Supplementary Information: On instabilities of deep learning in image reconstruction and the potential costs of AI

Vegard Antun, Francesco Renna, Clarice Poon, Ben Adcock and Anders C. Hansen

1 Methods

The specific setups for deep learning and neural networks in inverse problems are typically rather specialised for each type of network. However, the main idea can be presented in a rather general way. Given an under-sampled inverse problem

$$Ax = y, \qquad A \in \mathbb{C}^{m\times N}, \qquad m < N \tag{1}$$

there is typically an easy linear way of approximating x from the measurement vector y. We will denote this linear operator by $H \in \mathbb{C}^{N\times m}$. In the MRI case, when A is a subsampled discrete Fourier transform, often $H = A^*$. Note that in the MRI case x is complex valued and we actually display the magnitude image $|x|$. An example is illustrated in Figure 1. In the CT case H could be A^*, however, this gives very poor results (as demonstrated in Figure 1), and thus H is usually a discretisation of the filtered back projection (FBP). The problem is, as displayed in Figure 1, that the reconstruction $\tilde{x} = Hy$ may still be rather poor. The philosophy of deep learning is quite simple; improve this reconstruction by using learning. In particular, inspired by deep learning in image denoising [5], given training images $\{x_1, \dots, x_n\}$ and poor reconstructions $\{HAx_1, \dots, HAx_n\}$, train a neural network $f : \mathbb{C}^N \to \mathbb{C}^N$ such that

$$\|f(HAx_j) - x_j\| \ll \|HAx_j - x_j\|, \quad j = 1, \dots, n. \tag{2}$$

The hope is that (2) will hold for other images as well, not just the training examples $\{x_1, \dots, x_n\}$.

The construction process of the neural network f is typically done as follows. First one decides on a particular class (architecture) of neural networks $\mathcal{NN}$. Then one decides on a cost function $\mathrm{Cost} : \mathcal{NN} \times \mathbb{C}^N \times \mathbb{C}^m \times \mathbb{C}^N \to \mathbb{R}$ and tries to solve the optimisation problem of finding

$$f \in \operatorname*{argmin}_{h\in\mathcal{NN}} \sum_{j=1}^{n} \mathrm{Cost}(h, HAx_j, Ax_j, x_j). \tag{3}$$

The task of finding a good class $\mathcal{NN}$ and a good cost function Cost is an engineering art on its own. All the networks we test, except for AUTOMAP, are trained with some form of a "warm start" in form of a linear operator H, however, AUTOMAP is based on directly solving the problem

$$f \in \operatorname*{argmin}_{h\in\mathcal{NN}} \sum_{j=1}^{n} \mathrm{Cost}(h, Ax_j, x_j), \tag{4}$$

without any reference to the reconstructions HAx_j. We refer to Section 2 for details regarding the training of the networks. Note that the instability phenomenon is independent of the choice of (3) or (4), and the operator H may be viewed as just adding a layer to the network. Thus, we will in general talk about a network f that takes the measurements $y = Ax$ as input.

1.1 Describing the test

Before describing the algorithm for creating the unstable perturbations, it is convenient to have a short review of the framework for establishing instabilities for neural networks for image classification. For a detailed review of such methods, see [11] and the references therein. The basic idea is as follows. Let $g\colon \mathbb{R}^d \to [0,1]^C$ be an image classification network with C different classes, so that $g(x)$ is a C-dimensional vector containing the probabilities associated to the different classes for a given input image x. Let $\hat{k}_g\colon \mathbb{R}^d \to \{1, \dots, C\}$ where $\hat{k}_g(x) = \mathrm{argmax}_i(g(x)_i)$ is the image classifier. For a given norm $\|\cdot\|$ on $\mathbb{R}^d$, we can then define the optimal, meaning smallest, unstable perturbation $r^* \in \mathbb{R}^d$, for an image $x \in \mathbb{R}^d$ as

$$r^*(x) \in \operatorname*{argmin}_{r} \|r\| \text{ subject to } \hat{k}_g(x + r) \neq \hat{k}_g(x), \tag{5}$$

where we write $r^*(x)$ to indicate that this is a perturbation for the image x.

It is clear that one cannot apply the approach in (5) to the problem of finding instabilities of neural networks for the inverse problem. Indeed, (5) is designed for a decision problem a la "is there a cat in the image?". In inverse problems there is no decision problem but rather the following, slightly simplified issue:

$$\text{Reconstruct } x \text{ from } y = Ax, \qquad A \in \mathbb{C}^{m\times N}. \tag{6}$$

Thus, if we are given a neural network $f\colon \mathbb{C}^m \to \mathbb{C}^N$ designed to solve (6), and we want to search for instabilities imitating (5), we would end up with the problem of finding

$$\hat{r}(x) \in \operatorname*{argmin}_r \|r\| \text{ subject to } \|f(y+Ar) - f(y)\| \geq \delta, \tag{7}$$

for some $\delta > 0$, where $y = Ax$ for some x. Note that (7) has a clear disadvantage in that it may be infeasible for different values of δ. Hence, a slightly different, constrained Lasso inspired variant, may be worth considering;

$$\tilde{r}(x) \in \operatorname*{argmax}_r \|f(y+Ar) - f(y)\| \quad \text{subject to} \quad \|r\| \leq \theta, \tag{8}$$

for some $\tau > 0$. In the case of (8) we do not have any issues regarding infeasibility. However, a third option could be an unconstrained Lasso inspired version of (8) given by

$$r^*(y) \in \operatorname*{argmax}_r \frac{1}{2}\|f(y+Ar) - f(y)\|_2^2 - \frac{\lambda}{2}\|r\|_2^2 \tag{9}$$

(here we have specified the norm), where $\lambda > 0$. Note that (9) is not the only possibility. In particular, one could consider the more general setting

$$r^*(y) \in \operatorname*{argmax}_r \frac{1}{2}\|f(y+Ar) - p(x)\|_2^2 - \frac{\lambda}{2}\|r\|_2^2, \tag{10}$$

where $p : \mathbb{C}^N \to \mathbb{C}^N$. In this case r^* will obviously depend on p, and the quality of the artefacts produced by r^* may differ greatly as p changes. Indeed, this is the motivation for allowing this extra variable. In this paper we focus on (10) and consider $p(x) = f(Ax)$ (as in (9)) and $p(x) = x$.

However, the first part of our test could indeed be carried out by a different optimisation problem. Moreover, we anticipate that there will be other methods for creating instabilities for neural networks for inverse problems that will be as reliable and diverse as what we present here. Note that (10) is set up to find perturbations in the image domain. We do this deliberately as this provides an easy way to compare the original image with a perturbed image and deduce whether the reconstruction of the perturbed image is acceptable/unacceptable. However, one could set up (10) so that the perturbation is in the sampling domain as well. In the following we describe the test in detail and the methodology.

1.2 Stability with respect to tiny perturbations

The neural network $f\colon \mathbb{C}^m \to \mathbb{C}^N$ is a non-linear function. In practice this makes the problem of finding a global maximum of the optimization problem in (10) impossible, even for small values of m and N. In the following we will provide a method that aims at locating local maxima of (10) by using a gradient search method. In particular, given an image $x \in \mathbb{R}^N$, $A \in \mathbb{C}^{m\times N}$ and $y = Ax$ as in (1) let

$$Q_y^p(r) = \frac{1}{2}\|f(y+Ar) - p(x)\|_2^2 - \frac{\lambda}{2}\|r\|_2^2 \tag{11}$$

be the objective function. A most natural method to solve (10) is *gradient ascent with momentum*. Thus, the method uses the gradient of Q_y^p in conjunction with two parameters $\gamma > 0$ (the momentum) and $\eta > 0$ (the learning rate) in each step towards a local maximum.

Algorithm 1 Finding unstable perturbation for inverse problems

```
1: Input: Image: $x$, neural network: $f$, sampling matrix: $A$, maximum number of iterations: $M$.
2: Output: Perturbation $r_M$
3: Initialize: $y \leftarrow Ax, v \leftarrow 0, i \leftarrow 1, r_0 \sim \text{Unif}([0,1]^N), 0 < \lambda, \gamma, \eta, \tau$. Set $Q_y^p(r)$ as in Equation (11).
4: $r_0 \leftarrow \tau r_0$
5: while $i \leq M$ do
6:     $v_{i+1} \leftarrow \gamma v_i + \eta \nabla_r Q_y(r_i)$
7:     $r_{i+1} \leftarrow r_i + v_{i+1}$
8:     $i \leftarrow i + 1$
9: return $r_M$
```

This means that there are three parameters $\lambda > 0$, $\gamma > 0$ and $\eta > 0$ to be set, and hence the perturbation r found by the algorithm will depend on these. The complete algorithm is presented in Algorithm 1, where r_0 is initialised randomly. Note that the parameter τ used in Algorithm 1 is simply a scaling factor needed as the input images may have values in different ranges.

Note that for $u = y + Ar$, the gradient of Q_y^p is given by

$$\nabla_r Q_y^p = A^* \nabla_u g(u) - \lambda r, \quad g(u) := \|f(u) - p(x)\|_2^2 \tag{12}$$

where $\nabla_u g(u)$ can be computed efficiently using back propagation. Note also that at each iteration this gradient is left multiplied by the adjoint A^*.

Algorithm 2 Finding unstable perturbation for Radon problems

```
1: Input: Image: $x$, neural network: $f$, sampling matrix: $A$, FBP operator: $B$, maximum number of iterations: $M$.
2: Output: Perturbation $r_M$
3: Initialize: $y \leftarrow Ax$, $v \leftarrow 0$, $i \leftarrow 1$, $r_0 \sim$ Unif$([0,1]^N)$, $0 < \lambda, \gamma, \eta, \tau$. Set $g(u)$ as in Equation (12).
4: $r_0 \leftarrow \tau r_0$
5: while $i \leq M$ do
6:     $v_{i+1} \leftarrow \gamma v_i + \eta B \nabla_u g(y + Ar_i) - \lambda r_i$
7:     $r_{i+1} \leftarrow r_i + v_{i+1}$
8:     $i \leftarrow i + 1$
9: return $r_M$
```

Just as in the case when training neural networks using stochastic gradient descent with momentum, choosing the parameters γ and η is an art of engineering. We are in a similar situation with our algorithm, and the optimal choices of γ, η are based on empirical testing. Such experimenting with parameters also motivates experimenting with other parts of the algorithm. For example, when considering Radon measurements, we found that setting the optimal values of γ and η could be rather difficult. However, by replacing A^* in (12) by $B \in \mathbb{R}^{N\times m}$ being a discretisation of a filtered back projection (FBP), this problem could be overcome and we therefore use Algorithm 2 in the case of Radon samples.

