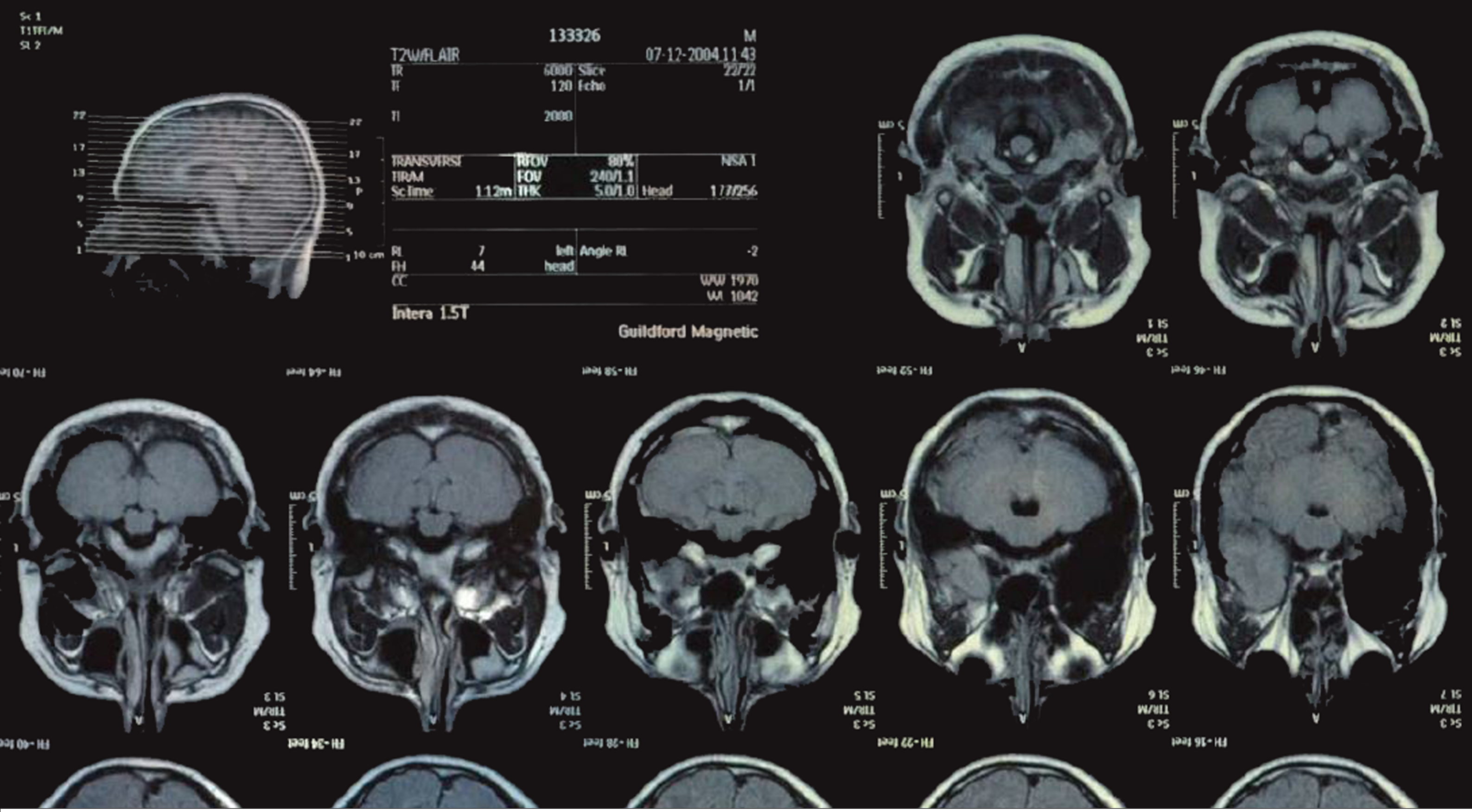
Super Resolution Techniques Applied to Low-Field MRI

Master Thesis

G. Ippolito

Delft University of Technology



Delft University of Technology

Challenge the future

Super Resolution Techniques Applied to Low-Field MRI

by

Giulia Ippolito

in partial fulfillment of the requirements for the degree of

Master of Science
in Applied Mathematics

at the Delft University of Technology,
to be defended publicly on Monday October 19, 2020 at 11:00.

Student number:	4949617	
Thesis committee:	Dr. ir. M. B. van Gijzen,	TU Delft, supervisor
	Ir. M. L. de Leeuw den Bouter,	TU Delft, supervisor
	Dr. Y. van Gennip,	TU Delft
	Prof. Dr. A. Webb,	LUMC

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

ABSTRACT

This work is part of the low-field MRI project, which aims to bring portable, affordable, low-field MRI scanners to low-income countries. Replacing the superconducting magnets of conventional scanners with standard ones can significantly reduce the costs, but it also has a negative impact on the Signal-to-Noise Ratio (SNR). In order to circumvent this problem, Super Resolution (SR) techniques may be used.

In this thesis, standard and Deep Learning (DL) SR techniques are presented. For standard SR, two types of regularization are considered: Tikhonov and Total Variation (TV). Then, the problem is solved using CGLS and ADMM algorithms respectively. From our analysis, we could conclude that TV outperforms Tikhonov regularization, yielding promising results.

Then, two 2D DL models, SRCNN and ReCNN, were implemented and trained on two commonly used SR datasets: T91 and Kirby21. Both networks managed to reconstruct the LR scans surprisingly well, with ReCNN yielding the best results when trained on both datasets. DL methods evidently outperform standard SR and can achieve a visual quality comparable to the one of a scan directly acquired in higher resolution.

A 3D extension of these networks was also considered, but, although it led to an improvement, it did not perform as well as the 2D models. We attribute this to the lack of time, which did not allow us to extensively explore this possibility.

PREFACE

A bit more than two years ago I decided to leave my city in Sicily to move here, in the Netherlands, and start my master in Applied Mathematics at TU Delft. Although, for many reasons, I consider this one of the best decisions of my whole life, it has not been an easy path. Homesickness and at times loneliness were there to be faced, but with a little help from my friends and the (digital) support of my family, I made it to the start of this thesis project

Having decided to graduate in the Numerical Analysis department, I turned to Martin van Gijzen and Merel de Leeuw den Bouter, intrigued by what I had heard of the low-field MRI project. Their contagious enthusiasm and interesting ideas convinced me to get on board with this research and now, after nine months, I can say I am very happy that I did.

Thank you, Martin and Merel, for always being extremely supportive and for helping me through every step of this journey. Even though I would have enjoyed our conversations much more in person, I always knew I could count on you, despite the weird times that we are all living. Thank you also to Thomas O'Reilly, for the hours spent imaging those nice fruits together.

Infine, un grazie dal profondo del cuore a tutti i membri della mia famiglia allargata, che, seppur lontani, non lo sono mai stati davvero. E grazie, Antonio. Spero di riuscire a dimostrarti ogni giorno perché.

G. Ippolito
Delft, October 2020

CONTENTS

1 Introduction 1
- 1.1 Motivation . . . 2
- 1.2 Research Questions . . . 3
- 1.3 Report Outline . . . 3
- 1.4 Hardware and Software . . . 3

2 MRI Background 5
- 2.1 Form of the Signal . . . 5
- 2.2 Spatial Information Encoding . . . 5
- 2.3 2D Image Reconstruction . . . 6
- 2.4 3D Case . . . 7
- 2.5 Low-field MRI . . . 7

3 Standard Super Resolution Techniques for MISR 9
- 3.1 Observation Model . . . 10
- 3.2 Least-Squares Formulation . . . 11
- 3.3 Regularization . . . 12
  - 3.3.1 Tikhonov Regularization . . . 12
  - 3.3.2 Total Variation Regularization . . . 13

4 Solution Methods 15
- 4.1 Krylov-Subspace Iterative Methods for Tikhonov Regularization . . . 15
  - 4.1.1 Generalization of the Problem . . . 15
  - 4.1.2 Standard CG . . . 16
  - 4.1.3 CGLS . . . 16
- 4.2 ADMM for Total Variation . . . 18

5 Low-Field MRI Dataset 21

6 Standard SR Results 25
- 6.1 Observation Model . . . 25
  - 6.1.1 3D Extension . . . 27
- 6.2 Phantom . . . 28
- 6.3 Low-Field Data 2D . . . 32
- 6.4 Low-Field Data 3D . . . 35

7 Deep Learning 37
- 7.1 Artificial Neural Networks . . . 38
- 7.2 Activation Functions . . . 39
- 7.3 Network Training . . . 41
  - 7.3.1 Gradient Descent . . . 42
  - 7.3.2 Backpropagation . . . 45
  - 7.3.3 Weight Initialization . . . 45
  - 7.3.4 Validation . . . 46
  - 7.3.5 Over and Underfitting . . . 47
- 7.4 Convolutional Neural Networks . . . 49
  - 7.4.1 Convolutional Layer . . . 50
  - 7.4.2 Max Pooling Layer . . . 51

8 Deep Learning Architectures for SISR 53
- 8.1 SRCNN . . . 53
  - 8.1.1 Architecture . . . 54
  - 8.1.2 Training . . . 55

8.2 ReCNN . . . 56
8.2.1 Architecture . . . 56
8.2.2 Training . . . 57
8.3 3D Extensions. . . . 58

9 Deep Learning Results **59**
9.1 Datasets and Preprocessing . . . 59
9.1.1 T91. . . . 60
9.1.2 Kirby21 . . . 60
9.2 Validation of the Models . . . 62
9.3 SRCNN-nat 2D . . . 63
9.3.1 Model Set-up . . . 63
9.3.2 Results: Test Data & Low-Field Data . . . 66
9.4 SRCNN-brain 2D . . . 72
9.4.1 Model Set-up . . . 72
9.4.2 Results: Test Data & Low-Field Data . . . 76
9.5 ReCNN-nat 2D . . . 81
9.5.1 Model Set-up . . . 81
9.5.2 Results: Test Data & Low-Field Data . . . 83
9.6 ReCNN-brain 2D . . . 88
9.6.1 Model Set-up . . . 88
9.6.2 Results: Test Data & Low-Field Data . . . 89

10 Conclusions and Recommendations **95**
10.1 Recommendations for Future Research . . . 97

Bibliography **99**

A Acquisition Parameters - MRI Scan **103**

B PSNR and SSIM - MRI Scans **105**
B.1 SRCNN-nat . . . 105
B.2 SRCNN-brain . . . 106
B.3 ReCNN-nat . . . 106
B.4 ReCNN-brain . . . 107

C Additional results on MRI Scans with Deep Learning **109**
C.1 SRCNN-nat . . . 109
C.2 SRCNN-brain . . . 110
C.3 ReCNN-nat . . . 110
C.4 ReCNN-brain . . . 111

D Acronyms and Abbreviations **113**

1

INTRODUCTION

Magnetic Resonance Imaging (MRI) is an imaging technique which produces three dimensional detailed anatomical images. It is one of the most powerful non-invasive technologies available at the current moment, but unfortunately it is also extremely expensive. Superconducting magnets are used in conventional MRI scanners which produce very high magnetic fields and guarantee resulting images of high quality. These magnets have a nominal cost of $ 1M per Tesla (T) of magnetic field, and scanners in common use require 1.5 or 3 T.[1] Furthermore, superconducting magnets need to be cooled by using liquid helium, which requires a high amount of energy.[2]

Lastly, a high level of expertise is required for operation and repair, making the scanners completely out of reach for many communities, especially in low-income countries. A significant reduction in cost is possible by replacing the superconducting magnets from the MRI system by either permanent or electromagnets, but this comes with a large reduction in available magnetic field strength.[3]

For the past four years teams at the TU Delft, Leiden University Medical Center (LUMC), Pennsylvania State University (PSU) and Mbarara University of Science and Technology (MUST) have been working on developing a low-cost, portable MRI scanner with a magnetic field in the milliTesla (mT) range (~ 50 mT). In Figure 1.1 the scanner currently at LUMC is shown. The main motivation behind this scanner lies in the will to be able to export it to African countries, such as Uganda, in order to make it possible to treat and monitor hydrocephalus in children.

In particular, the CURE children's hospital in Uganda, specialized in the treatment of infant hydrocephalus, uses Computed Tomography (CT) brain imaging, which is potentially harmful for developing children due to X-ray radiation.

Hydrocephalus (from the Greek 'hydro', meaning water, and 'cephalus', head) is a condition of which the primary characteristic is excessive accumulation of fluid in the brain. The 'water' is actually cerebrospinal fluid (CSF), a clear fluid that surrounds the brain and spinal cord.[4] The build-up of this liquid causes a widening of the brain ventricles, which in turn creates potentially harmful pressure on the tissues of the brain. Without treatment, permanent disability or even death may occur.[4]

It is estimated that hundreds of thousands of children in sub-Saharan Africa develop hydrocephalus every year.[5]

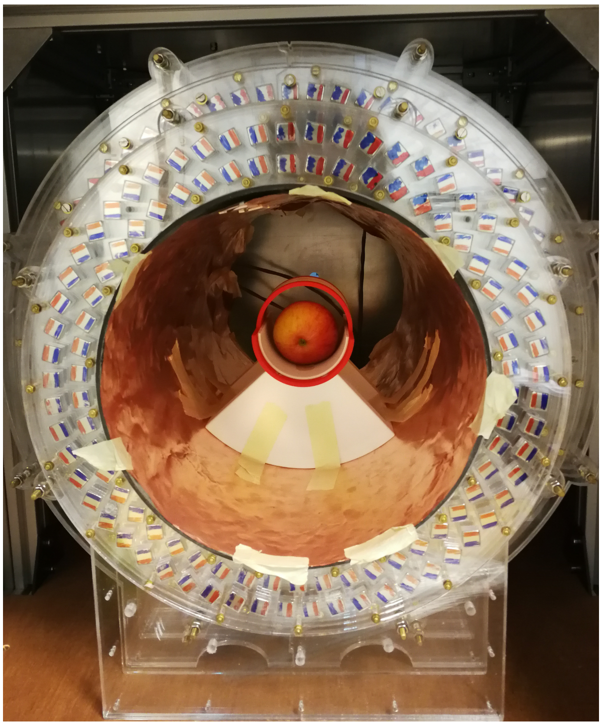


Figure 1.1: LUMC MRI scanner during a measurement of a phantom (an apple, in this case) which will be used throughout the report for our tests.

1.1. MOTIVATION

The motivation for this thesis project comes from the will to explore possibilities to improve both the resolution and the scanning times for the MRI scanner at LUMC. Indeed, the major problem with such low-field MRI systems is simply the Signal-to-Noise Ratio (SNR), since reducing the magnetic field strength from a typical clinical strength of 1.5 T to ~ 50 mT comes with a several hundred fold reduction in SNR.[3] Furthermore, it is also crucial to be able to reduce the scanning time as much as possible, in order to account for possible (likely) movements from the subject scanned. Even more so because the goal is to be able to effectively scan children up to 1 year of age. Simultaneously, the image quality must be high enough to allow to detect signs of hydrocephalus.

In [6] Super Resolution Reconstruction (SRR) techniques were already presented as a possibility to reach these goals. Unfortunately, by the time this research was carried out, it was not possible yet to generate proper images using the scanner at LUMC or PSU, hence it was not possible to test SRR techniques on actually working low-field machines. Since the start of the low-field MRI project major progress has been made, and it is now possible to do tests on real data (see Figure 1.2), which is the first innovation with respect to the previous work.

Furthermore, by the time this thesis project was started, an additional problem needed to be faced. As appears clear from Figure 1.2, we only have enough intensity in the central part of the scan. This is due to the small Field of View (FoV), but also to inhomogeneities in the magnetic field and gradient non-linearities along the bore of the scanner. This implies that, in order to have a complete observation of a subject (if the subject is bigger than the FoV), more than one scan is needed and those scans need to be combined.

During the period this research has been carried out, progress has been made on the scanner, and a new gradient coil was designed which helps overcoming this problem.[7]

In order to acquire multiple scans within a shorter time, it is possible to coarsen the resolution in the through-plane direction. The slice thickness can be substantially higher than the in-plane resolution in order to increase the SNR.[8] Moreover, even if the scanned subject fits in the FoV, multi image super resolution is still a valuable option. Indeed, in [8] a super resolution method for multi-slice MRI is described, which has a great advantage compared to full 3D acquisitions. That is that it is possible to interleave the acquisition of slices, which in turns makes it possible to record multi-slice images significantly faster than full 3D images

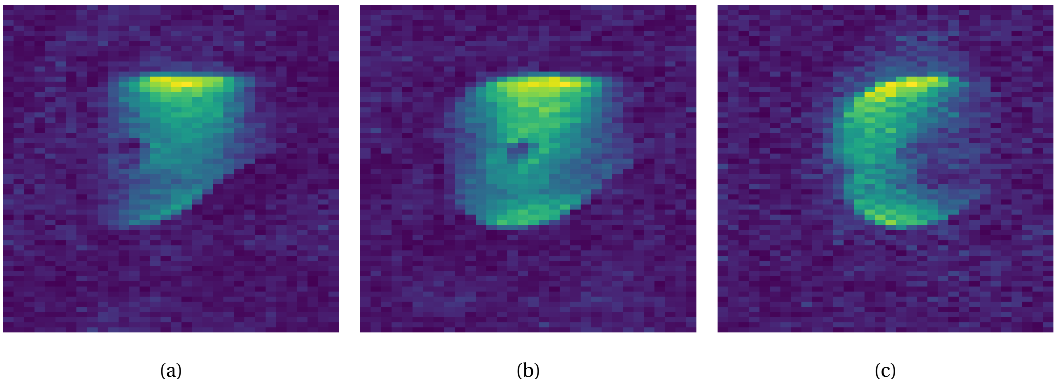

(a) (b) (c)

Figure 1.2: MRI scans of a pear, shifted of 1.7 cm one from the other. yz plane. Data acquired on 28-01-20.

with the same resolution.

Lastly, it has also been shown by several authors how, for a given acquisition time, it is possible to obtain super resolution reconstructed images which have a higher SNR than images directly acquired at the same resolution.[9]

Hence, the idea is to acquire multiple scans within a short time by coarsening the resolution in the through-plane direction, and then combine those scans into a high resolution image with isotropic voxels. This would result in a significant step forward on the path of reducing the scanning time and improve the quality of the images. That is why the focus of this research will be on super resolution techniques, which will be tested on real data produced by LUMC low-field MRI scanner.

1.2. RESEARCH QUESTIONS

As mentioned before, being able to acquire several low-resolution scans of a certain observation and to successfully combine them into a high resolution image would be potentially extremely beneficial for the development of the low-field MRI project. In this thesis project the possibilities of different super resolution techniques are explored, spacing from standard super resolution algorithms to more recently developed deep learning approaches. The final goal is to test (hopefully successfully) these techniques on 2D slices of low-field MRI data. Therefore, the research questions we aim to answer are:

1. Can Multi Image Super Resolution (MISR) yield comparable results to images directly acquired in higher resolution?
2. Can super resolution deep learning techniques yield better results than standard ones?

For the definition of MISR, see Chapter 3.

1.3. REPORT OUTLINE

After the Introduction, Chapter 2 covers the basics of how a conventional MRI scanner works, together with some information on the low-field version at LUMC. Next, in Chapter 3 and 4 respectively, the standard Super Resolution Reconstruction (SRR) technique and some solution methods are presented. Chapter 5 contains the low-field MRI dataset on which results are presented. In Chapter 7 and 8 the basics of deep learning are covered and the deep learning architectures implemented are described.

Deep Learning (DL) is a popular and steadily growing branch of machine learning, thus we only focused on the knowledge relevant to our scope. Chapters 6 and 9 contain the 2D results respectively on SRR and DL super resolution. Additionally, a few preliminary 3D results for standard super resolution are shown in Section 6.4. Finally, Chapter 10 is reserved to conclusions and further research recommendations.

1.4. HARDWARE AND SOFTWARE

All the algorithms mentioned in the report were implemented in Python, using Jupyter Notebooks. For the neural networks we made use of the Keras API [10], an open-source and user-friendly neural network library written in Python. To run the algorithms the GPU provided by Google Colab Pro was used. Google Colab offers free GPU and CPU computational resources, allowing for faster computations without the need of a local GPU. This in turn makes almost any laptop available nowadays able to train neural networks. The Pro version was preferred to the free one because it extends the maximum execution time from 12 to 24 hours and offers the possibility to upgrade the usable RAM from 12 to 26.75 GB and the GPU from Nvidia K80 to T4. This way it was possible to train in 3 hours networks which would have needed 8/9 hours of training on the free GPU and much more than that on a CPU. For further references on the Google Colab GPU specifications see [11].

2

MRI BACKGROUND

In this chapter we will briefly and concisely describe the process which allows us to obtain 2D and 3D data from a conventional MRI scanner. Finally, we will make some observations to link conventional and low-field MRI scanner. The chapter is mainly based on [12].

2.1. FORM OF THE SIGNAL

MRI scanners use the body's natural magnetic properties to produce detailed images from any part of the body.[13] The strong magnetic field inside the scanner has an effect on the body by forcing the protons in the hydrogen molecules to align with that field. This uniform alignment creates a magnetic vector oriented along the axis of the MRI scanner.[13] When a radiofrequency current is added to the magnetic field, the magnetic vector is deflected. When the radiofrequency source is switched off the magnetic vector returns to its resting state (the protons realign with the magnetic field). This process is called *free precession*, and it causes a signal to be emitted, and from this signal images are obtained.

An MRI scanner is made of three components: a main magnet, a magnetic field gradient system and a Radio Frequency (RF) system.

The main magnet generates the strong, uniform static magnetic field B_0. The gradient system is made of three orthogonal coils, which serve the purpose of generating time-varying magnetic fields which vary linearly in each spatial dimension. Finally, the RF system is made of a *transmitter* coil and a *receiver* coil. The former generates the B_1 field (also known as RF pulse), which is responsible for exciting the proton spins in the object (or person) that is to be imaged [6], while the latter transforms the energy released as the protons realign with the magnetic field into a signal.

The general form of the demodulated received signal is

$$S(t) = \int_{\text{object}} \rho(\boldsymbol{r}) e^{-i\gamma \Delta B(\boldsymbol{r}) t} d\boldsymbol{r}, \tag{2.1.1}$$

where $\rho(\boldsymbol{r})$ is the spin density of the imaged object, γ is the gyromagnetic ratio and $B(\boldsymbol{r})$ is called the receive field at location $\boldsymbol{r}$, since it can be interpreted as the magnetic flux density generated by the receiver coil carrying a unit current.

For an extensive background on the derivation of the signal model, the reader is referred to [12].

2.2. SPATIAL INFORMATION ENCODING

For 2D imaging, MRI scanners turn the 3D imaged object into a 2D image. In order to do so, a slice has to be selected and only the hydrogen protons in that slice will be excited. This selective excitation is done through two components: a gradient field and a shaped RF pulse. A RF pulse can only be frequency selective, and protons at different spatial locations will be excited in the same way if they resonate at the same frequency. This means that the RF pulse has to be made spatially selective by making the protons resonance frequency position-dependent. A way to do so is to augment the B_0-field with a linear gradient field during the excitation period.

After the RF pulse, spatial information can be encoded into the signal during the free precession period, which we mentioned earlier. Since the signal is in the form of a complex exponential, there are essentially two ways to encode this information: *frequency encoding* and *phase encoding*.[12]

Frequency encoding makes the oscillation frequency of an activated MR signal linearly dependent on its spatial origin, and it is used to determine one axis in the xy-plane of the selected slice. This is done by applying a linear gradient field to the magnetic field after the RF pulse, for instance a gradient G_x in the x-direction. As thoroughly described in [12], this leads to the demodulated signal

$$S(t) = \iint_{\text{object}} \rho(x,y)e^{-i\gamma G_x xt}dxdy. \tag{2.2.1}$$

Because the magnetic field strength only varies in one direction, Eq. 2.2.1 enables us to distinguish between different locations along the x-direction only. For the two-dimensional spatial localisation, phase encoding is needed.

