Lipschitz Constrained Convolutional Neural Networks for Plug-and-Play Methods

Sébastien Emery , Pakshal Bohra, Michael Unser
Lab Immersion I BIO-501
Biomedical Imaging Group, EPFL, Switzerland

***Abstract*—Variational methods have been widely used to solve inverse problems such as the ones encountered in image reconstruction. Alternating direction of multipliers (ADMM) is one such method used to solve those problems. Its modular structure allows to plug any off-the-shelf image denoising algorithm for a subproblem, it is then called Plug-and-Play ADMM. This kind of methods has encountered great empirical success in numerous applications. However, the conditions under which such algorithms are guaranteed to converge are less well-known. In this work, we are interested in extending the work of Ryu and al. [1] where they proved the convergence of plug-and-play methods using modern denoising convolutional networks where the global lipschitz constant is upper-bounded by 1 and a strong assumption on the data-fidelity term. We rather want to build a firmly expansive convolutional network to avoid strong assumptions on the data-fidelity while being guaranteed to converge. To do so, we will study the lipschitz constant of our networks to then build firmly expansive ones and test them in a Plug-and-Play framework. We will also add the use of Deep spline activations [2] instead of ReLu, which is a theory developed at the laboratory.**

I. Introduction

Many image reconstructions tasks try to recover an image $\mathbf{x} \in \mathbb{R}^n$ from a vector of measurements $\mathbf{y} \in \mathbb{R}^m$ which have been corrupted with some noise $\mathbf{n} \in \mathbb{R}^m$ after the image has been transformed through a forward model $\mathbf{H} \in \mathbb{R}^{mxn}$.

$$\mathbf{y} = \mathbf{H}\mathbf{x} + \mathbf{n} \tag{1}$$

This kind of problem is non-deterministic and often ill-posed [3]. Variational methods are a classical way of solving such problems. The idea is to solve it through minimization of an expression where the recovered image $\hat{\mathbf{x}} \in \mathbb{R}^n$ "best explains" the measurements. We can formulate this problem as a maximum-a-posteriori (MAP) estimation, where we want to maximize the posterior probability:

$$\begin{aligned} \hat{\mathbf{x}} &= \underset{x}{\operatorname{argmax}}\, p(\mathbf{x}|\mathbf{y}) \\ &= \underset{x}{\operatorname{argmin}} - \log p(\mathbf{y}|\mathbf{x}) - \log p(\mathbf{x}) \end{aligned} \tag{2}$$

for a conditional probability $p(\mathbf{y}|\mathbf{x})$ defining the forward imaging model and a prior density $p(\mathbf{x})$ on the image probability distribution.

It is straightforward to see that solving (2) is equivalent to the following optimization problem :

$$\hat{\mathbf{x}} = \underset{x}{\operatorname{argmin}}\, f(\mathbf{x}) + \gamma g(\mathbf{x}) \tag{3}$$

where $f(\mathbf{x}) = -\log p(\mathbf{y}|\mathbf{x})$, $g(\mathbf{x}) = -\log p(\mathbf{x})$ and $\gamma \geqslant 0$. This probabilistic point of view in (2) motivates the use of general uncontrained minimization problem as in (3), thus the variational methods.

In this work, we will focus on classical optimization algorithms such as alternating direction of multipliers (ADMM) and forward backward-splitting (FBS).

The idea of ADMM is to turn an unconstrained problem (2) into a constrained one :

$$(\hat{\mathbf{x}}, \hat{\mathbf{v}}) = \underset{x,v}{\operatorname{argmin}}\, f(\mathbf{x}) + \gamma g(\mathbf{v}), \text{subject to } \mathbf{x} = \mathbf{v} \tag{4}$$

We can then equivalently consider the augmented Lagrangian of (4) turning the constrained problem into an unconstrained one and solve it. It has been shown in [4] and [5], that minimizing Lagrangian function can be done by solving iteratively a series of three subproblems:

$$x^{k+1} = \underset{x}{\operatorname{argmin}}\, \sigma^2 g(x) + 1/2||x - (v^k - u^k)||_2^2 \tag{5}$$

$$v^{k+1} = \underset{v}{\operatorname{argmin}}\, \alpha f(v) + 1/2||v - x^{k+1} - u^k||_2^2 \tag{6}$$

$$u^{k+1} = u^k + x^{k+1} - v^{k+1} \tag{7}$$

with $\sigma^2 = \alpha\gamma$.

We can define the proximal operator of a function $h \in \mathbb{R}^n$ with $\alpha > 0$:

$$\mathrm{Prox}_{\alpha \mathrm{h}}(\mathrm{z}) = \underset{x}{\mathrm{argmin}}\{\alpha h(x) + (1/2)||x - z||_2^2\} \quad (8)$$

which is well defined if h is proper, closed and convex. Then we rewrite the ADMM algorithm as :

$$x^{k+1} = \mathrm{Prox}_{\sigma^2 \mathrm{g}}(v^k - u^k) \quad (9)$$
$$v^{k+1} = \mathrm{Prox}_{\alpha \mathrm{f}}(x^{k+1} + u^k) \quad (10)$$
$$u^{k+1} = u^k + x^{k+1} - v^{k+1} \quad (11)$$

This iterative scheme converges, if f and g follows the same mild assumptions for a well-defined proximal operator as stated above [5].

The subroutine (9) can be can be interpreted as a denoising step :

$$\mathrm{Prox}_{\sigma^2 \mathrm{g}} : \text{noisy image} \rightarrow \text{less noisy image}$$

In fact, by multiplying (5) by $\frac{1}{\sigma^2}$ and let $g(x) = ||x||_{TV}$, it becomes equivalent to the standard total variation denoising problem [6] or as stated in [7], we can interpret it as an MAP optimization for recovering a corrupted signal with additive Gaussian noise σ^2.

While the subroutine (10) can be seen as a consistency step :

$$\mathrm{Prox}_{\alpha \mathrm{f}} : \text{less consistent} \rightarrow \text{more consistent with data}$$

The idea of plug-and-play (PnP) methods is to replace the denoising step originating from the optimization problem by state-of-the-art ones with great empirical performance [7], [8] such as convolutional neural networks (CNNs) [9]. In fact, we will restrict our study to such denoisers. The denoiser $H_\sigma : \mathbb{R}^n \rightarrow \mathbb{R}^n$ has still the same interpretation:

$$H_\sigma : \text{noisy image} \rightarrow \text{less noisy image}$$

where $\sigma > 0$ is a noise parameter with greater value of sigma correspond to more agressive denoising.

The plug-and-play ADMM iterative scheme can be written as :

$$x^{k+1} = H_\sigma(v^k - u^k) \quad (12)$$
$$v^{k+1} = \mathrm{Prox}_{\alpha \mathrm{f}}(x^{k+1} + u^k) \quad (13)$$
$$u^{k+1} = u^k + x^{k+1} - v^{k+1} \quad (14)$$

While having great empirical performance, theory about convergence of plug-and-play methods is still insufficient. We will begin with the theory developed by Ryu et al [1] on the use of CNNs with lipschitz-constant upperbounded by 1 and strong assumption on $f(x)$ as a reference and try to develop more restrictive CNNs which allows to soften the assumptions on $f(x)$ and obtain convergence for a broader class of inverse problem. We will also try to incorporate Deep spline activations [2] to outperform the current reference model using rectified linear units (ReLu) activations.

II. Plug-and-play methods

We will consider two PnP methods for our experiments as done by [1]. The first one is PnP-ADMM and was already described previously. The second one is PnP-FBS which was already mentioned, the iterative scheme is as followed :

$$x^{k+1} = H_\sigma(I - \alpha \nabla f)(x^k) \quad (15)$$

for any $\alpha > 0$. An important point is that even if PnP-FBS and PnP-ADMM are two distinct methods, they share the same fixed point [8] (remark 3.1).

A. *Fixed point iteration*

We can interpret both PnP-ADMM or PnP-FBS as fixed-point iteration problem, where the fixed point(s) are the solution(s) of (3).

We can say that x^* is a fixed point of PnP-FBS if :

$$x^* = H_\sigma(I - \alpha \nabla f)(x^*)$$

is satisfied. Analogously for PnP-ADMM, (x^*, u^*) is a fixed point if:

$$x^* = H_\sigma(x^* - u^*)$$
$$x^* = \mathrm{Prox}_{\alpha \mathrm{f}}(x^* + u^*)$$

The question now is to know when, these iterative schemes converge to the fixed point.

B. *Definitions*

To understand the fixed point convergence, we first need to introduce a few definitions.

***Definition** 2.1 (Lipschitz):* A function $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ is called *Lipschitz continuous* if there exists a constant L such that :

$$||f(x) - f(y)||_2 \leqslant L||x - y||_2 \quad \forall x, y \in \mathbb{R}^n$$

The smallest L satisfying the inequality above is called the Lipschitz constant of *f*.