It should be mentioned that there are different variations of discretisations of the filtered backprojection for Radon problems. The discretisation $B \in \mathbb{R}^{N\times m}$ used in our experiment is the one provided by MatLab R2018b.

1.3 State-of-the-art comparison method

All of our tests are done against state-of-the-art benchmark methods using established techniques based on sparse regularisation and compressed sensing [7, 9, 30, 2].

There are many variations in the literature using X-lets and Total Variation (TV) techniques separately or in combination. Our main algorithm is based on the re-weighting technique suggested in [8]. This idea was refined in [26] and [25], by combining both X-lets (shearlets in this case) and TV. This is our main algorithm of choice used in this paper. We refer the reader to [25] for details, however, a short summary can be described as follows. The algorithm allows for both Fourier and Radon sampling, however, the current implementation only allows for single coil MRI in the Fourier case. The idea is to solve iteratively the problem

$$\underset{z}{\text{minimize}} \sum_{j=1}^{J} \lambda_j \|W_j \Psi_j z\|_1 + \mathrm{TGV}_\alpha^2(z) \text{ subject to } Az = y,$$

where the λ_js are weights, W_j is a diagonal weighting matrix, Ψ_j is the j'th subband in a shearlet transform [25], and TGV_α^2, $\alpha = (\alpha_1, \alpha_2)$ is the second order Total Generalised Variation operator. The TGV_α^2 operator consist of a first order term (TV) weighted by α_1 and a second order (generalised) term weighted by α_2. In each iteration step the weights λ_j and weighting matrices W_j are updated.

In particular, the minimisation problem is casted into an unconstrained formulation

$$\underset{z}{\text{minimize}} \sum_{j=1}^{J} \lambda_j \|W_j \Psi_j z\|_1 + \mathrm{TGV}_\alpha^2(z) + \frac{\beta}{2}\|Az - y\|_2^2,$$

and solved via split Bregman iterations. This means that the problem is decoupled into two portions, one accounting for the ℓ_1-norm term and one for the ℓ_2-norm term.

In particular, on denoting by Ψ' a composite operator including (1) the effect of multi-scale X-lets transform in different levels including the weights λ_j, (2) the first order (TV) term of TGV_α^2 and (3) the second order term of the same operator, and by adding a further splitting variable $d = \Psi' z$, it is possible to write the k-th split Bregman iteration as

$$\left\{\begin{array}{rcl} (z_{k+1}, d_{k+1}) & = & \arg\min_{z,d} \|Wd\|_1 + \frac{\beta}{2}\|Az - y_k\|_2^2 \\ & & + \frac{\mu}{2}\|d - \Psi' z - b_k\|_2^2, \\ b_{k+1} & = & b_k + \Psi' z_{k+1} - d_{k+1}, \\ y_{k+1} & = & y_k + y - Az_{k+1}. \end{array}\right. \tag{13}$$

During each iteration, the (z, d)-minimisation problem is solved using one or multiple non-linear block Gauss-Seidel iterations, which alternate between minimising z and d. Also, in contrast with the re-weighting strategy originally presented in [8], the weights in W are updated not only after convergence to the solution of the ℓ_1 minimisation problem, but weight updates are incorporated in the split Bregman iterations.

In the above iterations we have allowed for a slight abuse of notation. We are using $\mu = (\mu_1, \mu_2, \mu_3)$ and split the sum $\|d - \Psi' z - b_k\|_2^2$, into three separate parts, depending on which of the terms of Ψ' they come from, and weight each partial sum separately with $\frac{\mu_1}{2}$, $\frac{\mu_2}{2}$ and $\frac{\mu_3}{2}$, respectively (see equation (15) in [25] for details).

This method has been used for reconstruction from Fourier and Radon measurements, using two different setups. For the case of Fourier measurements, discrete shearlets are generated with 3 scales and with directional parameters [1 2 2] (see [26] and [25] for details). The optimisation parameters are set as follows:

- $(\mu_1, \mu_2, \mu_3) : (5000, 10, 20)$,
- (α_1, α_2): $(1, 1)$,
- β: 10^5,
- ϵ: 10^{-5},

Where ϵ is a parameter which is added in the denominator, of the updating rule, for the weights W, to avoid division by zero. Similarly, image reconstruction from Radon measurements are obtained by using shearlets with 4 scales and directional parameters [0 0 1 1] and with the following parameter setup:

- $(\mu_1, \mu_2, \mu_3) : (500, 0, 0)$,
- (α_1, α_2): $(1, 0)$,
- β: 50,
- ϵ: 10^{-8},

Notice in particular that $\mu_3 = \alpha_2 = 0$, hence we are only using shearlets and a TV term as our regularizes. In all setups, we run the algorithm until convergence, i.e. between 50 and 500 iterations.

The above approach is used in all examples except for the tests using the MRI-VN network which is designed for parallel MRI. For this imaging modality we have the following reconstruction problem. Let $\Omega \subseteq \{1, \ldots, N\}$, $|\Omega| = m$ and $P_\Omega \in \mathbb{R}^{m \times N}$ be the projection operator onto the canonical basis i.e. $P_\Omega x = (x_i)_{i \in \Omega}$. Let $F \in \mathbb{C}^{N \times N}$ be the discrete Fourier transform (DFT) matrix. The Fourier sampling matrix can then be written as $A_f = P_\Omega F$, for a given sampling pattern Ω. In parallel Fourier imaging we receive information from multiple coils elements at the same time. This is modeled by introducing diagonal sensitivity matrices $S_1, \ldots S_c \in \mathbb{C}^{N \times N}$ which weight the measurements, based on the environmental conditions of the sensing problem. The corresponding measurement matrix is then written as

$$A_{pf} = \begin{bmatrix} P_\Omega F & & \\ & \ddots & \\ & & P_\Omega F \end{bmatrix} \begin{bmatrix} S_1 \\ \vdots \\ S_c \end{bmatrix} \in \mathbb{C}^{m' \times N}.$$

where $m' = cm$. Note that for this sampling operator we might have $m' > N$, which means that the corresponding linear system may be overdetermined. Given an image $x \in \mathbb{R}^N$ we let $y = A_{pf} x$ and use the SPGL1 algorithm [37] for solving

$$\underset{z}{\text{minimize}} \|z\|_1 \quad \text{subject to} \quad \|A_{pf} \Psi^{-1} z - y\|_2 \leq \delta,$$

where $\Psi \in \mathbb{R}^{N \times N}$ is the wavelet transform corresponding to the periodised Daubechies 2 wavelet with 3 levels. In all the experiments we set $\delta = 0.01$.

1.4 Non-uniqueness of the test – parameter dependency

Note that the test we provide can never become unique. Indeed, we choose to solve (10) with different choices of p (see Figure 2), however, (7) and (8) could also be viable alternatives. Moreover, all of these approaches depend on parameters δ, θ and λ that have to be specified, and different values give different worst-case perturbations. In addition, Algorithm 1 and Algorithm 2 designed to solve (10) depend on the parameters γ, η and τ. Moreover, note also that there is no built-in halting criteria in Algorithm 1 and Algorithm 2, but rather the parameter M controlling the number of iterations. Thus, the stability test can never become a unique test, but instead a collection of algorithms depending on different parameters. Hence, an appropriate use of the test means running Algorithm 1 and Algorithm 2 varying the parameters. This is also what is done in this paper, however, only the results based on the final parameters are displayed in the figures. The final parameters chosen are listed in Table 1.

In Figure 4, we display different perturbations r_j produced by Algorithm 1 with different values of M corresponding to the experiments shown in Figures and in the main manuscripts. The values of λ, γ, η and τ are as in Table 1. Note the difference between the perturbations depending on the network. As the perturbations are tiny, they have been enlarged in order to get a visual impression.

1.5 Stability of state-of-the-art methods

The perturbation r computed by the algorithms from Section 1.1.2 are constructed specifically for an image x, sampling operator A and neural network f. Hence it might not be too surprising that the state-of-the-art methods are unaffected by this perturbation. Though our use of state-of-the-art methods in this work have mainly been to ensure that the inverse problem in itself is not ill posed, it is tempting to see if similar instabilities can occur for sparse regularization algorithms as well. Indeed, it could be that sparse regularization techniques are equally unstable to a worst case perturbation r, but that such a perturbation is rare enough so as to not have yet arisen in practice. Compressive sensing techniques have after all, only been tested by the scientific community for a little more than a decade, and by clinicians for the last few years.

Extending the algorithms from Section 1.1.2 so that they can be applied to general sparse regularization algorithms is a challenging task, as both algorithms need to compute the gradient of $g(u) = \|f(u) - p(x)\|_2^2$. Thus if f is not a neural network, we can no longer use the back propagation algorithm to easily compute this quantity. While some of the simplest sparse regularization algorithms can be written as neural networks, more sophisticated algorithms often contain internal *if-else* or *while* statements, making it hard to write them as neural networks.

Writing the above sparse regularization algorithms as neural networks are beyond the scope of this paper. Yet, we want to illustrate that sparse regularization algorithms can be tested if they are written as neural networks. To this end, we include an experiment from [1, 15], where an iterative algorithm for solving the following optimization problem

$$\underset{z\in\mathbb{C}^N}{\operatorname{minimize}}\, \mu\|Wz\|_1 + \|A\Psi^{-1}z - y\|_2, \qquad \mu > 0, \tag{14}$$

have been unrolled, using 1000 iterations, as a neural network. Here $W \in \mathbb{R}^{N\times N}$ is a diagonal weighting matrix, $A = P_\Omega F \in \mathbb{C}^{m\times N}$ is a subsampled Fourier transform and $\Psi \in \mathbb{R}^{N\times N}$ is the discrete Haar wavelet transform. To test the stability of this network, we have used the same data x and sampling pattern Ω as for the AUTOMAP network. Using Algorithm 1, we then computed perturbations v_j, $j = 1, 2, 3, 4$ all with the same ℓ_2-norm as the r_j's computed for AUTOMAP in Figure in the main paper. As can be seen in Figure 3, the network seems stable with respect to tiny perturbations.

1.6 Stability with respect to small structural changes

This part is fully explained in the main manuscript. However, we want to emphasise that the symbols used in the experiment are chosen in order to assure that networks can recover important details. These can of course be replaced by other symbols as long as the ability of an algorithm to reconstruct these symbols correlate with the ability to recover other small important details.

Important note: In our examples, it is irrelevant whether the symbols have been included in the training set or not. In fact, both the AUTOMAP and the Deep MRI networks have no problem recovering the symbols, see the first column of Figure and Figure 2, where a heart is artificially added, and the fourth row of Figure . Indeed, none of these networks have been trained on images containing the symbols, yet they can perfectly well recover them.

