DEEP LEARNING IN COMPUTATIONAL IMAGING

1 Introduction

1.1 Project in a nutshell

Computational imaging is an approach to the process of imaging a scene of interest, where the acquired data are not the sought images themselves, but rather contain only incomplete information about these images (*e.g.* due to time, cost, or physical constraints inherent to the sensing procedure). The data is often not acquired in the image domain itself, but rather in a transform domain (*e.g.* the Fourier domain). In this context, an ill-posed inverse problem appears for image formation, and advanced algorithms are needed to reconstruct an image from data. Important imaging modalities in science and technology, ranging from astronomy to medicine, fall into this category.

In the first part of the course, we have introduced the framework of convex optimisation, which provides a whole wealth of algorithms enabling to solve imaging inverse problems. We have in particular studied proximal splitting methods, such as the forward-backward algorithm, and the Alternating Direction Method of Multipliers (ADMM). The second part of the course will take the form of a group project which will enable us to explore how machine learning algorithms, more specifically deep neural networks, can provide an alternative framework to solve imaging inverse problems, or otherwise integrate and enhance optimisation algorithms. For the sake of the illustration, the project will focus on a specific computational imaging modality known as Magnetic Resonance (MR) Imaging, which is a state-of-the-art imaging technique in medicine.

1.2 MR imaging

MR imaging is a non-invasive non-ionising medical imaging technique that finds its superiority in the flexibility of its contrast mechanisms. It comes in various modalities ranging from structural imaging aiming at mapping detailed tissue structures, or diffusion imaging mapping the structural neuronal connectivity by probing molecular diffusion in each voxel of the brain, to dynamic imaging mapping for example the heart movements. MR imaging is a good example of how it is possible to play with physical phenomena to encode data. Here a phenomenon called “magnetic resonance”, which relates to the alignment of the spins of protons in a magnetic field, enables the sequential measurement of Fourier coefficients of the image of interest [1]. MR data acquisition is intrinsically long, sometimes prohibitively, due to its sequential nature. MR acquisition sequences that are fast and simultaneously enable high-resolution imaging constitutes a big challenge for medical research. In this context, the acquisition of incomplete Fourier data (*i.e.* of only part of the Fourier coefficients) is a commonly used acceleration strategy. This places the modality in the category of computational imaging techniques, with advanced algorithms required for image reconstruction from the data.

1.3 Deep Learning?

Recently, Deep Neural Networks (DNNs), with mixed architectures, including simple feed-forward networks, variational networks, generative adversarial networks, autoencoders, and convolutional neural networks (CNNs) such as U-net, were demonstrated solving inverse imaging problems with outstanding precision and for a variety of applications, in particular in medical imaging [2–5]. For illustration, a CNN will process data, in the form of an input “corrupted” image, through the sequential application

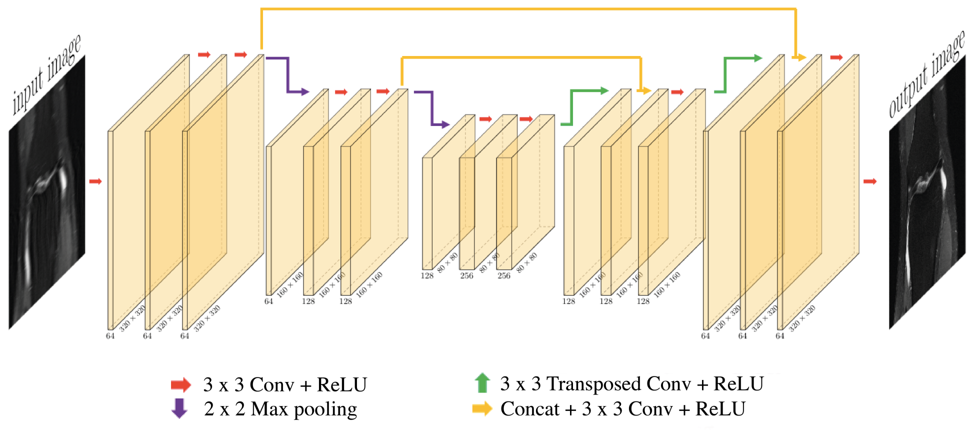

Figure 1: Illustration of a generic U-net CNN architecture. The network will be fed with an "input image" and produce an "output image". The input is progressively processed through the sequential application of simple activation functions (typically thresholding operations such as ReLU) and convolutions with small kernels (Conv) over a number of layers. On top of that, "Max pooling", "Transposed Conv", and "Concat" respectively perform down-sampling, up-sampling, and concatenation operations, which all help creating a complex structure of "feature maps" (yellow planes) over the layers. The parameters of the network (typically the convolution kernels that lead to the feature maps) will be *learned*, *i.e.* their values will be chosen to optimise the output of the network over a *training data set* of input/output images. The more complex the architecture, the larger the inference power of the network.