***Remark** 1:* We will only consider the *l2-norm* for the entire report.

***Definition** 2.2 (Non-expansive):* An operator $R : \mathbb{H} \to \mathbb{H}$ is said to be *non-expansive* if it satisfies the following inequality :

$$||R(x) - R(y)||_2 \leqslant ||x - y||_2 \quad \forall x, y \in H^2$$

i.e. the operator has a lipschitz constant of 1

***Definition** 2.3 (α-averaged):* An operator T is said to be *α-averaged* if it can be written as :

$$T = (1 - \alpha)I - \alpha R$$

where R is a non-expansive operator.

***Definition** 2.4 (firmly non-expansive):* An operator T is said to be *$\frac{1}{2}$-averaged* if it can be written as :

$$T = \frac{1}{2}(I + R)$$

where R is a non-expansive operator.

C. Fixed point convergence

We will now present the conditions under which the fixed point iteration is *guaranteed*.

Standard tools of monotone operator theory tells us that for PnP-FBS the denoiser needs to be an averaged operator and it is even more restrictive for PnP-ADMM which needs an $\frac{1}{2}$-averaged denoiser. The two following theorems were stated in [1] and [10].

***Theorem** 2.1:* Given an inverse problem of the same form as (4) and using the PnP-FBS algorithm given in (15) to solve it. The iterative scheme converges, if the denoiser can be written as:

$$H_\sigma = (1 - \alpha)I - \alpha R$$

i.e. it is an averaged operator

***Theorem** 2.2:* Given an inverse problem of the same form as (4) and using the PnP-ADMM algorithm given in (12-14) to solve it. The iterative scheme converges, if the denoiser can be written as:

$$H_\sigma = \frac{1}{2}(I + R)$$

i.e. it is an $\frac{1}{2}$-averaged operator

However, it has been demonstrated that most of the state-of-the-art denoisers does not fulfill either conditions. Chan et al. [4] demonstrated that the NLM denoiser is not non-expansive for example. Thus Ryu et al. [1] made a more reasonable assumption on $H_\sigma : \mathbb{R}^n \to \mathbb{R}^n$:

$$||(H_\sigma - I)(x) - (H_\sigma - I)(y)||_2^2 \leqslant \epsilon^2 ||x - y||_2^2 \quad (16)$$

$\forall x, y \in \mathbb{R}^n$ and for some $\epsilon \geqslant 0$. Since σ control the strength of the denoising, we can expect H_σ to be close of the identity for small σ. Under this assumption, it can be shown [1], that both iterative scheme denoted as $T : \mathbb{R}^n \to \mathbb{R}^n$ are contractive :

$$||T(x) - T(y)||_2 \leqslant \delta ||x - y||_2 \quad \forall x, y \in \mathbb{R}^n$$

for some $\delta < 1$. If x^* satisfies $T(x^*) = x^*$, then $x^k \to x^*$ geometrically by the classical Banach contraction principle. The two following theorems have been derived by Ryu et al. [1] to demonstrate the contractiveness of the methods :

***Theorem** 2.3:* Assume H_σ satisfies (16) for some $\epsilon \geqslant 0$. Assume f is μ-strongly convex, differentiable and ∇f is L-lipschitz, then PnP-FBS is a contraction, thus converges

***Theorem** 2.4:* Assume H_σ satisfies (16) for some $\epsilon \geqslant 0$. Assume f is μ-strongly convex then PnP-ADMM is a contraction, thus converges.

Assumptions made in theorems above are satisfied in problems such as image denoising and single photon imaging, but are not fullfilled in compress sensing, sparse interpolation and super-resolution. To address the convergence problem due to assumption's violation, we aim to construct $\frac{1}{2}$-averaged denoiser. To do that, we will study how to build denoising CNNs under which the iterative scheme chosen will converge using one of the theorem (2.1-2-4) stated in this section.

D. Denoiser Scaling

Many state-of-the-art denoisers such as CNNs do not have an intrinsic parameter related to the noise measurement's variance ($\mathbf{n}$). Thus, setting an optimal noise variance σ to train the CNN on, requires training multiple networks for different σ levels. To avoid retraining multiple CNNs at different noise level, [11] proposed and justified a scaled version of the denoiser :

$$D_\mu(z) = \frac{1}{\mu} D(\mu z)\,, z \in \mathbb{R}^n \quad (17)$$

where σ is omitted, $D : \mathbb{R}^n \to \mathbb{R}^n$ is the denoiser and $\mu \geqslant 0$ is the scaling parameter. This allows to control the strength of the noise within the PnP method, by only training once the denoiser on a initial guess or taking a pretrained one.

III. Convolutional Neural Networks

We will now try to train denoising CNNs that satisfy either assumption (16) or that are $\frac{1}{2}$-averaged. This will ensure by theorem (2.2) or theorem (2.3 and 2.4) with additional assumptions the convergence of the iterative schemes.

A. *CNN architecture*

We choose to use the same architecture as [1], which uses the **DnCNN** [9], which is a state-of-the-art denoising CNN on natural images. The classical architecture consists of 17-layer CNN with a residual mapping (Fig. 1). Given a noisy input $y = x + e$ with x being the clean image and e the noise, the the residual mapping outputs the noise, ie $R(y) = e$, thus $y - R(y)$ gives the clean image. We will also use what we call the $\frac{1}{2}$-averaged mapping, which we will justify later. The output is $R(y) = -e + x$, so that the $\frac{y+R(y)}{2}$ gives the clean image.

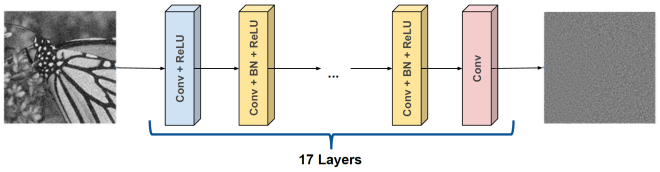

Fig. 1: Residual DnCNN Network Architecture, convolution: conv, activation: Relu, batch normalization: BN. Image taken from [1].

We also use the smaller network called SimpleCNN, which consists of 4-layer CNN without batch norm.

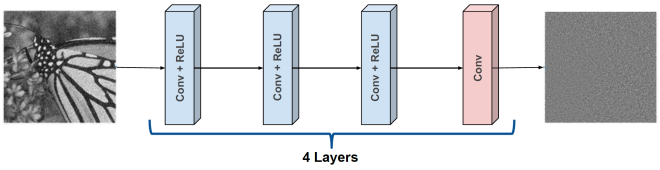

Fig. 2: Residual SimpleCNN Network Architecture, convolution: conv, activation: Relu. Image taken from [1].

We will also use two variants, we called DnCNN-Bspline and SimpleCNN-Bspline, where we replace the ReLU activation by Deep spline activations [2], a theory developed in the laboratory.

B. *PnP convergence of CNN*

The convergence of the PnP algorithms using CNN denoisers requires to satisfy either assumption (16) along with assumptions on $f(x)$ or the CNN needs to be an $\frac{1}{2}$-averaged operator.

The first strategy is to enforce the denoiser to satisfy assumption (16), while assuming the inverse problem of interest has a data-fidelity term which satisfy the required assumptions. To do so, we can rewrite the residual mapping $H_\sigma(y) = y - R(y)$ as $R(y) = y - H_\sigma(y) = (I - H_\sigma)(y)$, thus enforcing assumption (16) can be achieved by controlling the lipschitz-constant of the CNN ($R(y)$).

The second method is to enforce the denoiser to be firmly non-expansive. A strategy demonstrated in [12] state that a firmly non-expansive operator, can be constructed from a non-expansive network, ie. a 1-lipschitz operator as stated in theorem (2.2). If we can write the denoiser as $H_\sigma = \frac{1}{2}(I + R)(y)$ where $R(y)$ is non-expansive, then H_σ is firmly non-expansive. It is easy to see that rewriting the expression as $H_\sigma = \frac{1}{2}(y+R(y))$, we found the so-called $\frac{1}{2}$-averaged mapping mentioned earlier, where $R(y)$ is the output of the network, thus training an $\frac{1}{2}$-averaged mapping, while constraining the lipschitz-constant of the network will, give an $\frac{1}{2}$-averaged CNN.

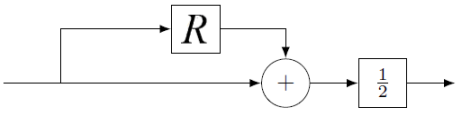

Fig. 3: Architecture of a firmly non-expansive network given a non-expansive network $\mathbf{R}$. Image taken from [12].

C. *Lipschitz-constant of CNN*

Restricting the lipschitz-constant of the CNN ensures the convergence of the PnP algorithms using either of the two mappings mentioned earlier. We will now describe theoretical results to establish an upperbound for CNNs which were stated in [13].

Firstly, we rewrite a CNN as the composition of its layers :

$$f(x) = f_n \circ f_{n-1} \circ ... \circ f_1(x)$$

Then, the following theorem holds for the lipschitz-constant of a composition.