1.7 Stability with respect to more samples

All of the networks we have tested, except AUTOMAP, are convolutional neural networks (CNNs), which means that the trained weights come from convolutional layers. This has the advantage of reducing the number of parameters we need to learn, compared to fully connected layers (dense matrices), and may for large image sizes be the only alternative. Moreover, these CNNs can easily be adapted to other subsampling patterns as explained below. Thus, one can easily apply a network that is trained on 25% subsampling, say, to input that uses, for example, 35% subsampling. The question is whether the quality of the reconstruction is kept when increasing the subsampling ratio. The reason for the flexibility of the CNNs in our test is that they all depend on the operator $H \in \mathbb{C}^{N\times m}$, as described in (2), by considering it as a non-learnable first layer. As the H is non-learnable, this allows for flexibility in our choice of m, since we know how to construct H for various values of m.

In the last row of Figure 4 in the main manuscript we have varied the number of samples m and measured the image quality of the networks reconstruction using the peak signal-noise-ratio (PSNR) between the magnitude images of the original and the reconstructed image. Figure 4 shows all of the networks, except the AUTOMAP network, which learns a mapping directly from the measurement domain without using a non-learned layer H. Below follows the description of each of the experiments visualised in the last row of Figure 4.

Ell 50: We created 25 sinograms of images containing ellipses similar to the data in the network's training and test set. The sinograms were created with 1000 uniformly spaced angles (views) using the formula for the Radon transform of an ellipse. We then considered an acceleration factor $k \in \{2, \dots, 30\}$, by subsampling every k-th line among the 1000 views. The FBP of the subsampled sinogram was given to the network and the PSNR of the reconstruction was computed against the FBP of all 1000 views.

Med 50: We used a test set, provided by the authors of [19], consisting of 25 CT images from the Mayo Clinic. These images were synthetically sampled, using a the discrete Radon transform from MATLAB, at the same angles as the Ell 50 network. The subsampled sinograms were mapped back into the image domain using a FBP and reconstructed using the network. The PSNR values were computed with the original image as ground truth.

DAGAN: We used 20 MR images of brain tissue from the test set, and subsampled these images using the 1D Gaussian sampling patterns provided by the authors of [38]. These patterns have been generated for subsampling rates 1%, 5%, 10%, 20%, 30%, 40% and 50%.

MRI-VN: We used image data from the networks test set, and picked one image slice from 10 different patients. The image data was subsampled with uniformly spaced lines (center lines was always included), at subsampling rates 5%, 10%, 15%, 20%, 25%, 30%, 35%, and 50%. The PSNR was computed with the magnitude image of the fully sampled images as reference.

Deep-MRI: We used 30 image slices from one MRI scan, and subsampled each slice using lines sampled according to a Gaussian distribution. Extra caution was taken, so that all lines sampled at a low sampling rate, was included at higher sampling rates. We sampled with an acceleration rate $2, \ldots, 14$.

It should be noted that measuring image quality is a delicate issue. We point out that no comparison based on the last row of Figure on the reconstruction quality should be made between the networks, as the PSNR is unfit to measure image quality between different types of images [17]. However, we are only interested in the change in PSNR, as a function of subsampling percentage, for each specific network.

1.8 Theoretical aspects

Our contribution documents the instability phenomenon in deep learning methods for inverse problems. However, the instability phenomenon can be explained theoretically as well. Indeed, the recent paper [12] provides theoretical foundations that explain the reasons for the instabilities. Moreover, the theoretical results confirm that the instabilities are stable in the sense that there will always be balls around the 'bad' perturbations such that adding elements in the ball to the 'bad' perturbation will yield another 'bad' perturbation. If the perturbation

$$r_{\mathrm{pert}} = r^1_{\mathrm{pert}} + r^2_{\mathrm{pert}},$$

where r^1_{pert} and r^2_{pert} are random variables there will typically be a non-zero probability that r_{pert} is a 'bad' perturbation depending on the probability distribution of r^1_{pert} and r^2_{pert}. We may have that r^1_{pert} is a Gaussian vector but r^2_{pert} could have a probability distribution (coming from patient movement, anatomic differences, etc) that is incredibly difficult to estimate. Thus, deeming unstable neural networks safe for use in medical imaging based on a probabilistic estimate is far from trivial, if not impossible.

The theoretical developments also demonstrate how difficult it is to cure the instability phenomenon. As the theoretical results in [12] confirm, one can predict which attempts on remedies that will have limited effect. This includes adversarial training (a technique that has been thoroughly investigated as a remedy for instabilities occurring in image classification), training with multiple random sampling patterns, enforcing consistency, adding small random perturbations to the measurements etc. [13, 3, 34, 4, 28]

2 Technical details needed to reproduce the results

2.1 Overview

This section contains all the extra material on neural networks that is useful in order to understand and reproduce all the experiments done in the paper. In particular, the it displays the variety of different architectures and training sets used in the various experiments. The neural networks considered are:

(i) *AUTOMAP* [39]: The AUTOMAP neural network we test is for low resolution single coil MRI with 60% subsampling. In the paper [39] one mentions 40% subsampling, but this apparent discrepancy is simply due to different interpretation of the word subsampling. We use the traditional meaning in sampling theory referring to x% subsampling as describing that the total amount of samples used are x% of full sampling. The network used in our experiment is trained by the authors of [39]. The details of the architecture and training data are summarised in §2.2.

(ii) *DAGAN* [38]: This network is for medium resolution single coil MRI with 20% subsampling. The network weights are not available online, however, complete instructions on how to reproduce the network used in [38] are accessible. Based on these instruction, we have retrained a network that reproduces the results in [38]. The details of the training data and architecture are summarised in §2.3.

(iii) *Deep MRI* [31]: This neural network is for medium resolution single coil MRI with 33% subsampling. The network used in our experiments is trained by the authors of [31], can be found online and we summarise the details on training data and architecture in §2.4.

(iv) *Ell 50* [19]: Ell 50 is a network for CT or any Radon transform based inverse problem. The number 50 refers to the number of lines used in the sampling in the sinogram. The training of the network is done by the authors of [19]. The network can be obtained online, and all the details can be found in §2.5.

(v) *Med 50* [19]: Med 50 has exactly the same architecture as Ell 50 and is used for CT, however the training is done on a different dataset. The network is trained by the authors of [19]. Details are summarised in §2.5.

(vi) *MRI-VN* [14]: This network is for medium to high resolution parallel MRI with 15 coil elements and 15% subsampling. In order to show a variety of subsampling ratios we have trained this network on a smaller subsampling percentage than what the authors of [14] originally (25% and 33%) did in their paper. As we already have 33%, and 20%, we want a test on even lower subsampling rates. All the remaining parameters are kept as suggested in the code provided by the authors of [14], except for the subsampling ratios and batch size (due to memory limitations). All the details are documented in §2.6.

All network weights are available from https://github.com/vegarant/Invfool.

2.2 AUTOMAP

2.2.1 Network architecture

The AUTOMAP network [39] is proposed for image reconstruction from Radon measurements, spatial non-Cartesian Fourier sampling, misaligned Fourier sampling and undersampled Cartesian Fourier samples. In this work we have tested the network trained for image reconstruction from undersampled Cartesian Fourier samples. In contrast with the other networks considered in this work, the AUTOMAP network provides a direct mapping of the Fourier measurements to the image domain without applying the adjoint operator H as a first step.

The authors of [39] have not made their code publicly available, and the weights from their paper [39] had not been stored. However, they kindly agreed to retrain their network for us and save the weights. The network architecture they trained deviates slightly in some of activation functions reported in their paper [39], however, the network was trained on the same data and sampling pattern. Below we describe the network architecture we received. The training parameters and data, are reported as in the paper [39].

The input of the AUTOMAP network, as described in [39] and in Figure 5 takes a complex $n \times n$ image of measurements as input. In the case of subsampling, one may interpret the $n \times n$ image as being zero padded in the coordinates that are not sampled. In the tests, $n = 128$, and in the actual implementation the input is represented by the complex measurement data $y \in \mathbb{C}^m$ with $m = 9855$ (60% of n^2) in this experiment. Such data is reshaped into a vector of length $2m$ with real entries before being fed into the network. The first two layers of the network a fully connected matrices of size $25000 \times 2m$ and $n^2 \times 25000$. The first fully connected layer is followed by a hyperbolic tangent activation function. The second fully connected layer is followed by a layer which subtracts the mean from the output of the second layer. The output is then reshaped into an $n \times n$ image.

Next follows two convolutional layers with filter size 5×5, 64 feature maps and stride 1×1. The first convolutional layer is followed by a hyperbolic tangent function, while the other is followed by a rectified linear unit (ReLU). Finally, the output layer deconvolves the 64 feature maps provided by the second convolutional layer with 7×7 filters with stride 1×1. The output of the network is an $n \times n$ matrix representing the image magnitudes.

2.2.2 Training parameters

The loss function used for training consisted of two terms, $\mathcal{L}_{\text{SE}}$ and $\mathcal{L}_{\text{PEN}}$. Here $\mathcal{L}_{SE}$ is the ℓ_2-norm of the difference between the ground truth magnitude image and the magnitude image predicted by the network. Similarly the $\mathcal{L}_{\text{PEN}}$ is an ℓ_1-penalty on the outputs of the activations following the second convolutional layer ($C2$). The total loss was then computed as

$$\mathcal{L}_{\text{TOTAL}} = \mathcal{L}_{\text{SE}} + \lambda \mathcal{L}_{\text{PEN}}$$

with $\lambda = 0.0001$. The network is trained using the RMSProp algorithm (see for example http://www.cs.toronto.edu/ tijmen/csc321/slides/lectu

2.2.3 Training data

The training dataset consists of $50,000$ images taken from 131 different subjects from the MGH-USC HCP public dataset [10][1]. For each image, the central 256×256 pixels were cropped and subsampled to a resolution of 128×128 pixels. Before training, the images were preprocessed by normalizing the entire dataset to a constant value defined by the maximum intensity of the dataset. Fourier data were obtained by subsampling the Cartesian k-space using a Poisson-disk sampling pattern with 60% undersampling [36].

In order to increase the network robustness against translation, the following data augmentation scheme was applied. New images were created from each image in the training dataset by tiling together four reflections of the original image. Then, the so obtained 256×256 image was cropped to a random 128×128 selection. The routines used to implement the AUTOMAP network were written in TensorFlow[2].

2.3 DAGAN

2.3.1 Network architecture

The DAGAN network was introduced in [38] to recover images from Fourier samples, with particular emphasis on MRI reconstruction applications. The DAGAN network assumes measurements $y = Ax$, where A is a subsampled discrete Fourier transform. The input of the network is represented by the noisy magnitude image $\tilde{x} = |Hy|$, which is obtained by direct inversion of the zero-filled Fourier data, in particular, $H = A^*$.