of simple nonlinear activation functions and small convolutions over a number of layers, to produce an output "corrected" image (see Figure 1). If the network is not too deep, *i.e.* the number of layers is not too high, its application may be extremely fast in comparison with an optimisation algorithm, opening the door to a significant scalability to the large image and data dimensions characterising modern applications. The computational cost is transferred to training the DNN offline. The training task itself is actually performed using optimisation algorithms! It requires large training databases, as well as powerful computation resources, in particular dedicated GPU systems. The lack of theoretical understanding of DNNs however raises the question of the reliability of the estimated image. In particular, the relation between the data and the underlying image is, in most applications, dependent on a variety of acquisition parameters. Training can only be performed over a limited range of conditions, making the question of the power of generalisation of a DNN to imaging conditions outside those considered during training a constant challenge.

Emerging "Plug-and-Play" (PnP) methods in optimisation propose to replace the proximal operator associated with the regularisation term in a proximal splitting optimisation method by a more general denoising operator characterising a more general prior image model [6–8]. They were also shown to provide outstanding imaging precision, in particular when the image model is learnt and the corresponding denoiser takes the form of a DNN [9–15]. Interestingly, the DNN being merely a denoiser, it becomes agnostic to the data acquisition conditions. The PnP approach thus solves the generalisation issue mentioned above.

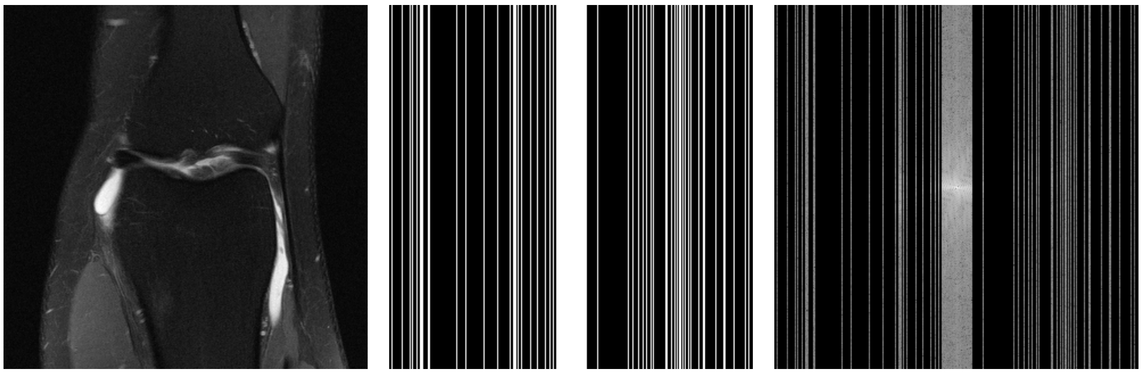

Figure 2: Left: the groundtruth $\boldsymbol{x}^o$ we wish to recover, center: the Fourier mask $\mathrm{diag}(\mathbf{M}^\dagger\mathbf{M})$, and right: the observed Fourier data correctly positioned in the Fourier plane $\mathbf{M}^\dagger\boldsymbol{y}$ (in logscale of the modulus).

2 The project

2.1 Problem formulation and objectives

Problem formulation. In MR imaging, one wishes to recover an image $\boldsymbol{x}^o \in \mathbb{R}^N$ from a noisy and sub-sampled version of its Fourier coefficients $\boldsymbol{y} \in \mathbb{C}^M$, with $M < N$. The acquisition process can be formulated as

$$\boldsymbol{y} = \boldsymbol{\Phi}^\dagger \boldsymbol{x}^o + \boldsymbol{n}, \quad \text{with} \quad \boldsymbol{\Phi}^\dagger = \mathbf{MF}, \tag{1}$$

where $\mathbf{F} \in \mathbb{C}^{N\times N}$ denotes the discrete Fourier transform, $\mathbf{M} \in \{0,1\}^{M\times N}$ is a matrix implementing the selection of a subset of Fourier coefficients (each of its M lines contains a single nonzero value indexing a spatial frequency to be sampled), and $\boldsymbol{n} \in \mathbb{C}^M$ is some additive i.i.d. Gaussian noise (both its real and imaginary parts follow a normal distribution with zero mean and variance $\sigma^2/2$). We consider a class of Fourier sampling patterns where observed frequencies are organised by lines in the Fourier plane of the image in order to fit technical constraints on the design of an MR acquisition sequence. All low frequency lines are sampled, while higher frequencies are selected uniformly at random (see Figure 2).