***Theorem** 3.1:* Let $f_1, f_2, ..., f_n$ be functions with respective Lipschitz constants $L_1, L_2, ..., L_n$. For a composition $g = f_1 \circ f_2 \circ ... \circ f_n$, the lipschitz constant is upperbounded by the product $\prod_{i=1}^{n} L_i$.

Thus, controlling the global lipschitz-constant of the CNN to be upperbounded by 1, can be achieved by controlling each layer individually [14]. Now, we will describe different types of layer constituting CNN.

The following theorems for fully-connected (3.2) and convolutional layers (3.3) holds according to definition 2.1 and remark 1.

***Theorem** 3.2:* The Lipschitz constant of an affine mapping $W : \mathbb{R}^m \to \mathbb{R}^n$ written as $W(x) = Wx + b$ is the largest singular value of W.

***Theorem** 3.3:* The Lipschitz constant of $W : \mathbb{R}^{n \text{x} c_{in}} \to \mathbb{R}^{n \text{x} c_{out}}, W \in \mathbb{R}^{c_{out} \text{x} c_{in} (2d+1)}$, with kernel size (2d+1) and weight tensor $w \in \mathbb{R}^{(2d+1) \text{x} c_{in} \text{x} c_{out}}$ is bounded by $\sqrt{(2d+1)}\sigma_1$ where σ_1 is the largest singular value of the unfolded weight matrix W, for a 1-dimensional input $\text{x} \in \mathbb{R}^{n \text{x} c_{in}}$

For each additional kernel dimension, we multiply the upperbound given in theorem 3.3 by the size of the additional dimension, ie. $\sqrt{(2d+1)}$ for a square kernel, which gives the following upperbound : $\sqrt{(2d+1)}^2 \sigma_1$.

The lipschitz-constant of the classical activations such as ReLu, hyperbolic tangent and sigmoid is their maximum slope as they are one-dimensional functions. We can easily verify that for the three activations the slopes are $\leqslant 1$.

D. Lipschitz-constrained CNN

In the previous section, we have established an upperbound on the lipschitz-constant of a CNN as the product of the individual upperbound of each layer and activation. We will now see how to practically constrain it.

1) Spectral Normalization

Given a CNN using ReLu non-linearities, the global lipschitz can be controlled by controlling the the upperbound layer-wise as stated above, in [15] and [14]. As stated in theorem 3.2 and 3.3, controlling the upperbound involve computing the highest singular value of the matrix W. To avoid the cost of computing the SVD decomposition, spectral normalization (SN) rather approximate the highest singular value of the weight matrix by power iteration as done in [1].

Given a weight matrix, $W_l \in \mathbb{R}^{m \text{x} n}$ of the l-th layer, vectors $u_l \in \mathbb{R}^m$, $v_l \in \mathbb{R}^n$ initialized randomly maintained in memory to estimate the leading first left and right singular vector of W_l. At each forward pass, SN is applied to all layers ($1 \leqslant l \leqslant L$):

Power iteration

1. Apply one step to update u_l and v_l :
   $v_l \leftarrow W_l^T u_l / ||W_l^T u_l||_2, u_l \leftarrow W_l v_l / ||W_l u_l||_2$
2. Normalize W_l with the estimated spectral norm :
   $W_l \leftarrow W_l^T / \sigma(W_l)$, where $\sigma(W_l) = u_l^T W_l v_l$

2) Real Spectral Normalization

While SN can be applied directly to fully-connected layers, it can't be for convolutional layers. As stated in theorem 3.3, the upperbound take into account the reshaping of of the 4-dimensional weight kernel (C_{out}, C_{in}, h, w) into $(C_{out}, C_{in} \text{x} h \text{x} w)$, thus the effective upperbound according to (3.3) is $(\sqrt{h} * \sqrt{w} * \sigma_1)$, where σ_1 is the highest singular value of the reshaped kernel. Thus, Ryu et al. [1] proposed a kernel version of the power iteration to estimate the true upperbound.

Given a convolutional $K_l : \mathbb{R}^{C_{in} \text{x} h \text{x} w} \to \mathbb{R}^{C_{out} \text{x} h \text{x} w}$ and its conjugate operator K_l^*, Real SN maintains $U_l \in \mathbb{R}^{C_{out} \text{x} h \text{x} w}$ and $V_l \in \mathbb{R}^{C_{in} \text{x} h \text{x} w}$ to estimate the leading first left and right singular vectors. Thus, at each forward pass the RealSN applies for each layer the two following steps:

Kernel power iteration

1. Apply one step of the power method with K_l :
   $V_l \leftarrow K_l^* U_l / ||K_l^* U_l||_2$
   $U_l \leftarrow K_l V_l / ||K_l V_l||_2$
2. Normalize K_l with the estimated spectral norm :
   $K_l \leftarrow K_l / \sigma(K_l)$, where $\sigma(K_l) = \langle U_l, K_l V_l \rangle$

3) Parseval Normalization

Another possibility to constrain the lipschitz upperbound of the CNN is called Parseval normalization (PN). The idea is to keep the rows of the weight matrix $W \in \mathbb{R}^{C_{out}, C_{in} \text{x} h \text{x} w}$ orthogonal, ie. the matrix W is orthonormal. Thus, the singular values of the matrix are all equal to 1. To do that, [16] proposed to update W, after each main update of the network. However, it is applied to the reshaped 2-dimensional kernel, thus we also need to divide by the the product of the square root of the kernel dimension, ie. $\frac{1}{\sqrt{h} * \sqrt{w}}$. Thus, after each main update of the backward pass, we freeze the network gradient and do the following steps :

Parseval Normalization

1. Reshape the 4-dimensional kernel in 2D :
$$W_l \leftarrow K_l$$
2. Update W_l to keep the rows orthogonal:
$$W_l \leftarrow (1+\beta)W_l - \beta W_l W_l^T W l$$
where $\beta \geqslant 0$ is a parameter that controls the strength of the penalty
3. Rescale W_l :
$$W_l \leftarrow \frac{1}{\sqrt{h}*\sqrt{w}} W_l$$
4. Reshape back W_l in 4-dimension:
$$K_l \leftarrow W_l$$

E. *Deep Spline*

Deep spline activations is a theory developed at the laboratory from the observation that ReLu CNN are global continuous and piecewise linear (CPWL). The goal is adjust neuron-wise the activations, while enforcing them to be CPWL. This will augment the capacity of the network and thus hopefully yield better results. In the following, we will briefly describe theoretical aspects, but refer to [2] for a more complete work.

The idea is to add a second-order-total-variation regularization for each learnable activation on the cost function. Using Unser's representer theorem for DNNs, the optimal activation form is a sum of individual adaptative piecewise-linear functions, which size is unknown prior to the optimization. Setting a uniform grid of knots with step size T and transforming the adaptative piecewise-linear representation to B-splines of degree one:

$$\beta^1(x) = \begin{cases} 1-|x| & \text{if } x \in [-1,1] \\ 0 & \text{otherwise} \end{cases} \tag{18}$$

We can represent each learnable activation as a sum of shifted B-splines of degree one (triangles on Fig.4) and linear extrapolation outside the grid. Thus we can represent each activation as vector of coefficients $c \in \mathbb{R}^{K+2}$, where K is the cardinality of the set of nodes and we add two extra triangles for the linear extrapolation outside the grid. At each forward pass, the coefficients are updated through backpropagation of the gradient.

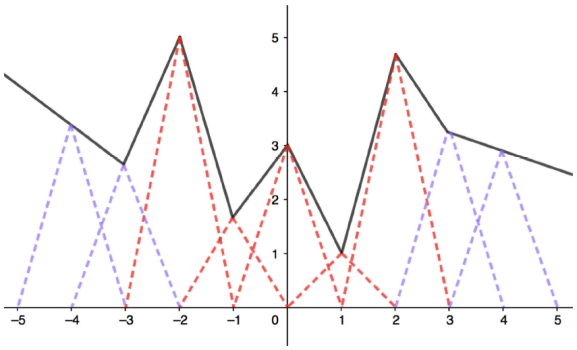

Fig. 4: B-spline activation function on a grid [-3,3] with T =1 and linear extrapolation beyond. Image taken from [2].

In our experiment, we will use per channel activations which are made up of 51 knots with step size 0.1. this means, that all neurons in a channel shared the same activation. Thus, our matrix of coefficients is of size (number of channels, number of knots).

To constrain the lipschitz-constant of the CNN, we also need to restrict the constant of the activations. For B-spline activations, each channel has a one dimensional function, which lipschitz-constant is the maximum slope of the activation. In this case, we can constrain each linear piece to have a slope $\leqslant 1$. At the end of each coefficients update through gradient backpropagation, we freeze the coefficients and apply one of the two methods presented below.