The basic idea behind phase encoding is to pre-frequency encode the signal for a short time interval. After an RF pulse, a phase-encoding gradient G_y is turned on for a short interval T_{pe}, and then turned off. During the interval $0 \leq t \leq T_{\text{pe}}$, the local signal is frequency encoded. As a result of this frequency encoding, signals from different y-positions accumulate different phase angles after a time interval T_{pe}. At time T_{pe}, the signal will have an initial phase angle

$$\varphi(y) = -\gamma G_y y T_{\text{pe}}. \tag{2.2.2}$$

$\varphi(y)$ is linearly dependent on the position y, so the signal is phase-encoded.
Similarly to the case of frequency-encoding, after demodulation we then have that the received signal is given by

$$S(t) = \iint_{\text{object}} \rho(x,y)e^{-i\gamma G_y y T_{pe}}dxdy. \tag{2.2.3}$$

Combined with the previous signal model, we can now determine both the x- and y-coordinates of each signal component. So if a frequency-encoding gradient is turned on in the x-direction and a phase-encoding gradient in the y-direction, we obtain

$$S(t) = \iint_{\text{object}} \rho(x,y)e^{-i(\gamma G_x xt+\gamma G_y y T_{pe})}dxdy. \tag{2.2.4}$$

It is possible to rewrite the signal in the following form

$$S(k_x,k_y) = \iint_{\text{object}} \rho(x,y)e^{-i(k_x x+k_y y)}dxdy \tag{2.2.5}$$

by performing the transformation

$$\begin{cases} k_x = \gamma G_x t \\ k_y = \gamma G_y T_{pe}. \end{cases} \tag{2.2.6}$$

Hence, we expressed the signal in the so-called *k-space* notation. The k-space is an abstract concept and refers to a data matrix containing the raw MRI data.[14]

2.3. 2D IMAGE RECONSTRUCTION

Since we are imaging a 3D object, of which we have selected one slice, ρ will actually also depend on z, i.e $\rho(x,y,z)$. Hence, Eq. 2.2.5 needs to be replaced by

$$S(k_x,k_y) = \iint_{\text{object}}\int_{\text{slice}} \rho(x,y,z)e^{-i(k_x x+k_y y)}dxdydz. \tag{2.3.1}$$

Denoting the desired image function by $I(x,y)$, in the slice-selective imaging case from [12] we have that

$$I(x,y) = \int_{z_0-\Delta z/2}^{z_0+\Delta z/2} \rho(x,y,z)dz. \tag{2.3.2}$$

Substituting Eq. 2.3.2 in Eq. 2.3.1, the basic imaging equation is the 2D Fourier transform

$$S(k_x, k_y) = \iint_{\text{object}} I(x, y) e^{-i2\pi(k_x x + k_y y)} dx dy, \tag{2.3.3}$$

where now

$$\begin{cases} k_x = \dfrac{\gamma}{2\pi} G_x t \\ k_y = \dfrac{\gamma}{2\pi} G_y T_{pe}. \end{cases} \tag{2.3.4}$$

A few more observations lead to the final Fourier reconstruction formula

$$\begin{aligned} I(x, y) &= \Delta k_x \Delta k_y \sum_{n=-N/2}^{N/2-1} \sum_{m=-M/2}^{M/2-1} S\left(n\Delta k_x, m\Delta k_y\right) e^{i2\pi\left(n\Delta k_x x + m\Delta k_y y\right)} \\ |x| &< \frac{1}{\Delta k_x}, |y| < \frac{1}{\Delta k_y}, \end{aligned} \tag{2.3.5}$$

which is the discretized version of the inverse Fourier Transform. $I(x, y)$ can then be obtained by performing an inverse FFT, i.e. using an inverse Fast Fourier Transform algorithm.

2.4. 3D CASE

The future goal after this thesis project would be to extend super resolution techniques to 3D data. The challenge in three-dimensional imaging lies in sorting out all the signal components from different spatial locations. This could be done by using slice-selective excitations for localization in the third dimension, leaving the other two dimensions to be done with encoding methods. Alternatively, it is possible to encode information along all three dimensions into the activated signals. Because of the nature of MRI signals, one dimension would be frequency encoded, while the other two could be either phase or frequency encoded. The general imaging equation for 3D would be in any case of the following form

$$S\left(k_x, k_y, k_z\right) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} I(x, y, z) e^{-i2\pi\left(k_x x + k_y y + k_z z\right)} dx dy dz. \tag{2.4.1}$$

2.5. LOW-FIELD MRI

The description of MRI given in this chapter also applies to low-field MRI, with some variations. In particular, the image is obtained from the signal by performing an inverse FFT for both the conventional and the low-field MRI. As anticipated in the Introduction, in low-field scanners superconducting magnets are replaced by standard magnets, and this causes the magnetic field to drop from 1.5 or 3 T to approximately 50 mT, leading to a substantial reduction in SNR. Furthermore, towards the edge of the scanner our gradients deviate from a linear pattern. Especially the gradient in the z-direction is very nonlinear towards the edges, which is consistent with the images in Figure 1.2. Moreover, the background field is more inhomogeneous and there is more noise in the signal of the low-field scanner. All these factors combined result in the relationship between signal and image not being perfectly governed by an FFT. As a consequence, we can notice irregularities and artifacts in the images.

For further and detailed information about low-field MRI, we refer the reader to [3, 7, 15].

3

STANDARD SUPER RESOLUTION TECHNIQUES FOR MISR

In this chapter the idea behind Super Resolution (SR) is explained and a Multi-Image Super Resolution (MISR) reconstruction technique is presented. From now on we will refer to the approach discussed in this chapter as "standard" super resolution. This chapter is mainly based on [6, 16], as well as on some other sources which will be cited later in the text.

Super resolution is a process for obtaining one High-Resolution (HR) image from one or more degraded Low-Resolution (LR) observations of a certain scene. We distinguish between Single-Image Super Resolution (SISR) and Multi-Image Super Resolution (MISR). As the names suggest, SISR employs a single image to generate an HR version of it, while MISR involves multiple LR images. A clear advantage of MISR over SISR is that a single image is often quite limited in the amount of information that it provides, whereas MISR can draw out otherwise unavailable information from the different image observations.[17]

The MISR algorithms are usually referred to as reconstruction-based SR algorithms, because the idea behind MISR is to have multiple LR observations of the same scene, and to use the non-redundant information contained in those images to "reconstruct" the original image. This ideally results in an HR image in which high frequency components are enhanced and lost ones are retrieved. In this case we talk about Super Resolution Reconstruction (SRR). We will see in Sections 3.2 and 3.3 how this process results in an ill-posed inverse problem, for which a multiplicity of solutions exist.[18]

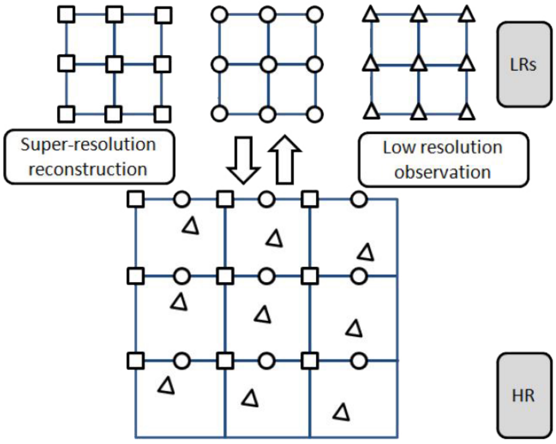


Figure 3.1: Sub-pixel shifts provide new information and lead to HR images. Source [19].

In order for each of the LR observations to contain new information, we need them to be shifted with subpixel precision one from the other. Indeed, if we would have integer shifts, every image would contain the same information and therefore it would not be possible to retrieve the desired HR image.

Depending on the kind of scene which is observed and by the mechanism of the observation (e.g. a camera, an MRI scan, etc...), those sub-pixel shifts may be due to *uncontrolled motions*, like movements of the imaged subjects, or *controlled motions*, like images acquired from orbiting satellites. If those motions, whether controlled or not, can be estimated with sub-pixel accuracy, it is possible to combine the downsampled LR images into an HR image.

3.1. OBSERVATION MODEL

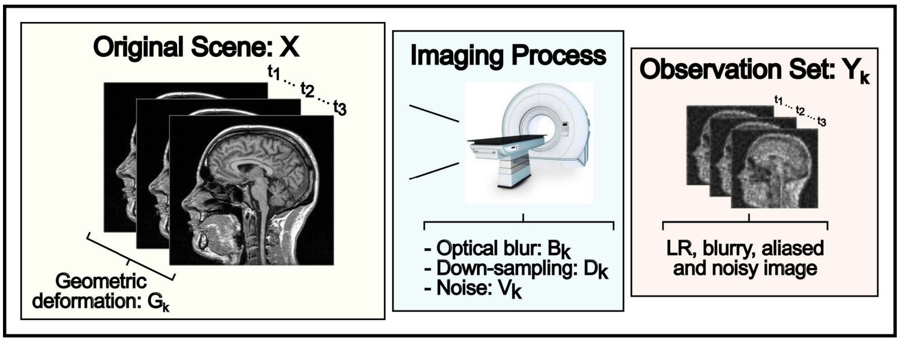


Figure 3.2: Scheme of the acquisition process. Source [9].

In order to talk about SRR techniques, the first step to be taken is the formulation of an *observation model* which links the original HR images to the LR images. We first introduce the following notation. X and Y_k, $k = 1,\dots,P$ denote the $L_1N_1 \times L_2N_2$ HR image and the $N_1 \times N_2$ LR images respectively in matrix form. L_1 and L_2 represent the downsampling factors for the horizontal and vertical directions respectively. $\boldsymbol{x}$ and $\boldsymbol{y}_k$, $k = 1,\dots,P$ denote instead the same images, but in lexicographical vector form. Hence, for instance we have

$$\boldsymbol{x} = \mathrm{vec}(X) = \begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_N \end{bmatrix},$$

with $N = L_1N_1 \times L_2N_2$ and where $\boldsymbol{x}_i$ is the ith column of X. In the following observation model it is assumed that $\boldsymbol{x}$ remains constant during the acquisition of the samples, except for motions and degradations allowed by the model. In Figure 3.2 a scheme of the acquisition process is shown. From a mathematical point of view, the model can be represented as

$$\boldsymbol{y}_k = D_kB_kG_k\boldsymbol{x} + \boldsymbol{n}_k, \qquad k = 1,\dots,P. \tag{3.1.1}$$

In Eq. 3.1.1, $\boldsymbol{n}_k$ is the noise, which is usually assumed to be independent, identically distributed Gaussian noise. D_k, B_k and G_k are three matrices that represent the transformations that lead from the original scene $\boldsymbol{x}$ to the LR images $\boldsymbol{y}_k$, with dimensions $N_1N_2 \times L_1N_1L_2N_2$, $L_1N_1L_2N_2 \times L_1N_1L_2N_2$ and $L_1N_1L_2N_2 \times L_1N_1L_2N_2$ respectively. As shown in 3.2, first a geometric deformation G_k is applied, which can be a sub-pixel translation, rotation, or both. B_k is the blurring matrix, which models the blurring occurring during the imaging process. This is done through the Point Spread Function (PSF). The PSF is a function that describes the blurring of one pixel over its surrounding pixels.[6] Finally, D_k models the downsampling operator. It is usually assumed that every LR image has the same downsampling factor and that the blurring operator is spatially invariant. So D_k and B_k do not actually depend on k. Hence, System 3.1.1 can be replaced by

$$\boldsymbol{y}_k = DBG_k\boldsymbol{x} + \boldsymbol{n}_k, \qquad k = 1,\dots,P. \tag{3.1.2}$$

Expressing the matrix product in the RHS of Eq. 3.1.2 as a single matrix yields

$$\boldsymbol{y}_k = A_k\boldsymbol{x} + \boldsymbol{n}_k, \qquad k = 1,\dots,P. \tag{3.1.3}$$

If we now vertically concatenate the vectors $\boldsymbol{y}_k$, the matrices A_k and the noise vectors $\boldsymbol{n}_k$ as follows

$$\boldsymbol{y} = \begin{bmatrix} \boldsymbol{y}_1 \\ \vdots \\ \boldsymbol{y}_P \end{bmatrix}, \quad A = \begin{bmatrix} A_1 \\ \vdots \\ A_P \end{bmatrix}, \quad \boldsymbol{n} = \begin{bmatrix} n_1 \\ \vdots \\ n_P \end{bmatrix},$$

we obtain the final system

$$\boldsymbol{y} = A\boldsymbol{x} + \boldsymbol{n}, \tag{3.1.4}$$

where the matrix A has dimensions $N_1N_2P \times L_1N_1L_2N_2$, $\boldsymbol{y}$ and $\boldsymbol{n}$ are $N_1N_2P \times 1$ vectors, and $\boldsymbol{x}$ is a $L_1N_1L_2N_2 \times 1$ vector.
The details concerning the implementation of the matrices themselves will be explored in Chapter 6.

3.2. LEAST-SQUARES FORMULATION

The aim of SRR is to solve System 3.1.4 for $\boldsymbol{x}$, therefore "inverting" the process which leads to the blurred, downsampled LR images and retrieving the HR image. We can immediately notice that the matrix A in Eq. 3.1.4 is not necessarily square, unless $P = L_1L_2$.

We recall the following definitions for a system

$$A\boldsymbol{x} = \boldsymbol{y}, \quad \text{with } A \in \mathbb{C}^{m\times n}, \quad \boldsymbol{x} \in \mathbb{R}^n, \quad \boldsymbol{y} \in \mathbb{C}^m. \tag{3.2.1}$$

Definition 3.1. A linear system is called *underdetermined* if the number of unknown variables exceeds the number of equations ($m < n$).

Definition 3.2. A system is called formally *determined* if the number of unknowns is the same as the number of equations ($m = n$).

Definition 3.3. A system is called *overdetermined* if there are more equations than unknowns ($m > n$).[5]

From these we can tell that System 3.1.4 might have zero, one, or infinitely many solutions. In particular, if A is invertible, then a determined system has a unique solution.
When A is not square, a system equivalent to 3.2.1 can be solved:

$$A^*A\boldsymbol{x} = A^*\boldsymbol{y}. \tag{3.2.2}$$

This system is Hermitian semi-positive definite (semi HPD) and it is called the system of *normal equations* associated with the *least-squares problem*

$$\min_{\boldsymbol{x}} \|\boldsymbol{y} - A\boldsymbol{x}\|_2^2. \tag{3.2.3}$$

Equation 3.2.2 is generally used when dealing with overdetermined systems. The minimizer of the problem is called the *least squares solution* because it minimizes the sum of the squares of the components of the residual error $\boldsymbol{r} = \boldsymbol{y} - A\boldsymbol{x}$. In this case, if the system is consistent (i.e. it has at least one solution), and $\operatorname{rank}(A) = n$ with $m \geq n$, then the least squares minimum norm (LSMN) solution is given by

$$\boldsymbol{x}_{\text{ls}} = (A^*A)^{-1}A^*\boldsymbol{y}. \tag{3.2.4}$$

If A^*A is invertible, then Eq. 3.2.4 is actually the solution of System 3.2.2.

Most of the times when dealing with SRR for MRI data, our system will be underdetermined. Similarly as we had before, if the system is consistent and $\operatorname{rank}(A) = m$ with $m \leq n$, the LSMN solution is

$$\boldsymbol{x}_{\text{ls}} = A^*(AA^*)^{-1}\boldsymbol{y}. \tag{3.2.5}$$

Indeed, if $\boldsymbol{y} = A\boldsymbol{x}$, we can set $\boldsymbol{x} = A^*\boldsymbol{u}$ and solve equation

$$AA^*\boldsymbol{u} = \boldsymbol{y}$$

for $\boldsymbol{u}$. Then $\boldsymbol{x}$ can be obtained by multiplying $\boldsymbol{u}$ by A^*.

A first attempt of solving such equations when AA^* or A^*A are not invertible can be done by computing the *pseudoinverse* of A. In particular, we will compute the Moore-Penrose inverse. We recall that the SVD of

A is

$$A = U\Sigma W^*, \tag{3.2.6}$$

with U, Σ and V respectively the left singular vectors, the singular values σ_i and the right singular vectors of A.
Given Eq. 3.2.6, the Moore-Penrose inverse of A is $A^+ = W\Sigma^+U^*$, and $\boldsymbol{x} = A^+\boldsymbol{y}$ is our least-squares minimum norm solution. Note that Σ^+ is the pseudoinverse of Σ, defined as

$$\Sigma^+ = \begin{bmatrix} \Sigma_m^+ & 0 \\ 0 & 0 \end{bmatrix}, \quad \Sigma_m^+ = \mathrm{diag}(1/\sigma_1, 1/\sigma_2, \dots, 1/\sigma_r), \quad \text{where} \quad r = \mathrm{rank}(A). \tag{3.2.7}$$

If we have A with $m \geq n$ and full column rank, then

$$A^+ = (A^*A)^{-1}A^*. \tag{3.2.8}$$

If instead A has $m \leq n$ and full row rank, then

$$A^+ = A^*(AA^*)^{-1}. \tag{3.2.9}$$

Hence, by using the notion of pseudoinverse we are able to compute the least-squares minimum norm solution even when AA^* and A^*A are not invertible.

However, this naïve method becomes unreliable when having large, noisy systems with very small singular values. Assuming A^+ is the left inverse of A, $\boldsymbol{x}_{\mathrm{ls}}$ is equal to

$$\boldsymbol{x}_{\mathrm{ls}} = A^+\boldsymbol{y} = W\Sigma^+U^*\boldsymbol{y} = \boldsymbol{x} + W\Sigma^+U^*\boldsymbol{n}, \tag{3.2.10}$$

where we used that $\boldsymbol{y} = A\boldsymbol{x} + \boldsymbol{n}$. The last term in the equation is called *inverted noise*, and it can be explicitly computed as

$$W\Sigma^+U^*\boldsymbol{n} = \sum_{i=1}^{r} \frac{\boldsymbol{u}_i^*\boldsymbol{n}}{\sigma_i}\boldsymbol{w}_i. \tag{3.2.11}$$

It appears clear that if the singular values are too close to 0, the inverted noise will grow and contaminate the reconstructed image.

3.3. REGULARIZATION

As we have seen, naïve solutions like the pseudoinverse method lead to images potentially heavily corrupted by noise. Furthermore, we introduce the following definition.

According to Jacques Hadamard, a problem is *well-posed* if:

1. A solution exists
2. The solution is unique
3. The solution's behaviour changes continuously with the initial conditions.

A problem is said to be *ill-posed* if one or more of these properties do not hold. System 3.1.4 is ill-posed, as inverse problems typically are. Indeed, the condition number of matrix A becomes very large, and A is said to be ill-conditioned. As a consequence, solutions become very sensitive to small perturbations in the right-hand side $\boldsymbol{y}$, and the third of the properties aforementioned does not hold. This is why we introduce the idea of *regularization*. Through the regularization term, we add additional information about $\boldsymbol{x}$, which results in more feasible solutions.

3.3.1. TIKHONOV REGULARIZATION

One of the best known regularization methods is the Tikhonov regularization. Adding the regularization term $\lambda\|F\boldsymbol{x}\|_2^2$ to Eq. 3.2.3 yields the following new minimization problem

$$\min_{\boldsymbol{x}} \frac{1}{2}\|\boldsymbol{y} - A\boldsymbol{x}\|_2^2 + \frac{1}{2}\lambda\|F\boldsymbol{x}\|_2^2, \tag{3.3.1}$$

where λ is the regularization parameter and F is the regularization matrix. We will from now on refer to the generalized form for Tikhonov regularization, in which F is typically a discrete finite difference approximation of a derivative operator. In the context of MRI scans, it can in particular be chosen to be a second-order difference matrix. In MRI images it is very likely for neighboring pixels to have the same value, and such a choice of a regularization matrix penalizes jumps between pixels.
The parameter λ gives the balance between minimizing $\|\boldsymbol{y}-A\boldsymbol{x}\|_2^2$ and minimizing $\|F\boldsymbol{x}\|_2^2$, hence the balance between a solution which fits the data well and enforcing prior information on the solution.

There exist several ways to determine the value for λ. One of the most popular choices is to employ the $L-$curve method.[20]
The *L-curve* is a log-log-plot of $\|\boldsymbol{y}-A\boldsymbol{x}\|_2$ versus $\|F\boldsymbol{x}\|_2$, with λ as the parameter. The curve basically consists of two parts: a “flat” part where the regularization errors dominates, and a “steep” part where the perturbation error dominates. The optimal regularization parameter is supposed to lie somewhere near the L-curve’s corner. The scale is chosen as such because it emphasizes the two parts and the corner of the curve.

Observe now that the objective function of minimization problem 3.3.1 is a convex function (linear combination of composition of l_2-norms), hence by setting its gradient equal to 0 we can find the global optimal solution to our problem. Therefore we have

$$(A^*A+\lambda F^*F)\boldsymbol{x} = A^*\boldsymbol{y} \tag{3.3.2}$$

which leads to

$$\boldsymbol{x} = (A^*A+\lambda F^*F)^{-1}A^*\boldsymbol{y}. \tag{3.3.3}$$

Unfortunately it is extremely computationally demanding to use the previous equation to find $\boldsymbol{x}$ by inverting $A^*A+\lambda F^*F$. In fact, if X is a $n\times m$ image, for what we have observed in Section 3.1, $A^*A+\lambda F^*F$ will be an $nm\times nm$ matrix. So if X is an HR image, then the previous matrix becomes very large.

3.3.2. TOTAL VARIATION REGULARIZATION

Another possible regularization is the Total Variation (TV) regularization, which leads to the following minimization problem

$$\min_{\boldsymbol{x}} \frac{1}{2}\|\boldsymbol{y}-A\boldsymbol{x}\|_2^2+\frac{1}{2}\lambda\|F\boldsymbol{x}\|_1. \tag{3.3.4}$$

Here F is a first-order difference matrix defined such that

$$\|F\boldsymbol{x}\|_1 = \sum_{k=1}^{n}\sum_{l=2}^{m}\left|X_{l,k}-X_{l-1,k}\right| + \sum_{l=1}^{m}\sum_{k=2}^{n}\left|X_{l,k}-X_{l,k-1}\right|. \tag{3.3.5}$$

Similarly to the Tikhonov case, this choice of F penalizes jumps between neighboring pixels, even though not as harshly. Indeed, in minimization problem 3.3.1, the l_2-norm makes the regularization term grow quadratically with the difference in value between neighboring pixels, while in Eq. 3.3.4 the penalization grows linearly, due to the l_1-norm.

4

SOLUTION METHODS

In the following chapter we will present some solution methods for the standard SRR problem. The main source for this chapter is [6].

4.1. KRYLOV-SUBSPACE ITERATIVE METHODS FOR TIKHONOV REGULARIZATION

As we have seen in the previous chapter, it is not feasible to directly compute $\boldsymbol{x}$ as in System 3.3.3 when dealing with high-resolution images. Nevertheless, it is still possible to solve System 3.3.2 by using iterative solvers.

The *Conjugate Gradient* (CG) method is a Krylov subspace method developed by Hestenes and Stiefel [21]. It is suitable for solving linear systems in the form $A\boldsymbol{u} = \boldsymbol{f}$, where A is a square HPD matrix.
In the following sections we will describe the CG and some of its variations.

4.1.1. GENERALIZATION OF THE PROBLEM

In Section 3.3.1 we have considered minimization problem 3.3.1

$$\min_{\boldsymbol{x}} \frac{1}{2}\|\boldsymbol{y} - A\boldsymbol{x}\|_2^2 + \frac{1}{2}\lambda\|F\boldsymbol{x}\|_2^2.$$

As carefully explained in [6], this is a special form of the following problem

$$\min_{\boldsymbol{x}} \frac{1}{2}\|\boldsymbol{y} - A\boldsymbol{x}\|_{C^{-1}}^2 + \frac{1}{2}\lambda\|\boldsymbol{x}\|_R^2, \tag{4.1.1}$$

where $R = F^*F$ and C is the covariance matrix of the noise. Similarly to what we have seen before, the optimality condition for 4.1.1 is.

$$(A^*C^{-1}A + \lambda R)\boldsymbol{x} = A^*C^{-1}\boldsymbol{y} \tag{4.1.2}$$

We can now observe that the minimization problem in 4.1.1 can be formulated as a constrained optimization problem

$$\begin{gathered}\min_{\boldsymbol{r},\boldsymbol{x}} \frac{1}{2}\|\boldsymbol{r}\|_C^2 + \frac{1}{2}\lambda\|\boldsymbol{x}\|_R^2 \\ \text{subject to } \boldsymbol{r} = C^{-1}(\boldsymbol{y} - A\boldsymbol{x}).\end{gathered} \tag{4.1.3}$$

Indeed,

$$\|\boldsymbol{r}\|_C^2 = \boldsymbol{r} * C\boldsymbol{r} = (\boldsymbol{y} - A\boldsymbol{x})^* C^{-1}(\boldsymbol{y} - A\boldsymbol{x}) = \|\boldsymbol{y} - A\boldsymbol{x}\|_{C^{-1}}^2.$$

If we now use the technique of lagrangian multipliers on this new problem, we find

$$\boldsymbol{r} = C^{-1}(\boldsymbol{y} - A\boldsymbol{x}), \quad \lambda R\boldsymbol{x} = A^*\boldsymbol{r}, \tag{4.1.4}$$

which if λR is invertible yields

$$\boldsymbol{x} = \frac{1}{\lambda} R^{-1} A^* \boldsymbol{r}, \quad \left(\frac{1}{\lambda} A R^{-1} A^* + C\right) \boldsymbol{r} = \boldsymbol{y}. \tag{4.1.5}$$

If we now define $\boldsymbol{z} := \frac{1}{\lambda} \boldsymbol{r}$, we can extend the result to the case $\lambda = 0$ and have

$$(A R^{-1} A^* + \lambda C) \boldsymbol{z} = \boldsymbol{y}, \quad \boldsymbol{x} = R^{-1} A^* \boldsymbol{z}. \tag{4.1.6}$$

From the equivalence between problems 4.1.1 and 4.1.3 follows the one between equations 4.1.2 and 4.1.6.