Objectives. The aim of this project is to implement 3 algorithms to solve the inverse imaging problem, *i.e.* for the recovery of $\boldsymbol{x}^o$ from $\boldsymbol{y}$, and to compare them in terms of both quality of the recovered image, and computation time. The 3 methods are: (**M1**) A pure optimisation method based on the ADMM algorithm; (**M2**) A pure deep learning method based on the U-net CNN; (**M3**) A hybrid PnP method where a simple CNN trained as a denoiser is plugged into ADMM to substitute the proximity operator of the regularisation term.

2.2 M1

In order to solve (1), one may introduce a sparsity regularisation term, in a well chosen sparsity basis. The estimation $\boldsymbol{x}^\star$ of $\boldsymbol{x}^o$ will be defined as a solution of the following optimisation problem:

$$\boldsymbol{x}^\star = \underset{\boldsymbol{x}}{\mathrm{argmin}} \ \|\boldsymbol{\Psi}^\dagger \boldsymbol{x}\|_1 \quad \text{subject to} \quad \|\boldsymbol{y} - \boldsymbol{\Phi}^\dagger \boldsymbol{x}\|_2 \le \varepsilon, \tag{2}$$

where $\Psi^\dagger$ is the (orthonormal) Db4 basis and $\varepsilon = \sigma\sqrt{M + 2\sqrt{M}}$. The ADMM algorithm for solving (2) reads

$$\begin{aligned}
\boldsymbol{x}_{t+1} &= \boldsymbol{\Psi}\, \text{shrink}\left(\boldsymbol{\Psi}^\dagger\left(\boldsymbol{x}_t - \delta\Re(\boldsymbol{\Phi}(\boldsymbol{s}_t + \boldsymbol{n}_t - \bar{\boldsymbol{\nu}}_t))\right), \delta\rho^{-1}\right)\\
\boldsymbol{s}_{t+1} &= \boldsymbol{\Phi}^\dagger \boldsymbol{x}_{t+1} - \boldsymbol{y}\\
\boldsymbol{n}_{t+1} &= \mathcal{P}_{B(0,\varepsilon)}(\bar{\boldsymbol{\nu}}_t - \boldsymbol{s}_{t+1})\\
\bar{\boldsymbol{\nu}}_{t+1} &= \bar{\boldsymbol{\nu}}_t - (\boldsymbol{s}_{t+1} + \boldsymbol{n}_{t+1}),
\end{aligned} \tag{3}$$

where $\Re$ denotes the real part, $\delta > 0$ satisfies $\delta \leq \sigma_{\max}^{-1}(\Re(\boldsymbol{\Phi}\boldsymbol{\Phi}^\dagger))$, and $\rho > 0$ is a free parameter. The value of this parameter will usually be fine-tuned to maximise the reconstruction quality over some test set of groundtruth images, *i.e.* a data set of T_e pairs $(\boldsymbol{y}_i, \boldsymbol{x}_i^o)_{i\in T_e}$. A properly tuned value of ρ should lead to $\boldsymbol{x}_i^\star \simeq \boldsymbol{x}_i^o\ \forall i \in T_e$. If the required accuracy is not reached, the value of ρ needs to be changed. Or more importantly the regularisation term and the objective function might have to be adapted!

Note that, in theory, the optimisation algorithm will converge after an infinite number of iterations. In a practical implementation the iterative process will typically be stopped when the relative variation of consecutive iterates is smaller than a fixed bound $\alpha > 0$, *i.e.* $\|x_{t+1} - x_t\|_2/\|x_{t+1}\|_2 \leq \alpha$. Note that other criteria can also be applied.

2.3 M2

What is a neural network? Given a specific architecture (sequence of nonlinear activation functions and convolution operations, see Figure 1), a neural network can be represented as a highly nonconvex function $\mathcal{G}$ of a set of parameters $\boldsymbol{\theta}$. The latter are optimised in order to have $\mathcal{G}$ solve a specific problem, which can be formalised as recovering some variable $\boldsymbol{x}^o$ from input data $\boldsymbol{y}$. Ideally, one seeks that the network output $\mathcal{G}(\boldsymbol{y}, \boldsymbol{\theta}^\star)$ be equal to $\boldsymbol{x}^o$. In order to find the optimal value $\boldsymbol{\theta}^\star$ of the parameters, the network is *trained* by minimizing a loss (objective) function l over a *training data set* of T_r pairs $(\boldsymbol{y}_i, \boldsymbol{x}_i^o)_{i\in T_r}$. A widely used loss function is the mean squared error over the data set between the network output $\mathcal{G}(\boldsymbol{y}, \boldsymbol{\theta})$ and the true variable $\boldsymbol{x}^o$:

$$\boldsymbol{\theta}^\star = \underset{\boldsymbol{\theta}}{\text{argmin}}\ \frac{1}{M}\sum_{i=1}^{M} \|\mathcal{G}(\boldsymbol{y}_i, \boldsymbol{\theta}) - \boldsymbol{x}_i^o\|_2^2. \tag{4}$$