Maximum Projection

1. For m in B-spline modules **do**
   2. For n = {1,2,...,number of channels} **do**
      3. **Initialize** S list of slopes
      4. For c = {1,2,...,C-1} **do**
         5. Compute absolute slope {c,c+1} abs(s)
         6. Append s to S
      7. s* $= max(S)$
      8. **If** s* $\leqslant 1$:
         do nothing
      9. **else** s* $\geqslant 1$:
         For c = {1,2,...,C-1} **do** :
         c = c/ s*

Quadratic constrained minimization problem

1. For m in B-spline modules **do**
   2. For n = {1,2,...,number of channels} **do**
      3. $\widetilde{c} = c$ where $c \in \mathbb{R}^{K+2}$
      4. Solve $||\bar{c} - \widetilde{c}||_2^2$, subject to $D\widetilde{c} \geqslant 1$ with D the first order finite difference
      5. c = $\bar{c}$

IV. *Results and Discussion*

The results presented are splitted in three main parts. The first experiments were done to study the methods proposed to constraint the lipschitz constant of the CNNs. Then, we studied the replacement of ReLu by

B-spline activations and the efficiency of the methods to constraint their lipschitz bound. Finally, given the results of the first two experiments, we conducted the plug-and-play experiments. As parameters and datasets used can change from one experiment to another, we'll specify for each experiment what has been used. The code (in Pytorch) is available on GitHub[1].

A. *Lipschitz experiments*

The goal of this experiment is to see whether SN, RealSN and Parseval are able to upperbound the lipschitz-constant of the CNN by 1.

1) *Implementation details*

We used the **SimpleCNN** architecture with residual and $\frac{1}{2}$-averaged mappings. The dataset used was the NYU fastMRI dataset [17], where we have randomly splitted the validation set into a validation and test set equally. We followed the strategy in [18], where the MRI knee images (320x320) are cropped in four sub-images (160x160) with additive white gaussian noise (AWGN) of level σ. We used Adam optimizer with batch size 25, default learning $10e^{-3}$ for 600 epochs to ensure the convergence of the CNNs.

2) *Method comparison*

We trained CNNs using the two mapping strategies combined with the three methods mentioned above plus no normalization (referred as "none") to have a baseline comparison. The noise level was set to $\sigma = 0.05$ and the Parseval strength parameter to $\beta = 0.5$. The singular values were computed using the kernel power iteration for 100 iterations on the trained networks.

		None		SN		RealSN		Parseval	
		R	H	R	H	R	H	R	H
Singular values	Layer 1	4.81	5.9	5.00	5.2	1.03	1.04	3.0	3.0
	Layer 2	19.42	27.14	14.05	12.17	1.01	1.0	2.94	2.98
	Layer 3	19.68	11.91	11.92	9.69	1.04	1.05	2.56	2.98
	Layer 4	1.2	0.63	3.19	3.51	1.01	1.01	1.06	2.96
Test MSE		12.46	12.49	12.55	12.63	13.57	15.63	40.83	33.33

TABLE I: Singular values of the four layers in SimpleCNN for the baseline and the three normalization methods, along with the reported MSE value for the test set.

As expected, SN is indeed unable to control the bound of convolutional layer, while RealSN can. The capacity of the network from no-normalization to RealSN decrease due to the stronger constraint on the upperbound of the lipschitz constant. On the other hand, Parseval seems to be much worse, thus constraining the weight matrix to be orthonormal might even be stronger than spectral normalization. Indeed, investigating the β parameter in the next experiment, will reveal unstable training optimization. We can observe, that residual mapping gives better result than is counterpart for the first three methods on the table, but is inverted in Parseval.

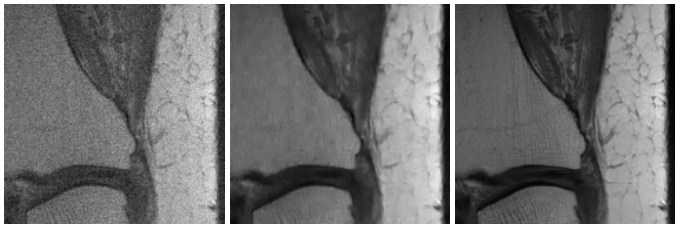

Fig. 5: Residual RealSN SimpleCNN network: From left to right (noisy input, prediction, target).

3) *Parseval strength parameter*

In the previous experiment, we set $\beta = 0.5$, we will see now non-exhaustively how the beta parameter influences the singular values.

Firstly, we set four $\beta = 0.0001, 0.1, 0.5, 0.6, 0.7, 1.0$, where the first value was used successfully in [16]. We used Adam optimizer with the same settings and architecture as in the method comparison with only the residual mapping.

Beta		0.0001	0.1	0.5	0.6	0.7	1.0
Singular values	Layer 1	6.25	3.00	3.00	3.00	NAN	NAN
	Layer 2	4.82	2.0	2.94	2.98	NAN	NAN
	Layer 3	4.24	1.65	2.56	2.96	1.88	1.34
	Layer 4	2.58	1.05	1.06	2.88	1.14	0.99
Test MSE		12.92	33.32	40.83	442.39	NAN	NAN

TABLE II: Singular values of the four layers in SimpleCNN with residual mapping for six beta values, along with the reported MSE value for the test set.

We see that increasing the parameter β from 0.0001 to 0.1 lowers the singular values, but beyond that it is not able to constraint them even stronger. However, the MSE loss on the test set indicate less capacity for higher values of β, thus the network might not be able to properly learn with a high β value. Indeed, for $\beta = 0.7$ and 1.0, the two first singular values and the MSE loss on the test set become undefined (table 2), while the training becomes unstable looking at the learning curve for $\beta = 1.0$ (Fig.6), which is similar for $\beta = 0.7$. The training loss in the first epoch is very high (explode) , while the validation loss doesn't seem to learn as the learning curve doesn't really decrease, if we look at the second scale in blue. We briefly tried to find a solution by lowering the learning rate or using SGD optimizer, but nothing seemed to correct this behaviour and constrained the constant to one. As RealSN was working, we didn't try to conduct a careful analysis to

[1]https://github.com/sebemery/Lipschitz-constrained-neural-networks

understand the instability of the training phase. However, there is room for improvement, but it will definitely more complicated than RealSN.

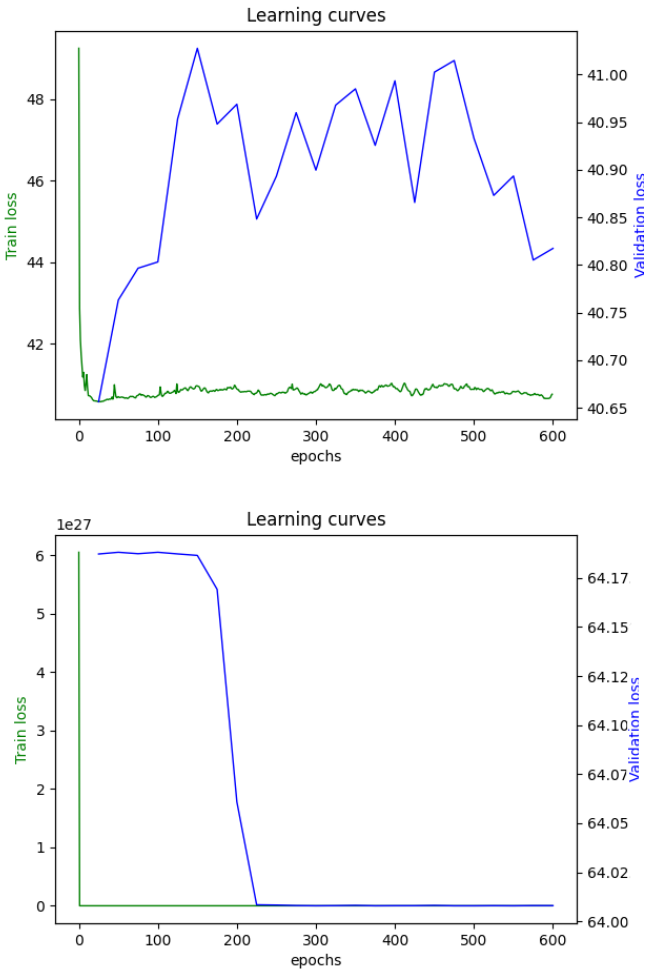

Fig. 6: Learning curves : train in green and validation in blue, the top panel is the for beta equals 0.5 and the bottom for beta equals 1.0.

B. *B-spline activation*

We aim to investigate the performance and computational cost of the two methods to constrain th lipschitz-constant of the B-spline activation, to choose the method with the best trade-off.

1) Performance of B-spline methods

In this experiment, we train **SimpleCNN** models with $\frac{1}{2}$-averaged mapping. The reason behind this choice is the training time. At this point of the project we were still using the (160x160) sub-images with $\sigma = 0.05$ from the NYU fastMRI dataset and it appears that changing the image's size (40x40) from the previous paper [2] to a bigger size, increases the computational cost non-negligibly. Thus we choose to train the mapping which appears to be harder to train from the previous experiment. All models were trained using Adam optimizer with a batch size of 6 and learning rate of $10e^{-3}$ for 600 epochs. The number of knots, the stepize, the weight decay and the hyperparameters controlling the total variation regularization were all used with the default values provided, respectively 51, 0.1, 0.005 and 0.001. For the quadratic constrained minimization problem, we use two different library to solve quadratic programming problems namely cvxpy/cvxpylayer and qpth.