4.1.2. Standard CG

CG is an iterative solver which finds a solution to the following minimization problem

$$\left\| \boldsymbol{u} - \boldsymbol{u}^k \right\|_A = \min_{\boldsymbol{y} \in K^k(A; \boldsymbol{r}^0)} \| \boldsymbol{u} - \boldsymbol{y} \|_A. \tag{4.1.7}$$

Thus, the method builds a sequence of improving approximate solution vectors $\boldsymbol{u}_0, \boldsymbol{u}_1, \dots$ and eventually it finds $\boldsymbol{u}^k$ in the Krylov subspace $K^k(A; \boldsymbol{r}^0)$ such that $(\boldsymbol{u} - \boldsymbol{u}^k)^* A (\boldsymbol{u} - \boldsymbol{u}^k)$ is minimized (i.e. in the A-norm). At each step, the residual is computed as

$$\boldsymbol{s}_k = \boldsymbol{f} - A \boldsymbol{u}_k. \tag{4.1.8}$$

Concerning the rate of convergence of the CG method, the following theorem holds

Theorem 4.1. *The iterates $\boldsymbol{u}_k$ obtained from the CG algorithm satisfy the following inequality:*

$$\| \boldsymbol{u} - \boldsymbol{u}_k \|_A \leq 2 \left(\frac{\sqrt{\kappa_2(A)} - 1}{\sqrt{\kappa_2(A)} + 1} \right)^k \| \boldsymbol{u} - \boldsymbol{u}_0 \|_A, \tag{4.1.9}$$

where $\kappa_2(A)$ is the condition number of the matrix A in the 2-norm.

Hence, a smaller condition number implies a faster convergence. In Algorithm 1 the complete standard CG is shown.

Algorithm 1 Standard CG

```
Require: $A \in \mathbb{C}^{n \times n}$, $\boldsymbol{u}_0 \in \mathbb{C}^n$, $\boldsymbol{f} \in \mathbb{C}^n$
Ensure: Approximate solution $\boldsymbol{u}_k$ such that $\left\| \boldsymbol{f} - A\boldsymbol{u}_k \right\| \leq TOL$
procedure CG($A, \boldsymbol{f}$)                                  ▷ Solve $A\boldsymbol{u} = \boldsymbol{f}$
Initialize: $\boldsymbol{s}_0 = \boldsymbol{f} - A\boldsymbol{u}_0$; $\boldsymbol{p}_0 = \boldsymbol{s}_0$; $\gamma_0 = \boldsymbol{s}_0^* \boldsymbol{s}_0$; $k = 0$;
    while $\sqrt{\gamma_k} > TOL$ and $k < k_{\max}$ do
        $\xi_k = \boldsymbol{p}_k^* A \boldsymbol{p}_k$
        $\alpha_k = \frac{\gamma_k}{\xi_k}$
        $\boldsymbol{u}_{k+1} = \boldsymbol{u}_k + \alpha_k \boldsymbol{p}_k$
        $\boldsymbol{s}_{k+1} = \boldsymbol{s}_k - \alpha_k A \boldsymbol{p}_k$
        $\gamma_{k+1} = \boldsymbol{s}_{k+1}^* \boldsymbol{s}_{k+1}$
        $\beta_k = \frac{\gamma_{k+1}}{\gamma_k}$
        $\boldsymbol{p}_{k+1} = \boldsymbol{s}_{k+1} + \beta_k \boldsymbol{p}_k$
        $k = k + 1$
    end while
end procedure
```

4.1.3. CGLS

When dealing with a system $A\boldsymbol{u} = \boldsymbol{f}$ where A is not hermitian positive definite, or not even square, it is possible to use a variant of the CG method to solve the system, namely CGLS. This variant of CG is obtained by applying the standard CG to the system of normal equations $A^* A \boldsymbol{u} = A^* \boldsymbol{f}$. Indeed, even if A is not, $A^* A$ is semi HPD and CG can be applied. As for standard CG, CGLS minimizes the 2−norm of the residual at every iteration. In fact

$$\|\boldsymbol{e}_k\|_{A^*A} = \|\boldsymbol{u}-\boldsymbol{u}_k\|_{A^*A} = (\boldsymbol{u}-\boldsymbol{u}_k)^*\, A^*A\,(\boldsymbol{u}-\boldsymbol{u}_k) = \left(\boldsymbol{f}-A\boldsymbol{u}_k\right)^*\left(\boldsymbol{f}-A\boldsymbol{u}_k\right) = \|\boldsymbol{s}_k\|_2^2. \tag{4.1.10}$$

So the error is minimized in the A^*A-norm, while the residual $\boldsymbol{r} = \boldsymbol{y} - A\boldsymbol{x}$ in the 2−norm.
In order to enhance the stability of the algorithm, it is possible to make some modifications.[6] Indeed, if we want to include Tikhonov regularization and solve minimization problem 4.1.1, we want to apply CG to 4.1.2 instead of standard normal equations. So in the general form $A\boldsymbol{u} = \boldsymbol{f}$ we substitute A with $A^*C^{-1}A + \lambda R$, $\boldsymbol{u}$ by x and $\boldsymbol{f}$ by $A^*C^{-1}\boldsymbol{y}$. From 4.1.3 we define

$$\boldsymbol{r}_k := C^{-1}(\boldsymbol{y} - A\boldsymbol{x}_k)$$

and we can introduce a recursion for the residual

$$\begin{aligned}\boldsymbol{s}_{k+1} &= A^*C^{-1}\boldsymbol{y} - (A^*C^{-1}A + \lambda R)\boldsymbol{x}_{k+1} = A^*C^{-1}(\boldsymbol{y} - A\boldsymbol{x}_{k+1}) - \lambda R\boldsymbol{x}_{k+1} \\ &= A^*\boldsymbol{r}_{k+1} - \lambda R\boldsymbol{x}_{k+1}.\end{aligned} \tag{4.1.11}$$

The full algorithm for solving Eq. 4.1.2 is shown in Algorithm 2.

Algorithm 2 CGLS

```
Require: $A \in \mathbb{C}^{m\times n}$, $C \in \mathbb{C}^{m\times m}$, $R \in \mathbb{C}^{n\times n}$, $\boldsymbol{x}_0 \in \mathbb{C}^n$, $\boldsymbol{y} \in \mathbb{C}^m$, $\lambda \in \mathbb{R}_{\geq 0}$
Ensure: Approximate solution $\boldsymbol{x}_k$ such that $\|A^*\boldsymbol{r}_k - \lambda R\boldsymbol{x}_k\| \leq TOL$
procedure CGLS($A, R, \boldsymbol{y}, \lambda$)
Initialize: $\boldsymbol{r}_0 = C^{-1}(\boldsymbol{y} - A\boldsymbol{x}_0)$; $\boldsymbol{s}_0 = A^*\boldsymbol{r}_0 - \lambda R\boldsymbol{x}_0$; $\boldsymbol{p}_0 = \boldsymbol{s}_0$; $\boldsymbol{q}_0 = A\boldsymbol{p}_0$ ;$\gamma_0 = \boldsymbol{s}_0^*\boldsymbol{s}_0$; $k = 0$;
    while $\sqrt{\gamma_k} > TOL$ and $k < k_{\max}$ do
        $\xi_k = \boldsymbol{q}_k^* C^{-1}\boldsymbol{q}_k + \lambda\boldsymbol{p}_k^* R\boldsymbol{p}_k$
        $\alpha_k = \frac{\gamma_k}{\xi_k}$
        $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k\boldsymbol{p}_k$
        $R\boldsymbol{x}_{k+1} = R\boldsymbol{x}_k + \alpha_k R\boldsymbol{p}_k$
        $\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k C^{-1}\boldsymbol{q}_k$
        $\boldsymbol{s}_{k+1} = A^*\boldsymbol{r}_{k+1} - \lambda R\boldsymbol{x}_{k+1}$
        $\gamma_{k+1} = \boldsymbol{s}_{k+1}^*\boldsymbol{s}_{k+1}$
        $\beta_k = \frac{\gamma_{k+1}}{\gamma_k}$
        $\boldsymbol{p}_{k+1} = \boldsymbol{s}_{k+1} + \beta_k\boldsymbol{p}_k$
        $\boldsymbol{q}_{k+1} = A\boldsymbol{p}_{k+1}$
        $k = k + 1$
    end while
end procedure
```

4.2. ADMM FOR TOTAL VARIATION

In the previous sections we presented methods to solve the system in 4.1.2, thus finding the optimal solution for minimization problem 4.1.1 with Tikhonov regularization. Nevertheless, in Section 3.3.2 we talked about another kind of regularization, namely Total Variation.
When dealing with this kind of regularization, it's not possible to directly apply CG or its variants due to the presence of the l_1-penalty. ADMM (Alternating Directions Method of Multipliers) is a possible method to solve minimization problem 3.3.4, which we recall being

$$\min_{\boldsymbol{x}} \frac{1}{2}\|\boldsymbol{y}-A\boldsymbol{x}\|_2^2+\frac{1}{2}\lambda\|F\boldsymbol{x}\|_1.$$

In [22] ADMM is explained. First, the objective function is split into two functions

$$h(\boldsymbol{x}) := \frac{1}{2}\|\boldsymbol{y}-A\boldsymbol{x}\|_2^2, \quad g(F\boldsymbol{x}) := \frac{\lambda}{2}\|F\boldsymbol{x}\|_1,$$

so that the minimization problem can be written as

$$\begin{gathered}\min_{\boldsymbol{x},\boldsymbol{v}} h(\boldsymbol{x})+g(\boldsymbol{v}) \\ \text{subject to } F\boldsymbol{x}=\boldsymbol{v}.\end{gathered} \tag{4.2.1}$$

The idea behind this method is to approximately perform the optimization of $(\boldsymbol{x},\boldsymbol{v})$ in two steps using alternating minimization of $\boldsymbol{x}$ and $\boldsymbol{v}$. This algorithm has been proved to converge to the same solution as 3.3.4, making it much easier to obtain. Indeed, the functions $h(\boldsymbol{x})$ and $g(\boldsymbol{v})$ are straightforward to minimize individually.

The problem in 4.2.1 can be rewritten as

$$\min_{\boldsymbol{x},\boldsymbol{v}} h(\boldsymbol{x})+g(\boldsymbol{v})+\frac{a}{2}\|F\boldsymbol{x}-\boldsymbol{v}\|^2, \tag{4.2.2}$$

where a penalty term is added to enforce the constraint in Problem 4.2.1, and a is a constant which regulates the influence of the penalty term on the minimization problem. This way, we managed to transform the optimization problem in 4.2.1 from constrained to unconstrained. Note that (if the couple $(\hat{\boldsymbol{x}},\hat{\boldsymbol{v}})$ is the solution to Problem 4.2.2) by making a very large, we should be able to enforce $F\hat{\boldsymbol{x}}\approx\hat{\boldsymbol{v}}$. However, independently from the magnitude of a, it will likely never be the case that $F\hat{\boldsymbol{x}}=\hat{\boldsymbol{v}}$, and the solution to the problem in 3.3.4 will never be exactly the same as the one of Problem 4.2.2.

For this very reason, it is necessary to add a new term in the problem so that the constraint $F\boldsymbol{x}=\boldsymbol{v}$ can be achieved:

$$\min_{\boldsymbol{x},\boldsymbol{v}} h(\boldsymbol{x})+g(\boldsymbol{v})+\frac{a}{2}\|F\boldsymbol{x}-\boldsymbol{v}+\boldsymbol{u}\|^2. \tag{4.2.3}$$

The expression minimized in Problem 4.2.3 is known as the *augmented Lagrangian* for our constrained problem and the role of $\boldsymbol{u}$ is equivalent to the one of a Lagrangian multiplier. As a consequence, the correct value for $\boldsymbol{u}$ can be determined through Algorithm 3. For more details on this approach see [22].

Algorithm 3 Split-Variable Augmented Lagrangian Algorithm

```
procedure AUGMENTED LAGRANGIAN
Initialize: $\boldsymbol{u}=0$; $\hat{\boldsymbol{v}}=0$; $k=0$.
    while $k < k_{\max}$ do
        $(\hat{\boldsymbol{x}},\hat{\boldsymbol{v}}) = \arg\min_{\boldsymbol{x},\boldsymbol{v}} \{h(\boldsymbol{x})+g(\boldsymbol{v})+\frac{a}{2}\|F\boldsymbol{x}-\boldsymbol{v}+\boldsymbol{u}\|_2^2\}$
        $\boldsymbol{u}=\boldsymbol{u}+(F\hat{\boldsymbol{x}}-\hat{\boldsymbol{v}})$
    end while
end procedure
```

As mentioned earlier, the intuition at this point is to carry out the optimization of $(\boldsymbol{x},\boldsymbol{v})$ in two steps, as shown in Algorithm 4.

Let us now examine how to solve the two separate optimization problems.
Since $g(\boldsymbol{v})=\frac{\lambda}{2}\|\boldsymbol{v}\|_1$, the second equation in Algorithm 4 can be written as

$$(\hat{\boldsymbol{x}},\hat{\boldsymbol{v}}) = \arg\min_{\boldsymbol{v}} \frac{\lambda}{2}\|\boldsymbol{v}\|_1+\frac{a}{2}\|F\hat{\boldsymbol{x}}-\boldsymbol{v}+\boldsymbol{u}\|_2^2. \tag{4.2.4}$$

Algorithm 4 Split-Variable ADMM

```
procedure ADMM
Initialize: $\boldsymbol{u} = 0$; $\hat{\boldsymbol{v}} = 0$; $k = 0$.
    while $k < k_{\max}$ do
        $\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} \left\{ h(\boldsymbol{x}) + \frac{a}{2} \|F\boldsymbol{x} - \hat{\boldsymbol{v}} + \boldsymbol{u}\|_2^2 \right\}$
        $\hat{\boldsymbol{v}} = \arg\min_{\boldsymbol{v}} \left\{ g(\boldsymbol{v}) + \frac{a}{2} \|F\hat{\boldsymbol{x}} - \boldsymbol{v} + \boldsymbol{u}\|_2^2 \right\}$
        $\boldsymbol{u} = \boldsymbol{u} + (F\hat{\boldsymbol{x}} - \hat{\boldsymbol{v}})$
    end while
end procedure
```

From [22] we have that

$$\hat{\boldsymbol{v}} = S_{\frac{\lambda}{a}}(F\hat{\boldsymbol{x}} + \boldsymbol{u}), \tag{4.2.5}$$

where

$$S_\tau(z) = \operatorname{sign}(z) \max\{|z| - \tau, 0\} \tag{4.2.6}$$

is the so-called shrinkage function. Concerning the first equation, by writing $h(\boldsymbol{x})$ explicitly we have

$$\min_{\boldsymbol{x}} \frac{1}{2} \|\boldsymbol{y} - A\boldsymbol{x}\|_2^2 + \frac{a}{2} \|F\boldsymbol{x} - \hat{\boldsymbol{v}} + \boldsymbol{u}\|_2^2. \tag{4.2.7}$$

This optimization problem is convex, hence imposing the derivative with respect to $\boldsymbol{x}$ equal to 0 guarantees the optimal solution:

$$\left(A^* A + aF^* F\right) \boldsymbol{x} = A^* \boldsymbol{y} + aF^* (\boldsymbol{v} - \boldsymbol{u}). \tag{4.2.8}$$

Finally, Eq. 4.2.8 can be solved using a conjugate gradient method. In Algorithm 5 the CGLS described in Section 4.1.3 is adapted to this problem.

Algorithm 5 CGLS for ADMM

```
Require: $A \in \mathbb{C}^{m\times n}$, $C \in \mathbb{C}^{m\times m}$, $R \in \mathbb{C}^{n\times n}$, $F \in \mathbb{C}^{p\times n}$, $\boldsymbol{x}_0 \in \mathbb{C}^n$, $\boldsymbol{y} \in \mathbb{C}^m$, $\boldsymbol{u} \in \mathbb{C}^p$, $\boldsymbol{v} \in \mathbb{C}^p$, $a \in \mathbb{R}_{\geq 0}$
Ensure: Approximate solution $\boldsymbol{x}_k$ such that $\|A^* \boldsymbol{r}_k - aR\boldsymbol{x}_k + aF^*(\boldsymbol{v} - \boldsymbol{u})\| \leq TOL$
procedure CGLS$(A, F, \boldsymbol{y}, \boldsymbol{u}, \boldsymbol{v}, a)$
Initialize: $\boldsymbol{r}_0 = C^{-1}(\boldsymbol{y} - A\boldsymbol{x}_0)$; $\boldsymbol{s}_0 = A^* \boldsymbol{r}_0 - aR\boldsymbol{x}_0 + aF^*(\boldsymbol{v} - \boldsymbol{u})$; $\boldsymbol{p}_0 = \boldsymbol{s}_0$; $\boldsymbol{q}_0 = A\boldsymbol{p}_0$ ;$\gamma_0 = \boldsymbol{s}_0^* \boldsymbol{s}_0$; $k = 0$;
    while $\sqrt{\gamma_k} > TOL$ and $k < k_{\max}$ do
        $\xi_k = \boldsymbol{q}_k^* C^{-1} \boldsymbol{q}_k + a\boldsymbol{p}_k^* R\boldsymbol{p}_k$
        $\alpha_k = \frac{\gamma_k}{\xi_k}$
        $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k$
        $R\boldsymbol{x}_{k+1} = R\boldsymbol{x}_k + \alpha_k R\boldsymbol{p}_k$
        $\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k C^{-1} \boldsymbol{q}_k$
        $\boldsymbol{s}_{k+1} = A^* \boldsymbol{r}_{k+1} - aR\boldsymbol{x}_{k+1} + aF^*(\boldsymbol{v} - \boldsymbol{u})$
        $\gamma_{k+1} = \boldsymbol{s}_{k+1}^* \boldsymbol{s}_{k+1}$
        $\beta_k = \frac{\gamma_{k+1}}{\gamma_k}$
        $\boldsymbol{p}_{k+1} = \boldsymbol{s}_{k+1} + \beta_k \boldsymbol{p}_k$
        $\boldsymbol{q}_{k+1} = A\boldsymbol{p}_{k+1}$
        $k = k + 1$
    end while
end procedure
```

5

LOW-FIELD MRI DATASET

On the 28th of January 2020 some measurements were performed at LUMC using a pear (and some other fruit and vegetables) as phantom. In Figure 1.2 of Chapter 1 a few samples were already shown, where we can see how the pear is shifted in the z-direction of 1.7 cm.

The data acquired have a resolution of $64 \times 64 \times 32$ in the x, y and z-directions respectively. We have isotropic voxels in the xy plane (2×2 mm), and anisotropic voxels in the xz and yz planes (2×4 mm). The coarser resolution in the z-direction helped reduce the scanning time, which is 2 times less than if we had already acquired the data with voxels of $2 \times 2 \times 2$ mm^3.

Unfortunately, the shift between the different samples was too big to be able to apply SRR (which we remind, requires sub-pixel shifts), which led to new measurements carried out on the 27th of February 2020.

Again, a pear and an apple were imaged with isotropic voxels in the xy plane (2×2 mm), and anisotropic voxels in the xz and yz planes (2×8 mm). The resolution is $64 \times 64 \times 16$ pixels. Shifts of 2 mm in the z-direction were made, thus we obtained 4 measurements shifted of $1/4$ of a pixel one from the other (sub-pixel shifts). Furthermore, a $2 \times 2 \times 2$ mm^3 sample was acquired, in order to compare the SRR result with it. In this literature review we will only consider the scans of the apple, leaving the other data for future experiments. Table 5.1 groups some data concerning the various measurements. In Appendix A further details about the acquisition parameters for one of the LR apple samples are shown.

Concerning the shifts between the pear and apple samples, it is not in truth correct to assert that shifts of 2 mm were made. Indeed, when moving the imaged objects we could rely on a ruler to make the shifts as precise as possible, but it is still clearly hard to manage to perform movements on such a small scale. Therefore, it was deemed necessary to correctly register the shifts between the samples.

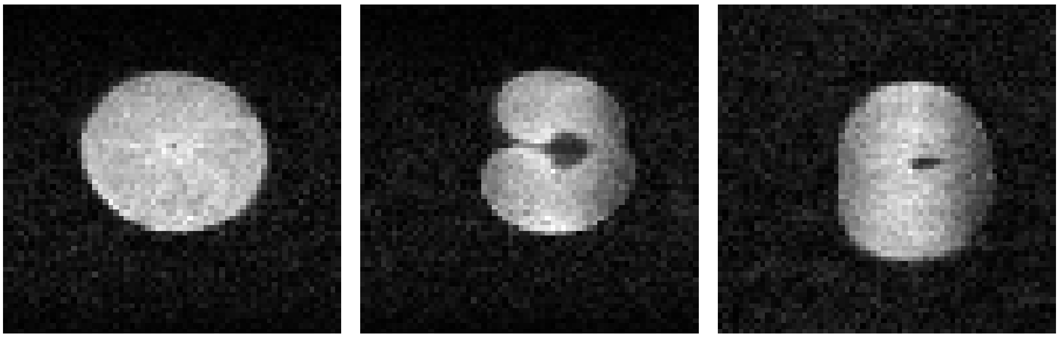
Image registration is the problem of inferring the coordinate transformation between two (or more) noisy and shifted (or distorted) signals or images.[23] The algorithm which will be used in this report is presented in [24], and it is a fast and efficient version of the standard algorithm for registration between translated images: compute an upsampled cross-correlation between the image to register and a reference image by means of a Fast Fourier Transform (FTT), and locate its peak. An implementation of this algorithm is available at the `scikit-image` Python library.

This efficient registration method allowed us to obtain new, more precise values for the exact shifts between samples and thus apply the SRR algorithms to the best of our possibilities. Nevertheless, the imaged objects might have also rotated slightly while shifting them, hence a registration technique which only takes shifts into account cannot be considered fully reliable. In the future, the use of a different, more complete, registration method might certainly lead to an improvement in these estimations.

Lastly, it is important to stress that the HR scans mentioned earlier are not exactly the ground-truths to the LR scans, hence it should be taken into account when comparing the reconstructions with the HR images. In the next pages slices of the apple scans from the 27th of February are shown, and those are the scans which will be used in Chapters 6 and 9 to test our SR algorithms.

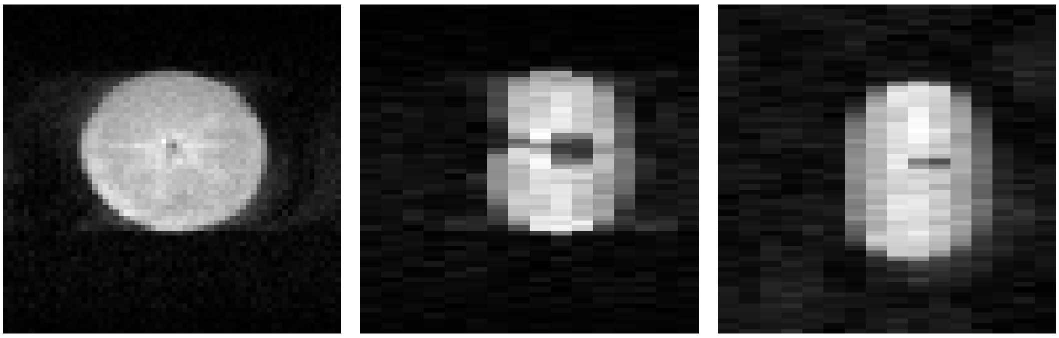
Table 5.1: MRI measurements performed at LUMC on 28-01 and 27-02-2020.

Date	Object	Voxel size	Resolution	FOV read	Shifts	N. of scans
28-01-20	Pear	$2\times2\times4$ mm^3	$64\times64\times32$	128	0,17,34,51,68,85 mm	6
28-01-20	Pointy pepper	$2\times2\times4$ mm^3	$64\times64\times32$	128	0,17,34,51,68,85 mm	6
27-02-20	Pear	$2\times2\times2$ mm^3	$64\times64\times64$	128	0 mm	1
27-02-20	Pear	$2\times2\times8$ mm^3	$64\times64\times16$	128	0, 1.92, 3.12, 4.08 mm	4
27-02-20	Apple	$2\times2\times2$ mm^3	$64\times64\times64$	128	0 mm	1
27-02-20	Apple	$2\times2\times8$ mm^3	$64\times64\times16$	128	0, 1.52, 3.68, 5.92 mm	4



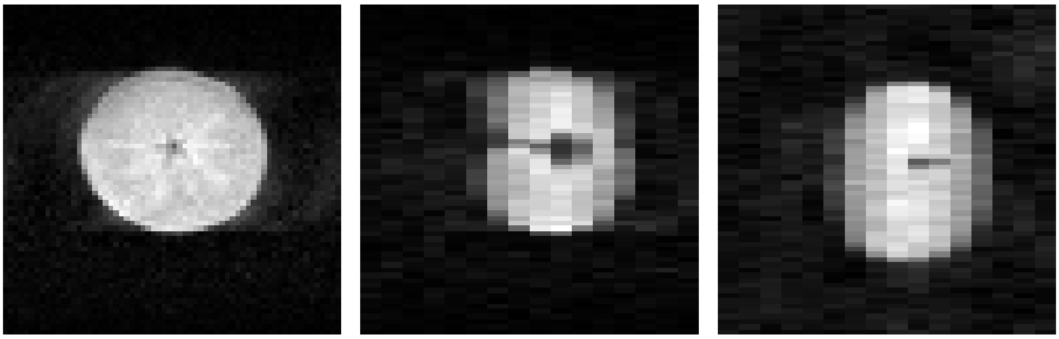
(a) xy plane (b) yz plane (c) xz plane

Figure 5.1: HR apple scan. Position 0.