Algorithms such as Stochastic Gradient Descent (SGD) and Adaptive Moment Estimation (ADAM) solve problem (4) efficiently (even though this problem is nonconvex). Just as ADMM, these algorithmic structures contain free parameters that will affect the optimal value $\boldsymbol{\theta}^\star$. Once the network is trained, it is evaluated on a separate *testing data set*, *i.e.* a data set of T_e pairs $(\boldsymbol{y}_i, \boldsymbol{x}_i^o)_{i\in T_e}$ on which the network has *not* been trained. A properly trained network should satisfy $\mathcal{G}(\boldsymbol{y}_i, \boldsymbol{\theta}^\star) \simeq \boldsymbol{x}_i^o\ \forall i \in T_e$. If the required accuracy is not reached, the free parameters need to be re-tuned and the network re-trained. Or more importantly the architecture $\mathcal{G}$ of the network might have to be adapted!

Application to imaging inverse problems. To solve an imaging inverse problem, a neural network will aim to estimate the sought image $\boldsymbol{x}^o$ from the incomplete data $\boldsymbol{y}$. For our problem, the training and testing data sets should be built as $T_r + T_e$ input/output pairs $(\boldsymbol{y}_i, \boldsymbol{x}_i^o)_{i\in T_r+T_e}$, where $\boldsymbol{y}_i$ is related to the groundtruth image $\boldsymbol{x}_i^o$ through (1). Note that working on pairs where both input and output are represented as images greatly simplifies the network architecture and training procedure. One can simply

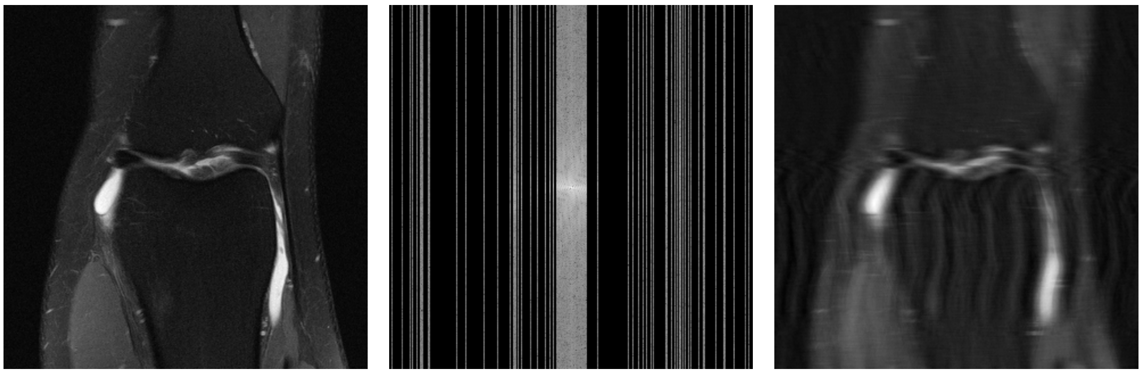

Figure 3: Left: the groundtruth $\boldsymbol{x}^o$ we wish to recover, middle: the observed Fourier data correctly positioned in the Fourier plane $\mathbf{M}^\dagger \boldsymbol{y}$ (in logscale of the modulus), and right: the backprojected image $\tilde{\boldsymbol{x}} = \Re\{\boldsymbol{\Phi}\boldsymbol{y}\}$.

pre-process the data $\boldsymbol{y}$ and project them onto the image domain to produce the so-called "backprojected image"

$$\tilde{\boldsymbol{x}} = \Re\{\boldsymbol{\Phi}\boldsymbol{y}\} = \Re\{\boldsymbol{\Phi}(\boldsymbol{\Phi}^\dagger \boldsymbol{x}^o + \boldsymbol{n})\} \in \mathbb{R}^N, \tag{5}$$

and work with input/output pairs $(\tilde{\boldsymbol{x}}_i, \boldsymbol{x}_i^o)_{i \in T_r + T_e}$ (see Figure 3 for an illustration).

The full design of an appropriate network architecture is a complicated question lying beyond the scope of the current project. We will choose and adapt here an existing architecture dubbed U-net [16], which has shown good results in medical imaging [2]. The explicit U-net architecture proposed is detailed in Appendix A. The standard loss (4) will be used.