[1]https:// db.humanconnectome.org/
[2]https://www.tensorflow.org

The recovery algorithm presented in [38] is based on a conditional generative adversarial network (GAN) model, which consists of a generator network, used for the image reconstruction, and a discriminator network, measuring the quality of the reconstructed image. The generator network adopted in [38] has a U-net structure, similar to that used in [19], and its objective is to produce the recovered image. In [38] the authors propose two almost identical architectures, and train them with different loss functions. Below we will describe their "refinement" architecture trained with what is referred to as Pixel-Frequency-Perceptual-GAN-Refinement loss in the paper. The refined version is also our choice, as this architecture and training performed the best in the paper and in our tests as well. The network was not made publicly available, and based on advice from the authors of [38] we trained the network ourselves.

The architecture of the generator network, which is reported in Figure 6, contains 8 convolutional layers and 8 deconvolutional layers each followed by batch normalization (BN) [18]. The batch normalization layers after the convolutional layers are followed by leaky ReLU (lReLU) activations with slope equal to 0.2 for $x < 0$, while the batch normalization layers after the deconvolutions are followed by a ReLU activation. The generator network also contains skip connections, i.e., connections that copy the output of a layer directly to the input of a layer further down in the hierarchy. The skip connections are used to concatenate mirrored layers (see Figure 6). The filter kernels used for the convolutional and deconvolutional layers have size 4×4 with stride 2×2. The number of filters in each convolutional/deconvolutional layer increases/decreases according to Figure 6.

The last deconvolutional layer is followed by a hyperbolic tangent activation function. A global skip connection, adding the input to the network and the output from the hyperbolic tangent function, is then followed by a ramp function clipping the output values of the image to the range $[-1, 1]$. Adding this last skip connection means that the network is actually approximating the residual error between the network input $\tilde{x} = Hy$ and the image of interest.

The generator network is trained jointly with a discriminator network, which aims to distinguish between the output of the generator network and ground truth images. For further information on this network, we refer to [38].

2.3.2 Training parameters

The loss function used to train the DAGAN network consists of four different terms. First, an image domain mean square error (MSE) loss, $\mathcal{L}_{\text{iMSE}}$, which accounts for the ℓ_2 distance between the output of the generator network and the ground truth image. Second, a frequency domain MSE loss, $\mathcal{L}_{\text{fMSE}}$, which enforces consistency between the output of the generative network in the frequency domain and the acquired Fourier measurements. Third, a perceptual loss term, $\mathcal{L}_{\text{VGG}}$, which is computed by using a pretrained VGG-16 described in [33]. In particular, the VGG-16 network was trained over the ImageNet dataset[3] and the output of its conv4 layer was used to compute the loss term by considering the ℓ_2-norm of the difference between the VGG-16 output corresponding to the ground truth image and the generator network output. Finally, the fourth term, $\mathcal{L}_{\text{GEN}}$ is computed using a cross entropy loss on the output of the discriminator network. Adding these four terms together gives us the loss

$$\mathcal{L}_{\text{TOTAL}} = \alpha\mathcal{L}_{\text{iMSE}} + \beta\mathcal{L}_{\text{fMSE}} + \gamma\mathcal{L}_{\text{VGG}} + \tau\mathcal{L}_{\text{GEN}}, \qquad \alpha, \beta, \gamma, \tau > 0.$$

We used the same values for α, β, γ and τ as in [38], in particular, $\alpha = 15$, $\beta = 0.1$, $\gamma = 0.0025$ and $\tau = 1$. The generator and the discriminator network were jointly trained by alternating gradient optimization. In particular, the Adam [21] optimizer was adopted, with initial learning rate 0.0001, momentum 0.5, and minibatch size 25. The learning rate was halved every 5 epochs. We applied the same early stopping rule as given in their implementation[4]. This is based on measuring the $\mathcal{L}_{\text{iMSE}}$ loss between the training set and validation set. We used the early stopping number 10. In total this resulted in 15 epochs of training.

2.3.3 Training data

The DAGAN network was trained using data from a MICCAI 2013 grand challenge dataset[5]. We removed all images from the dataset where less than 10% of the pixel values were non-black. In total we therefore used 15912 images for training and 4977 images for validation. The dataset consisted of T_1-weighted MR images of different brain tissues.

The following data augmentation techniques were used to increase the amount of training data: image flipping, rotation, shifting, brightness adjustment, zooming, and elastic distortion [32].

The discrete Fourier transform of the training images were subsampled using 1D Gaussian masks, i.e., masks containing vertical lines of data in the k-space randomly located over the image according to a Gaussian distribution. In our tests we trained a network to do recovery from 20% subsampling. The code used to implement DAGAN was written using the TensorLayer[6] wrapper and TensorFlow, and was made publicly available by the authors of [38].

2.4 DeepMRINet

2.4.1 Network architecture

The Deep MRI net is used to recover images from their subsampled Fourier measurements. Its architecture is built up as a cascade of neural networks, whose input is represented by the blurry image obtained by direct inversion of the measurements, i.e., $\tilde{x} = Hy$.

[3]http://www.image-net.org/
[4]https://github.com/nebulaV/DAGAN
[5]http://masiweb.vuse.vanderbilt.edu/workshop2013/index.php/
[6]http://tensorlayer.readthedocs.io

The networks in the cascade are convolutional neural networks (CNN) designed as follows

$$CNN_i(\tilde{x}) = \tilde{x} + C^{(i)}_{rec}\rho(C^{(i)}_{rec-1} \cdots \rho(C^{(i)}_1 \tilde{x} + b^{(i)}_1) \cdots + b^{(i)}_{rec-1}) + b^{(i)}_{rec},$$

where $\rho(z) = \max\{0, z\}$ is the ReLU activation function, whereas $C^{(i)}_k$ and $b^{(i)}_k$represent trainable convolutional operators and biases, respectively, for the ith network. These networks are then tied together and interleaved with *data consistency layers (DC)*, which have the objective to promote consistency between the reconstructed images and the Fourier measurements. The DC layers are defined as

$$DC_\lambda(\tilde{x}, y, \Omega) = F^{-1} g_\lambda(F\tilde{x}, y, \Omega), \quad \text{where} \quad g_\lambda(z, y, \Omega) = \begin{cases} z_k & k \notin \Omega \\ \frac{z_k + \lambda y_k}{1+\lambda} & k \in \Omega \end{cases}.$$

Here F represents the Fourier operator, and Ω is the set of indices corresponding to the measurements acquired in the k-space. We point out that in the limit $\lambda \to \infty$, the g_λ function simplifies to y_k if $k \in \Omega$ and z_k otherwise.

In practice, a DC layer performs a weighted average of the Fourier coefficients of the image obtained as the output of a CNN in the cascade and the true samples y. The parameter λ can either be trained or kept fixed. In [31], it is not specified whether λ is learned or not, however, from the code[7] it is clear that λ is chosen to be ∞.

The complete network can now be written as

$$f(y, \Omega) = DC_\lambda(CNN_n(\cdots DC_\lambda((CNN_1(Hy)), y, \Omega) \cdots), y, \Omega),$$

and its architecture is reported in Figure 7. In particular, the architecture used to produce the results in [31] and those reported in this paper contains 5 CNNs interleaved with 5 DC layers. Each CNN contains 5 convolutional layers, all with kernel size 3×3 and stride 1×1. The first 4 layers are using 64 filters and are followed by a ReLU activation function. The fifth convolutional layer in each CNN contains 2 filters, representing the real and imaginary part of the image. This fifth layer is not followed by any activation function, however its output is added to the input to the CNN using a skip connection.

2.4.2 Training parameters

In our experiments we used a pre-trained network that was trained (and published online) by the authors of [31] with training parameters documented in the paper [31]. The DeepMRINet was trained using a loss function with two terms, $\mathcal{L}_{\text{MSE}}$ and $\mathcal{L}_{\text{WEIGHTS}}$. The $\mathcal{L}_{\text{MSE}}$ term computed the mean squared error (MSE) between the true (complex valued) image and the predicted (complex valued) image, while the $\mathcal{L}_{\text{WEIGHTS}}$ computed the ℓ_2-norm of the weights. The loss function was then computed as

$$\mathcal{L}_{\text{TOTAL}} = \mathcal{L}_{\text{MSE}} + 10^{-7}\mathcal{L}_{\text{WEIGHTS}}.$$

The network weights were initialized using He initialization [16] and the Adam [21] optimizer was used for training. This optimizer takes as input a learning rate (step size) α, and two exponential decay parameters β_1 and β_2 related to a momentum term. We refer to [21] for further explanations of these parameters. The network was trained with $\alpha = 10^{-4}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and batch size equal 10.

2.4.3 Training data

The DeepMRINet was trained using data from five subjects from the MR dataset used in [6], which consists of 10 fully sampled short-axis cardiac cine scans. Each of these scans was then preprocessed, using the SENSE [27] software, into 30 temporal (complex-valued) frames of size 256×256. Synthetic MRI measurements were then obtained by sampling retrospectively the reconstructed images in k-space according to a Cartesian undersampling masks. During training, whereas a fixed undersampling rate of 33% was used, different undersampling masks were randomly generated in order to allow the network to recover images from measurements obtained with different undersampling masks. In particular, training images were fully sampled along the frequency-encoding direction but undersampled in the phase-encoding direction, according to the scheme described in [20] (center frequencies were always included in the subsampling patterns).

To prevent overfitting, data augmentation was performed by including rigid transformations of the considered images in the training datasets.

The code used to implement the DeepMRINet was written in Python using the Theano [8] and Lasagne[9] libaries.

2.5 FBPConvNet – The Ell 50 and Med 50 networks

The Ell 50 and Med 50 networks were proposed in [19] under the name FBPConvNet. The networks are trained to reconstruct images from Radon measurements. The networks have identical architecture and are trained using the same algorithm, with the same set of hyper parameters. The only difference between the training of the two networks, is the dataset they have been trained on. Below, we will describe the architecture and the training procedure of both the networks. We will then describe the datasets for the two networks in separate sections.

[7]https://github.com/js3611/Deep-MRI-Reconstruction

[8]http://deeplearning.net/software/theano

[9]https://lasagne.readthedocs.io/en/latest

2.5.1 Network architecture

The Ell 50 and Med 50 networks are trained for reconstructing x from measurements $y = Ax$ where $A \in \mathbb{R}^{m\times N}$ is a Radon[10] sampling operator, sampling 50 uniformly spaced radial lines. Rather than learning a mapping from y to x directly, the networks takes advantage of a discrete filtered back projection[11] $H \in \mathbb{R}^{N\times m}$, as described in the methods section, to obtain a noisy approximation $\tilde{x} = Hy$ to x. The operator H can be seen as a non-learnable first layer in the network.