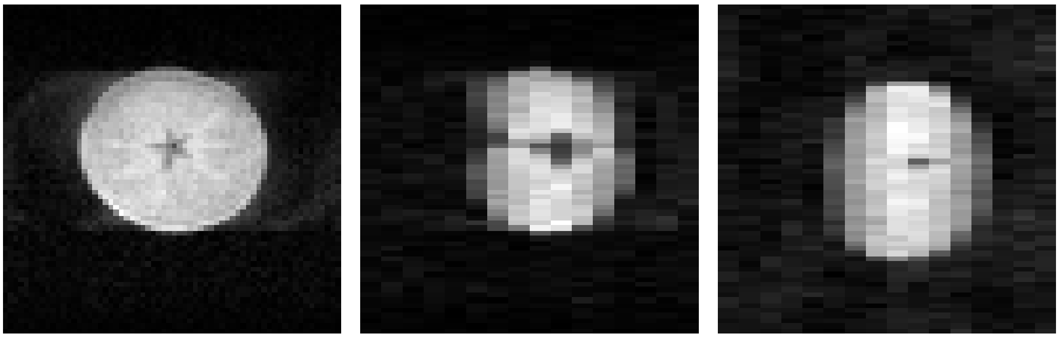
(a) xy plane (b) yz plane (c) xz plane

Figure 5.2: LR apple scan. Position 0 (0 mm shift).



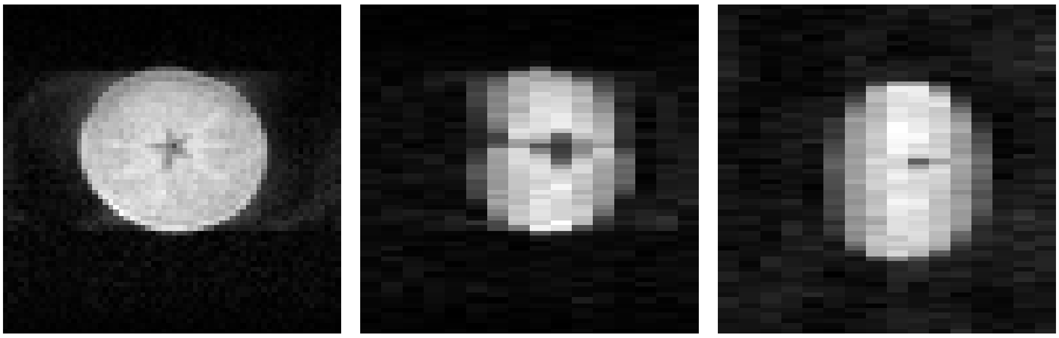
(a) xy plane (b) yz plane (c) xz plane

Figure 5.3: LR apple scan. Position 1 (2 mm shift).



(a) xy plane (b) yz plane (c) xz plane

Figure 5.4: LR apple scan. Position 2 (4 mm shift).



(a) xy plane (b) yz plane (c) xz plane

Figure 5.5: LR apple scan. Position 3 (6 mm shift).

6

STANDARD SR RESULTS

In this chapter we will present the results of standard Super Resolution on both the Shepp-Logan phantom and low-field MRI data.

6.1. OBSERVATION MODEL

Firstly, we implemented the observation model. The three matrices D, B and G_k in System 3.1.2 where built. Note that from this moment on when referring to Eq. 3.1.2 we will make the assumption of $\boldsymbol{n} = \mathbf{0}$. Other important assumptions will be made: we will only consider gray-scale images, since we will be dealing with MRI data. Furthermore, we will consider real values for the pixels. This assumption was made in order to uniform standard and deep learning techniques for future comparisons. Indeed, the networks described in Chapter 8 will be trained on real-valued images.
Blurring and downsampling will be in the z-direction only. Lastly, a shift to the right in the z-direction is chosen as geometric deformation.

Since we are dealing with gray-scale images (normalized between 0 and 1), every pixel only carries intensity information and its value is a single number. When shifting, it is clear that part of the image will go beyond the borders, and on the left we will have pixels missing. Therefore the images were padded with zeros. That is coherent with the type of data that we are considering, since in MRI scans the background is black (i.e. pixel value 0) and the central part is the one generally containing the information.

Having a shift of a in the right direction, we can decompose it into integer and decimal part:

$$a = \lfloor a \rfloor + a_{\text{dec}}. \tag{6.1.1}$$

When the image is shifted of a fraction of the pixel length, there will be an overlap between the pixel themselves. Given a pixel (i, j) of image X, its new value will be a weighted average of the value of the neighboring pixels on their left and right. Calling $\tilde{X}$ the shifted image, we have

$$\tilde{X}_{i,j} = (1 - a_{\text{dec}})\, X_{i,j-\lfloor a \rfloor} + a_{\text{dec}}\, X_{i,j-\lfloor a \rfloor -1}. \tag{6.1.2}$$

The downsampling was implemented by doing an averaging of neighboring pixels. Having an HR image of size $L_1N_1 \times L_2N_2$, where L_1 and L_2 are the downsampling factors in the y and x direction respectively, then the downsampled image is obtained by averaging over L_1L_2 pixels. In our case, since we only had a shift in the z-direction, downsampling was implemented in 1D only, so $L_2 = 1$, $L_1 = L$.

Finally, the blurring was modeled through the PSF. As already mentioned in Section 3.1, the PSF describes the blurring of one pixel over its surrounding pixels. The function was modeled as a zero-mean Gaussian with standard deviation σ, as suggested by most articles including [9]. Similarly to the downsampling operator, also the blurring was modeled in one direction only. This led to a PSF array $\boldsymbol{p}$, which will be our Gaussian kernel. p_i is given by

$$p_i = \frac{1}{\sqrt{2\pi\sigma^2}}\, e^{-\frac{(i-c)^2}{2\sigma^2}}. \tag{6.1.3}$$

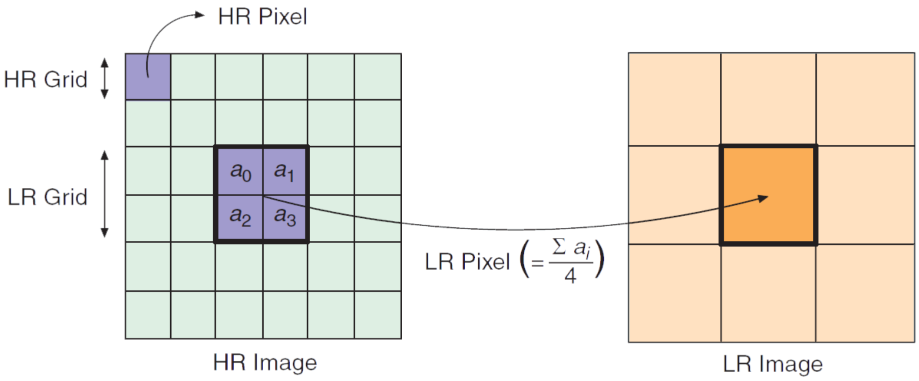


Figure 6.1: Downsampling by averaging when $L_1 = L_2 = 2$. Source [16].

Here, c is the central index of the kernel.
In order to obtain the blurred image $\overline{X}$, $\boldsymbol{p}$ is convoluted with the image X as described in [25]. The convolution is performed by rotating $\boldsymbol{p}$ by 180°, then placing it on top of the image horizontally so that the central element in P overlaps with element $X_{i,j}$. Finally, the corresponding components are multiplied and the results are summed, resulting in $\overline{X}_{i,j}$.

An important thing to notice is that when placing $\boldsymbol{p}$ over X, some of the values of the PSF array could not match with any value in X and go beyond the borders of the image. That is why it is necessary to include boundary conditions in our discussion. We chose the so-called "Zero Boundary Conditions", hence the image was padded with zeros. Putting the image X into lexicographic form $\boldsymbol{x}$ as in System 3.1.2, the blurring matrix B was built so that $B\boldsymbol{x}$ would result in the blurred image. Below an example of how to obtain the matrix B when X is a 2×3 matrix and $\boldsymbol{p}$ has length 5.

$$\boldsymbol{p} = \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \end{bmatrix}, \qquad X = \begin{bmatrix} x_1 & x_3 & x_5 \\ x_2 & x_4 & x_6 \end{bmatrix}, \qquad \boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix}$$

$$\begin{bmatrix} \overline{x}_1 \\ \overline{x}_2 \\ \overline{x}_3 \\ \overline{x}_4 \\ \overline{x}_5 \\ \overline{x}_6 \end{bmatrix} = \underbrace{\begin{bmatrix} p_3 & & p_2 & & p_1 & \\ & p_3 & & p_2 & & p_1 \\ p_4 & & p_3 & & p_2 & \\ & p_4 & & p_3 & & p_2 \\ p_5 & & p_4 & & p_3 & \\ & p_5 & & p_4 & & p_3 \end{bmatrix}}_{B} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix}. \tag{6.1.4}$$

The value for σ was set to 1.3 for both "testing" data and real low-field data. The reason behind this choice lies in the attempt to model the same degradation that the low-field LR scans underwent. In order to do so, this procedure was followed. The yz plane of the LR apple scans showed in Chapter 5 were considered. By setting $D = I, B = I$ and the matrices G_k with the appropriate shifts, the system in 3.1.2 was solved for $\boldsymbol{x}$. This way we obtained a new LR image which combines information from all the LR scans. Next, the yz plane of the HR scan showed in Figure 5.1 blurred by applying Gaussian blur with with different values for σ, ranging from 0.1 to 3.0. Finally, the blurred image was downsampled by averaging as described earlier. Afterwards, both the PSNR and the SSIM (see Section 8.1.2) were calculated between the original LR image and the blurred, downsampled HR image. The results allowed us to identify the range of optimal values for σ as the one going from 1.2 to 1.8. Indeed, for $\sigma = 1.2$ we obtained the highest PSNR and a high value for SSIM, while for 1.8 the SSIM was the highest, while the PSNR was still at a good level. Out of this range, both the metrics scored lower values.

The same procedure was repeated for the xz plane and we found (1.4,2.7) as a range. From a visual assessment it was clearly possible to see how any value above 1.4 was in truth too high and led to extremely blurred images. In this case, the PSNR showed to be a more reliable metric. Eventually, a value of 1.3 was deemed to lead to an image with resembles the LR scan the most (see Figure 6.2).

Clearly, the one described here is not a rigorous blur estimation procedure. Moreover, the procedure is highly influenced by a potentially incorrect registration. Indeed, the quality of the solution of the system might suffer from an improper registration of the LR images, but also an incorrect registration between the combined LR image and the downsampled, blurred HR image can drastically impact the choice of the value for σ. Nevertheless, we still deem it a useful method for restricting the suitable values for the standard deviation to a smaller, specific range.

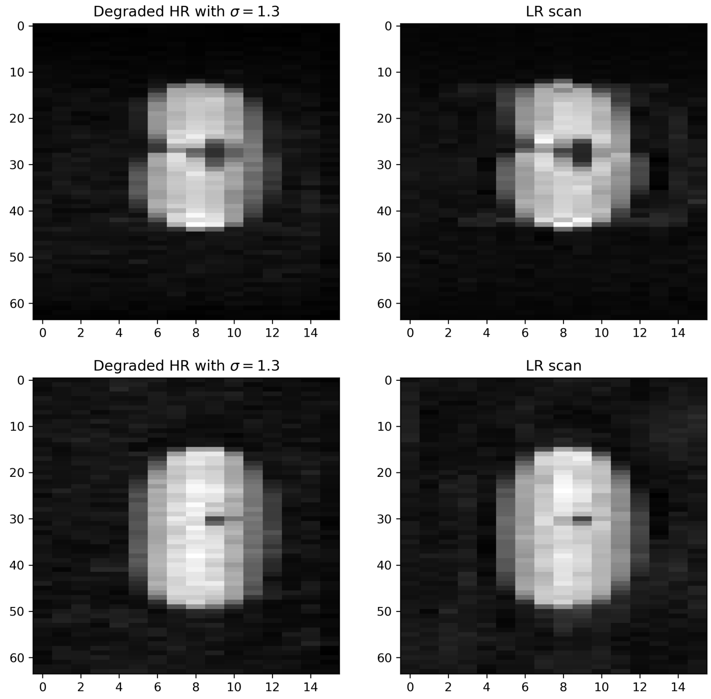


Figure 6.2: Comparison between degraded HR apple scan and LR apple scan. (top) yz plane, (bottom) xz plane.

6.1.1. 3D EXTENSION

In the previous section and in general throughout the whole text, we have been referring to 2D data, namely images. The observation model in Section 3.1 and 6.1 can nevertheless be extended to 3D data as well. Indeed, it is sufficient to replace matrices D, B and G_k by suitable block diagonal matrices. If we consider the matrices G_k (G_k^{2D}) from Section 3.1, each of them have size $L_1N_1L_2N_2 \times L_1N_1L_2N_2$, where L_1 and L_2 are the downsampling factors and N_1 and N_2 the number of rows and columns of the LR images Y respectively.

If instead of having an image X, we have 3D data $\tilde{X}$ of size $L_1N_1 \times L_2N_2 \times L_3N_3$, the matrices G_k (G_k^{3D}) will have size $L_1N_1L_2N_2L_3N_3 \times L_1N_1L_2N_2L_3N_3$ and they will be block diagonal matrices of the form in Eq. 6.1.5.

$$G_k^{3D} = \begin{bmatrix} G_k^{2D} & 0 & \cdots & 0 \\ 0 & G_k^{2D} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & G_k^{2D} \end{bmatrix}. \tag{6.1.5}$$

An analogous reasoning can be made for the other matrices. This simple extension will allow us in Section 6.4 to apply SRR techniques directly on the 3D low-field data and obtain some preliminary results.

6.2. PHANTOM

For testing purposes, a 128 × 128 Shepp-Logan phantom was used as HR image (see Figure 6.3). Four LR images were created, by shifting, blurring and downsampling the HR version. The first was created by blurring and downsampling, but no shifting. The other three images had shifts in the z-direction of $0.25, 0.5$ and 0.75 pixels with respect to the first. Note that in order to create a LR image which is shifted of a from the original image, the HR image needs to be shifted of $\bar{a} = aL$, where L is the downsampling factor. This way, the LR image resulting from the downsampling will be shifted of exactly a.

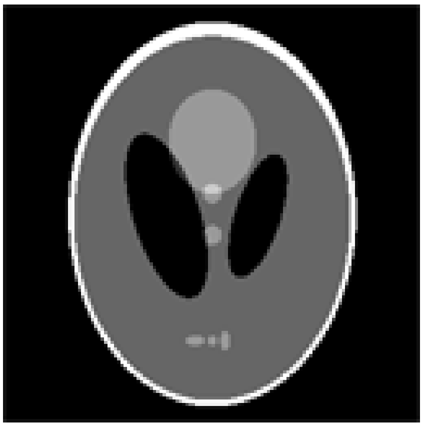
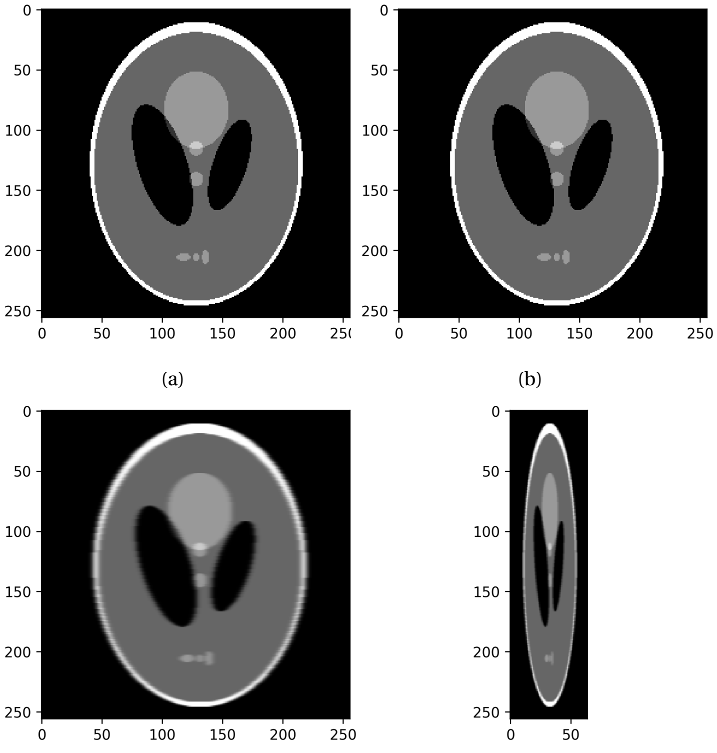


Figure 6.3: HR Shepp-Logan phantom. 128 × 128.

The downsampling factor was chosen as $L = 4$, and the standard deviation for the blurring $\sigma = 1.3$. The final LR images are then 128 × 32. The process is illustrated in Figure 6.4.



(a)

(b)

(c)

(d)

Figure 6.4: From (a) to (d), image is shifted by $3 = 4 \cdot 0.75$ pixels, blurred and downsampled.

Next, the four LR images were then transformed into vectors by lexicographical ordering and stacked, leading to the vector $\boldsymbol{y}$ of Eq. 3.1.4. The matrices D, B and G_k where multiplied and stacked, resulting in A. $\boldsymbol{y}$ and A have dimensions respectively $16,384 \times 1$ and $16,384 \times 16,384$. In fact $L * 128 * 32 = 16,384$, just like $128 * 128 = 16,384$.

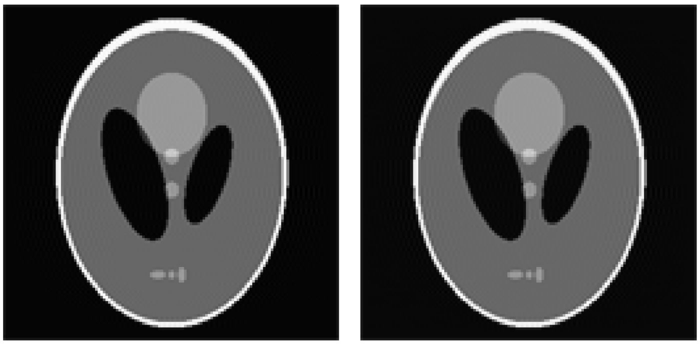
Firstly, we decided to compare the result of the Python LSQR iterative solver for least-squares with sparse matrices with the result of the CGLS without regularization. Indeed, the two methods are mathematically equivalent and a comparison between the two allows us to check the correctness of our algorithm. For further information on LSQR refer to [26]. The default tolerance for the `scipy.sparse.linalg.lsqr` function is 10^{-8}, and it took ~ 4.5 sec. on Google Colab Pro to reach that tolerance. Due to the availability of the original HR image, we were able to compare the reconstructed images not only visually, but also through the relative error between the reconstructed image $\boldsymbol{x}_\mathrm{r}$ and the original image $\boldsymbol{x}$. For the LSQR solver, the relative error in the l_1-norm was 0.012. Then, we set the same tolerance for CGLS and compared the two performances. The number of maximum iteration was set to 10000. It took the CGLS 1796 iterations and ~ 5.5 sec. to reach a relative error of 0.015. The resulting images are shown in Figure 6.5. It is clear that they are qualitatively comparable from a visual point of view. We notice no significant difference and the results are highly satisfactory.

Table 6.1: l_1 and l_2-norm of the relative error between $\boldsymbol{x}$ and $\boldsymbol{x}_\mathrm{r}$ for CGLS and ADMM on the Shepp-Logan phantom for Tikhonov and Total Variation (TV) regularization.

$\boldsymbol{\lambda}$	10^{-20}	10^{-15}	10^{-7}	10^{-5}
Tikhonov l_1	0.014	0.043	0.373	0.494
Tikhonov l_2	0.013	0.026	0.181	0.334
$\boldsymbol{\lambda = a}$	10^{-20}	10^{-15}	10^{-7}	10^{-5}
TV l_1	0.014	0.025	0.333	0.548
TV l_2	0.013	0.018	0.160	0.396

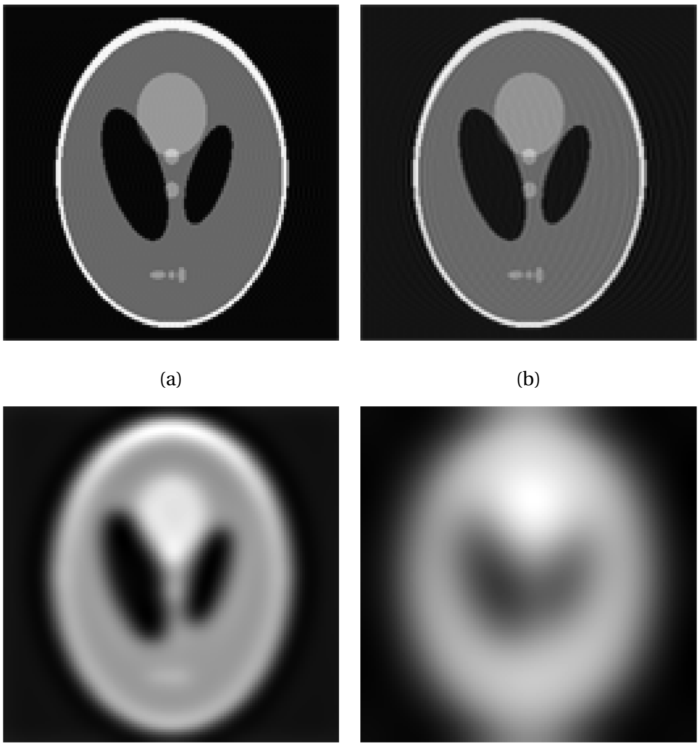
Afterwards, we wanted to analyze and compare the results for CGLS and ADMM. For the former, a second-order difference matrix with Neumann boundary conditions was chosen as regularization matrix F. For the latter, a first-order difference matrix as described in Section 3.3.2, and the number of iterations was set to 10. For CGLS, the same number of maximum iterations was used, and the same tolerance of 10^{-8}.
Figures 6.6 and 6.7 show four reconstructions for different values of λ, and Table 6.1 contains the relative errors.
Both CGLS and ADMM perform well for $\lambda = 10^{-20}, 10^{-15}$, giving visually good results, as well as quite low relative errors. The features of the HR image are clearly recognizable and the images are very similar to the ones in 6.5. In both Figures 6.6c and 6.7c we see how the image starts getting considerably blurrier, although the TV reconstruction is slightly more defined. Then the blurring increases drastically in Figures 6.6d and 6.7d.

In this particular case we obtain better results by not regularizing or by using a very low λ (which actually does not yield any regularization). The edges in the resulting images are much sharper in these cases. This was to be expected, since the original image is not smooth, and regularizing means imposing a certain level of smoothness on the image.



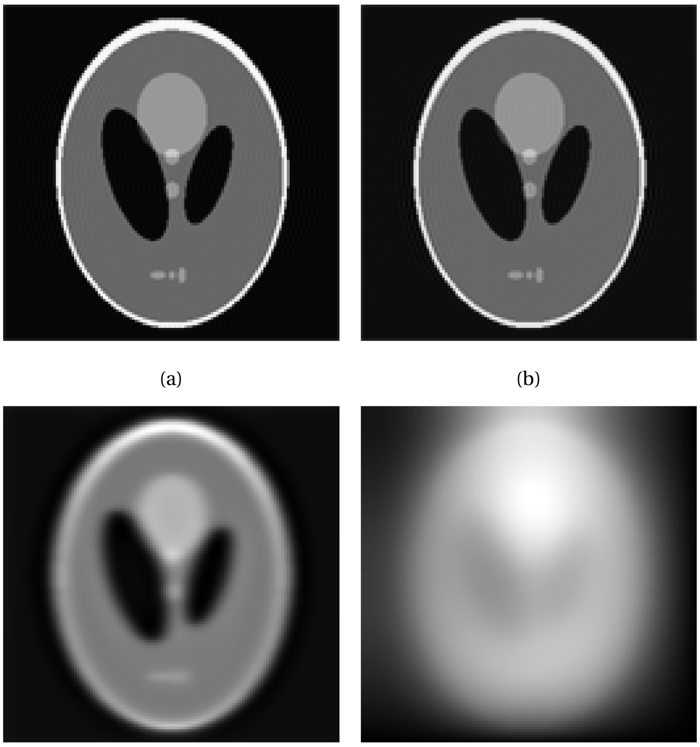
(a) Relative error in l_1-norm: 0.012 (b) Relative error in l_1-norm: 0.015

Figure 6.5: (a) `scipy.sparse.linalg.lsqr` vs (b) non-regularized CGLS.



(a) (b) (c) (d)

Figure 6.6: From (a) to (d), phantom reconstructions (Tikhonov) for different values of λ: $10^{-20}, 10^{-15}, 10^{-7}, 10^{-5}$.



(a) (b)

(c) (d)

Figure 6.7: From (a) to (d), phantom reconstructions (TV) for different values of λ: $10^{-20}, 10^{-15}, 10^{-7}, 10^{-5}$.

6.3. Low-Field Data 2D

After experimenting with the Shepp-Logan phantom, the next step was using real data. The set of apple scans described in Chapter 5 was used. SRR was applied in the yz plane. We have four LR images of size 64×16. Hence, we have again a downsampling factor $L = 4$ in the z-direction. The final matrix A is 4096×4096, $\boldsymbol{y}$ is a 4096×1 vector, just like the unknown $\boldsymbol{x}$.

We took the same steps as in the previous section, by first comparing the non-regularized CGLS with the LSQR solver, and then analyzing the results for the regularized CGLS and ADMM. Because of the availability of an HR scan ($2 \times 2 \times 2$ voxels), we could compare it with the reconstructed images and use it as a guideline to set our parameters. We could also compute the l_1 and l_2- norm of the relative error between the HR scan and the reconstructions. This was done after registering the HR scan in order for it to match the SR results. Furthermore, we employ the L-curve criterion to try and have an indication for the value of λ for CGLS.