After few trials on networks trained for a few epochs, we determined that the cvxpylayer library was faster to solve the optimization problem of the coefficients, thus we'll use this library for the main experiments.

Due to the expensive training time, we also tried to share the activation across the modules and the channels to diminish the number of parameters for each methods. Thus, the same activation module is called after each convolutional module and this single module is now a vector $c \in \mathbb{R}^{1x(K+2)}$, which is optimized.

Model	Test MSE	Test SNR (dB)
ReLu	**15.63**	19.75
B-spline	12.45	20.78
B-spline shared	12.5	20.76
B-spline maxproj	**15.07**	19.98
B-spline maxproj shared	170.93	9.58
B-spline orthoprojection	16.84	19.41
B-spline orthoprojection shared	27.31	17.21

TABLE III: Training performance for different activations for RealSN SimpleCNN halfaveraged models with noise input 0.05, ReLu network's batch size is 25 while B-spline is 6. Orthoprojection stands for the quadratic programming optimization, while shared refer to the activation shared across modules and channels

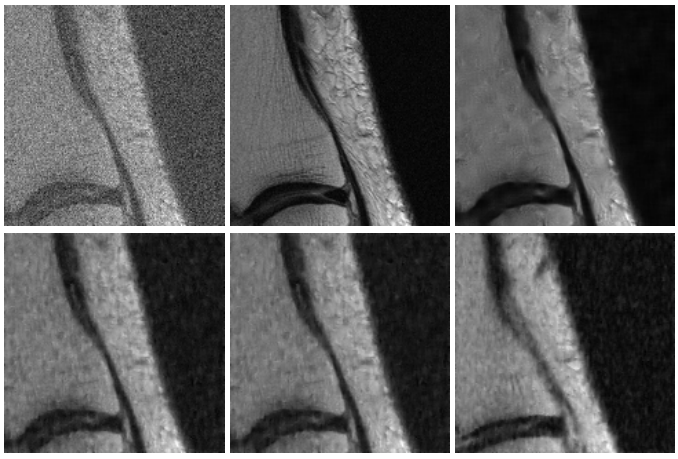

Fig. 7: KneeMRI image for the four models without sharing in table 3. Top panel : input, groundtruth , bspline. Bottom panel : ReLu, bspline maxproj, bspline orhtoprojection

Even if the ReLu network on table 3 was trained

with a higher batch size, we can see, that only the maximum projection method is able to slightly improve the performance. Surprisingly, sharing the activation seems to render the learning harder in each case, but especially for the maximum projection, this might be due to the simplicity of the method compared to quadratic optimization. So based on this experiment, maximum projection would be the method of choice, even if the B-spline models are sensitive to parameters optimization and we could get better results. However, we do not had time to do this here.

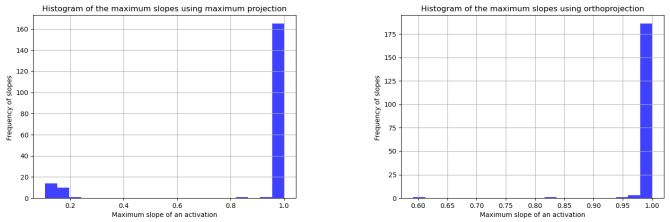

Fig. 8: Histograms of the maximum slopes for each module and each channels in the SimpleCNN architecture. We have 3 modules with 64 channels, thus 192 maximum slopes. Left panel is for the maximum projection and the right one for orthoprojection.

2) *Computational cost of B-spline methods*

In this experiment, we train **SimpleCNN** and **DnCNN** architectures with $\frac{1}{2}$-averaged mapping using Adam optimizer, for the same reason as above. We use for both standard ReLU , B-spline, B-spline maxproj and B-spline orthoprojection as activation. We use the same BSD dataset as in [1] with $\sigma = 15/255$. The batch size was set to 128 and the learning rate to $10\mathrm{e}^{-3}$ for 5 epochs to divide the training time by five and obtain the per epoch computational cost. The number of knots, the step size , the weight decay and the total variation's hyperparameters were set to the default values.

Model	SimpleCNN	DnCNN
ReLu	4min40s	7min44s
B-spline	5min49s	12min53s
B-spline maxproj	5min53s	13min7s
B-spline orthoprojection	25min19s	1h55min

TABLE IV: Computational cost per epoch of using four different activations combined with SimpleCNN and DnCNN architecture with $\frac{1}{2}$-averaged mapping.

We can see that the cost of using B-spline activations is mild using (40x40) images and that maximum projection's cost is similar to the unconstrained activations. However, especially for the DnCNN model, the cost of quadratic programming is prohibitive for us to use it in further experiments.

C. *Plug-and-Play Experiments*

Previous experiments allowed us to determine the normalization strategy and the possible benefit of using B-spline activations. We set the normalization to RealSN and whenever we use B-spline activations, we'll constraint the lipschitz bound with maximum projection.

1) *Compressed sensing MRI*

Magnetic resonance imaging (MRI) is a widely non-invasive technique used. The measurements retrieve from the machine are a sampling of the K-space of the image. However, the data acquisition is slow, thus a complete Cartesian sampling takes a long time. Compressed sensing (CS) aims to downsample the the measurements and accelerate the process. However, the lack of medical images in most cases does not allow an end-to-end training. A solution is to use the PnP framework where we plug a denoiser H_σ trained on natural images [1] and then use it on medical data.

The CS MRI mathematical measurement model is :

$$\mathrm{y} = \mathcal{F}_p \mathrm{x}_{\mathrm{true}} + \varepsilon_e \tag{19}$$

where $\mathrm{x}_{\mathrm{true}} \in \mathbb{C}^n$ is the real image, $\mathcal{F}_p : \mathbb{C}^n \to \mathbb{C}^m$ is the linear forward model (Fourier k-domain subsampling), $\mathrm{y} \in \mathbb{C}^m$ is the measured data and $\varepsilon_e \sim \mathcal{N}(0, \sigma_e I_m)$. We want to recover $\mathrm{x}_{\mathrm{true}}$ from y by minimizing the following data-fidelity term :

$$f(x) = \frac{1}{2}||\mathrm{y} - \mathcal{F}_p \mathrm{x}||_2^2 \tag{20}$$

The gradient and proximal operator of $f(x)$ are given in [19] and [20] respectively. We used random, radial and cartesian subsampling patterns (Fig. 8) with a 30% sampling rate as in [20]. The noise level σ_e is taken as 15/255.

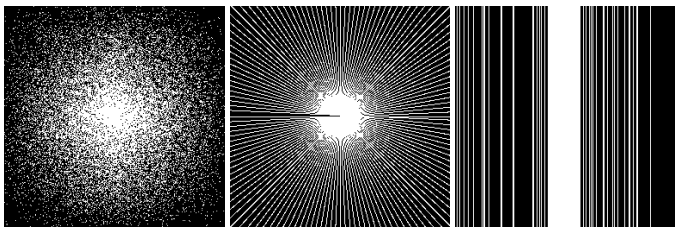

Fig. 9: Sampling patterns : On the left the random, in the middle the radial and on the right the Cartesian. Patterns were taken from experiment in [20]

We tested both PnP algorithm on a brain and bust MRI images (Fig.9) as in [1].

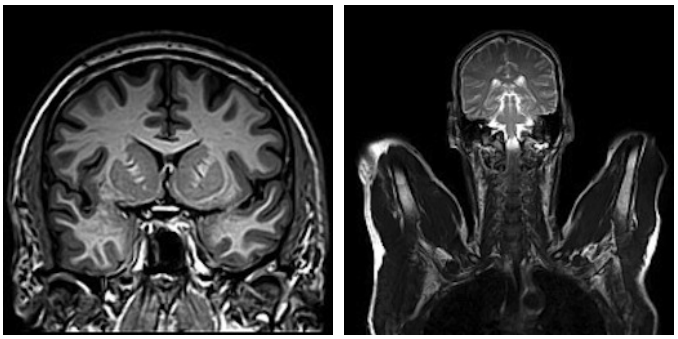

Fig. 10: MRI images: On the left a brain images and on the right a bust images. Images taken from experiment in [1].

We trained the denoisers H_σ with both **SimpleCNN** and **DnCNN** architectures combined to residual and $\frac{1}{2}$-averaged mappings. Each model was trained twice with and without bias added to convolutional layers. We used the BSD500 dataset with the same preprocessing as in [1]. Unless specified, model with bias were trained with Adam optimizer for 50 epochs with a step scheduler dividing the learning rate by 10 and beginning at 0.001. Networks without bias were trained similarly, but for 70 epochs. We used the same approach than [1] to set the noise level used to train our model. For PnP-ADMM the noise $\sigma = \sigma_e = 15/255$, while for PnP-FBS it is set as $\sigma = \sigma_e/3 = 5/255$. We took the alpha parameter given in [1] as $\alpha = 2.0$ in PnP-ADMM and $\alpha = 0.4$ in PnP-FBS. The number of iterations in both PnP algorithms is set to 100. The PSNR results are displayed on table 5 and table 6 for respectively models without and with bias and compared to the zero-filling baseline.