The network contain several convolutional and deconvolutional layers, all of which (except the last) are followed by a batch normalization (BN) layer and a ReLU activation function. The (de)convolutional layers use filter size 3×3, stride 1×1 and a varying number of filters. We will not describe the full architecture in detail, as it can be seen in Figure 8, with the relevant number of filters, skip connections, max-poolings (2×2) and concatenations. We do, however, point out that the network applies a final global skip connection, so that rather than learning a mapping from $\tilde{x}$ to x the network is trying to learn the “noise" $x - \tilde{x}$.

2.5.2 Training parameters

The network weights were provided by the authors of [19] and obtained based on the training procedure as described in their paper [19]. The loss function used to train the networks is the ℓ^2 difference between the network output and the ground truth, and the networks are trained using the stochastic gradient descent algorithm with momentum. The learning rate varies from 0.01 to 0.001, whereas the momentum is set to 0.99, and the minibatch size is equal to 1. During training, gradients are clipped to the interval $[-I_{\max}, I_{\max}]$ with $I_{\max} = 10^{-2}$, to prevent the divergence of the cost function. The networks are trained for 101 epochs, and the code used to implement the networks is written in MATLAB using the library MatConvNet[12].

2.5.3 Ell 50 – Training data

The Ell 50 network is trained from the filtered back projection of 475 synthetic sinograms containing the Radon transform of ellipses of random intensity, size, and location. The dynamic range of the back projected images is adjusted so that image values are contained in the interval $[-500, 500]$. The Radon transform of an ellipse has an analytic formula, and hence this formula was used to create sinograms of such images using 1000 uniformly spaced lines (views). Measurement data are obtained by retaining 50 radial lines out of the 1000 views. The ground truth images were obtained by applying filtered back projection to fully sampled sinograms (i.e., 1000 radial lines). This approach is motivated by the fact that in applications, one will never have access to the underlying actual ground truth image. Data augmentation is also applied to the training data, by considering horizontal and vertical mirroring of the original images.

2.5.4 Med 50 – Training data

Med 50 is trained on synthetic images obtained from 475 real in-vivo CT images from the Low-dose Grand challenge competition database provided by the Mayo Clinic. The sinograms used for this training were synthetically generated from high quality CT-images using MATLAB `radon` command. The same approach as for the Ell 50 network was used, where one sampled 1000 view and used this as ground truth. The network was trained from 50 of these views.

2.6 MRI Variational Network (MRI-VN)

2.6.1 Network architecture

The MRI Variational Network (MRI-VN) presented in [14] is designed to reconstruct images from undersampled MRI data, sampled using 15 coil elements. Thus, we use the sampling operator $A = A_{pf}$ as described in the methods section, with $c = 15$.

The network structure is inspired by the unfolding of a variational minimization problem including a fidelity term and a regularization term defined according the Fields of Experts model [29]. In particular, each iteration of the corresponding Landweber method [23] corresponds to a layer of the resulting neural network. More specifically, the implementation considered in this work consists of $T = 10$ layers/iterations that can be expressed as follows:

$$u^{t+1} = u^t - (K^t)^T \Psi^t(K^t u^t) + \lambda^t A^*(Au^t - y), \quad 0 \le t < T \tag{15}$$

where $u^0 = Hy$ is the complex image obtained by applying $H = A^*_{pf}$. We will describe each of the remaining components of this network separately.

We start by noticing that the images $u^t \in \mathbb{C}^N$ (stacked as a vector in this simplified description) are complex valued, and can therefore described by its real and imaginary components u^t_{re} and u^t_{im}, respectively. We will alternate between the representations.

The operator $K^t \colon \mathbb{C}^N \to \mathbb{R}^{N\times N_k}$ acts as follows on u^t,

$$K^t u^t = K^t_{\text{re}} u^t_{\text{re}} + K^t_{\text{im}} u^t_{\text{im}},$$

where $K^t_{\text{re}}, K^t_{\text{im}} \colon \mathbb{R}^N \to \mathbb{R}^{N\times N_k}$, are learnable convolutional operators, with N_k filters (channels), filter size 11×11 and stride 1×1. We will comment on the value of N_k later.

[10]We used MATLABs `randon` command to represent this operator
[11]We used MATLABs `iradon` with linear interpolation and a ‘Ram-Lak‘ filter to represent this operator
[12]http://www.vlfeat.org/matconvnet

The $\Psi^t\colon \mathbb{R}^{N\times N_k} \to \mathbb{R}^{N\times N_k}$ is a non-linear activation function in the network. For each filter/channel, $i = 1, \ldots, N_k$ it applies the non-linear function

$$\phi_i^t(z) = \sum_{j=1}^{N_w} w_{ij}^t \exp\left(-\frac{(z-\mu_j)^2}{2\sigma^2}\right),$$

pointwise to each component z in that channel. Here $\{w_{ij}^t\}_{i=1,j=1}^{N_k,N_w}$, with $N_w = 31$, are weights which are learnt during the training phase. The nodes μ_j are non-learnable, and distributed in an equidistant manner on the interval $[-I_{\max}, I_{\max}]$, for a fixed value $I_{\max}$, commented on below. The σ is also non-learnable and equals $\frac{2I_{\max}}{N_w-1}$.

The operator $(K^t)^T\colon \mathbb{R}^{N\times N_k} \to \mathbb{C}^N$, maps $z \mapsto (K_{\text{re}}^t)^T z + \mathrm{i}(K_{\text{im}}^t)^T z$, where i is the imaginary unit, and $(K_{\text{re}}^t)^T, (K_{\text{im}}^t)^T$ are the transpose of $K_{\text{re}}^t, K_{\text{im}}^t$, respectively. The matrices A, A^* are the matrix A_{pf} and its adjoint, while λ^t is a learnable scalar. The remaining operations should be clear from Equation (15).

During training, each of the filters in K_{re}^t and K_{im} were restricted to have zero mean and have unit Euclidean norm. This was done to avoid a scaling problem with the weights w_{ij}.

To reproduce this network, we use the code published by the authors of [14][13]. Parts of this code uses slightly different parameters, than what was used in the original paper. In particular, the value $N_k = 24$ was chosen, rather than $N_k = 48$, as used in the paper. The value of $I_{\max}$, was also changed from 150 in the paper, to 1 in the code. The change of the $I_{\max}$ value is motivated by another change, also made in the published implementation, namely the scaling of the k-space values. In [14] the k-space volumes (with n_{sl} slices) was normalized by the factor $\sqrt{n_{sl}}10000/\|y_{\text{volume}}\|_2$, whereas in the code this have been changed to scaling each k-space slice y with $1/\|Hy\|_2$. This change has been made to make the their implementation more streamlined. Whenever there has been a conflict between the two sources, we have chosen the version found in the code.

2.6.2 Training parameters

The MRI-VN network is trained using the ℓ_2-norm as loss function. In particular, since MRI reconstruction are typically assessed through magnitude images, the error is evaluated by comparing smoothed version of magnitude images

$$|x|_\epsilon = \sqrt{(\text{Re}(x))^2 + (\text{Im}(x))^2 + \epsilon},$$

with $\epsilon = 10^{-12}$. The network parameters that minimize the loss function are determined using the inertial incremental proximal gradient (IIPG) optimizer (see [14, 22] for details). Optimization is performed for 1000 epochs, with a step size of 10^{-3}. Training data is arranged into minibatches of size 5. In the original paper, the batch size was set to 10, but due to memory limitations we had to adjust this.

2.6.3 Training data

The authors in [14] considered 5 datasets for different types of parallel MR imaging protocols, and trained one VN for each dataset. In this work, we have trained a VN for one of these protocols, namely *Coronal Spin Density weighted with Fat Suppression*. The training data consisted of knee images from 10 patients. From each patient we used 20 slices of the knee images, making up a total of 200 training images. The raw k-space data for each slice consisted of 15, k-space images of size 640×368, each with a precomputed sensitivity map. The sensitivity map was computed by the authors of [14], using ESPIRiT [35].

The raw data was obtained using a clinical 3T system (Siemens Magnetom Skyra) using an off-the-shelf 15-element knee coil. The raw data was subsampled retrospectively by zeroing out 85% of the k-space data. In [14] they test both a regular sampling scheme and a variable density pattern as proposed in [24]. In our work, we used a regular sampling scheme, where the 28 first central k-space lines were sampled, and the remaining lines were placed equidistantly in k-space. No data augmentation was used.

The code was implemented in Python with a custom made version of Tensorflow[14], which was partly implemented in C++/CUDA with cuDNN support. All the code and data have been made available online by the authors of [14].

[13]https://github.com/VLOGroup/mri-variationalnetwork

[14]https://github.com/VLOGroup/tensorflow-icg

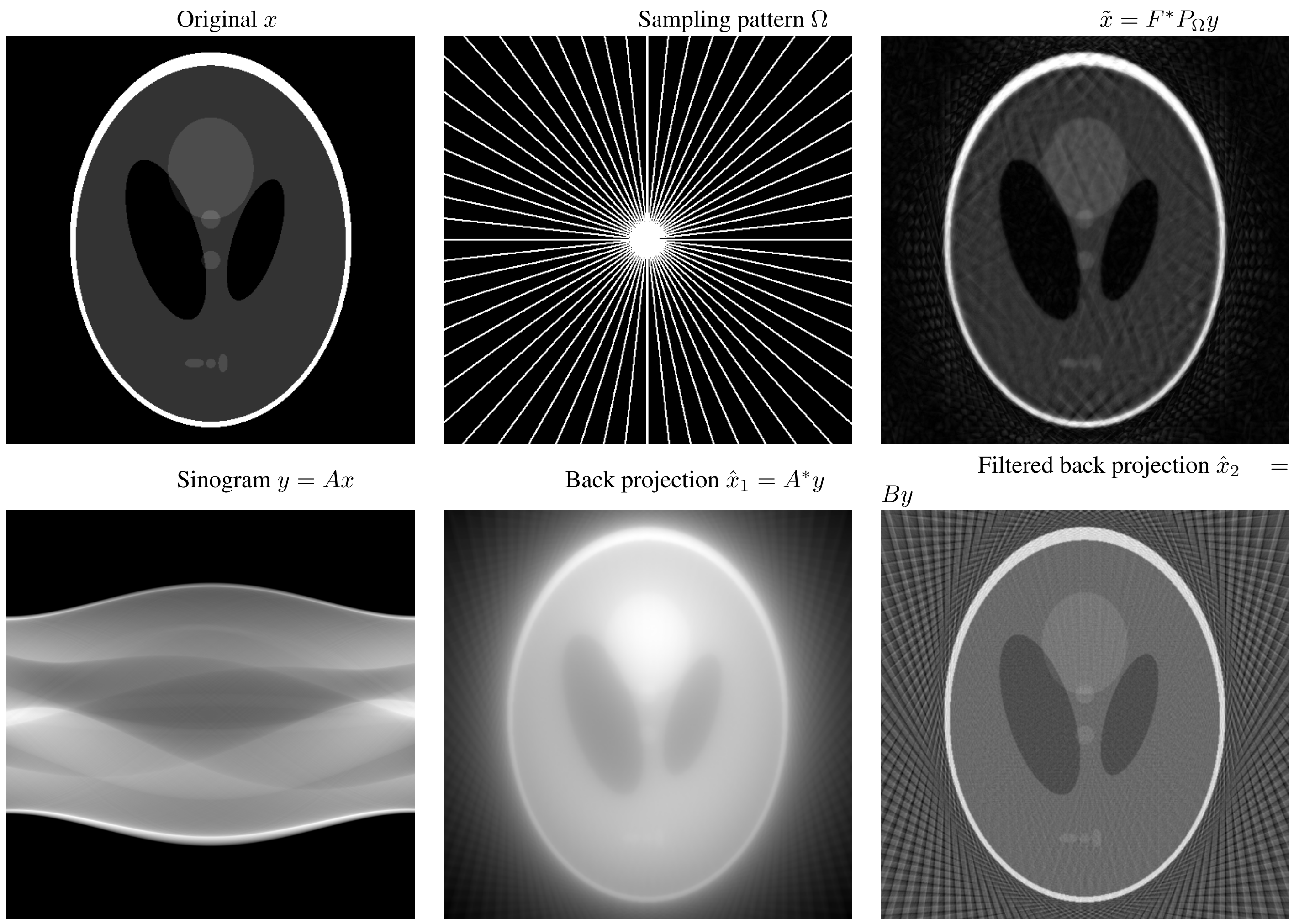
Original x
Sampling pattern Ω
$\tilde{x} = F^* P_\Omega y$
Sinogram $y = Ax$
Back projection $\hat{x}_1 = A^* y$
Filtered back projection $\hat{x}_2 = By$