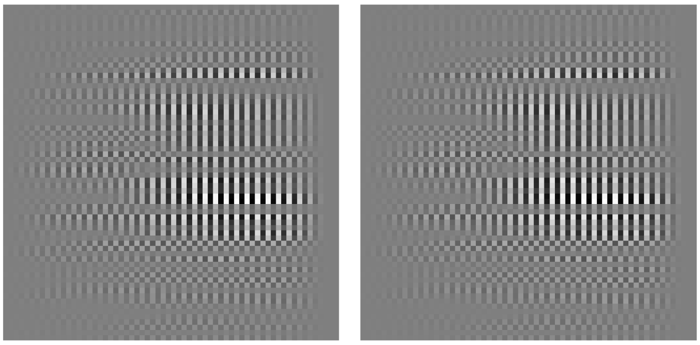
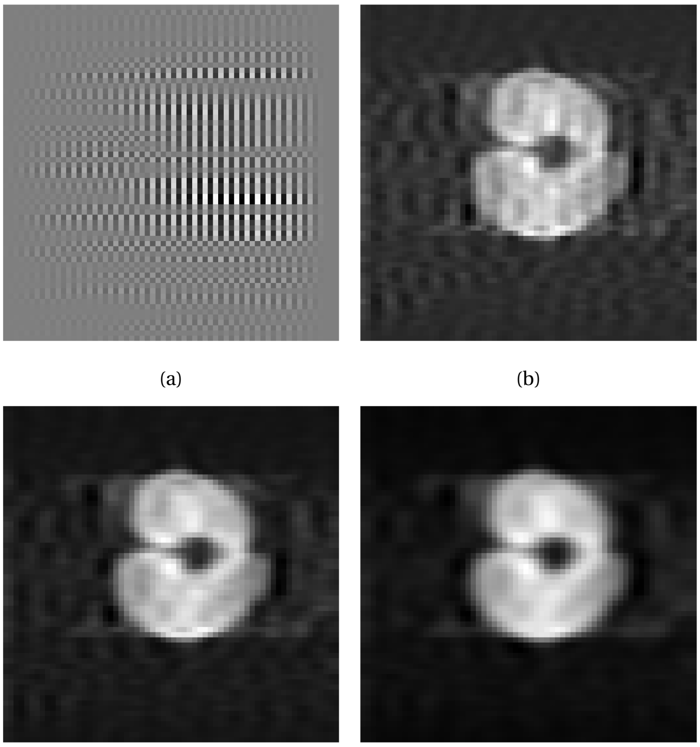

(a) Relative error in l_1-norm: ~ 8679 (b) Relative error in l_1-norm: ~ 8681

Figure 6.8: (a) `scipy.sparse.linalg.lsqr` vs (b) non-regularized CGLS.


(a) (b)

(c) (d)

Figure 6.9: From (a) to (d), apple reconstructions (Tikhonov) for different values of λ: $10^{-20}, 1.8 \cdot 10^{-10}, 1.2 \cdot 10^{-9}, 6.7 \cdot 10^{-9}$.

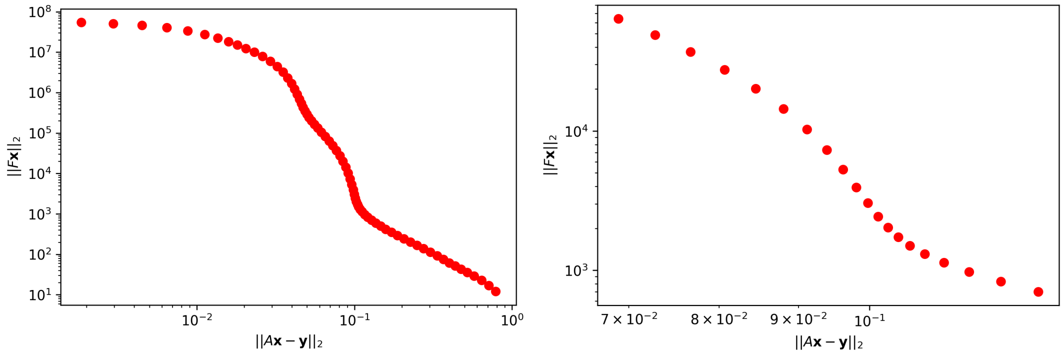
We can clearly see from Figure 6.8 that, unlike the results we had for the phantom, in this situation the absence of a regularization results in both the algorithms not being able to reconstruct the image at all. It was decided to investigate the reason behind this incredibly poor result.

Firstly, we considered the hypothesis of an incorrect registration, which might have been related to the stripes we see in the non-regularized CGLS result, as well as in both Figure 6.9a and 6.9b (even if in a lighter form in the latter). It was then tried to manually set the values for the shifts to 2 mm, as it was assumed at first in Chapter 5. Then, a different registration method was used ([27]) based on phase-only correlation, which gave slightly different results than the first method used. Nevertheless, all these strategies did not succeed in solving this problem, but gave almost analogous reconstructions. Based on these attempts, we considered it reasonable to exclude the registration as the source of the poor quality reconstruction.

In Chapter 3 it was already pointed out that the SRR problem is ill-posed. If we calculate the condition number of matrix A as the 2-norm of the matrix itself multiplied by the 2-norm of its pseudoinverse, we will obtain a value of approximately 10^5. The condition number gives an indication of how much solving a linear system will magnify the noise in the data. In Chapter 5 we have seen that even the high-resolution apple scan is visibly noisy, as it often happens with data coming from real world experiments. It might then be that because of this reasons the noise dominates the reconstruction and generates very unexpected artifacts. Nevertheless, more research would be needed to arrive to a definitive conclusion on this.

Table 6.2: l_1 and l_2-norm of the relative error between $\boldsymbol{x}$ and $\boldsymbol{x}_r$ for CGLS-Tikhonov on the apple scans.

$\boldsymbol{\lambda}$	$\mathbf{10^{-20}}$	$\mathbf{1.8 \cdot 10^{-10}}$	$\mathbf{6.9 \cdot 10^{-10}}$	$\mathbf{6.7 \cdot 10^{-9}}$
Tikhonov l_1	8007.00	2.93	2.75	2.71
Tikhonov l_2	9551.50	3.05	3.03	3.03
$\boldsymbol{\lambda = 10a}$	$\mathbf{10^{-20}}$	$\mathbf{1.8 \cdot 10^{-10}}$	$\mathbf{6.9 \cdot 10^{-10}}$	$\mathbf{6.7 \cdot 10^{-9}}$
TV l_1	8681.00	3.11	2.79	2.72
TV l_2	10483.28	3.08	3.04	3.03



(a)

(b)

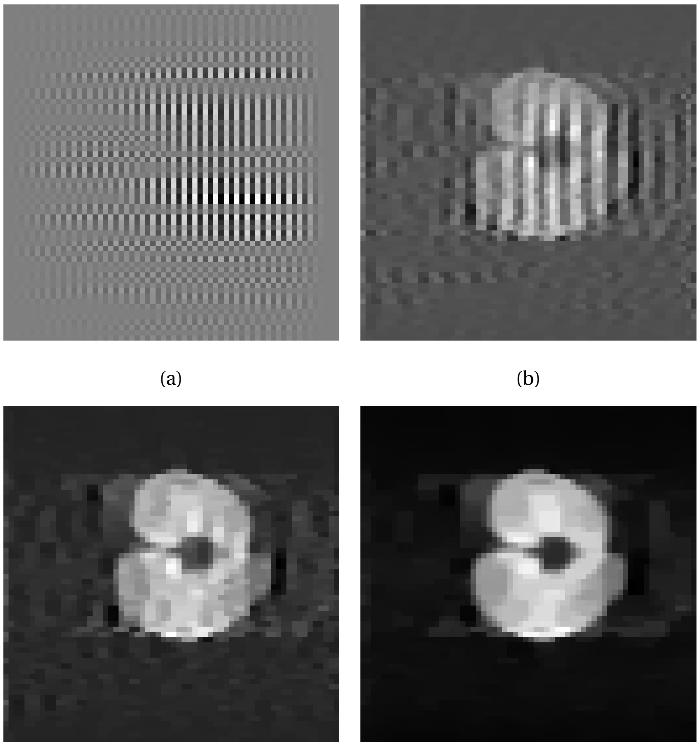
Figure 6.10: (a) L-curve plot for CGLS with Tikhonov regularization, (b) zoom from values 30 to 50.

Concerning the regularized CGLS, in Figure 6.10, we plot the 2-norm of the residual against the 2-norm of the regularized solution for 70 different values of λ. The values are evenly spaced on a log-scale from 10^{-20} to 10^{-3}. In the L-curve plot we can notice how we have a small corner going from around number 30 to 50 on the scale. By zooming in, we can see how the corner values go from 42 to 45. These values are supposed to be optimal for λ according to the L-curve criterion, and they are equal to $2.2 \cdot 10^{-10}$, $3.9 \cdot 10^{-10}$, $6.9 \cdot 10^{-10}$ and $1.2 \cdot 10^{-9}$. In Figure 6.9 we notice how the visual results are actually in accordance with the values above mentioned. Indeed, for a λ slightly smaller than $2.2 \cdot 10^{-10}$ we obtain the result in 6.9b, where the image is certainly more defined than in 6.9a but still very noisy. For λ corresponding to value 45 of the scale, we have what we deem to be the optimal reconstruction among those. A higher λ adds blurring and has an increasing smoothing effect. By looking at Table 6.2 we can interestingly observe how the relative error for 6.9d is lower than for 6.9c. It is actually true that the colors of 6.9d (i.e. the pixel values) resembles the ones of Figure 5.1 slightly more, having less noise in the central row and a subject with softer color transitions, which might be

the reason for a lower relative error. After all, we remind the reader that the HR apple scan is not exactly the ground-truth for the LR scans, therefore the relative errors are to be considered a guideline in assessing the reconstructions.
We conclude that a value for λ around $1.2 \cdot 10^{-9}$ gives the best result for this type of regularization.
In Figure 6.11 the results of ADMM are shown. We multiplied the values of λ examined for Tikhonov regularization by 10 and set $a = \frac{\lambda}{10}$. We deemed to pick these values because in our opinion they yield the best regularization for total variation.

The difference between Figure 6.9 and 6.11 is drastic. The jumps between neighboring pixels are not penalized as harshly as for Tikhonov regularization, and the reconstructions appear much smoother. The edges are sharper and the blur is almost absent. Furthermore, as it should be expected from total variation regularization, we clearly notice a staircasing effect. Concerning the relative errors, they are consistently slightly higher than those of Tikhonov regularization, but the difference is marginal.
We argue that ADMM with $\lambda = 6.7 \cdot 10^{8}$ and $a = \lambda/10$ yields a reconstruction of higher visual quality compared to CGLS, despite the artifacts typical of this kind of regularization.

Concerning the time needed for CGLS and ADMM to produce their results, depending on the values for λ and a, the first took between 2.6 and 0.04 sec. to converge, while the last took between 46 and 0.6 sec. for 10 iterations.



(a) (b)

(c) (d)

Figure 6.11: From (a) to (d), apple reconstructions (TV) for different values of λ: $10^{-19}, 1.8 \cdot 10^{-9}, 1.2 \cdot 10^{-8}, 6.7 \cdot 10^{-8}$. We set $a = \lambda/10$.

Overall, we have shown that regularization is certainly needed, hence regularized CGLS and ADMM outperform the LSQR solver. Both images 6.9c and 6.11d represent a huge improvement with respect to the LR images of Chapter 5.

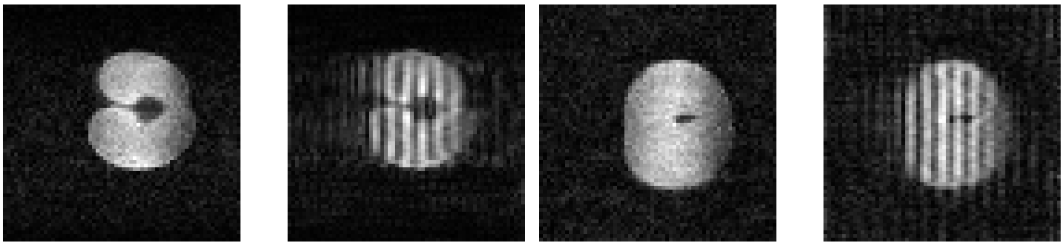
6.4. LOW-FIELD DATA 3D

After working on 2D low-field MRI for standard SR, it was decided to try to extend the codes to the whole 3D data. Nevertheless, the focus of this thesis was on super resolution for 2D data. Hence, we will not discuss these results in much detail, as we recognize that more research would be needed in order to fully exploit the possibilities of 3D standard SR. Here we present some preliminary results obtained for both Tikhonov and TV regularization for standard SR.

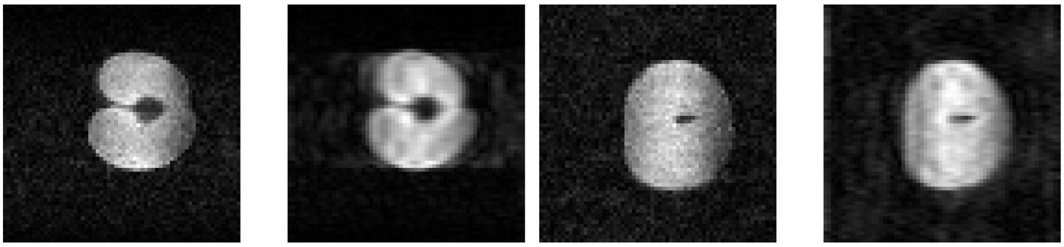
The same log-scale from 10^{-20} to 10^{-3} is considered, and in Figure 6.12 we show the results for values of λ corresponding to 40, 45 and 50 on the scale.

Interestingly, the value for lambda yielding the best reconstruction results is again $\lambda = 1.2 \cdot 10^{-9}$. The "striping" effect is now particularly evident for lower values of λ, and we speculate that the reason behind this might be the same as for 2D data. In general, for the appropriate value for λ. the 3D reconstructions are very similar to their 2D counterpart, except for the noise in the central band of the slices.

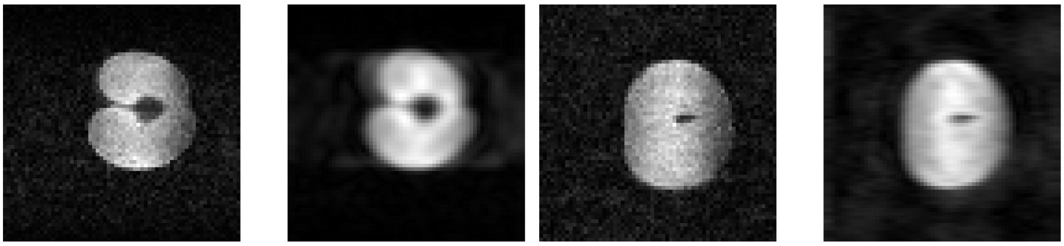
Overall we are satisfied with these findings, obtained with a simple and fast method (it took around 6.5 sec. for CGLS to converge).



(a) $\lambda = 7.2 \cdot 10^{-11}$ (b) $\lambda = 7.2 \cdot 10^{-11}$



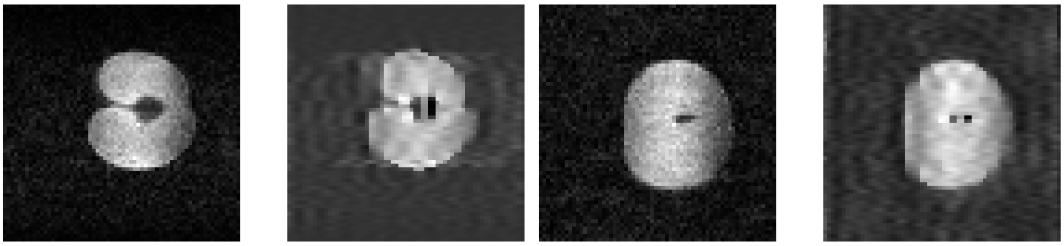
(c) $\lambda = 1.2 \cdot 10^{-9}$ (d) $\lambda = 1.2 \cdot 10^{-9}$



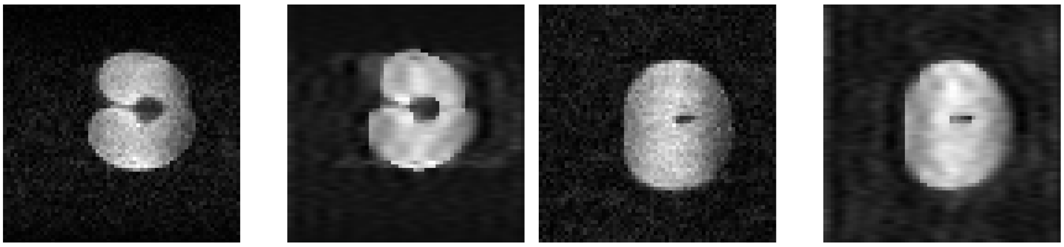
(e) $\lambda = 2.1 \cdot 10^{-8}$ (f) $\lambda = 2.1 \cdot 10^{-8}$

Figure 6.12: 3D apple reconstructions (Tikhonov) for different values of λ, compared to the HR scan. (a,c,e) yz plane, (b,d,f) xz plane.

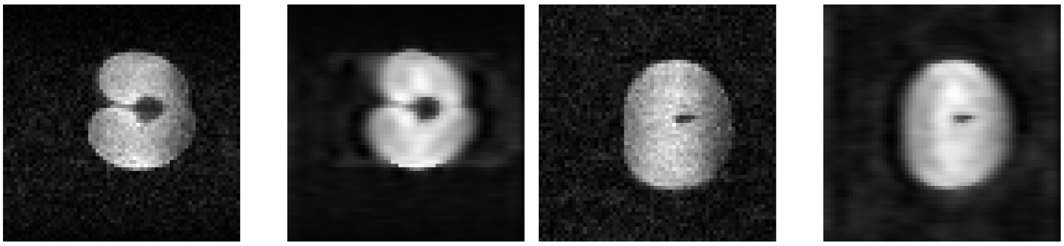
Concerning Total Variation, the parameters were chosen in order to get the most visually pleasing results. We set $a = \lambda/15$, and the values for λ as in Figure 6.13. The number of iterations for ADMM had to be increased to 20, for a total of around 3 minutes (significantly more time than for Tikhonov regularization).
In Figure 6.13 we can notice an increased staircasing effect compared to the 2D TV of Figure 6.11. The reconstruction is not very realistic, and overall arguably worse than the one of Figure 6.12, despite the slight reduction in blurring. Nonetheless, interestingly the "striping effect" is not so severely present, and only appears for very low regularization values.



(a) $\lambda = 6.5 \cdot 10^{-8}$ (b) $\lambda = 6.5 \cdot 10^{-8}$



(c) $\lambda = 2.0 \cdot 10^{-7}$ (d) $\lambda = 2.0 \cdot 10^{-7}$



(e) $\lambda = 1.1 \cdot 10^{-6}$ (f) $\lambda = 1.1 \cdot 10^{-6}$

Figure 6.13: 3D apple reconstructions (TV) for different values of λ, compared to the HR scan. (a,c,e) yz plane, (b,d,f) xz plane. We set $a = \lambda/15$.

7

DEEP LEARNING

In this chapter the basics of Deep Learning (DL) are explained and some typical architecture elements presented. The chapter is mainly based on [2, 28], as well on some other sources which will be cited later in the text.

In Chapter 3 standard Super Resolution techniques have been presented. The assumption that was made is to have several LR observation of the same scene, and from them try to reconstruct an HR image. Conversely, Single Image Super Resolution (SISR) has been gaining increasing research attention, particularly over the past two decades.

With the rapid development of deep learning techniques in recent years, deep learning based SR models have been actively explored and often achieve the state-of-the-art performance on various benchmarks of SR.[29] In this chapter we will present a brief introduction to deep learning and show how it is possible to employ deep learning techniques for SISR.

Deep learning is a sub-field of machine learning concerned with algorithms inspired by the structure and function of the brain called Artificial Neural Networks (ANNs).[30] The reason behind the name ANNs is that neural networks have originally been inspired by the goal of modeling biological neural systems. Nevertheless, the analogy between biological and artificial neurons is a loose one and artificial neurons represent a very coarse model of biological ones.

Deep learning techniques learn how to represent the data as a nested hierarchy of concepts within the layers of the neural network.[31] The idea is that the artificial neural networks send the input data through different so-called hidden layers of the network, with each layer hierarchically defining specific features of images. Hence, the computer learns progressively more difficult concepts by using the hidden layers, which extract increasingly abstract features.

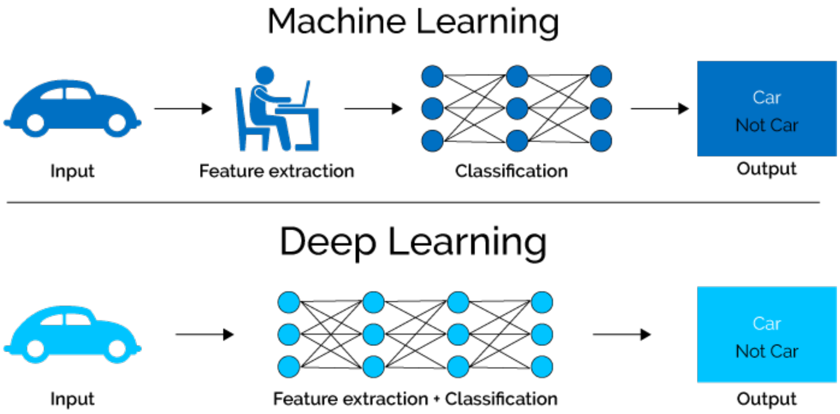


Figure 7.1: Machine learning versus deep learning. Source [32].

The development of deep learning was motivated in part by the failure of traditional algorithms. Machine learning algorithms work very well on a wide variety of important problems. However, they have not succeeded in solving problems such as recognizing speech or objects. [33] In order to give an example of the reason behind this limitation, we can imagine to have the need to categorize a collection of pictures of cats and dogs. What happens is that any machine learning algorithm needs structured data. Hence, cats and dogs need to be labeled as such in a way which would allow the algorithm to effectively distinguish them, by assigning salient features to the two categories. It appears evident how such a "simple" problem can easily become very complicated depending on the type of data and the complexity of the features.

On the other hand, since deep learning methods learn underlying hierarchical features from data, they do not necessarily need labeled data. As a result they allow to eliminate the need of domain experts to develop manual features, as represented in Figure 7.1. This can be of great advantage in situations in which there is lack of such experts.

7.1. ARTIFICIAL NEURAL NETWORKS

In order to simply and clearly explain how artificial neural networks work, it is best to explain one of their basic forms, namely the Multilayer Perceptron (MLP).

Multilayer perceptrons, also often called *deep feedforward networks* or Feedforward Neural Networks (FNN), are the quintessential deep learning models, based on a collection of connected neurons, nodes or units, where data and calculations flow from input to output only.[33] Neurons are organized in layers and connected in an acyclic graph. Hence, the outputs of some neurons can become inputs to other neurons, but in MLPs cycles are not allowed.[28] The neural network architectures (or neural network *models*) that will be presented in Chapter 8 are of course more complex, but the core idea behind them has many similarities with the one behind feedforward networks.

These networks are an evolution of perceptrons, which are single neuron models, made by an input layer having input units and an output layer having a single output unit.

When we "feed" an input to a perceptron, what happens is that the neuron performs a dot product with the input and its weights and adds the bias, resulting in z (see Figure 7.2), and finally applies the so-called *activation function* (AF) f.

Perceptrons were born as binary classification algorithms, hence the activation function used was the Heaviside step function. We will observe soon why, in the context of more complex neural networks, we require the activation function to be nonlinear and differentiable.

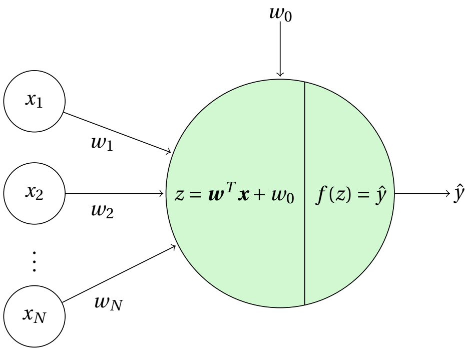


Figure 7.2: Visual scheme of a perceptron.

The relation between input and output is defined in Eq. 7.1.1.

$$\hat{y}(\boldsymbol{x}, \Theta) = f(\underbrace{\boldsymbol{w}^T\boldsymbol{x} + w_0}_{z}), \tag{7.1.1}$$

where $\boldsymbol{x} = \{x_i\}_{i=1}^N$ in the input vector, $\hat{y}$ is the output, $\boldsymbol{w} = \{w_i\}_{i=1}^P$ is the vector of weights, w_0 is the bias and $\Theta = \{\boldsymbol{w}, w_0\}$ is the parameter set.

The idea is that, given the classifier $\hat{y} = \hat{y}(\boldsymbol{x})$, the perceptron defines a mapping $\hat{y} = \hat{y}(\boldsymbol{x}, \Theta)$ from the input $\boldsymbol{x}$ to the output $\hat{y}$, thus finding the best approximation to the classifier by learning the best parameters Θ for it.

A perceptron is a simple binary classification algorithm only capable of learning linearly separable patterns, which is indeed a limitation when instead dealing with non-linear problems. It was observed that simply adding a second layer of nodes is sufficient to solve a lot of otherwise non-separable problems, giving birth to the idea of a multilayer perceptron, simply shown in Figure 7.3.