Surprisingly, models with residual mappings converges as in [1], although the function $f(x)$ does not satisfy assumptions in theorems 2.3 and 2.4, thus is not guaranteed to converge. As expected, our $\frac{1}{2}$-averaged networks converge. Moreover, the models converge fast as shown in Fig.11 for the PSNR and increment curves.

Results obtained for network without bias (table 5) are comparable to what was obtained in [1]. However, B-spline activations doesn't seem to consistently improve the performances compare to ReLu. Indeed, we can see on table 5, that for the residual mapping the performance is either slightly better, slightly lower or unchanged, but for the $\frac{1}{2}$-averaged models, it is consistently lower. Those networks are particularly difficult to train using the same optimizer's settings. We did not had time to tune them, but we'll see, that in table 6 were able to correct the training behaviours of these models. In ReLu models, the residual mapping seems to be a bit better than $\frac{1}{2}$-averaged apart from RealSN SimpleCNN for PnP-FBS. This is expected as the networks are even more constraint in $\frac{1}{2}$-averaged mapping. In PnP-ADMM, bigger networks tends to have similar performance, while for smaller networks unbounded lipschitz network are worse with residual and better when constrained.

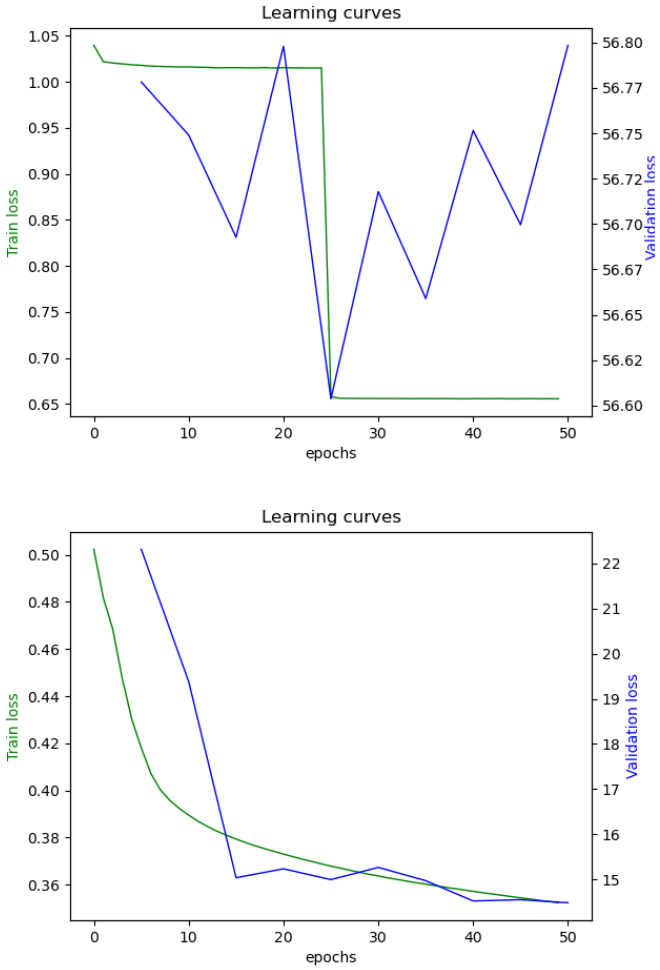

Fig. 12: Model: PnP-FBS RealSN DnCNN Bspline. Top Panel : Learning curves with Adam optimizer with lr = 0.001. Bottom panel: Learning curves with SGD optimizer with lr = 0.0001

Adding bias to the convolutional layer, seems to enhance the performance of the small network, while for bigger networks it does not bring significant improvements. In PnP-ADMM, bigger unconstrained networks are slightly better with $\frac{1}{2}$-averaged mapping, while being better with residual when constrained. For smaller network, the performance of unconstrained network is comparable for both mapping, while it is a bit better for the residual mapping when constrained. B-spline activations do not improve performance in DnCNN,but improve performance for $\frac{1}{2}$-averaged mapping in SimpleCNN.

Sampling approach		Random							
Image		Brain				Bust			
CNN architecture		R	R*	H	H*	R	R*	H	H*
Zero-filling		9.60				7.02			
PnP-FBS	DnCNN	19.75	19.75	18.61	18.61	16.67	16.67	15.25	15.25
	Real-SN DnCNN	**19.85**	**19.85**	**19.39**	**19.39**	16.67	16.67	**16.35**	**16.35**
	Real-SN DnCNN Bspline	19.77	19.77	14.43	14.70	**16.70**	**16.76**	13.56	13.72
	SimpleCNN	16.43	16.43	16.15	16.15	13.08	13.08	12.80	12.80
	Real-SN SimpleCNN	16.48	16.48	17.27	17.27	13.09	13.09	14.24	14.24
	Real-SN SimpleCNN Bspline	18.82	18.86	3.22	4.85	16.31	16.40	3.20	4.32
PnP-ADMM	DnCNN	19.82	19.94	19.83	19.92	**17.08**	**17.13**	**17.10**	**17.13**
	Real-SN DnCNN	**19.86**	**19.97**	**19.85**	**19.94**	17.05	17.10	17.04	17.08
	Real-SN DnCNN Bspline	19.78	19.92	19.31	19.36	16.95	17.01	16.90	16.91
	SimpleCNN	16.49	16.49	16.94	16.94	12.27	12.27	12.73	12.73
	Real-SN SimpleCNN	17.75	17.75	16.04	16.04	14.89	14.89	13.76	13.76
	Real-SN SimpleCNN Bspline	16.28	16.58	9.79	12.18	13.52	14.49	8.95	10.70

Sampling approach		Radial							
Image		Brain				Bust			
CNN architecture		R	R*	H	H*	R	R*	H	H*
Zero-filling		9.29				6.21			
PnP-FBS	DnCNN	19.06	19.06	18.08	18.08	16.15	16.17	14.82	14.82
	Real-SN DnCNN	19.05	19.05	**18.68**	**18.68**	16.16	16.20	**15.83**	**15.91**
	Real-SN DnCNN Bspline	**19.12**	**19.12**	14.13	14.37	**16.17**	**16.36**	12.84	13.12
	SimpleCNN	16.09	16.09	15.69	15.69	13.24	13.24	12.83	12.83
	Real-SN SimpleCNN	15.74	15.74	16.58	16.58	12.84	12.84	13.78	13.78
	Real-SN SimpleCNN Bspline	18.01	18.06	3.20	4.85	15.92	15.96	3.07	4.12
PnP-ADMM	DnCNN	18.92	19.15	18.96	19.15	16.63	16.72	**16.68**	**16.76**
	Real-SN DnCNN	18.93	**19.22**	**19.08**	**19.27**	**16.68**	**16.76**	16.65	16.75
	Real-SN DnCNN Bspline	**18.94**	19.19	18.54	18.54	16.61	16.71	16.37	16.37
	SimpleCNN	17.04	17.04	17.12	17.12	14.07	14.07	14.19	14.19
	Real-SN SimpleCNN	16.95	16.95	15.23	15.23	14.52	14.52	12.84	12.84
	Real-SN SimpleCNN Bspline	15.51	15.84	8.77	11.67	12.91	13.75	7.89	9.49

Sampling approach		Cartesian							
Image		Brain				Bust			
CNN architecture		R	R*	H	H*	R	R*	H	H*
Zero-filling		8.67				6.03			
PnP-FBS	DnCNN	14.66	14.73	**14.32**	**14.32**	14.00	14.00	13.08	13.08
	Real-SN DnCNN	**14.77**	**14.77**	14.20	14.21	**14.13**	**14.13**	**13.76**	**13.76**
	Real-SN DnCNN Bspline	14.37	14.37	11.98	12.08	13.89	13.97	11.11	11.35
	SimpleCNN	13.37	13.37	13.22	13.22	11.70	11.70	11.30	11.30
	Real-SN SimpleCNN	13.38	13.38	12.90	12.90	11.71	11.71	11.44	11.44
	Real-SN SimpleCNN Bspline	14.52	14.52	3.10	4.79	13.77	13.82	3.01	3.99
PnP-ADMM	DnCNN	14.91	15.34	**15.00**	**15.14**	13.94	14.32	13.95	**14.26**
	Real-SN DnCNN	**15.10**	**15.41**	14.94	15.12	14.15	14.36	**14.06**	14.25
	Real-SN DnCNN Bspline	15.07	15.25	14.81	14.81	**14.17**	**14.43**	13.94	13.94
	SimpleCNN	13.08	13.08	13.11	13.11	11.84	11.84	12.00	12.00
	Real-SN SimpleCNN	12.75	12.75	12.11	12.11	11.89	11.89	10.01	10.01
	Real-SN SimpleCNN Bspline	12.52	12.52	8.69	10.48	10.88	10.97	7.39	8.51

