a narrower (16 filters) variant

As could be expected, the neural network performs better the more filters there are on every convolutional layer (Figure 10). The performance increase is however much more significant between the narrow and the base model, than between the base and the wide model.

The number of parameters increases as the square of the number of filters per layer: the wide network has 376 849 learnable parameters against 168 289 for the base model. For this reason, the training time also increases significantly between the models: approximately 400s, 820s, and 1880s per epoch respectively.

3.4.2 *Network depth*

To test the influence of the network's depth on its performance, two variants to the base model were trained. The first is deeper, and has an additional convolutional layer before the max-pooling, between the max-pooling and the up-sampling and after the upsampling (4-4-3 for a total of 11 convolutional layers). The second is shallower, having one convolutional layer less in every section (2-2-1 for a total of 5 convolutional layers).

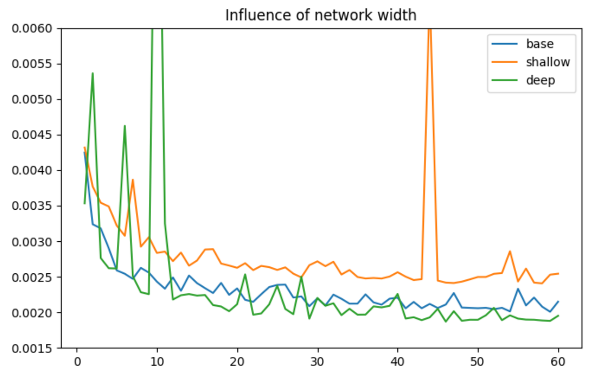

Fig. 11. Results of training for 60 epochs on the base model (8 convolutional layers), a shallow (5 convolutional layers) and a deep (11 convolutional layers) model

The results indicate that depth has a very similar effect to width on transformation quality (Figure 11). The number of parameters however increases much less: the deep network has 251 521 learnable parameters, which is a linear increase from the 168 289 of the base model in the number of convolutional layers, when accounting for the fact that the first and last layers have a reduced number of parameters.

Even though the time complexity increase is nearly linear in the number of convolutional layers to process a large image, it is far greater for smaller patches. Indeed, as the network gets deeper, in order to produce the same size outputs, the input must become increasingly large because each convolution crops the image, through the lack of padding on the input. In practice, this means that the deeper network was trained on $46 \times 46 \times 46$ voxel patches (instead of $38 \times 38 \times 38$ for the base model, and $30 \times 30 \times 30$ for the shallow variant), which leads to a great increase in training time: around 2040 seconds per epoch against the base 820, or 280s for the shallow model.

3.4.3 *Alternative architectures*

Experiments were also undertaken with two alternative architectures, the first attempt being to use progressive dimensionality reduction and increase, by using three max-pooling layers and three Up-Sampling layers, always on one single dimension, as seen in Figure 12. The results were nearly indistinguishable, both in loss throughout training and in training or processing time. For these reasons, the idea wasn't pursued any further.

The second attempt was inspired by the work of Kaiming He et al. (2015) on residual connections [8]. The idea behind a residual connection is that some information skips a part of the network (one or more layers), and is then added back. In our case, we tried concatenating the output of the third convolutional layer to the input of the seventh, essentially skipping the pooling and up-sampling (Figure 13). The intuition behind this was that the network would learn both the abstract features thanks to the auto-encoder part, while retaining high definition information on the image thanks to the residual connection, in order to produce more precise output. It didn't turn out to improve the synthetic output however, with both training and validation loss nearly identical to that of the network without a residual connection, and at the cost of somewhat increased training time (over 1000s per epoch).

3.5 Importance of dataset size and overfitting

As described earlier, in the base experiment, the Neural Network is trained on ninety patients. To determine the impact of the size of

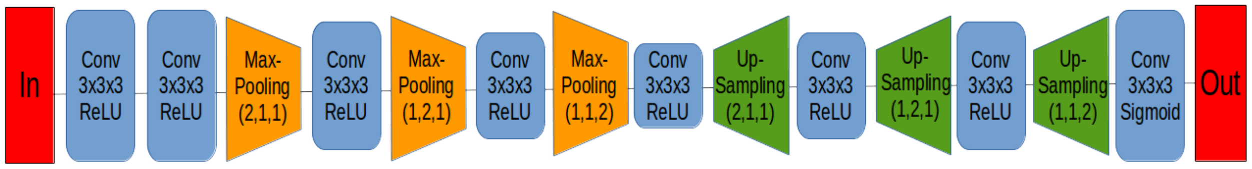

Fig. 12. Schematic of the architecture using a progressive pooling and up-sampling