2.4 M3

The PnP version of ADMM [9, 11] simply reads,

$$\begin{aligned}
\boldsymbol{x}_{t+1} &= \mathcal{G}\left(\boldsymbol{x}_t - \delta \Re(\boldsymbol{\Phi}(\boldsymbol{s}_t + \boldsymbol{n}_t - \bar{\boldsymbol{\nu}}_t))\right) \\
\boldsymbol{s}_{t+1} &= \boldsymbol{\Phi}^\dagger \boldsymbol{x}_{t+1} - \boldsymbol{y} \\
\boldsymbol{n}_{t+1} &= \mathcal{P}_{B(0,\varepsilon)}(\bar{\boldsymbol{\nu}}_t - \boldsymbol{s}_{t+1}) \\
\bar{\boldsymbol{\nu}}_{t+1} &= \bar{\boldsymbol{\nu}}_t - (\boldsymbol{s}_{t+1} + \boldsymbol{n}_{t+1}),
\end{aligned} \tag{6}$$

where the stepsize $\delta > 0$ satisfies $\delta \leq \sigma_{\max}^{-1}(\Re(\boldsymbol{\Phi}\boldsymbol{\Phi}^\dagger))$ and $\varepsilon = \sigma\sqrt{M + 2\sqrt{M}}$, and a denoising neural network $\mathcal{G}$ is substituted to the prox of the regularisation term in the objective function (2), imposing a *learned image model*. As the network is to be applied at each iteration of ADMM (rather than only once as in M2), we choose a simpler "DnCNN" architecture [10, 15]. This will in particular induce a faster application of $\mathcal{G}$, which will help contain the overall computation time. The explicit DnCNN architecture proposed is from [15], and detailed in Appendix B.

Recall that a network is trained by minimizing a loss function over a training data set T_r using optimisation algorithms containing additional free parameters that need tuning. Once the network is trained, it is evaluated on a testing data set T_e, and if the required accuracy is not reached, the free parameters need to be re-tuned and the network re-trained, or more importantly the architecture of the network might have to be adapted!

The training and testing data sets are built as input/output pairs $(\hat{\boldsymbol{x}}_i, \boldsymbol{x}_i^o)_{i \in T_r + T_e}$, where $\hat{\boldsymbol{x}}_i$ are noisy versions of the groundtruth images $\boldsymbol{x}_i^o$:

$$\hat{\boldsymbol{x}}_i = \boldsymbol{x}_i^o + \bar{\boldsymbol{n}}_i, \tag{7}$$

with $\bar{\boldsymbol{n}}_i \in \mathbb{R}^N$ some additive i.i.d. Gaussian noise, whose components follow a normal distribution with zero mean and variance $\bar{\sigma}^2$. Note that this variance $\bar{\sigma}^2$ should be adapted to the variance σ^2 of the noise on the observed data.

Note that the convergence in PnP version of ADMM is not proven for standard network architectures such as DnCNN [7, 8, 11, 15]. It is therefore safe to stop the algorithm after a fixed number of iterations, rather than "hoping" for the relative variation of consecutive iterates to be smaller than a fixed bound.

3 Implementation, data, and computational resources

The algorithms should be implemented in Python. All data and files needed for the project are provided on dropbox and organised in the following directories:

- **implementation/** contains the Python files **M1.py**, **M2.py**, **M3_train.py** and **M3_test.py**, for your implementation and validation of the 3 methods; the 4 files contain useful Python functions; **M2.py** also contains a partial implementation of the U-net architecture proposed in Appendix A, for you to finalise; **M3_train.py** also contains a partial implementation of the DnCNN architecture proposed in Appendix B, for you to finalise;

- **trainingset/** contains a training data set of approximately 7000 MR images of size 320×320 borrowed from [17] to serve as groundtruths for training both the U-net and DnCNN. From these, you will need to build the backprojected images to serve as an input to the Unet according to equation (5) in the training phase of M2, and the noisy images to serve as an input to the DnCNN according to equation (7) in the training phase of M3;

- **testingset/** contains a testing data set of 20 MR images of size 320×320 borrowed from [17] to serve as groundtruths for the validation of the 3 methods.

You will be granted access to external computational resources. Details on these resources and instructions to access and use them are provided in a dedicated guide available separately.

4 DNN online tutorials

For building background knowledge on DNNs (CNNs in particular, including their Python implementation with PyTorch) please refer to the following tutorials:

- PyTorch Tutorials

- Stanford course on CNNs

	SNR (dB)		SSIM		Reconstruction time (s)	
Method	Average	Standard dev.	Average	Standard dev.	Average	Standard dev.
M1						
M2						
M3						
M4						

Table 1: SNR, SSIM, and computation time results

5 Beyond M1, M2, M3

The design and development of M1, M2, and M3 raises a significant amount of questions with regards to possible improvements. Can the objective function for M1 be enhanced? Can the network architectures, training data sets (input/output), and/or training losses for M2 and M3 be enhanced? Would another PnP algorithm enable increased performance of M3? Can DNNs and optimisation algorithms be interfaced in a different way? These questions lay the ground for the development of new methods that would possibly outperform M1, M2, and M3...