TABLE V: CS-MRI results without bias (30% sample with additive Gaussian noise e = 15) in PSNR (dB), R : Residual , H : Half-averaged, * : scaled version

Sampling approach		Random							
Image		Brain				Bust			
CNN architecture		R	R*	H	H*	R	R*	H	H*
Zero-filling		9.60				7.02			
PnP-FBS	DnCNN	**19.84**	**19.84**	**18.87**	**18.87**	**16.82**	**16.82**	15.60	15.79
	Real-SN DnCNN	19.60	19.60	18.15	18.15	16.49	16.49	15.10	15.10
	Real-SN DnCNN Bspline$^+$	17.35	17.94	14.24	14.52	15.48	15.76	12.98	13.40
	SimpleCNN	19.21	19.25	18.81	18.81	16.61	16.68	**15.89**	**15.89**
	Real-SN SimpleCNN	18.57	18.60	16.91	17.31	16.23	14.24	13.83	14.49
	Real-SN SimpleCNN Bspline	19.05	19.05	17.55	17.55	16.57	16.57	14.81	14.81
PnP-ADMM	DnCNN	**19.80**	**19.91**	**19.83**	**19.90**	16.99	17.03	**17.09**	**17.11**
	Real-SN DnCNN	19.75	19.85	19.10	19.56	**17.10**	**17.10**	16.47	16.76
	Real-SN DnCNN Bspline	19.14	19.48	19.21	19.21	16.34	16.42	16.83	16.85
	SimpleCNN	19.49	19.49	19.44	19.45	16.96	16.96	16.95	16.95
	Real-SN SimpleCNN	18.79	18.86	16.87	17.00	16.48	16.48	14.69	14.77
	Real-SN SimpleCNN Bspline	18.74	18.77	17.11	17.21	16.52	16.52	14.97	15.04

Sampling approach		Radial							
Image		Brain				Bust			
CNN architecture		R	R*	H	H*	R	R*	H	H*
Zero-filling		9.29				6.21			
PnP-FBS	DnCNN	**19.21**	**19.21**	**18.30**	**18.35**	**16.33**	**16.33**	15.07	15.34
	Real-SN DnCNN	18.73	18.73	17.41	17.41	15.98	15.98	14.26	14.28
	Real-SN DnCNN Bspline$^+$	16.81	17.15	12.74	12.88	14.60	15.06	11.99	12.40
	SimpleCNN	18.42	18.46	18.08	18.08	16.09	16.09	**15.45**	**15.45**
	Real-SN SimpleCNN	17.74	17.79	16.42	16.70	15.75	15.77	13.37	14.02
	Real-SN SimpleCNN Bspline	18.13	18.21	16.91	16.91	16.09	16.10	14.34	14.34
PnP-ADMM	DnCNN	**18.85**	**19.10**	**18.91**	**19.09**	**16.61**	**16.72**	**16.61**	**16.67**
	Real-SN DnCNN	18.75	18.92	17.98	18.56	16.55	16.55	15.94	16.38
	Real-SN DnCNN Bspline	18.26	18.52	18.39	18.51	16.18	16.26	16.27	16.34
	SimpleCNN	18.54	18.64	18.45	18.57	16.44	16.46	16.43	16.46
	Real-SN SimpleCNN	17.79	17.90	15.96	16.09	15.93	16.00	13.92	13.99
	Real-SN SimpleCNN Bspline	17.69	17.75	16.28	16.37	15.97	16.02	14.28	14.32

Sampling approach		Cartesian							
Image		Brain				Bust			
CNN architecture		R	R*	H	H*	R	R*	H	H*
Zero-filling		8.67				6.03			
PnP-FBS	DnCNN	**15.00**	**15.00**	14.27	14.27	14.22	14.22	13.43	13.52
	Real-SN DnCNN	14.19	14.43	13.70	13.70	13.54	13.58	12.87	12.87
	Real-SN DnCNN Bspline$^+$	13.55	13.67	9.51	10.64	12.22	12.73	9.20	9.38
	SimpleCNN	14.82	14.85	**14.55**	**14.55**	**14.30**	**14.30**	**13.72**	**13.72**
	Real-SN SimpleCNN	14.38	14.46	12.99	13.13	13.82	13.84	11.46	11.75
	Real-SN SimpleCNN Bspline	14.80	14.80	13.15	13.15	13.80	13.94	11.92	11.92
PnP-ADMM	DnCNN	**15.07**	**15.19**	**14.70**	**15.13**	**13.99**	**14.19**	13.61	13.74
	Real-SN DnCNN	14.74	14.99	14.32	15.03	13.75	13.93	13.63	13.80
	Real-SN DnCNN Bspline	14.21	14.59	14.49	14.49	12.82	13.33	13.39	13.41
	SimpleCNN	14.65	14.97	14.67	14.92	13.82	14.06	**13.82**	**14.08**
	Real-SN SimpleCNN	13.83	13.96	12.57	12.67	12.73	13.00	10.63	10.79
	Real-SN SimpleCNN Bspline	13.80	13.83	12.71	12.79	12.85	13.14	11.0 0	11.16

TABLE VI: CS-MRI results with bias (30% sample with additive Gaussian noise e = 15) in PSNR (dB), R : Residual , H : Half-averaged, * : scaled version, Real-SN DnCNN Bspline$^+$: train with SGD without momentum with a learning rate of 0.0001 for the residual mapping and 0.00001 for $\frac{1}{2}$-averaged.

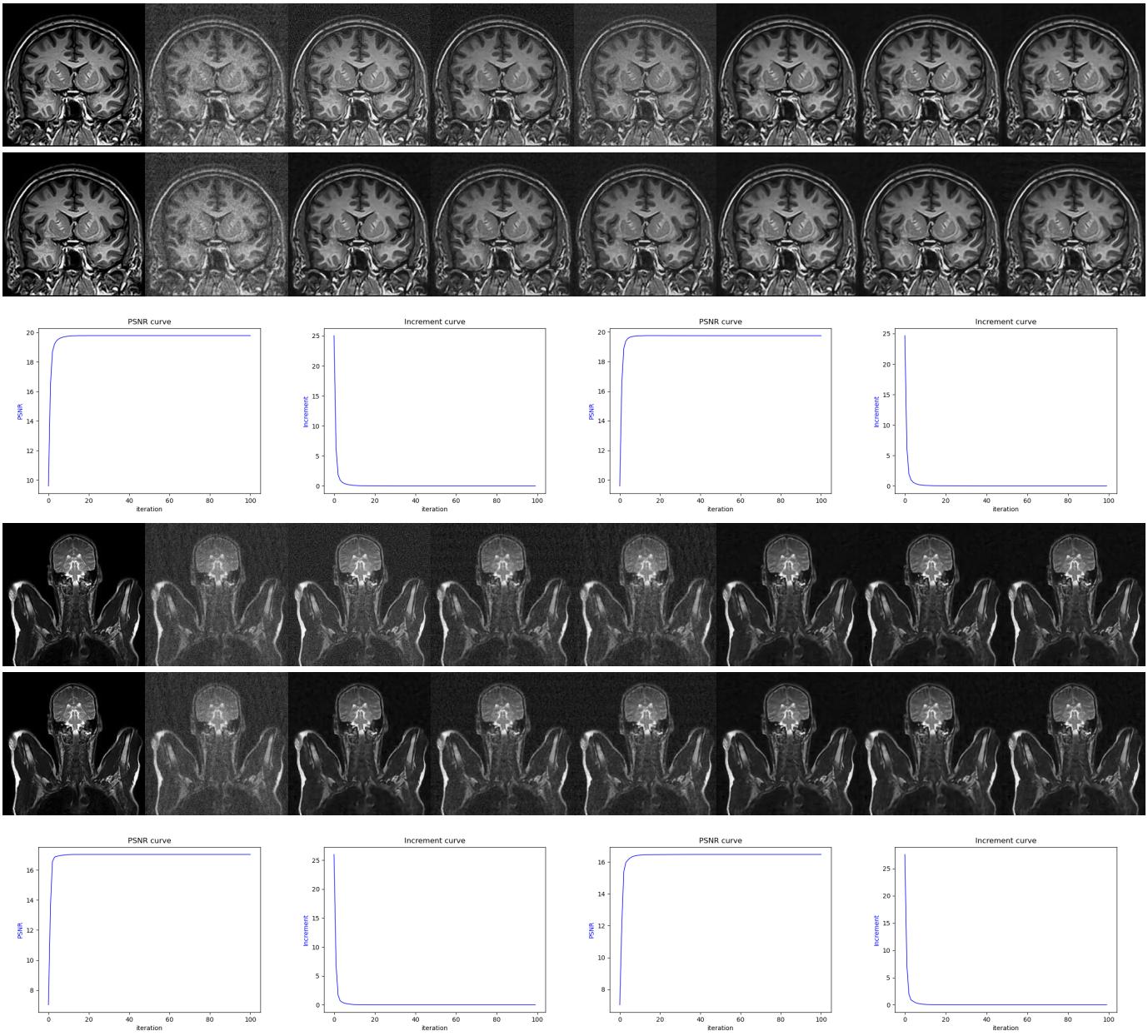

Fig. 11: Images : Ground Truth, Zero-filling, SimpleCNN, RealSN SimpleCNN, RealSN SimpleCNN Bspline, DnCNN, RealSN DnCNN, RealSN DnCNN Bspline (from left to right), without bias (top row) and with bias (bottom row). Curves : RealSN DnCNN without bias (left part), RealSN DnCNN with bias (right part).Top Panel : Brain Random residual PnP-ADMM experiment. Bottom Panel : Bust Random $\frac{1}{2}$-averaged PnP-ADMM experiment