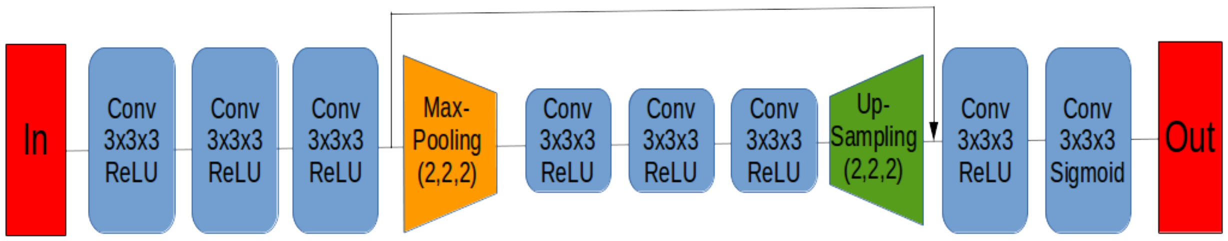

Fig. 13. Schematic of the architecture using a residual connection

the dataset on the results, we tried training it on only eleven patients. In order to have comparable data, a similar number of patches were extracted from the first ten patients as was from the ninety, by setting the extraction step to 8 (versus 16 previously). In total, 84 890 patches were obtained in this case, against 89 018 previously. The validation was done on the same patches from the ten last patients as in the prior experiments.

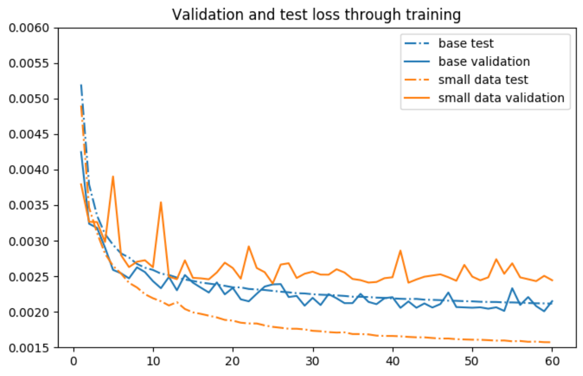

Fig. 14. Results of training for 60 epochs on the base model and the same model on a smaller dataset (11 images). Test = training loss and Validation = validation loss, on the same validation data

The results, seen in Figure 14, show that the network performs significantly better on the validation data when it is trained on more patients. Furthermore, contrary to the base experiment where test and validation data remain nearly identical throughout training, the network exhibits overfitting when trained on the smaller dataset.

Overfitting describes the phenomenon where a Neural Network adapts too strongly to its input data, that isn't general enough. For example, a classifier that learns to distinguish between cats and dogs, but is only trained on images of black dogs and white cats will probably identify a black cat as a dog, or a white dog as a cat. To prevent this problem, Neural Networks should be trained on datasets that are as general as possible. It can be recognized by the divergence of training and validation loss throughout the training process.

In this case, it seems like eleven patients is not enough to build a robust generalization for this network. Results on the small dataset could surely be improved by using techniques such as kernel regularization [3], or dropout [18], to prevent overfitting, but the results will always remain less good than that of the same network trained on a bigger dataset. We applied both these techniques to the base network, but on the larger dataset they showed worse convergence, so they were not used in further networks.

The best way of improving results with scarce training data is probably through unsupervised pre-training, where the network can learn to familiarize itself with the type of data it will process. This has shown to act as a regularizer, and prevent over-fitting [6]. Contrary to dropout and kernel regularization, this should also not negatively affect results on a larger dataset, but we did not implement such a training scheme, due to lack of time.

4 Result analysis

In this section, we will compare the synthesized T2 contrasts of our method, after training, to the synthetic T2 images produced by another state-of-the-art method, MIMECS [2], that uses an example-based approach. In order to synthesize contrasts, example based approaches use an atlas, consisting of an mri image with the original contrast, and another one with the desired contrast for a same patient, to build a dictionary that maps patches from the image with one contrast to the other. This dictionary is then used to transform patches from the input image into the desired contrast.

A comparison between both synthetic T2-w contrasts, the ground truth and the input images can be seen in Figure 2. The output of our method should be distinguishably more accurate, which is also numerically measurable through the mean squared error: our method outperforms MIMECS on the validation dataset (the ten last patients), even after training for only one epoch (Figure 3).