Challenge. You should attempt to develop and implement a new method M4 to outperform M1, M2, and M3.

6 Report

Your results. You are asked to provide the results of validation of your developments over the 20 images from the testing data set. More precisely, you need to provide, for each method (M1, M2, M3, and your M4):

1. the average and standard deviation of the Signal-to-Noise Ratio (SNR (dB)) and Structural Similarity Index Measure (SSIM) of the reconstructions (use Table 1). Note that the SNR of reconstruction $\boldsymbol{x}^\star$ is defined as SNR $= 20\log_{10}\left(\|\boldsymbol{x}^o\|_2/\|\boldsymbol{x}^o - \boldsymbol{x}^\star\|_2\right)$. The more involved definition of SSIM can be found, *e.g.* on Wikipedia, and Python packages, such as scikit-image, have a built-in function for it. Both metrics are standard estimators of the reconstruction quality;

2. the average computation time and standard deviation (use Table 1);

3. the reconstructed images $\boldsymbol{x}_i^\star$ in comparison with the groundtruth images $\boldsymbol{x}_i^o$ and the backprojected images $\tilde{\boldsymbol{x}}_i = \Re\left\{\boldsymbol{\Phi}\boldsymbol{y}_i\right\}$. For convenience, you only need to show 3 out of the 20 images.

Last but not least, it is of prime importance that you discuss your results, in particular how the methods compare to one another, both in terms of visual reconstruction quality and the quantitative SNR and SSIM metrics. You are required to discuss their strengths and limitations, and how you think they could be improved.

The document structure. You need to provide a project report containing the following sections: (i) Introduction, (ii) Objectives, (iii) Methodology and implementation, (iv) Validation and results, (v) Discussion and conclusion, (vi) References, (vii) Attendance and contributions. The document should have maximum 20 pages (font Arial, font size 11, top-bottom-lef-right margins 2cm). The "Validation and results" section should include a statement confirming that the algorithms were developed, the validation was performed, and results obtained, using the external computational resources provided. The "Attendance and contributions" section should detail, for each student, the number of progress meetings attended as well as their contributions to both the work (algorithms and codes) and report. This is understood as a joint statement agreed by the whole group.

Upload. The project is set as an assignment on the VLE (Virtual Learning Environment). You need to upload one zip file named "B31XO-groupname-project-20XX.zip" containing (i) the 4 Python files M1_sol.py, M2_sol.py, M3_sol.py, and M4_sol.py, reproducing the results of your report for each method, along with any additional files containing auxiliary functions called, (ii) the codes used for the training of your networks (the M2_train.py for M2, and M3_train.py for M3), (iii) your report in pdf format named "B31XO-groupname-report-20XX.pdf".

7 B31XO resit information - project reloaded

The B31XO resit will consist in performing a variant of this project, following all the structure and guidelines proposed in this document, but: (i) in individual (as opposed to group) mode, (ii) considering an unconstrained version of problem (2) (using, or not, ADMM) for M1 and M3, (iii) attempting another method M4 than the one proposed by your group during the semester. M2 can be ignored for the resit.