In PnP-FBS, ReLu networks consistently perform better with the $\frac{1}{2}$-averaged mapping. For smaller networks, B-spline activations seem to enhance the performance compare to ReLU, while the performance is worse for bigger networks. Indeed, B-spline DnCNN networks are more difficult to train with default settings (Fig.12 Top panel), it seems that the network is dependent of the initialization and then stagnate. This behaviour has been observed in multiple experiments with Adam and lower learning rate. Switching to stochastic gradient descent (SGD) optimizer and lowering the learning rate seems to fix the problem. However, the network still under performs, investigating both B-spline and optimizer's parameters could allow us to obtain B-spline networks outperforming ReLu.

*2) **Denoiser scaling***

In this experiment, we ran a scaled version of all our denoisers on a grid [0.1 - 3.0] with a step size T = 0.1 for the parameter μ to see if we can slightly improve the performance. On table 5 and 6, we can observe that we were either able to slightly increase or stay at the same level for the performance. This was expected as

we already knew the noise level σ_e. We can see, that the peak of the scaling is around $\mu = 1$ as expected (Fig. 13). In this case the optimal parameter was $\mu^* = 1.1$ (Fig. 13). However, not all scaling curves were a smooth as for this experiment. Indeed, for a broad range a value after 1.5 the scaling was failing to infinite/nan value, this was especially observed for $\frac{1}{2}$-averaged mapping. This behaviour should be investigated to determine, if the scaling method can be generalized to different denoisers and when trained with a noise level far from the optimal value. However, this shows that we might be able to have a fast and simple way to enhance the performance of PnP methods.

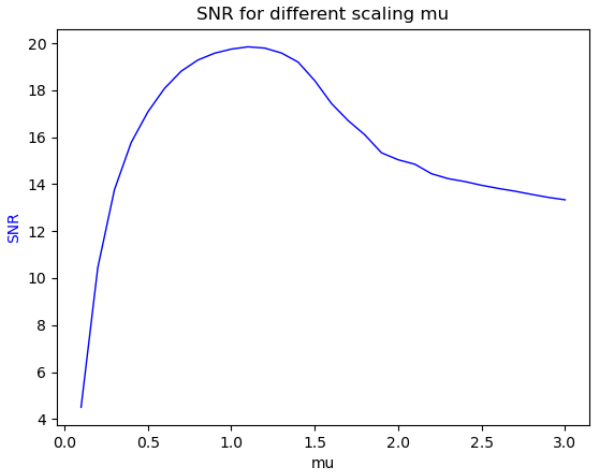

Fig. 13: SNR value for different values of the scaling parameter mu for the PnP-ADMM scaled version of residual RealSN DnCNN with random sampling.

V. Conclusion

We have seen, that real spectral normalization (RealSN) is the only method able to constrain the lipschitz upperbound of the convolutional layer. This has allowed us to to train two different network's mappings, where under some constraints or not on $f(x)$, the PnP algorithm is guaranteed to converge. We were able to compare the performance between the two mappings and results shown that $\frac{1}{2}$-averaged ReLu CNN's performance is comparable or slightly lower in most of the cases. Indeed, the residual mapping is still the reference as it performs equally or better in general and the $\frac{1}{2}$-averaged mapping has proven to be slightly more difficult to train. On the other hand, we have seen that using lipschitz constraint B-spline activation is is cost efficient for the maximum projection approach. However, it does not bring the improvement we could have imagined. As a matter of fact, it has improved the performance in very few cases, but in most cases the PnP performance was limited by unstable training.

Investigating optimizer and model's parameters tuning for B-spline activations, can be considered in further work and could lead to outperform ReLu based networks. Another interesting possibility could be to find applications where residual based CNNs do not converge, thus the use of our $\frac{1}{2}$-averaged networks could solve the problem, as it is theoretically more general.

References

[1] E. Ryu, J. Liu, S. Wang, X. Chen, Z. Wang, and W. Yin, "Plug-and-play methods provably converge with properly trained denoisers," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. Long Beach, California, USA: PMLR, 09–15 Jun 2019, pp. 5546–5557. [Online]. Available: http://proceedings.mlr.press/v97/ryu19a.html

[2] P. Bohra, J. Campos, H. Gupta, S. Aziznejad, and M. Unser, "Learning activation functions in deep (spline) neural networks," *IEEE Open Journal of Signal Processing*, vol. 1, pp. 295–309, 2020.

[3] M. T. McCann and M. Unser, "Biomedical image reconstruction: From the foundations to deep neural networks," *Foundations and Trends® in Signal Processing*, vol. 13, no. 3, p. 283–357, 2019. [Online]. Available: http://dx.doi.org/10.1561/2000000101

[4] S. H. Chan, X. Wang, and O. A. Elgendy, "Plug-and-play admm for image restoration: Fixed-point convergence and applications," *IEEE Transactions on Computational Imaging*, vol. 3, no. 1, pp. 84–98, 2017.

[5] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends® in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2011. [Online]. Available: http://dx.doi.org/10.1561/2200000016

[6] L. I. Rudin, S. Osher, and E. Fatemi, "Nonlinear total variation based noise removal algorithms," *Physica D: Nonlinear Phenomena*, vol. 60, no. 1, pp. 259–268, 1992. [Online]. Available: https://www.sciencedirect.com/science/article/pii/016727899290242F

[7] S. V. Venkatakrishnan, C. A. Bouman, and B. Wohlberg, "Plug-and-play priors for model based reconstruction," in *2013 IEEE Global Conference on Signal and Information Processing*, 2013, pp. 945–948.

[8] T. Meinhardt, M. Moller, C. Hazirbas, and D. Cremers, "Learning proximal operators: Using denoising networks for regularizing inverse imaging problems," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[9] K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang, "Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising," *IEEE Transactions on Image Processing*, vol. 26, no. 7, pp. 3142–3155, 2017.

[10] J. Hertrich, S. Neumayer, and G. Steidl, "Convolutional proximal neural networks and plug-and-play algorithms," 2020.

[11] X. Xu, J. Liu, Y. Sun, B. Wohlberg, and U. S. Kamilov, "Boosting the performance of plug-and-play priors via denoiser scaling," 2020.

[12] M. Terris, A. Repetti, J.-C. Pesquet, and Y. Wiaux, "Building firmly nonexpansive convolutional neural networks," in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020, pp. 8658–8662.

[13] M. Lohne, "Parseval reconstruction networks: Improving robustness of deep learning based mri reconstruction towards adversarial attacks," Master's thesis, Department of Mathematics, 2019.

[14] H. Gouk, E. Frank, B. Pfahringer, and M. J. Cree, "Regularisation of neural networks by enforcing lipschitz continuity," 2020.

[15] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida, "Spectral normalization for generative adversarial networks," 2018.

[16] M. Cisse, P. Bojanowski, E. Grave, Y. Dauphin, and N. Usunier, "Parseval networks: Improving robustness to adversarial examples," 2017.

[17] J. Zbontar, F. Knoll, A. Sriram, T. Murrell, Z. Huang, M. J. Muckley, A. Defazio, R. Stern, P. Johnson, M. Bruno, M. Parente, K. J. Geras, J. Katsnelson, H. Chandarana, Z. Zhang, M. Drozdzal, A. Romero, M. Rabbat, P. Vincent, N. Yakubova, J. Pinkerton, D. Wang, E. Owens, C. L. Zitnick, M. P. Recht, D. K. Sodickson, and Y. W. Lui, "fastmri: An open dataset and benchmarks for accelerated mri," 2019.

[18] Y. Sun, J. Liu, and U. S. Kamilov, "Block coordinate regularization by denoising," 2019.

[19] Y. Liu, Z. Zhan, J.-F. Cai, D. Guo, Z. Chen, and X. Qu, "Projected iterative soft-thresholding algorithm for tight frames in compressed sensing magnetic resonance imaging," *IEEE Transactions on Medical Imaging*, vol. 35, no. 9, pp. 2130–2140, 2016.

[20] E. M. Eksioglu, "Decoupled algorithm for mri reconstruction using nonlocal block matching model: Bm3d-mri," *J. Math. Imaging Vis.*, vol. 56, no. 3, p. 430–440, Nov. 2016. [Online]. Available: https://doi.org/10.1007/s10851-016-0647-7