4.1 Numerical comparison

The performance of two of the networks (base model and deep) was numerically assessed, after 60 epochs of training, as well as that of MIMECS, on the validation data (ten last patients). For this, three metrics were used: the mean absolute error, the mean squared error, and the signal-to-noise ratio. This last metric has the advantage of not being scale-dependent: if the intensity of all pixels is doubled, both in the synthetic image, and the ground truth image, the signal-to-noise ratio will not change, whereas the mean absolute error will double, and the mean squared error will quadruple. This makes it a viable basis for comparison, even when the intensities are not normalized to the same values. The signal-to-noise ratio of an image is given by the following formula, where $mean$ is the mean intensity value.

$$SNR = 10 \times \log_{10}(\frac{mean^2}{mse}) \tag{10}$$

All metrics are calculated only on the voxels of the image that are part of the brain, that are identified by a non-null intensity values.

TABLE 1
Numerical comparison of two neural networks and MIMECS on ten patients (all values averages)

Method	MAE	MSE	SNR
Base	0.02943	0.002585	16.55
Deep	0.02724	0.002349	16.80
MIMECS	0.04506	0.007612	11.65

It is important to note that, since the mean squared error was used as loss function, the neural networks will favor reducing it, and there is an inherent bias in using this metric (or the signal to noise ratio that is based on it) as a basis for comparison with other methods. It is however linked with perceptual quality, and only diverges significantly from the prior in some cases where, for example, two images differ only through their shading, leading to a large mean squared error, even though both have great structural similarity.

4.2 Error localization

To make sure no such effect is biasing the results, an error map was computed between the output of the base neural network and the ground truth: every voxel has for intensity the absolute difference between its intensities in both images. The same error map was computed between the MIMECS synthetic image and the ground truth, both using the same scale between 0 and 0.2 (Figure 15). As a reminder, the intensity of the ground truth is normalized for the mean to be of 1/3.

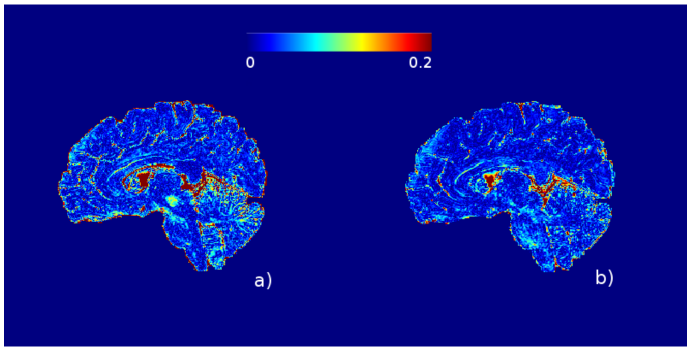

Fig. 15. Absolute difference between output and ground truth. a) MIMECS, b) Base neural network

These maps show that the error localization is similar for both methods, with a large part concentrated around the ventricles. Our method however seems to be more accurate on every part of the brain than MIMECS, and also shows noticeably less errors around the gray matter / white matter frontier (nearly not apparent in the second error map, contrary to the first).

This indicates that the better scores achieved by the neural networks are gained (at least partially) through more visually accurate image synthesis, and not only some measurement bias that favors them. It also shows that there is still significant room for improvement, since a large part of the error is not only due to noise (in which case the error map would be more uniform), but concentrated in parts where the network fails to transform the contrasts with high precision.

4.3 Running time

Even though training the neural network may take a significant amount of time (nearly fourteen hours to train the base model for 60 epochs), it is highly efficient when synthesizing contrasts from input: ten contrasts are processed in only a couple of minutes, putting the running time at a dozen seconds per image. On the other hand, example based approaches don't require any separate training phase, but are significantly slower when it comes to synthesizing a new contrast: it takes MIMECS over an hour on our hardware to process one single input contrast. This is several orders of magnitude longer than the base network.

It is also worth noting that in our experiments, even one epoch of training was sufficient to outperform MIMECS, which means one could theoretically train and use the network in less time than it takes to synthesize a contrast using MIMECS, while producing better output. This should however be nuanced by the fact that the neural network was running on a powerful GPU, thanks to deep learning libraries that implemented efficient parallelized computing for neural networks, whereas MIMECS was only running on the CPU.

5 Conclusion

The results indicate that deep learning, and in particular convolutional auto-encoders, are very well suited for the problem of MRI contrast synthesis. Many recently developed techniques in this field have also contributed to making the Neural Network perform better: the use of Rectified Linear Units [13], ADAM optimizer [12], an adapted initialization scheme [9] and Batch Normalization [11], which also makes us believe that future advances in the general field of Deep Learning will likely further improve upon the results we were able to achieve.

The main limitation of this technique is the requirement of having a large training dataset, but this can probably be partially overcome through additional work on the network or by developing a more efficient training scheme, that takes advantage of the robustness of unsupervised pre-training.