A MLP consists of at least three layers of nodes: an input layer, a hidden layer and an output layer, where by *hidden layer* we simply refer to any intermediate layer between the input and the output layer. Except for the input nodes, each node uses a nonlinear activation function, which is what, combined with the multiple layers, mark the difference between the MLP and the perceptron. These features allow the MLP to distinguish data that is not linearly separable.

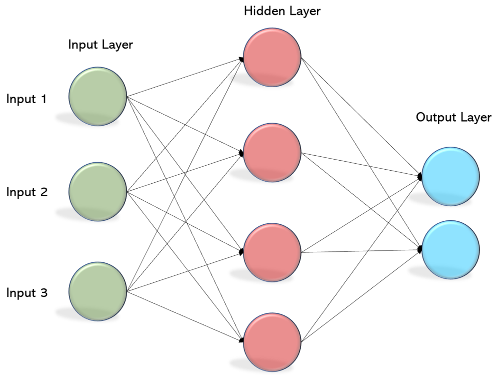


Figure 7.3: A 2-layer multilayer perceptron. Source [34].

Clearly, a multilayer perceptron generally has more than one hidden layer and several input and output units. Since, as the name suggests, the input layer is only responsible for receiving the inputs, it does not count when referring to the number of layers of a MLP. Hence, a M-layer MLP consists of $M-1$ hidden layers and 1 output layer. All the operations are carried out in the hidden and output layers, or, in the case of a single perceptron as in Figure 7.2, only in the output layer. M is called the *depth* of the network and the number of nodes in each layer is called the *width* of the layer. In Equation 7.1.2 we show the output of the i^{th} neuron within a layer m in the case of a M-layer MLP.

$$y_i^{(m)} = f_i^{(m)}(z_i^{(m)}) \quad \text{with} \quad z_i^{(m)} = \sum_{j=1}^{n^{(m-1)}} w_{ij}^{(m)} y_j^{(m-1)} + w_{i0}^{(m)}. \tag{7.1.2}$$

In the equation, $w_{ij}^{(m)}$ denotes the weighted connection between the i^{th} node in layer m and the j^{th} node in layer $m-1$, $w_{i0}^{(m)}$ is the bias associated with node i of layer m, $n^{(0)}$ is the number of inputs of the input layer and $n^{(m-1)}$ is the number of outputs of layer $m-1$. Furthermore, $y^{(0)}$ is the input of the network and $y_j^{(m-1)}$ is the j^{th} output of layer $m-1$. Finally, $f_i^{(m)}$ is the activation function associated with level m.

7.2. ACTIVATION FUNCTIONS

Activation functions can have various forms depending on the task of the network, and they have been evolving to face the new, increasing challenges posed by training of neural networks and by the growing requests for very specific deep learning architectures.

Historically, the logistic sigmoid function (see Eq. 7.2.1) is a common activation function. That is because it has a biological interpretation as the firing rate of a neuron. Indeed, neurons go from not firing at all if the received signal is below a certain threshold (output 0), to fully-saturated firing at an assumed maximum frequency (output 1).[28] The sigmoid is often referred as the “squashing” function, since it squashes the input into range between 0 and 1.

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{7.2.1}$$

However the sigmoid AF has fallen into disuse, since it suffers of major drawbacks, such as vanishing gradients during backpropagation (see Section 7.3). Additionally, the weights of sigmoid neurons need to be properly initialized to prevent saturation. Indeed, if the initial weights are too large then most neurons would become saturated and the network will barely learn. [28] The hyperbolic tangent function in Eq. 7.2.2 was proposed to remedy some of these drawbacks, and in fact it has been shown to give better training performance for multi-layer neural networks. [35] Furthermore, as opposed to the sigmoid, the hyperbolic tangent output is zero-centered, a desirable property as observed in Section 7.3. Nevertheless the hyperbolic tangent does not succeed in solving the vanishing gradient problem, which led to further research for an activation function which would not suffer of this relevant drawback.

$$\tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{7.2.2}$$

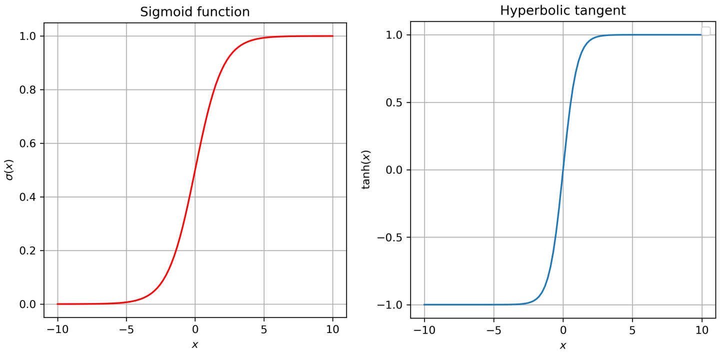


Figure 7.4: Plots for the sigmoid and the hyperbolic tangent activation functions, respectively.

In 2010 the Rectified Linear Unit (ReLU) activation function was proposed [36] and since then it has been the most widely used activation function, and has achieved state-of-the-art results in many deep learning applications. [35]

$$f(x) = \max(0, x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{if } x < 0. \end{cases} \tag{7.2.3}$$

This activation function, defined in Eq. 7.2.3 and plotted in 7.5, performs a threshold operation to each input element where values less than zero are set to zero. Indeed, the ReLU owes its name to the fact that it represents a nearly linear function which "rectifies" the values of the inputs less than zero, which makes the ReLU a non-saturated AF and in turn eliminates the vanishing gradient problem encountered earlier with the sigmoid and hyperbolic tangent function. [28] Another significant advantage of this activation function is that it allows faster computations due to the absence of complex operations such as computing exponentials or divisions. Moreover, it introduces sparsity in the hidden units and consequently allows for a lighter network.

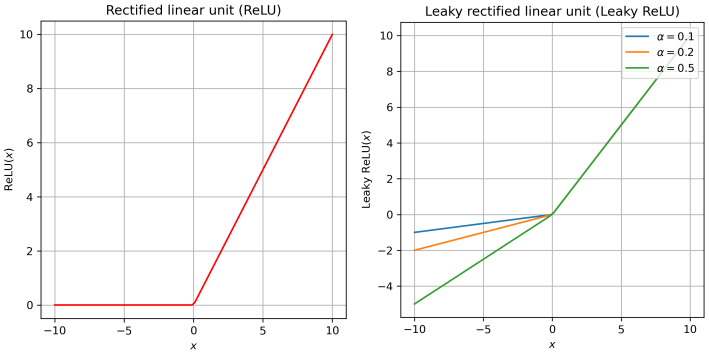


Figure 7.5: Plots for the ReLU and the Leaky ReLU with different values of α, respectively.

Unfortunately, despite showing a marked improvement compared to the previously analyzed functions, two main drawbacks affect the ReLU. Firstly, it easily overfits compared to the sigmoid function, an issue which has been nevertheless shown to be mitigated by the use of regularization techniques. Secondly, ReLU units can be fragile during training and thereby cause some of the gradients to “die”. What happens is that if certain weights of the network lead to negative inputs, the ReLU units will have zero output, and therefore zero gradient. As we will see in Section 7.3, neural networks learn through what is called backpropagation, a technique which uses gradient descent to optimize the networks weights. As a consequence, the weights will not get adjusted during descent if the gradient is zero, hence the gradient flowing through the unit will be zero from that point on causing the death of the neuron. This issue is often encountered when the learning rate of the chosen optimizer is set too high, and it becomes less prominent when this parameter is properly set. Furthermore, a correct weight initialization is of great importance in order to limit the possibility of running into this problem.

As an attempt to solve the dead neurons issue, the Leaky ReLU activation function was presented in 2013. This function introduces a small negative slope compared to the ReLU (see Figure 7.5), regulated by a parameter α. The presence of this parameter guarantees that none of the gradients will be zero at any time during the training. However, in practice ReLU is still widely used, and often preferred to its variants for multiple reasons. ReLU is generally faster than LeakyReLU and also, [28] suggests that the performance of Leaky ReLU is not consistent and should not *a priori* be preferred to ReLU.

Finally, another widely used activation function is the Softmax function, typically used in multi-class models, where it returns probabilities of each class with the target class having the highest probability. The function expressed in Eq. 7.2.4 is indeed used to compute probability distribution from a vector of real numbers, as it produces an output which is a range of values between 0 and 1, with the sum of the probabilities been equal to 1. [35]

$$f(x_i) = \frac{\exp(x_i)}{\sum_j \exp\left(x_j\right)} \tag{7.2.4}$$

7.3. NETWORK TRAINING

In Section 7.1 we have talked about parameters of an artificial neural network. These parameters are the weights and biases associated with each neuron in the network. They are variables estimated from the dataset by the network itself, and are learned during training of the network. However, a network is also affected by the so-called *hyperparameters*.

Examples of such hyperparameters are the activation function of a neuron, the number and size of layers of the network, and several others which will be referred later in the chapter. They differ from parameters because they are configuration variables external to the network and require to be set manually before the training.

The training of a network is the way the network learns, which in turn refers to the way the parameters are optimized in order to make accurate predictions. We make a distinction between *supervised* and *unsupervised* learning. In supervised learning, the machine is trained using labeled data in order to predict outcomes for unforeseen data. As an example, if we consider an image dataset made of cats, dogs and other animal “classes” in which each picture has assigned a label defining the kind of animal, the network will learn to associate specific features to each of the classes thanks to the labels. Later on, such a network might be used for classification, or even for anomaly detection.

On the other hand, usually unlabeled data is used to train networks in unsupervised learning. In this setting, the network is given a dataset containing many features, then learns useful properties of the structure of this dataset. From a mathematically intuitive point of view, unsupervised learning involves observing several examples of a random vector $\boldsymbol{x}$, and attempting to implicitly or explicitly learn the probability distribution $p(\boldsymbol{x})$, while supervised learning involves observing several examples of a random vector $\boldsymbol{x}$ and an associated target value or vector $\boldsymbol{y}$, and learning to predict $\boldsymbol{y}$ from $\boldsymbol{x}$, usually by estimating $p(\boldsymbol{y}|\boldsymbol{x})$.[33]

The architectures which will be shown in Chapter 8 are based on supervised learning, hence our explanation of the learning process of a network will refer to this class of models.

From a mathematical point of view during training our network learns a mapping from inputs $\boldsymbol{x}$ to targets $\boldsymbol{y}$ given a labeled set of input-targets pairs $\mathscr{D} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^{N}$, where by target we refer to the desidered value for the response associate to one input. $\mathscr{D}$ is called the training set and N is the number of training samples.

The goal of the training of a supervised network is the optimize the parameters of the model by minimizing a certain cost or *loss function* in order for the calculated output $\hat{\boldsymbol{y}}$ to be as close as possible to the target $\boldsymbol{y}$.[33]

Accordingly to the specific task different loss functions are employed, but two of the most common are the

Mean Squared Error (MSE) and the cross-entropy loss function, defined respectively as follows

$$E(\Theta) = \sum_{i=1}^{N} E_n(\Theta) = \sum_{n=1}^{N}\sum_{k=1}^{C} \left(\hat{y}_k(\boldsymbol{x}_n, \Theta) - y_{n,k}\right)^2, \tag{7.3.1}$$

$$E(\Theta) = \sum_{i=1}^{N} E_n(\Theta) = \sum_{n=1}^{N}\sum_{k=1}^{C} y_{n,k} \log\left(\hat{y}_k(x_n, \Theta)\right), \tag{7.3.2}$$

where $y_{n,k}$ denotes the k^{th} entry of the target $\boldsymbol{y}_n$ associated with input $\boldsymbol{x}_n$, Θ is the vector of parameters (weights and biases) and C is the number of targets of the output layer. The MSE is simple to implement and the idea behind it is intuitive and clear, since it simply takes the square of the difference between output and target and averages it on the whole dataset. The cross-entropy loss function is often used for classification problems when outputs are interpreted as probabilities of the inputs belonging in an indicated class. Here, $y_{n,k}$ is equal to 1 if the output is supposed to be categorized as belonging to class k, 0 otherwise.

It is interesting to notice that when dealing with linear classifiers, such as the perceptron in Figure 7.2 or logistic regression models, we usually have convex loss functions. That is not the case for neural networks with non-linear activation functions, which cause most interesting loss functions to become non-convex.[33] A consequence of the non-convexity of the training of neural network models is the absence of a guarantee of convergence and a sensitivity to the values of the initial parameters.

7.3.1. GRADIENT DESCENT

The most widely used algorithm known for solving the optimization problem mentioned earlier is Gradient Descent (GD), where model weights are updated each iteration using the backpropagation algorithm.
Let us consider Eq. 7.3.1 again. It is known that

Theorem 7.1. *For $f \in \mathrm{C}^2$ and $\|\boldsymbol{h}\|$ small:*

$$f(\boldsymbol{x}+\boldsymbol{h}) = f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^\top \boldsymbol{h} + \frac{1}{2}\boldsymbol{h}^\top \nabla^2 f(\boldsymbol{x})\boldsymbol{h} + O\left(\|\boldsymbol{h}\|^2\right).$$

Corollary 7.1.1. *Consider $f : \mathbb{R}^n \to \mathbb{R}, f \in C^1$ (resp. $f \in C^2$). If $\boldsymbol{x} \in \mathbb{R}^n$ is a local minimizer then $\nabla f(\boldsymbol{x}) = \boldsymbol{0}$ (resp. $\nabla^2 f(\boldsymbol{x}) \succeq \boldsymbol{O}$).*

Hence $\nabla f(\boldsymbol{x}) = 0$ is a necessary condition for $\boldsymbol{x}$ to be a local minimizer. The sufficiency only holds for convex functions. Therefore in our case we want to have

$$\nabla E(\boldsymbol{w}) = 0. \tag{7.3.3}$$

The ideal scenario would be to "simply" find the analytical solution(s) to the previous equation, but that is unfortunately unfeasible due to the complexity of the error function and the numerical cost of the gradient evaluation. That is where the GD comes in. The GD is a first-order iterative optimization algorithm for finding a local minimum of a differentiable function. The idea behind the algorithm is to adjust the parameters according to the direction of steepest descent, namely the direction that shows the largest decrease in the error measure. The following lemmas hold

Lemma 7.2. *Consider $f : \mathbb{R}^n \to \mathbb{R}$, $f \in C^1$, and $\boldsymbol{x}, \boldsymbol{d} \in \mathbb{R}^n$. Then*

1. $\frac{df}{d\boldsymbol{d}} = \nabla f(\boldsymbol{x})^T \boldsymbol{d}$;
2. *If $\frac{df}{d\boldsymbol{d}} < 0$, then $\boldsymbol{d}$ is a strict descent direction of f at $\boldsymbol{x}$;*
3. *If $\frac{df}{d\boldsymbol{d}} > 0$, then $\boldsymbol{d}$ is a strict ascent direction of f at $\boldsymbol{x}$ (and thus is not a strict descent direction of f at $\boldsymbol{x}$).*

Lemma 7.3. *For $f \in C^1(\mathbb{R}^n, \mathbb{R})$ and $\boldsymbol{x} \in \mathbb{R}^n$ s.t. $\nabla f(\boldsymbol{x}) \neq 0$ we have*

$$\arg\min_{\mathrm{d}} \left\{\nabla f(\mathbf{x})^\top \mathbf{d} : \|\mathbf{d}\|_2 = 1\right\} = -\frac{\nabla f(\mathbf{x})}{\|\nabla f(\mathbf{x})\|_2}.$$

$\boldsymbol{d} = -\frac{\nabla f(\boldsymbol{x})}{\|\nabla f(\boldsymbol{x})\|_2}$ is the direction of steepest descent.

Therefore we have

$$\boldsymbol{w}[i+1] = \boldsymbol{w}[i] + \Delta\boldsymbol{w}[i] \tag{7.3.4}$$

$$\Delta\boldsymbol{w}[i] = -\eta\nabla E(\boldsymbol{w}[i]), \tag{7.3.5}$$

where i is the iteration of GD and $\eta > 0$ is a hyperparameter of the network called the *learning rate* (often set between 0 and 1.0), which regulates how much the weights are adjusted with respect to the loss gradient. Hence at each iteration i we update in the direction $-\nabla E(\boldsymbol{w}[i])$, which according to Lemma 7.3 is the direction of steepest descent of E at $\boldsymbol{w}$.

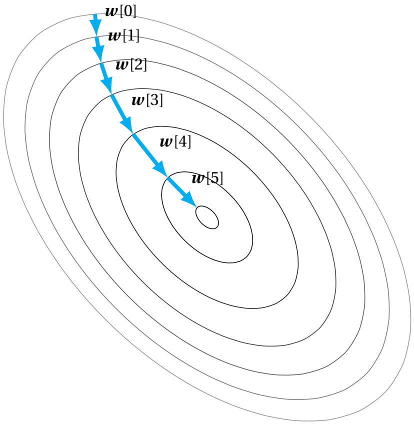


Figure 7.6: Gradient descent (GD) on a series of level curves. Under the hypothesis of E being a convex function, GD can converge to the global solution.

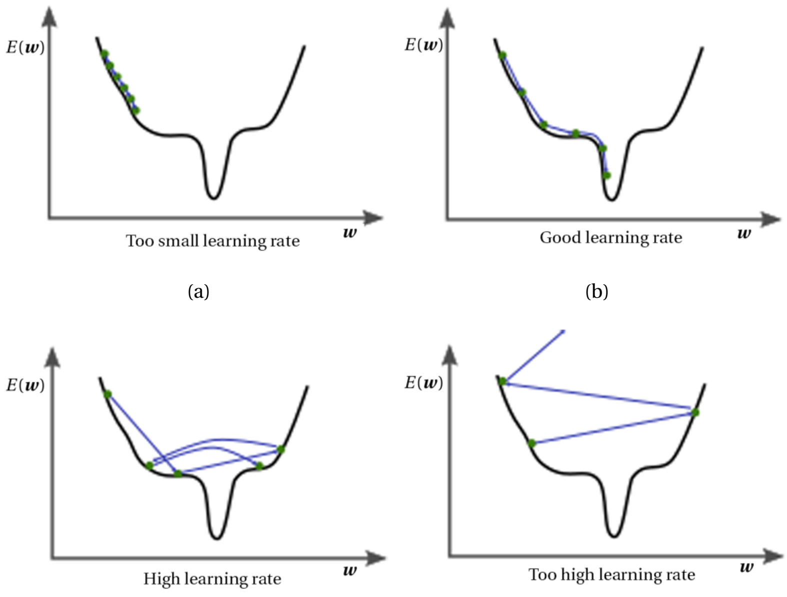
There exist three versions of the GD algorithm: batch (or "vanilla") gradient descent, stochastic gradient descent (SGD) and mini-batch gradient descent.

Batch gradient descent computes the gradient of the loss function with respect to the weights for the entire training dataset. Hence Equations 7.3.4–7.3.5 are calculated for $E(\boldsymbol{w}) = \sum_{n=1}^{N} E_n(\boldsymbol{w})$. This algorithm is guaranteed to converge to the global minimum for convex functions and to a local minimum for non-convex functions. As we need to calculate the gradients for the whole dataset to perform just one update, batch gradient descent can be very slow and unfeasible to apply to large datasets.[37] The second variant of GD differs from batch GD because it performs a parameter update for each training sample, based on the error $E_n(\boldsymbol{w})$. This allows the SGD to be much faster than batch GD, which performs redundant computations by recomputing gradients for similar samples. However, the added speed of SGD comes with a cost. The name stochastic comes from the fact that the gradient based on a single training sample is a "stochastic approximation" of the actual cost gradient.[38] Because of its stochastic nature, the path to the global minimum consists of frequent updates with a high variance that cause the objective function to fluctuate heavily. These fluctuations complicate the convergence to the exact minimum, because they cause SGD to overshoot. Nevertheless, it has been proved that decreasing the learning rate of SGD leads to the same convergence behavior as batch GD, with the almost certain convergence to a local or the global minimum for non-convex and convex optimization respectively.

Mini-batch gradient descent is a combination of both batch and stochastic GD. This algorithm performs an update for every mini-batch of M training samples, hence the weights are updated based on the error $E_M(\boldsymbol{w}) = \sum_{n\in M} E_n(\boldsymbol{w})$. As a consequence, it allows for a more stable convergence by reducing the variance of the parameter updates. It can be noted that SGD is a particular case of mini-batch GD where the mini-batch contains a single sample. Furthermore, it is common to refer to mini-batch GD as SGD, where it is implied that mini-batches are used.

The size of the mini-batch is a hyperparameter but it is usually not deemed to have a much relevant impact on the training of a model. It is usually based on memory constraints or manually set to some specific value by the programmer. Common batch sizes are 32, 64, 128 or 256. Powers of 2 are used in practice because many vectorized operation implementations work faster when their inputs are sized in powers of 2.[28, 33]

The number of times a learning algorithm works through the entire training set is the number of *epochs*. During each iteration, one mini-batch is processed by the algorithm, hence the number of iterations to complete one epoch is given by n_iterations=n_trainining_data/batch_size.



(a) (b)

(c) (d)

Figure 7.7: Effects of different learning rates on the training of a network. A too high learning rate might lead to divergence.

The learning rate was said earlier to indicate how quickly the model is adapted to the problem. This hyperparameter has a strong impact on the training of a network and can make the difference between convergence or divergence of the model. Quoting [33], “The learning rate is perhaps the most important hyperparameter. If you have time to tune only one hyperparameter, tune the learning rate.” As you can intuitively see from Figure 7.7, a too small learning rate might take a very high amount of iterations to converge and consequently an excessive and sometimes unfeasible amount of time. Conversely, a high learning rate might lead to the network overshooting the minimum, oscillating around the optimal solution without reaching it (Figure 7.7c), often even leading to divergence (Figure 7.7d).

Contrarily to what was said for batch an stochastic GD, mini-batch GD does not guarantee good convergence and it is highly dependent on the choice of the learning rate, which can be a difficult one to make as we have already showed through Figure 7.7. It is common to adopt a trial-and-error procedure to estimate an appropriate value, usually leaning between 0.1 and 10^{-5}, or even 10^{-6} in specific cases. What is considered to be a good approach is to try all learning rates on a logarithmic scale between the values mentioned earlier and to monitor the learning curves that plot the loss function as a function of epochs.[33]

Nevertheless, it has been seen in practice that it is usually necessary when choosing mini-batch GD as an optimizer to gradually decrease (decay) the learning rate over time (i.e. over the epochs). This is done by setting a learning rate schedule and reducing the learning rate during training accordingly. However, this sort of schedules need to be set in advance by the programmer, and are thus unable to adapt to a dataset characteristics.[37]

This kind of considerations is what led to a growing interest towards a different class of optimizers, namely algorithms with adaptive learning rate. The idea behind these algorithms is to adapts the learning rate to the parameters, performing smaller updates for parameters associated with frequently occurring features, and larger updates for parameters associated with infrequent features.[37] Some well known optimizers which belong to this class are Adagrad, RMSprop and Adam. In particular, Adam [39] has recently seen broader adoption for deep learning applications in computer vision and natural language processing. It is a method for efficient stochastic optimization that only requires first-order gradients with little memory requirement. As a per-parameter optimizer, it computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients.[39] The name derives indeed from the phrase “adaptive moment estimation”.[33]
Mini-batch GD and Adam will be the optimizers of choice for the results in Chapter 9.

7.3.2. BACKPROPAGATION

Let us consider a FNN that receives the input $\boldsymbol{x}$ and produces the output $\hat{y}$. As we know from the previous sections, information flows forward through the network. The initial information provided by $\boldsymbol{x}$ propagates to the hidden units of the network and finally produces the output. This first step is called forward propagation. Then, during training forward propagation continues until a scalar cost $E(\Theta)$ is obtained. The backpropagation algorithm allows the error to flow backwards through the network.[33] This way, the gradient can be computed and the iterative adjustment process of the weights can continue.

For an extensive description of backpropagation we refer the reader to [33, 40]. Once the gradient is computed, it is used to update the weights of the network as described in Section 7.3.1. Note that there is often confusion about the exact meaning of the term backpropagation, and it is common to refer to it when describing the whole learning algorithm for multi-layer neural networks. However, we shall use this term specifically to describe the evaluation of derivatives.

The backpropagation algorithm can be decomposed in four main steps:

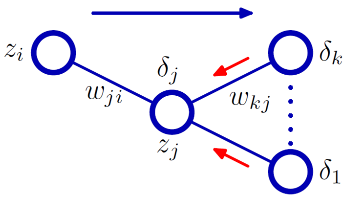


Figure 7.8: Calculation of δ_j for hidden layer j by backpropagation. The blue arrow indicates the direction of forward propagation, while the red ones indicate the backward propagation of error information. Source [40].

- ▹ The first step is the forward propagation of the information from the input value $y^{(0)}$ to all the hidden units. This is done by successively applying

$$y_i^{(l)} = f_i^{(l)}(z_i^{(l)}) \text{ with } z_i^{(l)} = \sum_{j=0}^{m^{(l-1)}} w_{ij}^{(l)} y_j^{(l-1)}, \tag{7.3.6}$$

  where we used the same notation as in Section 7.1.

- ▹ Secondly, the error of each unit in the output layer is computed as follows

$$\delta_k = \hat{y}_k - y_k, \tag{7.3.7}$$

  where $\hat{y}_k$ is the target value for unit k.