Figure 1: (Under-sampled MRI and CT problem) We consider sampling $y = Ax$ for two different sampling modalities: the Fourier transform (upper figure simulating MRI) and the Radon transform (lower figure simulating CT) . Upper left: x is the original image. Upper middle: The white dots corresponds to the frequencies we sample i.e. the indices of the sampling pattern Ω. Upper right: The poorly reconstructed image $\tilde{x} = A^* y$, where $A = P_\Omega F$ and F is the 2-dimensional discrete Fourier transform transform and P_Ω is the projection onto the span of $\{e_j\}_{j\in\Omega}$ where the e_js denote the canonical basis. Lower left: Sinogram of Radon measurements using 729 radial lines, $y = Ax$ where A is the discrete Radon sampling matrix. Lower middle: The back projected blurry image $\hat{x} = A^* y$ obtained by using a radon matrix with 50 uniformly spaced lines. Lower right: The slightly sharper image $\hat{x}_2 = By$ using the filtered back projection with 50 uniformly spaced lines.

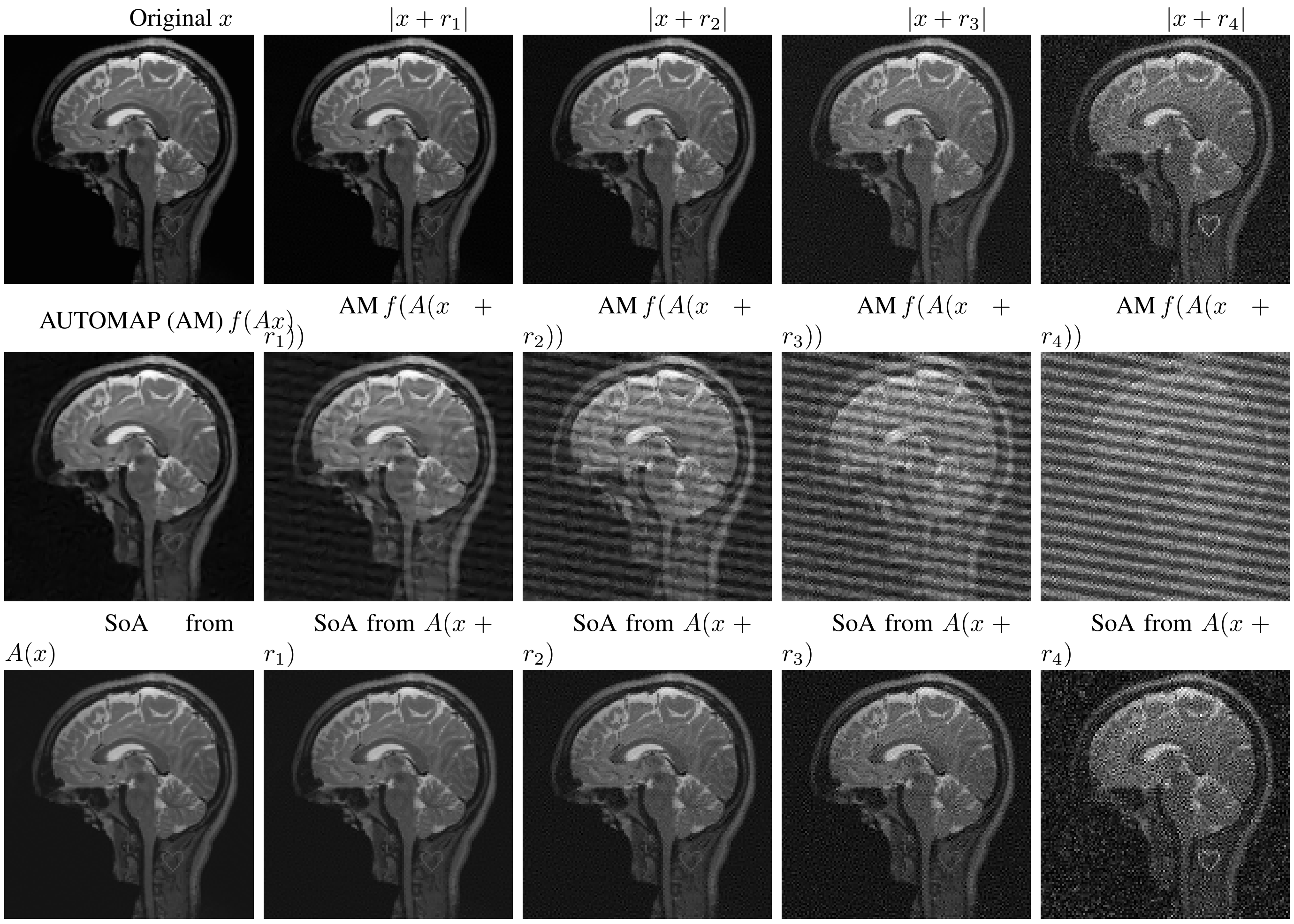

Figure 2: The experiment from Figure . is repeated, however, by using a different p in Algorithm 1. In particular, Figure . is produced by using $p(x) = x$, however, this figure is produced by using $p(x) = f(Ax)$. Note the substantial difference in the quality of the artefacts in the AUTOMAP reconstruction compared to Figure .

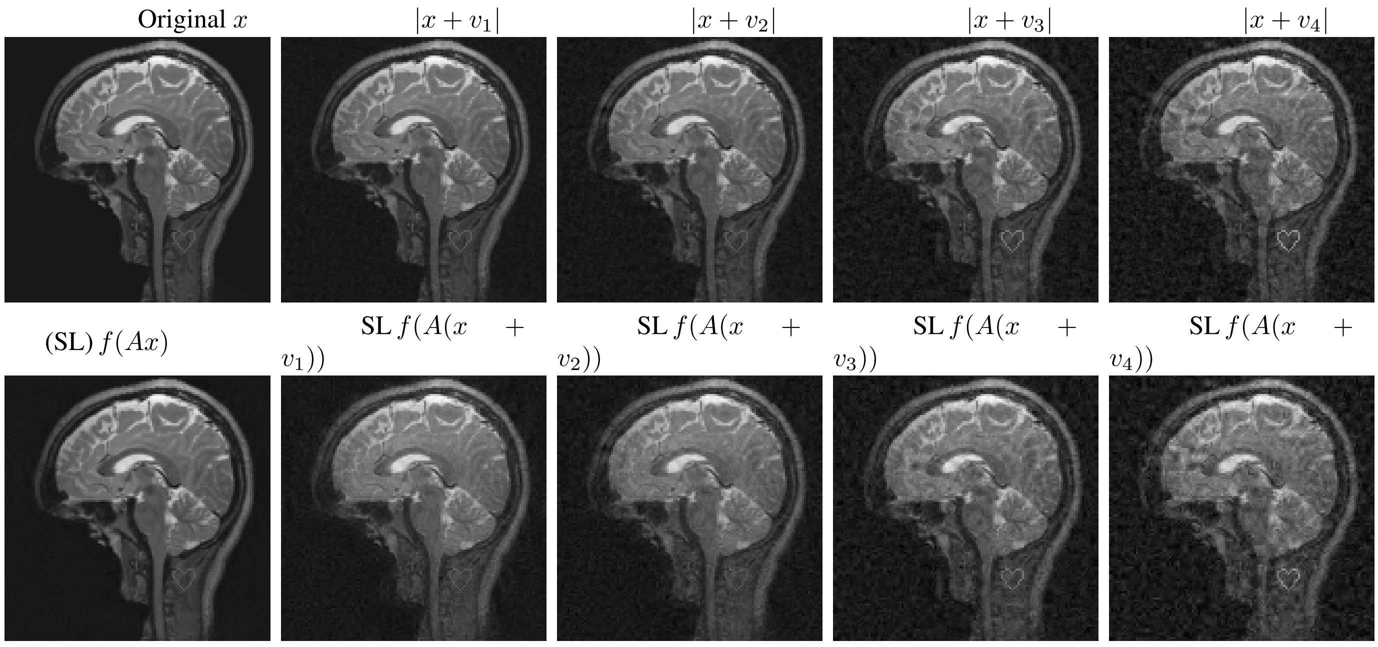

Figure 3: The recovery mapping f is based on unraveling an optimisation algorithm for solving the (square root) LASSO (SL) optimization problem in (14) used in compressed sensing. The perturbations v_j have been chosen so that $\|v_j\|_2 = \|r_j\|_2$ for $j = 1, 2, 3, 4$, where r_j are the perturbations used for Figure 3 in the main paper. The sampling operator $A \in \mathbb{C}^{m \times N}$ is the same in both experiments. As is evident from the images, this compressed sensing type recovery mapping f has a local Lipschitz constant that is quite reasonable, whereas the AUTOMAP network from Figure 3 in the main paper has a very large local Lipschitz constant.