This method also seems more apt to benefit from increases in computing power, allowing for both deeper and wider networks, than example-based approaches, that often converge as of a given processing time, and require modifications to the method to provide significant improvement in results. As has been the case for other computer vision problems, we further conjecture that more specialized and complex architectures may perform better on this problem.

Finally, with the advantages of better performance and running time, at least on our data, and with more prospects for improvement, deep learning seems like the best method for modality synthesis. These improvements upon current state-of-the-art methods may make the use of synthetic contrasts more widespread, as it becomes an increasingly viable replacement for acquired MRI data. The effectiveness of synthetic data produced through this method for common medical applications (e.g. intermodality analysis) is definitely a problem worthy of further investigation.

REFERENCES

[1] H. C. Burger, C. J. Schuler, and S. Harmeling. Image denoising: Can plain neural networks compete with bm3d? In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2392–2399, June 2012.

[2] François Chollet et al. Keras, 2015.

[3] Corinna Cortes, Mehryar Mohri, and Afshin Rostamizadeh. L2 regularization for learning kernels. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, UAI '09, pages 109–116, Arlington, Virginia, United States, 2009. AUAI Press.

[4] C. Dong, C. C. Loy, K. He, and X. Tang. Image super-resolution using deep convolutional networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(2):295–307, Feb 2016.

[5] Ronen Eldan and Ohad Shamir. The power of depth for feedforward neural networks. 12 2015.

[6] Dumitru Erhan, Aaron Courville, Yoshua Bengio, and Pascal Vincent. Why does unsupervised pre-training help deep learning? In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 201–208, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[7] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[8] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. *ArXiv e-prints*, December 2015.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV '15, pages 1026–1034, Washington, DC, USA, 2015. IEEE Computer Society.

[10] Juan Eugenio Iglesias, Ender Konukoglu, Darko Zikic, Ben Glocker, Koenraad Van Leemput, and Bruce Fischl. Is synthesizing mri contrast useful for inter-modality analysis? *Medical image computing and computer-assisted intervention : MICCAI ... International Conference on Medical Image Computing and Computer-Assisted Intervention*, 16 Pt 1:631–8, 2013.

[11] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, 07–09 Jul 2015. PMLR.

[12] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[13] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems*, NIPS'12, pages 1097–1105, USA, 2012. Curran Associates Inc.

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[15] D. J. Kroon and C. H. Slump. Mri modalitiy transformation in demon registration. In *2009 IEEE International Symposium on Biomedical Imaging: From Nano to Macro*, pages 963–966, June 2009.

[16] S. Roy, A. Carass, and J. L. Prince. Magnetic resonance image example-based contrast synthesis. *IEEE Transactions on Medical Imaging*, 32(12):2348–2363, Dec 2013.

[17] Snehashis Roy, Yi-Yu Chou, Amod Jog, John A. Butman, and Dzung L. Pham. Patch based synthesis of whole head mr images: Application to epi distortion correction. *Simulation and synthesis in medical imaging : first International Workshop, SASHIMI 2016, held in conjunction with MICCAI 2016, Athens, Greece, October 21, 2016, Proceedings. SASHIMI (Workshop)*, 9968:146–156, 2016.

[18] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[19] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016.

[20] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Mach. Learn. Res.*, 11:3371–3408, December 2010.

APPENDIX

I mainly worked alone on the contrast synthesis problem. The data (two registered contrasts for all 100 patients as well as the brainmasks) were provided by my supervisor. Another PhD student of my supervisor (Chi-Hieu Pham) worked on MRI image super-resolution, and also implemented a network for contrast synthesis as part of his research. Other PhD students that were in the same room as I, worked on different medical imaging problems (registration problems for example). Chi-Hieu helped me learn to use some of the software to visualize the data (3D mri images viewed with ITK-snap). I also attended two reading group sessions in which articles about machine learning were discussed (after reading them independently), and several oral presentations : four given by candidates for a new position about the research they had accomplished, on information processing, and one given by a PhD student for a conference, about his work on dynamic ankle registration. This gave me a small overview of the numerous problems in both machine learning and medical imaging. Discussions over lunch and coffee breaks were also an opportunity to learn more about what is done in these fields, and I received several helpful suggestions and article recommendations. The impression I got of the academic world during this internship is very positive. Even though I was already familiar with it (bot my parents are scientists), I enjoyed the freedom to explore a research area and tackle problems as I deemed fit, while always being able to rely on the help of my supervisor or others scientists or students when needed. This also contributed to a very friendly and constructive atmosphere in the laboratory.