- ▹ Thirdly, the error for all the hidden units can be obtained by backpropagating the errors from deeper layers in the network (see Figure 7.8), by using the following formula:

$$\delta_i^{(l)} = f'(z_i^{(l)}) \sum_{k=1}^{m^{(l+1)}} w_{ki}^{(l+1)} \delta_k^{(l+1)}. \tag{7.3.8}$$

- ▹ Lastly, the derivatives of the loss function with respect to the weights are evaluated:

$$\frac{\partial E}{\partial w_{ij}^{(l)}} = \delta_i^{(l)} y_j^{(l-1)}. \tag{7.3.9}$$

7.3.3. WEIGHT INITIALIZATION

As explained in Section 7.3.1, the parameters in a neural network are optimized by updating them according to the direction of steepest descent. Before updating the parameters though, it is necessary to initialize them for every layer. This is a crucial step, because it is well known that arbitrary initializations can slow down or even completely stall the convergence process.[41] Typically, the biases for each unit are set to heuristically chosen constants [33], and only the weight initializers require fine-tuning. There are upsides and drawbacks for both too large and too small initial weights. If initial weights are too large this could result in exploding values during forward propagation or back-propagation. Large weights may also result in extreme values that cause the activation function to saturate, causing complete loss of gradient through saturated units.[33] On the other hand, too small initial values will hinder the gradients to flow backwards beneficially, and the network will take longer to converge (it might even not converge at all).

Weight initialization is an area of active research, and numerous methods have been proposed to deal with this problem.[41] A very intuitive way to initialize weights is to do so drawing them from a Gaussian distribution with 0 mean and finite variance. This is done in Keras through the RandomNormal initializer, which takes as parameters the mean and standard deviation of the distribution.

To prevent the gradients of the network from vanishing or exploding, it is necessary to ensure that the variances of the outputs from the different layers are approximately the same. Indeed, arbitrary initializations can result in the deeper layers receiving inputs with small variances, which in turn slows down backpropagation and retards the overall convergence process.[41] Hence, when considering our normal distribution $\mathcal{N}(0,\sigma^2)$, σ^2 should be chosen as such to enforce the aforementioned condition.

Glorot and Bengio performed a systematic analysis of this problem [42], and showed that for a linear activation function, the optimal value of $\sigma^2 = 1/N$, where N is the number of nodes feeding into the considered layer. This initialization scheme is referred to as *Xavier initialization*, and it is one of the most widely used. In order to understand the reason behind this, let us consider the inner product $s = \boldsymbol{w}^T\boldsymbol{x}$ between the weights $\boldsymbol{w}$ and input $\boldsymbol{x}$. s gives us the raw activation of a neuron before the non-linearity. If we now consider the variance of s, we have

$$\begin{aligned}
\mathrm{Var}(s) &= \mathrm{Var}\Big(\sum_i^n w_i x_i\Big) \\
&= \sum_i^n \mathrm{Var}(w_i x_i) \\
&= \sum_i^n [E(w_i)]^2 \mathrm{Var}(x_i) + [E(x_i)]^2 \mathrm{Var}(w_i) + \mathrm{Var}(x_i)\mathrm{Var}(w_i) \\
&= \sum_i^n \mathrm{Var}(x_i)\mathrm{Var}(w_i) \\
&= (n\mathrm{Var}(w))\mathrm{Var}(x).
\end{aligned} \tag{7.3.10}$$

In these calculations we made multiple assumptions. In the third step we assumed that $E[x_i] = E[w_i] = 0$, and in the last that all w_i, x_i are i.i.d (independently identically distributed). From this derivation we see that to ensure that $\mathrm{Var}(s) = \mathrm{Var}(x)$, we need to have

$$\mathrm{Var}(w) = \frac{1}{n} \tag{7.3.11}$$

Note that ReLU units have positive mean, hence the first assumption does not hold. In [43] an initialization specific for ReLU activation function is derived, and it is argued that when using ReLU the best weight initialization strategy is to initialize the weights randomly but with variance $\sigma^2 = 2/N$. We will refer to this as *He initialization*.

Despite this, we will see in Chapter 9 that for shallow enough networks Xavier initialization does perform well.

7.3.4. VALIDATION

Once a model has been trained, in order to judge its generalization abilities it is necessary to validate the model. Indeed, the final goal of training a network is to give useful and meaningful predictions on unseen data.

The concept of a validation set is to give a “preview” of how well the model is going to perform on data which are not part of the training dataset. The need for a second set originates from the fact that the performance on the training set is not a good indicator of predictive performance on unseen data, due to the problem of overfitting which will be addressed in Section 7.3.5.

If we are in possess of a large amount of data, a common, intuitive strategy would be to shuffle the the samples and split the data into three parts: training, validation and test dataset. This approach is also called the *hold-out validation*, as the validation and test dataset are simply “held out”. The training set is the biggest and is the one used to update the weights of the model. The other two datasets are usually much smaller than the training one, and of similar size. Common training-to-validation ratios go from 90/10 to 75/25, depending on how big the dataset is.

During training the validation set is frequently used to evaluate the performance of the model through calculating the loss. After the training, based on the results on the validation set the programmer fine-tunes the hyperparameters of the network. Finally, once the final model has been trained, the test set is used for a final accuracy evaluation of the model. The test set is not used for learning or for any architectural or hyperparameter decisions. This last set is needed because if the model design is iterated many times using a limited size data set, then some overfitting to the validation data can occur and evaluating on these data would give a biased result.[40]

Nevertheless, this method is often considered naïve ans suffers from one significant flaw: if little data is available, then your validation and test sets may contain too few samples to be statistically representative of the data at hand [44] and might give a relatively noisy estimate of predictive performance.[40]

What is then commonly done is to instead perform the so-called k-fold cross-validation. The idea behind this technique is to partition the dataset into k parts (folds). Then, $k-1$ of these folds are used as training dataset and the remaining fold is used at the end of the training to evaluate the performance of the model. Then, the same thing is repeated for all the k possible choices for the validation set . This way, each data point will be in each of the sets at least one time. Then, the performance for the different runs are averaged. The procedure is shown in Figure 7.9.

In general, k-fold cross-validation is preferable to the simple hold-out validation, as it gives the model the opportunity to train on multiple train-validation splits, while using only one validation set might make the evaluation of the model dependent on the particular set used.

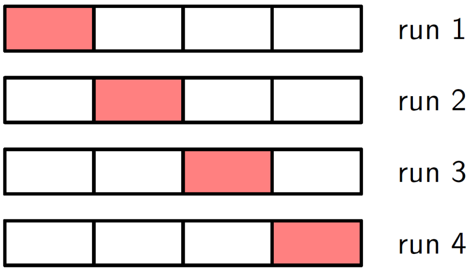


Figure 7.9: k-fold cross-validation for $k = 4$. Source [40].

7.3.5. OVER AND UNDERFITTING

In machine learning, the bias error is an error resulting from incorrect assumptions in the learning algorithm. In other words, the bias tells us how far are the predicted values from the target values. When a model predicts the training data well, is it said that it has a low bias. When instead it fails to capture the underlying pattern of the training data, we say that the model underfits or has a high bias.[2] There can be several possible reasons for a model to underfit, the more common of which are: the model used is too simple for the kind of data used, or the features of the training samples are not informative enough.[45]
Increasing the complexity of the model is often an appropriate solution to tackle and solve this problem. When instead the model performs well on the training set, but has a high loss on the validation and test set, it is said to have a high variance or that it overfits. The reasons for this are opposite to the ones which lead to underfitting: the model is too complex for the data, or the samples have many relevant features but there is lack of data.

The variance is an error of the model due to its sensitivity to small fluctuations in the training set. [45] For instance, high variance can cause an algorithm to model the random noise in the training data rather than important features which would allow it to make meaningful predictions on new data. There exist various solutions to the overfitting problem: a simpler model, dimensionality reduction of the data, more training data or regularization.[45] The last approach is the most commonly used, also because the other possibilities are not always feasible or require major changes in the structure of the network.

Regularization is defined in [33] as "any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error". Regularization methods force the learning algorithm

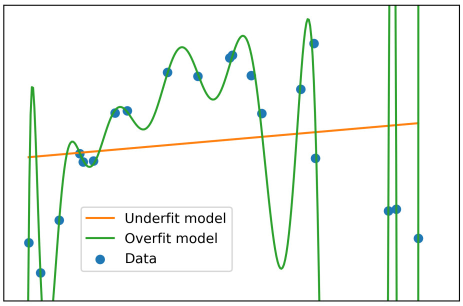


Figure 7.10: The plot shows examples of under, over and good fitting. When overfitting, the model best follows the training data but it is too dependent on them and probably will not generalize well. Source [46].

to build a less complex model. This often leads to slightly higher bias but significantly reduces the variance. This problem is known in the literature as the bias-variance trade-off.[45]

WEIGHT REGULARIZATION

The idea behind weight regularization is to modify the objective function by adding a penalizing term whose value is higher when the model is more complex. A model with larger weights is more complex than one where weights are smaller, so the new loss function acts by penalizing large weights through the addition of a penalty term $\Omega(\Theta)$. [33] If we denote the regularized loss function by $\tilde{E}(\Theta)$, we have

$$\tilde{E}(\Theta) = E(\Theta) + \alpha\Omega(\Theta), \tag{7.3.12}$$

where $\alpha \in [0, \infty[$ is a hyperparameter which determines the magnitude of the penalization. Based on the form of $\Omega(\Theta)$ we distinguish different kinds of weight regularizations. Two of the most common were already mentioned in Chapter 3, namely L_1 and L_2 regularization, which respectively yield

$$\Omega(\Theta) = \|\Theta\|_1 \tag{7.3.13}$$

and

$$\Omega(\Theta) = \|\Theta\|_2^2. \tag{7.3.14}$$

L_1 regularization encourages sparse weights by assigning a zero value to many of them, while L_2 penalizes larger weights more severely, but results in less sparse weights.

DROPOUT

Dropout is an extremely effective regularization technique introduced in [47]. The name refers to “dropping out” units in a neural network, namely temporarily removing them from the network, along with all their incoming and outgoing connections. The concept is simple: each time a new training sample is fed to the network, some units are temporarily excluded from the computation. The higher the percentage of units excluded the higher the regularizing effect.[45]. The probability of retaining a unit is a fixed probability $p \in [0, 1]$ independent of other units. The value for p can be tuned on the validation set, or simply set at 0.5, which seems to be close to optimal for a wide range of networks and tasks.[47]

When dropout is applied to a network, a “thinned” network is sampled from it, consisting of the units which survived dropout. If we consider a neural net with n units, it can be seen as a collection of 2^n possible thinned neural networks.[47] Since for every training sample a new thinned network is sampled and trained, training a neural network with dropout is equivalent to training 2^n thinned networks which extensively share weights and where each network gets trained very rarely or even not at all.

In this thesis we will not discuss these techniques further, since they will not be used in practice in the models we will describe and test in Chapters 8 and 9 respectively.

EARLY-STOPPING

We know that when training a neural network as the number of epochs increases, the loss decreases, meaning that the network is learning and fitting the training data well. If we consider a hold-out split setting, where we have our validation set, what we can often see is that there comes a point in which the training loss keeps on

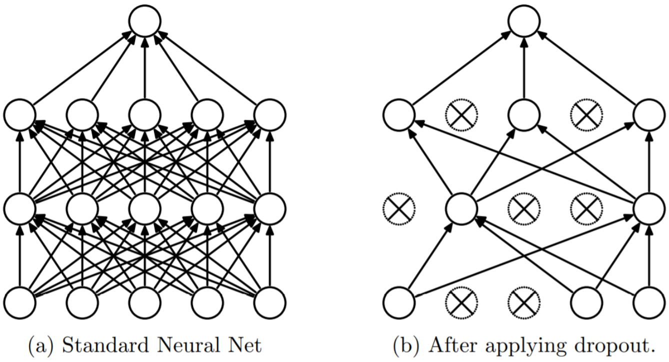


Figure 7.11: Dropout in the case of a FNN with two hidden layers. For each training sample, a new thinned network is sampled and trained (b). Source [47].

decreasing but the validaton loss starts increasing instead. The model will stop generalizing and start learning the statistical noise in the training dataset. We are in presence of overfitting.

If the weights of the network are saved after each epoch during training, it is possible to stop the training once the model starts overfitting. This approach is called early-stopping, and it is one of the most popular and effective regularization techniques, also because it does not require any changes to the model or objective function which can change the learning dynamics of the system.

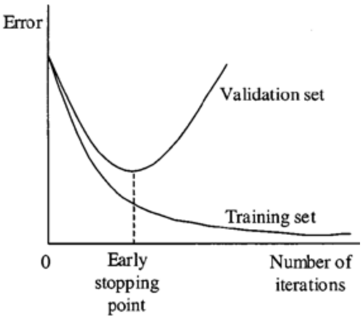


Figure 7.12: Representation of early-stopping. Source [48].

7.4. CONVOLUTIONAL NEURAL NETWORKS

In the previous sections the multilayer perceptron was presented. The MLP has a so-called fully-connected architecture. Fully Connected Neural Networks (FCNNs) are a type of artificial neural network where the architecture is such that all the nodes, or neurons, in one layer are connected to the neurons in the next layer. [49]

Convolutional Neural Networks (CNNs) are a different class of deep neural networks usually applied to analyzing visual imagery. These architectures, also known as shift invariant or Space Invariant Artificial Neural Networks (SIANN), are very similar to the feedforward networks described in the previous sections. Indeed, they are made of several layers of neurons that have learnable weights and biases, neurons receive inputs, perform a dot product and then apply an activation function.

Nevertheless, the assumption of having images as input allows the possibility to encode certain properties into the architecture of the network, and in turn consistently reduce the amount of learnable parameters in the network compared to fully-connected networks. [28, 50]

In fact, when training a fully-connected network on image data the number of parameters can quickly reach very high figures. As an example, consider an input image of size 250 × 250 × 3, where 250 is the height and width and 3 is the number of color channels. A single neuron in a fully-connected hidden layer will then have $25 \cdot 250 \cdot 3 = 187.500$ weights, that will be added to the weights of all the other neurons in the network.

Such an amount of parameters not only makes the training of the network tremendously expensive from a computational point of view, but it also makes the network more prone to overfitting.

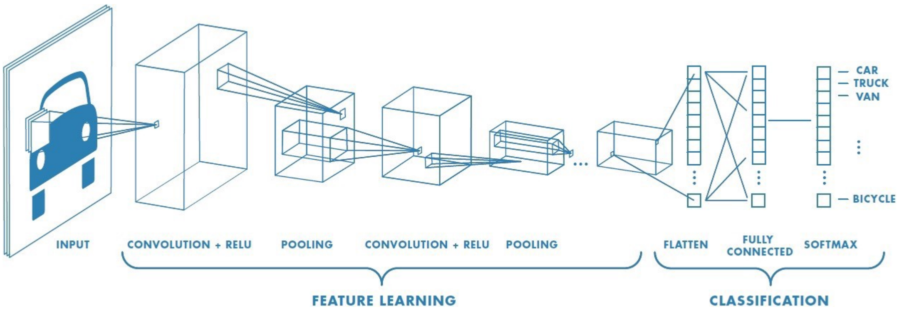


Figure 7.13: Scheme of a CNN architecture. Source [51].

In our example we considered a three-dimensional input image, where the last dimension is the number of color channels. Indeed, unlike the neural networks presented in the previous sections, the layers of a CNN have neurons organized in volumes, each of them with its own width, height and depth. As you can intuitively see from Figure 7.13 and as will be explained further in this section, the neurons in a layer of a CNN are only connected to a small region of the layer before it, allowing for a much lighter network in terms of parameters. [28]

The training of a convolutional neural network can generally be divided into two parts: feature extraction and classification. In order to perform these different operations a CNN is typically made of three different kinds of layers: convolutional layers, max pooling layers and fully-connected layers.

The latter has already been presented in Section 7.1, and it is usually the last layer of a CNN, responsible for the classification once the features have been extracted (see Figure 7.13). For classification purposes, the activation function commonly used is the Softmax. The remaining two new kinds of layers will be described shortly in the remainder of the section.

7.4.1. Convolutional Layer

Convolutional layers (ConvLayers for short) represent the bulk of a CNN. As the name suggests, they perform a convolution between the layer input and the filters of the convolutional layer itself. Indeed, a set of learnable so-called *filters* or kernels are associated to each ConvLayer. Filters are usually small spatially and extend through the full depth of the input volume, so for example a filter on a first layer of a CNN with as input a RGB image might have size 5x5x3, where 5 is the width and height and 3 (the number of color channels) is the depth.

The concept of convolution was already introduced briefly in Section 6.1. Similarly to what was described there, during the forward pass each filter is “slid” across the width and height of the input volume and compute dot products between the entries of the filter and the input at any position. By performing this operation for each filter, we obtain the 2-dimensional so-called *feature maps* or activation map. This map represents the responses of that filter at every spatial position. Examples of feature maps are shown in Figure 7.14. We can see how the first ConvLayer is responsible for capturing the low-level features such as edges and simple shapes. By going deeper into the network, the feature map starts looking more like an abstract representation of the cat more than an actual cat. The network is learning high-level concepts related to the class of the image, such as the particular shape of the ears, eyes and nose, the pattern of the fur. In short, what makes of a cat a cat.

Feature maps for each filter are stacked along the depth dimension and finally produce the output volume.

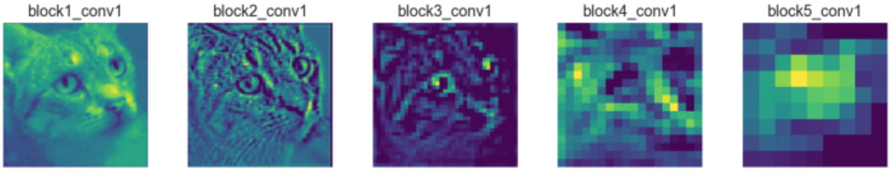


Figure 7.14: Feaure maps for a CNN trained on cat images. Source [52].

The number and size of the filters are hyperparameters of a ConvLayer. The *receptive field* is another hyperparameter which tells us the input area of a neuron. As already anticipated, conversely to fully-connected layers, in a ConvLayer each neuron receives input from only a restricted subarea of the previous layer. The extent of the connectivity along the depth axis is always equal to the depth of the input volume, while the width and height are reduced (see Figure 7.13).

Another two important hyperparameters in a ConvLayer are the *stride* and the *padding*.

The stride is simply the number of pixels shifts over the input volume. So a stride of 1 means that filters are moved one pixel at a time.

When convolving an image with a filter, we will usually observe a reduction in the output size compared to the input. This happens when the so-called *valid* padding is applied. If the stride is 1, then the size of the output will be reduced by $f-1$, where f is the filter size. Instead, it would be possible to zero-pad the input so that the input and output have the same size. This kind of padding is called *same* padding. Then, if same padding is set and the stride is 1, the number of zero padded will be $(f-1)/2$. If *full* padding is chosen instead, then the number of zero padded will be $f-1$ and the output will have its size augmented by $f-1$. Hence, padding allows to control the size of the final output.

It should be noted that also the number of strides has an influence on the final output size. More generally, Eq. 7.4.1 gives the final output size $\tilde{n}$ based on the padding p, the stride s, the filter size f and the input size n.

$$\tilde{n} = \frac{n+2p-f}{s} + 1 \tag{7.4.1}$$

7.4.2. MAX POOLING LAYER

Pooling layers have the function to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and consequently also help preventing overfitting. [28] They are often used after convolutional layers, creating a structure of the kind shown in Figure 7.13. A *max pooling* layer in particular returns the maximum value from the portion of the image covered by the kernel. Another possible option are *average pooling* layers, which return the average of all the values instead of the maximum. Nevertheless, it is generally believed that max pooling layers perform considerably better than average pooling, also because of the de-noising properties of the max pooling.[51]

8

DEEP LEARNING ARCHITECTURES FOR SISR

As already anticipated in Chapter 3, there are undeniable advantages in multi-image super resolution compared to its single-image version. Furthermore, as we already know in general image super resolution (and specifically SISR) is very challenging and inherently ill-posed, since there are always multiple HR images corresponding to a single LR image.[18] However, there has been significant progress in single image super resolution as deep learning methods and deep neural networks have been brought into use. Deep learning based SR models have been actively explored and often achieve the state-of-the-art performance on various benchmarks of SR.[29] On the other hand, except for some very recent work (see [17, 53]), there has not been a comparable progress in MISR networks.[17]

As we know from Chapter 7, deep learning super resolution algorithms are learning-based. They contain a training step in which the relationship between some HR examples (from a specific class) and their LR counterparts is learned. They are also example-based, because they learn mapping functions from external low and high-resolution sample pairs. A variety of deep learning methods have been applied to tackle SR tasks, also when dealing with MRI data. These methods differ from each other in major aspects such as different types of network architectures, different types of loss functions or different types of learning principles. Among the most advanced and recent works in deep learning SR for MRI we refer to [54].

In this chapter two different deep learning architectures for SISR will be described: Super Resolution Convolutional Neural Network (SRCNN)[55] and ReCNN (Residual Convolutional Neural Network)[56].

8.1. SRCNN

In 2014 the first convolutional neural network for super resolution was presented.[55] The idea behind this algorithm is to model through a CNN the same pipeline of the Sparse-Coding-based (SC) method. The SC is one of the representative external example-based SR algorithms. Here, overlapping patches are densely cropped from the input image and pre-processed by subtracting the mean and normalizing. Next, the patches are encoded by a low-resolution dictionary, while the sparse coefficients are passed into a high-resolution dictionary for reconstructing high-resolution patches. Lastly, the overlapping reconstructed patches are aggregated to produce the final output. More details on this algorithm can be found in [57, 58].

In [55] it is shown how this pipeline is equivalent to a deep convolutional neural network, namely the SRCNN. This network directly learns an end-to-end mapping between low and high-resolution images, but conversely to the SC method this is not achieved by explicitly learning dictionaries, but implicitly through the hidden layers of the network.

Undoubtedly there has been major progress in the field of deep learning for super resolution since the publication of this method in 2015. Nevertheless, it was deemed interesting to explore the potentialities of this network due to the simplicity of both the idea behind it and its implementation. Indeed, with moderate number of filters and layers and very little pre-processing the network has been proved to substantially outperform bicubic interpolation with little computational time, and even provide superior accuracy compared to state-of-the-art example-based models of the time in which it was published.

In [18] an improved version of the model is presented. One of the main contributions was the extension of

SRCNN to process three channels of color images simultaneously, a desirable property for a super resolution algorithm. However, since the final scope of this report is to able to improve the resolution of low-field MRI scans, it was considered more appropriate to consider gray-scale images (as it was done already in Chapter 6). Hence, we will not make use of this possibility offered by SRCNN. Later on in the text our observations and references to the SRCNN architecture will be based on [18].

One important distinction usually made between different types of supervised deep learning SR algorithms concerns the upsampling method used. There exist indeed several options regarding this point. In this report we will focus on architectures where upsampling is a pre-processing step and where models have the role of refining the upsampled images and reconstruct high quality details.

8.1.1. ARCHITECTURE

Let us consider a single LR image and upscale it by bicubic interpolation to the desired size. We will denote the interpolated image by Y and the ground-truth HR image by X. For the sake of simplicity from now on will refer to Y as a LR image, despite it having the same size of X. The goal of SISR is to recover from Y an image $f(Y)$ as similar as possible to X, thus learning f. In the sparse-coding-based method this is achieved through three main phases: patch extraction and representation, non-linear mapping, and reconstruction. We will now briefly explain how all of these operations can be combined to form a single convolutional neural network. More detailed and complete information can be found in [18].

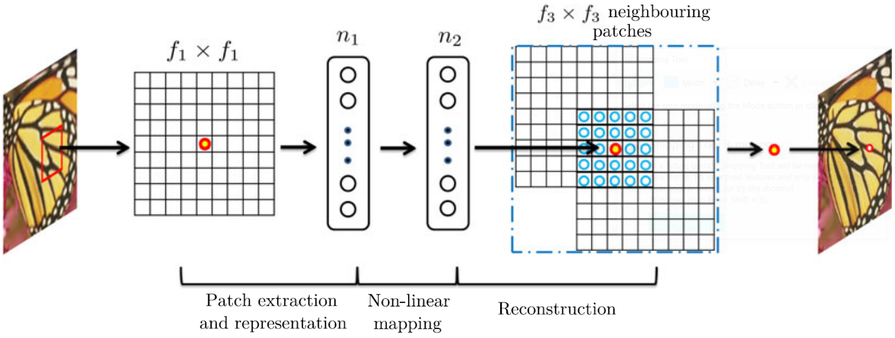

$f_3 \times f_3$ neighbouring patches
$f_1 \times f_1$
n_1
n_2
Patch extraction and representation
Non-linear mapping
Reconstruction

Figure 8.1: An illustration of sparse-coding-based methods in the view of a convolutional neural network. Here the dataset is the T91 dataset of natural images (see Section 9.1 for more details).

In the first phase overlapping patches are extracted from Y. Then, the sparse coding solver of choice will project each patch onto a LR dictionary. If we now consider a $f_1 \times f_1$ LR patch and a dictionary of size n_1, this operation is equivalent to applying n_1 linear filters ($f_1 \times f_1$) on the input image. From a mathematical point of view, the first layer of this CNN is expressed as an operation F_1:

$$F_1(Y) = \max(0, W_1 * Y + B_1), \tag{8.1.1}$$

where W_1 and B_1 represent the filters and biases respectively and '$*$' denotes the convolution operation. W_1 corresponds to n_1 filters of support $c \times f_1 \times f_2$, where c is the number of channels of the input image. The output is composed of n_1 feature maps. B_1 is an n_1-dimensional vector, whose each element is associated with a filter. The ReLU activation function is applied on the filter responses $(\max(0, \cdot))$. Note that the ReLU can be equally considered as part of the first or second operation (non-linear mapping). In this last case, the first convolution becomes purely linear. ReLU makes convergence much faster, while still presenting good quality.