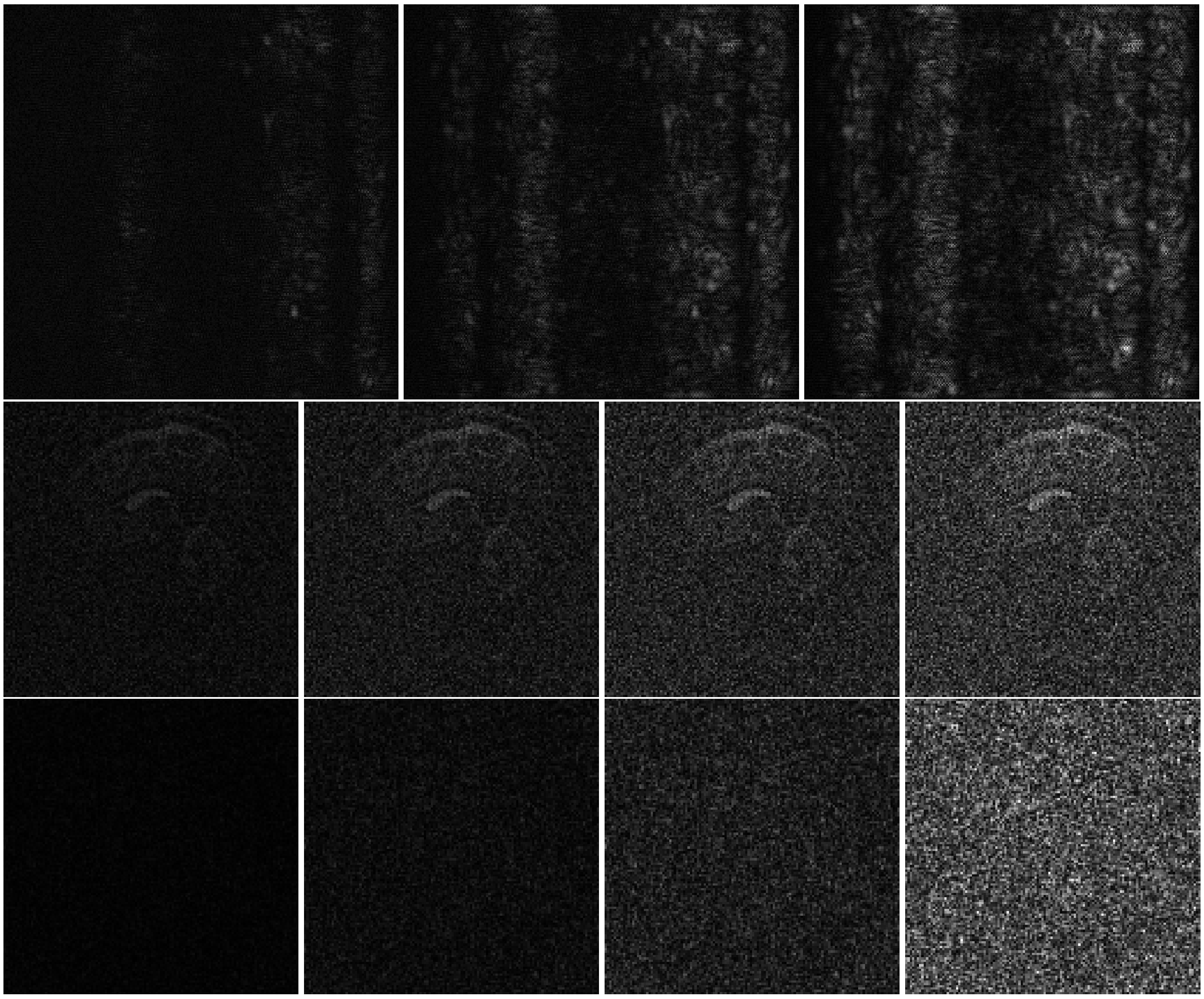

Figure 4: First row: We visualise the perturbations $|r_1|$, $|r_2|$ and $|r_3|$ used in Figure in the main manuscript to create the instabilities for the Deep MRI network. These perturbations have been rescaled to all lie in the same intensity range. The number of iterations in Algorithm 1 is given by $M = 2000, 4000, 6000$. Second and third row: We visualise the perturbations $|r_1|, |r_2|, |r_3|$ and $|r_4|$ used in Figure . (second row) and Figure 2 (third row) to create the instabilities for the AUTOMAP network. These perturbations have been rescaled to all lie in the same intensity range. The number of iterations in Algorithm 1 is given by $M = 12, 16, 20, 24$, for the second row and $M = 160, 170, 177, 183$ for the third row. The values of λ, γ, η and τ used in in Algorithm 1 are given in Table 1.

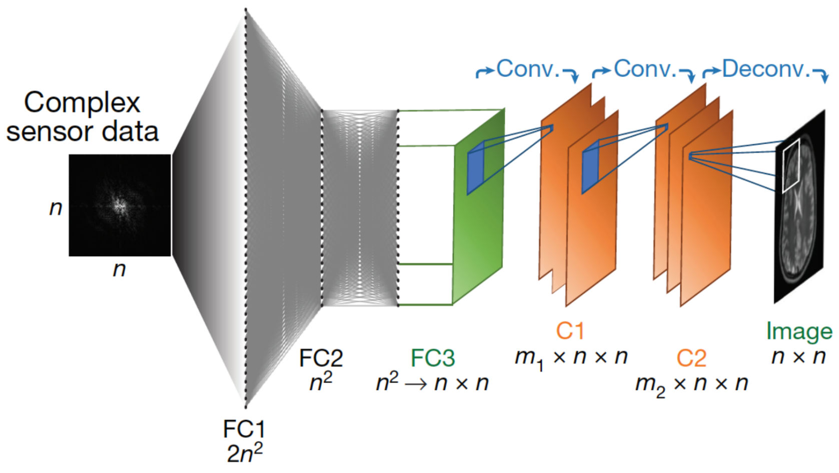

Figure 5: The AUTOMAP architecture (figure from [39]).

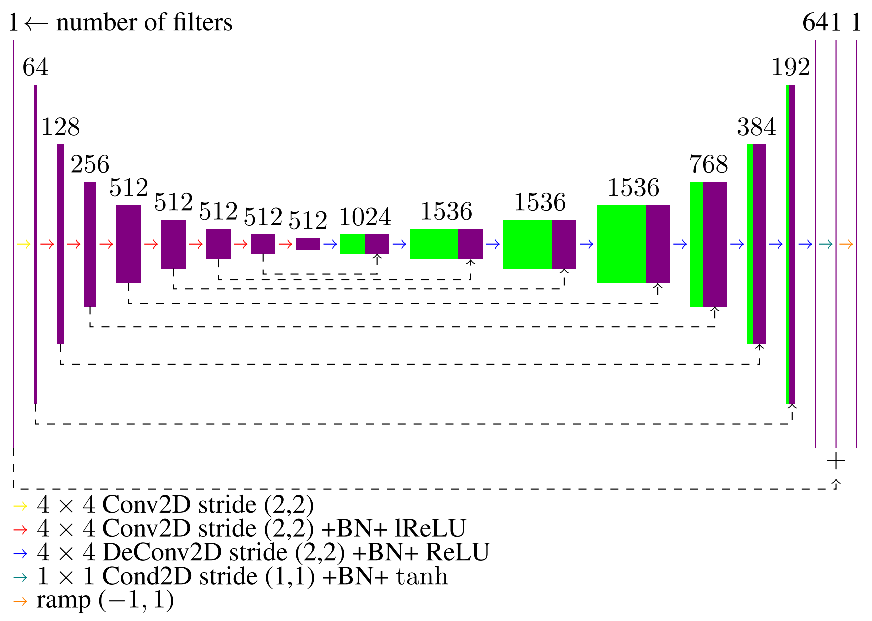
1 ← number of filters
64
128
256
512
512
512
512
512
1024
1536
1536
1536
768
384
192
641
1
+
4 × 4 Conv2D stride (2,2)
4 × 4 Conv2D stride (2,2) +BN+ lReLU
4 × 4 DeConv2D stride (2,2) +BN+ ReLU
1 × 1 Cond2D stride (1,1) +BN+ tanh
ramp (−1, 1)

Figure 6: DAGAN architecture. Here lReLU is the leaky ReLU function with slope parameter equal to 0.2.

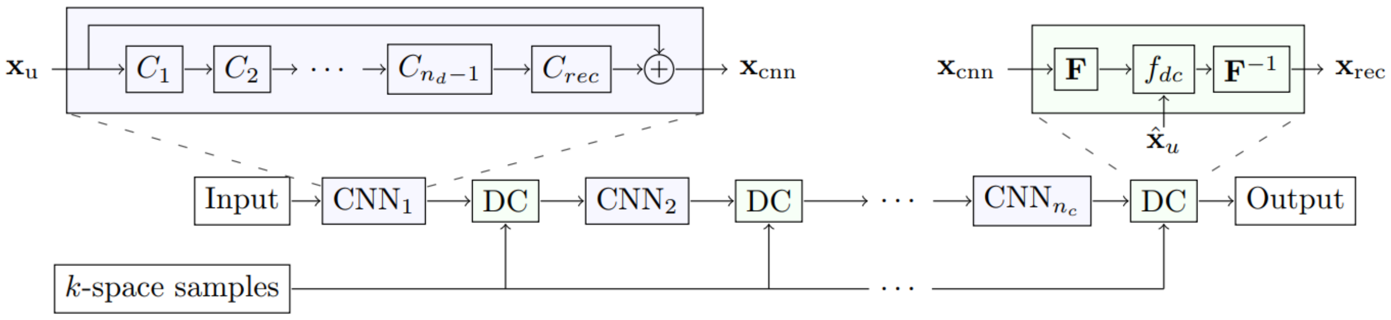

Figure 7: The DeepMRINet architecture (figure from [31]).