A Proposed U-net architecture

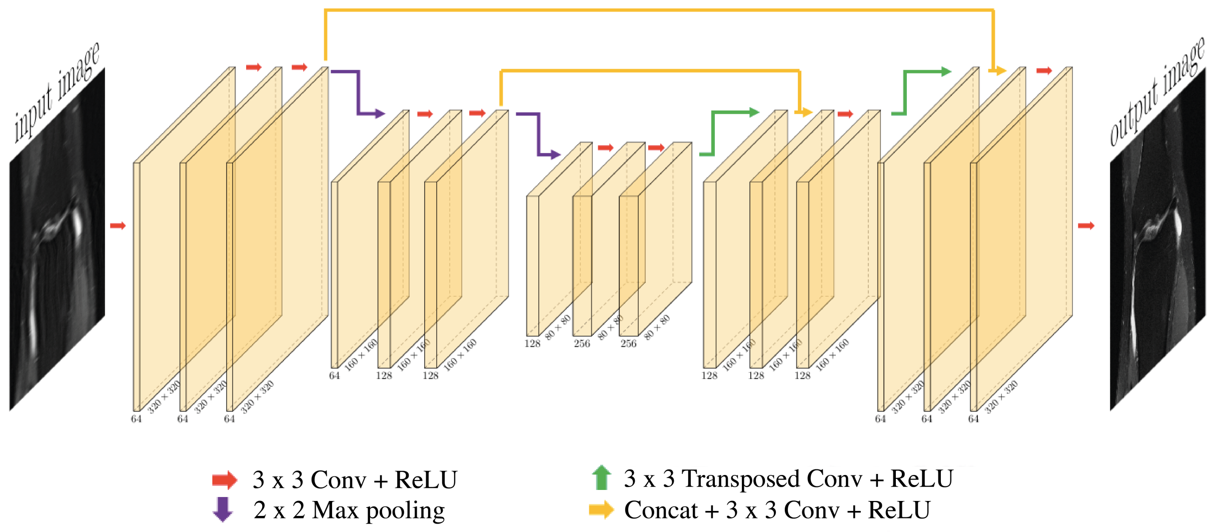

Figure 4: Illustration of the chosen U-net architecture, inspired from [2]. The network is designed as a number of layers of operations (arrows) distributed in 3 depths, each creating a feature map (yellow planes). Indicated below each map is its number of channels (between 64 and 256) as well as its dimension (between 320×320 and 80×80). **At depth 1**, the input image of dimension 320×320 is processed through 3 layers consisting in a sequence of 2D Convolution and ReLU operations (red arrows, see online tutorials for a detailed explanation of these operations), each outputting a feature map with 64 channels. **At depth 2**, after a 2×2 Max pooling (purple arrow), maps are of dimension 160×160 and processed through 2 layers of 2D Convolution and ReLU operations (red arrows), to create a feature map with 128 channels. **At depth 3**, after a second 2×2 Max pooling (purple arrow), maps are of dimension 80×80 and processed through 2 layers of 2D Convolution and ReLU operations (red arrows), to create a feature map with 256 channels. Maps are then up-sampled to dimension 160×160 by a layer of 2D Transposed Convolution with stride 2 (green arrow, see online tutorials) and ReLU operations, bringing them **back to depth 2** with 128 channels. This up-sampled output is concatenated to the 128 channels of the last feature map from depth 2 and processed through 1 layer of 2D Convolution and ReLU operations (yellow arrow), to create a feature map with 128 channels. This output is further processed by 1 layer of 2D convolution and ReLU operations (red arrow), outputting 128 channels. Maps are up-sampled a second time to dimension 320×320 by a layer of 2D Transposed Convolution with stride 2 (green arrow) and ReLU operations, bringing them **back to depth 1** with 64 channels. This up-sampled output is concatenated to the 64 channels of the last feature map from depth 1 and processed through 1 layer of 2D Convolution and ReLU operations (yellow arrow), to create a feature map with 64 channels. This output is further processed by 1 layer of 2D convolution and ReLU operations (red arrow), outputting 64 channels. Eventually, this last feature map is processed through 1 layer of 2D Convolution and ReLU operations (red arrow), to create a feature map with only 1 channel... the output image.