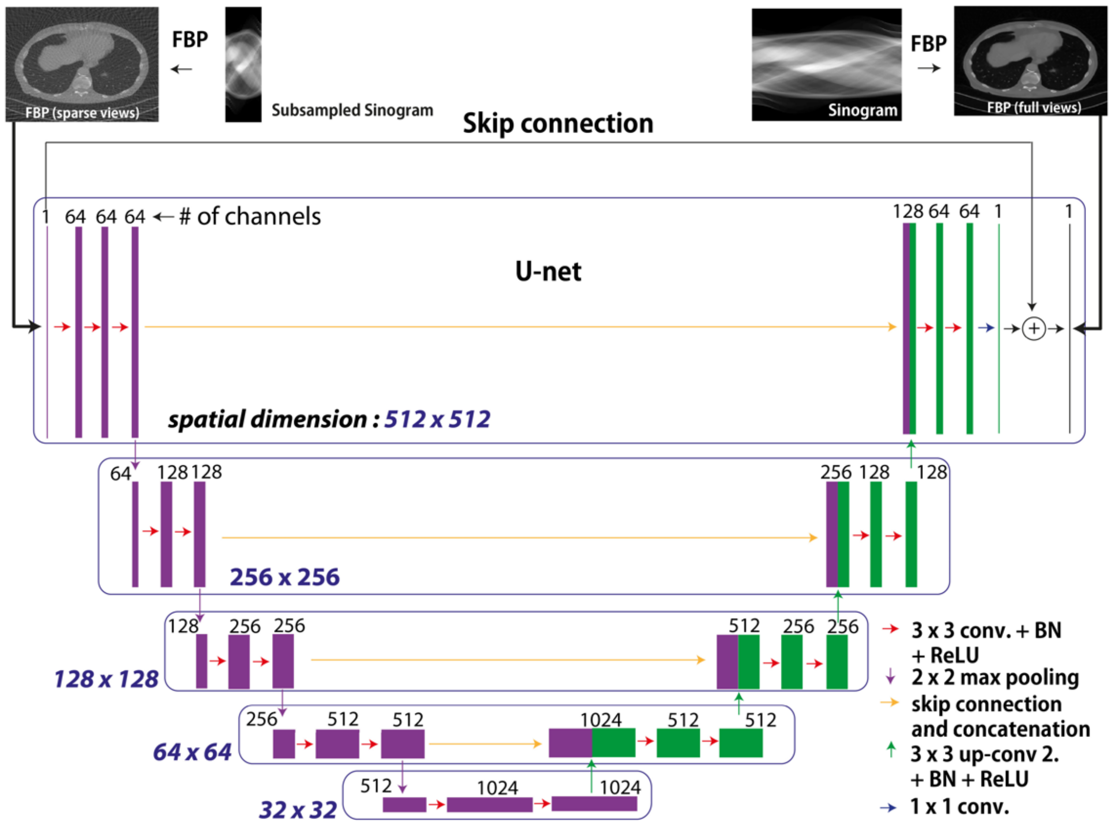

Figure 8: The Ell 50 and Med 50 architecture (figure from [19]).

Table 1: Summary of the different choices of parameters leading to the results reported in Figures , , 4 and 2.

Neural Network	λ	γ	η	τ	$p(x)$
Deep MRI	0.001	0.9	0.01	0.01	$f(Ax)$
AUTOMAP Fig.	0.1	0.9	0.001	10^{-5}	$f(Ax)$
AUTOMAP Fig. 2	0.1	0.9	0.001	10^{-5}	x
MRI-VN	1	0.9	0.005	0.001	$f(Ax)$
MED 50	20	0.9	0.005	0.005	$f(Ax)$

References

[1] B. Adcock and A. C. Hansen. *Compressive imaging*. Cambridge University Press, 2020. In press.

[2] B. Adcock, A. C. Hansen, C. Poon, and B. Roman. Breaking the coherence barrier: A new theory for compressed sensing. *Forum of Mathematics, Sigma*, 1-84, 2017.

[3] J. Adler and O. Öktem. Solving ill-posed inverse problems using iterative deep neural networks. *Inverse Problems*, 33(12):124007, 2017.

[4] S. A. Bigdeli, M. Zwicker, P. Favaro, and M. Jin. Deep mean-shift priors for image restoration. In *Advances in Neural Information Processing Systems*, pages 763–772, 2017.

[5] H. C. Burger, C. J. Schuler, and S. Harmeling. Image denoising: Can plain neural networks compete with BM3D? In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2392–2399, 06 2012.

[6] J. Caballero, A. N. Price, D. Rueckert, and J. V. Hajnal. Dictionary learning and time sparsity for dynamic MR data reconstruction. *IEEE Transactions on Medical Imaging*, 33(4):979–994, 2014.

[7] E. J. Candès, J. Romberg, and T. Tao. Robust uncertainty principles: exact signal reconstruction from highly incomplete frequency information. *IEEE Transactions on Information Theory*, 52(2):489–509, 2006.

[8] E. J. Candes, M. B. Wakin, and S. P. Boyd. Enhancing sparsity by reweighted ℓ^1 minimization. *Journal of Fourier Analysis and Applications*, 14(5-6):877–905, 2008.

[9] D. L. Donoho. Compressed sensing. *IEEE Transactions on Information Theory*, 52(4):1289–1306, 2006.

[10] Q. Fan, T. Witzel, A. Nummenmaa, K. Van Dijk, J. D. Van Horn, M. K. Drews, L. H. Somerville, M. A. Sheridan, R. M. Santillana, J. Snyder, T. Hedden, E. E. Shaw, M. O. Hollinshead, V. Renvall, R. Zanzonico, B. Keil, S. Cauley, J. R. Polimeni, D. Tisdall, R. L. Buckner, W. V. J., W. L. L., T. A. W., and B. R. Rosen. MGH–USC human connectome project datasets with ultra-high b-value diffusion MRI. *Neuroimage*, 124:1108–1114, 2016.

[11] A. Fawzi, S. M. Moosavi Dezfooli, and P. Frossard. The robustness of deep networks – A geometric perspective. *IEEE Signal Processing Magazine*, 34(6), 11 2017.

[12] N. M. Gottschling, V. Antun, B. Adcock, and A. C. Hansen. The troublesome kernel: why deep learning for inverse problems is typically unstable. *arXiv: 2001.01258*, 2020.

[13] H. Gupta, K. H. Jin, H. Q. Nguyen, M. T. McCann, and M. Unser. CNN-based projected gradient descent for consistent CT image reconstruction. *IEEE transactions on medical imaging*, 37(6):1440–1453, 2018.

[14] K. Hammernik, T. Klatzer, E. Kobler, M. P. Recht, D. K. Sodickson, T. Pock, and F. Knoll. Learning a variational network for reconstruction of accelerated MRI data. *Magnetic Resonance in Medicine*, 79(6):3055–3071, 2018.

[15] K. M. Haug. Stability of adaptive neural networks for image reconstruction, 2019. Master's thesis, , https://www.duo.uio.no/handle/10852/69486.

[16] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *IEEE International Conference on Computer Vision*, pages 1026–1034, 2015.

[17] Q. Huynh-Thu and M. Ghanbari. Scope of validity of PSNR in image/video quality assessment. *Electronics Letters*, 44(13):800–801, 2008.

[18] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *The 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456. PMLR, 07 2015.

[19] K. H. Jin, M. T. McCann, E. Froustey, and M. Unser. Deep convolutional neural network for inverse problems in imaging. *IEEE Transactions on Image Processing*, 26(9):4509–4522, 2017.

[20] H. Jung, J. C. Ye, and E. Y. Kim. Improved k–t BLAST and k–t SENSE using FOCUSS. *Physics in Medicine & Biology*, 52(11):3201, 2007.

[21] D. Kingma and J. Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[22] E. Kobler, T. Klatzer, K. Hammernik, and T. Pock. Variational networks: connecting variational methods and deep learning. In *German Conference on Pattern Recognition*, pages 281–293. Springer, 2017.

[23] L. Landweber. An iteration formula for fredholm integral equations of the first kind. *American Journal of Mathematics*, 73(3):615–624, 1951.

[24] M. Lustig, D. Donoho, and J. M. Pauly. Sparse MRI: The application of compressed sensing for rapid MR imaging. *Magnetic Resonance in Medicine: An Official Journal of the International Society for Magnetic Resonance in Medicine*, 58(6):1182–1195, 2007.

[25] J. Ma and M. März. A multilevel based reweighting algorithm with joint regularizers for sparse recovery. *arXiv preprint arXiv:1604.06941*, 2016.

[26] J. Ma, M. Marz, S. Funk, J. Schulz-Menger, G. Kutyniok, T. Schaeffter, and C. Kolbitsch. Shearlet-based compressed sensing for fast 3D cardiac MR imaging using iterative reweighting. *Physics in Medicine & Biology*, 63(23):235004, 2018.

[27] K. P. Pruessmann, M. Weiger, M. B. Scheidegger, P. Boesiger, et al. SENSE: sensitivity encoding for fast MRI. *Magnetic Resonance in Medicine*, 42(5):952–962, 1999.

[28] J. Rick Chang, C.-L. Li, B. Poczos, B. Vijaya Kumar, and A. C. Sankaranarayanan. One network to solve them all–solving linear inverse problems using deep projection models. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5888–5897, 2017.

[29] S. Roth and M. J. Black. Fields of experts. *International Journal of Computer Vision*, 82(2):205, 2009.

[30] L. I. Rudin, S. Osher, and E. Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D: Nonlinear Phenomena*, 60(1-4):259–268, 1992.

[31] J. Schlemper, J. Caballero, J. V. Hajnal, A. Price, and D. Rueckert. A deep cascade of convolutional neural networks for MR image reconstruction. In *International Conference on Information Processing in Medical Imaging*, pages 647–658. Springer, 2017.

[32] P. Y. Simard, D. Steinkraus, and J. C. Platt. Best practices for convolutional neural networks applied to visual document analysis. In *Seventh International Conference on Document Analysis and Recognition*, pages 958–963, 08 2003.

[33] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[34] K. C. Tezcan, C. F. Baumgartner, R. Luechinger, K. P. Pruessmann, and E. Konukoglu. MR image reconstruction using deep density priors. *IEEE Transactions on Medical Imaging*, 38(7):1633–1642, 7 2019.

[35] M. Uecker, P. Lai, M. J. Murphy, P. Virtue, M. Elad, J. M. Pauly, S. S. Vasanawala, and M. Lustig. ESPIRiT-an eigenvalue approach to autocalibrating parallel MRI: where SENSE meets GRAPPA. *Magnetic Resonance in Medicine*, 71(3):990–1001, 2014.

[36] M. Uecker, F. Ong, J. I. Tamir, D. Bahri, P. Virtue, J. Y. Cheng, T. Zhang, and M. Lustig. Berkeley advanced reconstruction toolbox. In *Proc. Intl. Soc. Mag. Reson. Med*, volume 23, page 2486, 2015.

[37] E. van den Berg and M. P. Friedlander. Probing the pareto frontier for basis pursuit solutions. *SIAM Journal on Scientific Computing*, 31(2):890–912, 2008.

[38] G. Yang, S. Yu, H. Dong, G. Slabaugh, P. L. Dragotti, X. Ye, F. Liu, S. Arridge, J. Keegan, Y. Guo, and D. Firmin. DAGAN: Deep de-aliasing generative adversarial networks for fast compressed sensing MRI reconstruction. *IEEE Transactions on Medical Imaging*, 37(6):1310–1321, 6 2018.

[39] B. Zhu, J. Z. Liu, S. F. Cauley, B. R. Rosen, and M. S. Rosen. Image reconstruction by domain-transform manifold learning. *Nature*, 555(7697):487, 03 2018.