B Proposed DnCNN architecture

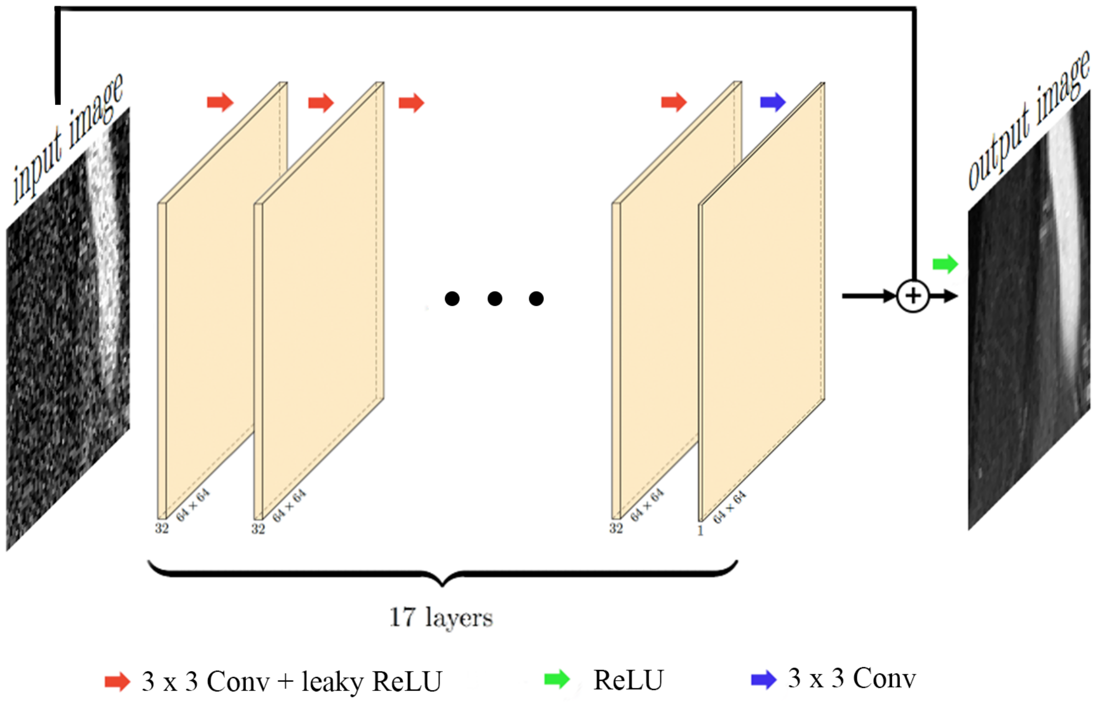

Figure 5: Illustration of the chosen DnCNN architecture, from [15]. The overall architecture consists of a total of 17 layers, each creating a feature map (yellow planes). While the first 16 layers involve 2D Convolution and Leaky ReLU operations (see red arrows; see online tutorials for a detailed explanation of these operations), the last layer involves only 2D Convolution operations (see blue arrow). Indicated below each feature map is its number of channels (between 1 and 32) as well as its dimension (64×64). The output of the 17th layer is added to the input image, through a so-called "skip connection". This means that the network is actually learning the difference between input and output. Such networks are called "residual networks". A final ReLU operation is applied after the skip connexion, that ensures the positivity of the output image. **Note on the image dimension:** due to the i.i.d. nature of the noise, and with the aim to reduce computation cost, a standard procedure for training a denoiser is to train it on small images (here of size 64×64), with a training data set built from patches of the images (here of size 320×320) in the original training data set. Once trained, the denoising network can be applied to images of any size. This follows from realising that the weights learned are those of convolution kernels, and the convolution can simply be performed on any image size. Notice on the contrary that, in case the network is trained to recover directly the image from the observed data, as in M2, the measurement operator induces nonlocal dependencies in the artefacts to be removed from the backprojected image (input to the network), possibly extending over the whole image, therefore preventing the same approach of training on patches.

References

[1] Liang and Lauterbur, "Principles of magnetic resonance imaging: a signal processing perspective", SPIE Optical Engineering Press, 2000.

[2] Jin et al., "Deep convolutional neural network for inverse problems in imaging", IEEE TIP 26 (2017) 4509.

[3] Hauptmann et al., "Multi-Scale Learned Iterative Reconstruction", accepted in IEEE TCI, arXiv:1908.00936.

[4] Yang et al., "DAGAN: Deep De-Aliasing Generative Adversarial Networks for Fast Compressed Sensing MRI Reconstruction", IEEE TMI 37 (2018) 1310.

[5] Knoll et al., "Deep Learning Methods for Parallel Magnetic Resonance Image Reconstruction", arXiv:1904.01112.

[6] Venkatakrishnan et al., "Plug-and-play priors for model based reconstruction", in Proc. IEEE Global SIP (2013) 945.

[7] Reehorst & Schniter, "Regularization by denoising: Clarifications and new interpretations", IEEE TCI 5 (2018) 52.

[8] Sun et al., "An online plug-and-play algorithm for regularized image reconstruction", IEEE TCI 5 (2019) 395.

[9] Meinhardt et al., "Learning Proximal Operators: Using Denoising Networks for Regularizing Inverse Imaging Problems", in Proc. ICCV (2017), arXiv:1704.03488.

[10] Zhang et al., "Learning Deep CNN Denoiser Prior for Image Restoration", in Proc. CVPR (2017), arXiv:1704.03264.

[11] Ryu et al., "Plug-and-Play Methods Provably Converge with Properly Trained Denoisers", in Proc. ICML (2019), arXiv:1905.05406.

[12] Gupta et al., "Cnn-based projected gradient descent for consistent CT image reconstruction", IEEE TMI 37 (2018) 1440.

[13] Chang et al., in Proc. ICCV (2017), "One network to solve them all–solving linear inverse problems using deep projection models", in Procs. ICCV (2017), arXiv:1703.09912.

[14] Yazdanpanah et al., "Deep Plug-and-Play Prior for Parallel MRI Reconstruction", in Procs. ICCV (2019), arXiv:1909.00089.

[15] Terris et al., "Image reconstruction algorithms in radio interferometry: from handcrafted to learned denoisers", arXiv:2202.12959.

[16] Ronneberger et al., "U-net: Convolutional networks for biomedical image segmentation", in Procs. MICCAI (2015), arXiv:1505.04597.

[17] Zbontar et al., "fastMRI: An open dataset and benchmarks for accelerated MRI", arXiv:1811.08839.