Next, the sparse coding solver will iteratively process the n_1 coefficients. The outputs of this solver are n_2 coefficients, and usually $n_1 = n_2$ in the case of sparse coding. These n_2 coefficients are the representation of the high-resolution patch. In this sense, the sparse coding solver behaves as a special case of a non-linear mapping operator with spatial support 1×1. From the point of view of SRCNN, the second layer applies n_2 filters with spatial support 1×1 to the n_1-dimensional vectors obtained earlier. This process is expressed through F_2:

$$F_2(Y) = \max\left(0, W_2 * F_1(Y) + B_2\right), \tag{8.1.2}$$

where W_2 contains n_2 filters of size $n_1 \times f_1 \times f_2$ and B_2 is n_2-dimensional. Again, the output will be made of n_2-dimensional vectors that, conceptually, are a representation of an HR patch that will be used for reconstruction. Note that this interpretation is only valid for 1 × 1 filters, but can be generalized to larger filters like 3 × 3 or 5 × 5. In that case, the non-linear mapping is on a 3 × 3 or 5 × 5 "patch" of the feature map instead of on a patch of the input image.

Lastly, the above n_2 coefficients (after sparse coding) are projected onto another HR dictionary to produce a high-resolution patch. The overlapping HR patches are then averaged to generate the final HR image, which is expected to be similar to the ground-truth X. If the HR patches used for reconstruction are of size $f_3 \times f_3$, then the linear filters have an equivalent spatial support of size $f_3 \times f_3$. The averaging can be considered as a predefined filter on a set of feature maps, where each position is the flattened vector form of an HR patch. That is why we can define a convolutional layer to produce the final image:

$$F(Y) = W_3 * F_2(Y) + B_3, \tag{8.1.3}$$

where W_3 corresponds to c filter of size $n_2 \times f_3 \times f_3$ and B_3 is a c-dimensional vector.

A very interesting thing to notice is that although the above three operations are motivated by different intuitions they all lead to the same form as a convolutional layer. By combining these layers we obtain the architecture in Figure 8.2, namely the SRCNN.

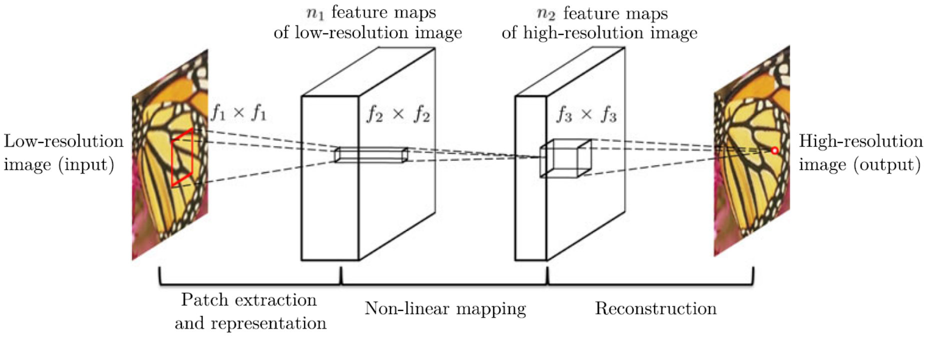


Figure 8.2: A scheme of the SRCNN architecture.

8.1.2. TRAINING

As we know from Chapter 7, in order to learn the end-to-end mapping function F, it is necessary to estimate the network parameters $\Theta = \{W_1, W_2, W_3, B_1, B_2, B_3\}$. This is achieved through minimizing the loss between the reconstructed images $F(Y, \Theta)$ and the corresponding ground-truth high-resolution images X. The loss function used in [18] is the MSE defined in Eq. 7.3.1. It was already mentioned in Chapter 7 that the mean squared error is one of the most commonly used loss functions. However, in this particular context there is a further reason for this choice of a loss function, namely that it favors a high PSNR.

The PSNR (Peak-signal-to-noise-ratio) is arguably the most widely-used metric for quantitatively evaluating image restoration quality, and is at least partially related to the perceptual quality.[18] It represents the ratio between the maximum possible power of a signal and the power of corrupting noise that affects the fidelity of its representation. It is defined via the maximum pixel value (denoted as L) and the MSE between images. Given the ground-truth image X with N pixels, and its reconstruction $\hat{X}$, the PSNR between X and $\hat{X}$ is given by

$$\text{PSNR} = 10 \cdot \log_{10}\left(\frac{L^2}{\text{MSE}(X, \hat{X})}\right), \tag{8.1.4}$$

where L equals to 255 in general cases using 8-bit representations. As we can see from the definition of PSNR, this metric is only related to the pixel-level MSE. As a consequence, it often leads to poor performance in representing the reconstruction quality in real scenes, due to its inability to grasp the presence (or lack) of many details relevant to the human perception. Nevertheless, due to the necessity to compare with literature works and the lack of completely accurate perceptual metrics, PSNR is still currently the most popular evaluation criterion for SR models.[29]

Another Image Quality Assessment (IQA) metric worth mentioning is the so-called Structural Similarity Index Metric (SSIM). This metric was proposed for measuring the structural similarity between images, based on independent comparisons in terms of luminance, contrast, and structures.[29] Let us consider again X and $\hat{X}$, with X having N pixels. The SSIM is defined as

$$\mathrm{SSIM}(X,\hat{X}) = l(X,\hat{X})c(X,\hat{X}), s(X,\hat{X}), \tag{8.1.5}$$

where

$$\begin{cases} l(X,\hat{X}) = \dfrac{2\mu_X\mu_{\hat{X}} + C_1}{\mu_X^2 + \mu_{\hat{X}}^2 + C_1} \\ c(X,\hat{X}) = \dfrac{2\sigma_X\sigma_{\hat{X}} + C_2}{\sigma_X^2 + \sigma_{\hat{X}}^2 + C_2} \\ s(X,\hat{X}) = \dfrac{\sigma_{X\hat{X}} + C_3}{\sigma_X\sigma_{\hat{X}} + C_3} \end{cases} . \tag{8.1.6}$$

Here, $l(X,\hat{X})$ is the luminance comparison function, $c(X,\hat{X})$ is the contrast comparison function and $s(X,\hat{X})$ is the structure comparison function. The first measures the closeness of the two images' mean luminance (μ_X and $\mu_{\hat{X}}$), and it is equal to 1 and maximal only for $\mu_X = \mu_{\hat{X}}$. The second measures the closeness in contrast between the two images, and the last measures the correlation coefficient between X and $\hat{X}$. The positive constants C_1, C_2 and C_3 are used to avoid a null denominator. The range of values for the SSIM goes from 0 to 1, with 1 is achieved only for $X = \hat{X}$.

Nowadays there is no unanimous consent on which metric to choose. On one hand, there have been studies revealing that, as opposed to the SSIM, the MSE (and consequently the PSNR) performs badly in discriminating structural content in images. For instance, it has been observed how various types of degradations applied to the same image can yield the same value of the MSE. Other studies have shown that the MSE, and consequently the PSNR, have the best performance in assessing the quality of noisy images.[59] In our analysis of the results of Chapter 9 we will consider both metrics in order to estimate the quality of the predictions of our network, as well as a visual assessment.

In [57] the optimizer used for the network is SGD with the standard backpropagation and the weights are initialized by drawing randomly from a Gaussian distribution with zero mean and standard deviation 0.001. We will see in Chapter 9 that using minibatch SG and Adam leads to very good results in a shorter amount of time and that there exist better options for the weight initialization.

8.2. RECNN

The authors who gave birth to SRCNN made attempts of preparing deeper versions of their network, but failed to observe superior performance, and in some cases deeper models gave even inferior performance. Eventually, they concluded that deeper networks do not result in better performance. While in Chapter 9 we will test this claim by training different architectures for SRCNN, in 2016 Kim et. al. presented a new, deep network for super resolution, which they called Very Deep Super Resolution (VDSR) network.[60]
The idea behind this model is to cascade small filters many times in a deep network structure, in order to exploit contextual information over large image regions in an efficient way.[60] It is argued that for a large scale factor, it is often the case that information contained in a small patch is not sufficient for detail recovery.

In this report we will present a variation of the VDSR, which we will refer to as ReCNN (Residual Convolutional Neural Network). This model was inspired by [56], where the 3D version of this network is implemented and tested. In the remainder of the section the architecture and training process of the ReCNN will be explained.

8.2.1. ARCHITECTURE

The ReCNN model consists of 10 layers all of the same type, except for the first and last: 64 filters of size 3 × 3 followed by a ReLU activation function. The last layer is instead made of a single filter with the same kernel size, and a linear activation function. From a mathematical point of view, it can described as

$$\begin{cases} F_1(\mathrm{Y}) = \max(0, W_1 * \mathrm{Y} + B_1) \\ F_i(\mathrm{Y}) = \max(0, W_i * F_{i-1}(\mathrm{Y}) + B_i) \quad \text{for } 1 < i < L \\ F_L(\mathrm{Y}) = W_L * F_{L-1}(\mathrm{Y}) + B_L \end{cases} , \tag{8.2.1}$$

where L is the number of layers and W_i and B_i are the weights and biases of the network. W_i corresponds to n_i filters of support $f_i \times f_i \times c$, where c is again the number of channels of the input layer. Finally, $\max(0,\cdot)$

refers to the activation function ReLU applied to the filter responses of each convolutional layer except for the last.[56] In our case, $L = 10$, $n_i = 64$ for $1 \le i < L$, $c = 1$ and $f_i = 64$.

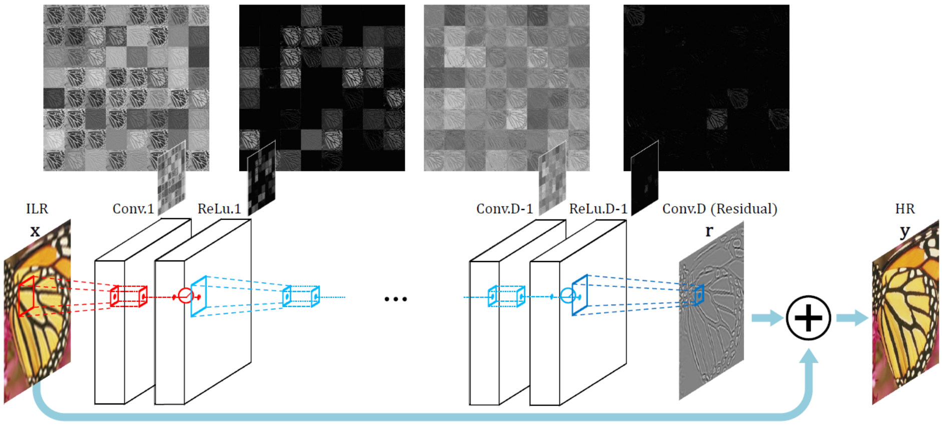


Figure 8.3: A scheme of the VDSR architecture.[60] The ReCNN has the same structure with smaller filter sizes.

8.2.2. TRAINING

A significant problem when dealing with very deep network is a painfully slow convergence. Simply increasing the learning rate is usually not a preferable solution, because it can lead to potentially exploding gradients. That is a problem which occurs when large error gradients accumulate and result in very large updates to neural network model weights during training. The gradients increase exponentially as they propagate down the model and, eventually, explode. When the magnitudes of the gradients accumulate, an unstable network is likely to occur, which can cause poor prediction results and be incapable of effective learning.[61] For further information on this problem we refer the reader to [62].

In [60] this issue was addressed by means of residual-learning and adjustable gradient clipping. Concerning the latter, the basic idea is to recall that the gradient does not specify the optimal step size, but only the optimal direction. Hence, when the traditional gradient descent algorithm proposes to make a very large step, gradient clipping “clips” the step size to a certain range $[-\alpha, \alpha]$, so that it is small enough that it is less likely to go outside the region where the gradient indicates the direction of approximately steepest descent.[33] Instead of a predetermined range, adjustable gradient clipping works by clipping the gradients to $[-\frac{\alpha}{\gamma}, \frac{\alpha}{\gamma}]$, where γ is the learning rate. This method compensates for small learning rates caused by learning rate decay. Indeed, as we know during training learning rate is multiplied to adjust the step size, so if a smaller learning rate is used training time can increase considerably. Adjustable gradient clipping enables faster learning by clipping gradients to a larger interval when small step sizes are used, while at the same time preventing exploding gradients.

These strategies are meant to work when using SGD for training. As we will see in Chapter 9 though, Adam will prove to be a good choice as optimizer for ReCNN, therefore adjustable gradient clipping will be discarded.

In super resolution the main goal is to estimate high-frequency components from LR observations. The idea of residual-learning is to, instead of learning directly and end-to-end mapping from the LR space to the HR one, estimate a mapping from the LR space to the missing high-frequency components, also called the residual between HR and LR data.

We define a residual image $R = X - Y$ where, since the two images are supposedly very similar, most values in R will be zero or very small anyway. The goal of the network is to predict this residual image. In order to do so, the loss function becomes

$$\frac{1}{2}\|R - f(Y)\|^2, \tag{8.2.2}$$

where $f(Y)$ is the prediction of the network.

In terms of the architecture of the network, the model takes the LR image Y as input and outputs the residual. Then, the residual is summed to the input image generating the reconstructed image, and the loss is computed as the Euclidean distance between the reconstructed image and the ground-truth. In terms of network training, this means minimizing the MSE between $R + Y = X$ and $f(Y)$ over the training dataset.

8.3. 3D EXTENSIONS

Both the networks presented allow for 3D extensions. In particular, in [56, 63] a 3D version of SRCNN (SRCNN3D) is implemented by simply replacing 2D convolutional layers with 3D ones and keeping the same architecture.
By performing the same operation for ReCNN it is possible to obtain the 3D network described in [56].

Because of lack of time, it has not been possible to fully explore these 3D networks. Some preliminary results have been obtained, which indicate the potential of such models, but which do not deliver the same performance and image quality as the 2D models. Thus, they were excluded from this report.

9

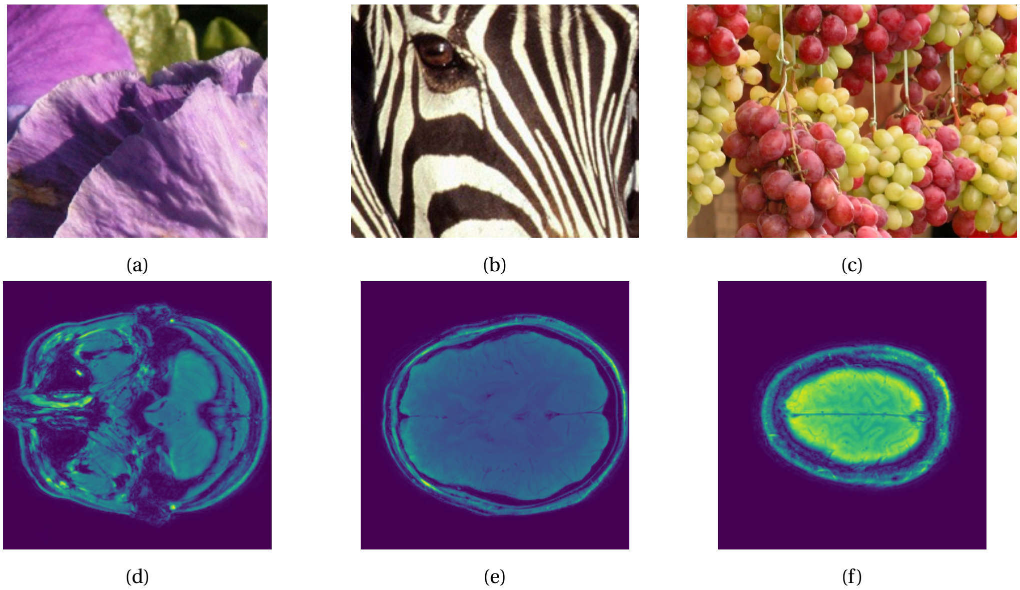
DEEP LEARNING RESULTS

In this chapter we present the 2D results obtained by training the network models mentioned in Chapter 8 on two different datasets: The T91 dataset of natural images and the Kirby21 MRI brain dataset. Both datasets are commonly used for super resolution bench-marking.

9.1. DATASETS AND PREPROCESSING

As already mentioned in Chapter 8, T91 is the dataset used to train SRCNN in [18, 55]. The set consists of 91 natural images of fruit, flowers, animals and people. The images are usually not of the same size and vary greatly in both height and width. T91 is typically used to train convolutional neural networks for classification or image reconstruction problems.

The Kirby21 dataset is made of imaging sessions on 21 healthy volunteers with no history of neurological disease. Imaging modalities include MPRAGE, FLAIR, DTI, resting state fMRI, B0 and B1 field maps, ASL, VASO, quantitative T1 mapping, quantitative T2 mapping, and magnetization transfer imaging. Following [56, 63], our choice was to apply the reconstruction algorithms on T1-weighted MRI images. These MPRAGE (Magnetization Prepared Gradient Echo) data were acquired using a 3-T MR scanner (Achieva, Philips Healthcare, Best, The Netherlands) with a $1.0\times1.0\times1.2$ mm^3 resolution over a FOV of $240\times204\times256$ mm acquired in the sagittal plane. The dataset is freely available at [64]. In Figure 9.1 a few samples for each of the mentioned datasets are shown.


(a) (b) (c)

(d) (e) (f)

Figure 9.1: (a,b,c) samples from the T91 dataset, (d,e,f) samples from the Kirby21 dataset (slices).

9.1.1. T91

It is widely known that there is a close correlation between the amount of training data and the performance of a deep learning model. Indeed, depending on the complexity of the problem analyzed or the amount of parameters we want the model to learn, the data required for training increase. In general, deep learning usually benefits from big data training [18]. Moreover, we have already mentioned in Section 7.3.5 how the size of the training dataset might have a relevant influence on the quality of the training itself, resulting in possible over or underfitting. That is why a dataset made of 91 samples is not enough for SRCNN nor ReCNN to be able to make meaningful and accurate predictions on new data, and data augmentation operations will be needed to enlarge the dataset to an appropriate size.

SRCNN

In order to build the final natural images training dataset for SRCNN, a data augmentation operation is performed. First of all, the images are all converted to gray-scale and normalized between 0 and 1. Then, from each of the samples, overlapping patches of size 33×33 are extracted with a stride of 14. By doing this we are capable of creating a training set made of 18567 samples, against the 91 that we had at the start. Such patches have been shown [18] to be big enough for the network to converge. Furthermore, we consider two more sets: the Set5 and Set14 of natural images. The samples in these sets are very similar to the ones in T91. We will use them for testing and validation respectively. The validation set undergoes an analogous process as the training set, except for a bigger stride of 21, leading to a total of 6419 samples. Conversely, test data is "fed" as a whole to the network after training. Subsequently, each of the patches is downsampled by means of bicubic interpolation and an anisotropic 1D Gaussian blur is applied in the z-direction, as it was done already in Chapter 6. Concerning the value for the standard deviation σ, two versions of each network where created: one trained with $\sigma = 1.3$, and one where a blur with a different value for σ is applied to each patch. Lastly, the blurred downsampled patches are upsampled to the original HR dimensions before they are fed to the network fo training. We call the network trained on this data SRCNN-nat.

Note that, in order to avoid border effects, none of the convolutional layers of SRCNN have padding, which implies that the output of the network will be smaller than the input. In particular it will have size $((f_{\text{sub}} - f_1 - f_2 - f_3 + 3)^2 \times c) = (f_{\text{sub}} - f_1 - f_2 - f_3 + 3)^2$, since we are dealing with gray-scale images ($c = 1$). Here $f_{\text{sub}} = 33$ is the size of the patches and the f_i follow the notation of Section 8.1. To make sure that the MSE between the HR patches and the outputs of the network is evaluated correctly pixel-to-pixel, the former are cropped to the size of the latter.

By performing these operations we are replicating the same pre-processing as in [18], except for the values of the standard deviation and the scaling factor. It could be observed that the choice of a bicubic interpolation for downsampling (and upsampling) differs from the choice made in Chapter 6 of an averaging operator. Nevertheless, by downsampling the HR apple scan in the two different ways, we observed a value of 1 for the SSIM and of $6 \cdot 10^{-16}$ for the MSE when comparing the resulting images (see Figure 9.2). Therefore, we decided to continue with the two different downsampling methods and deem a comparison between the results of the different algorithms valid.

RECNN

Concerning ReCNN-nat, an almost identical procedure to SRCNN-nat was carried on, with only two differences.

First, in ReCNN same zero-padding is applied to every convolutional layer, resulting in an output of the same size as the input. In fact, conversely to what happens with SRCNN, in [60] it is claimed that zero-padding works surprisingly well, and allows for a network which can correctly predict also the pixels near the image boundary. Hence, the HR patches do not need to be cropped.

Second, we consider different values for the strides. ReCNN-nat was trained on two datasets. The first (Dataset 1), with the same strides as SRCNN-nat, the second (Dataset 2) with a smaller stride for the training set. Indeed, our guess was that in order to guarantee convergence of the network, a smaller stride might be in truth needed. The ReCNN is a much more complex and deeper network than SRCNN, and might be prone to overfitting when trained on a not large enough dataset. Hence, for Dataset 2 we set strides of 9 and 21 for the training and validation set respectively.

9.1.2. KIRBY21

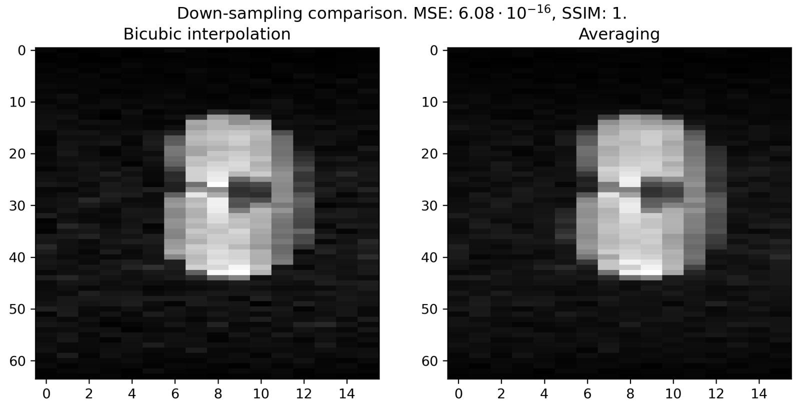


Figure 9.2: Comparison of downsampling operators on the HR apple scan.

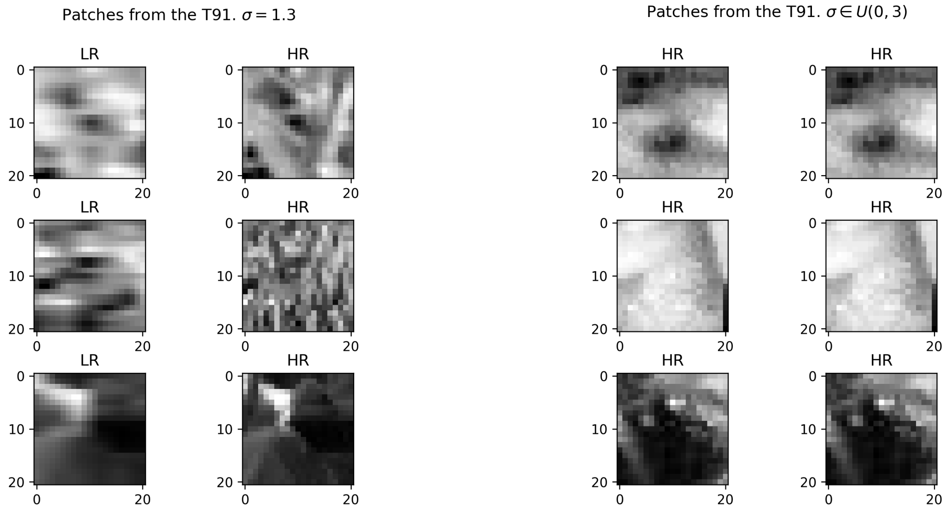


Figure 9.3: (left) LR patches from T91 with corresponding ground-truths for fixed $\sigma = 1.3$. (right) same for multiple blurs.

SRCNN

To the best of our knowledge, the Kirby21 dataset was utilized as a training set for the SRCNN for the first time in [63], where the model is referred to as SRCNN-brain. In the aforementioned paper, a 3D version of the SRCNN was implemented for super resolution on MRI scans, and compared to a network proposed by the authors themselves. In order to obtain a training dataset for the 2D SRCNN-brain (and later be able to make comparisons with the 2D results in Chapter 6), we took 8 T1-weighted scans and sliced them along the axial plane, forming a dataset of 760 samples. Such an amount of data is still insufficient for out network to converge, hence similar data augmentation operations as earlier were performed, by extracting 33×33 patches. In this case, in order to obtain a dataset comparable in size to the T91 one, large values for the strides would have to be taken. As a consequence, many potentially relevant parts of the images would have to be excluded from training because they would not fall into a patch. For this reason, it was decided to, instead of extracting patches with a certain stride, obtain the patches by randomly cropping the images. This way we also managed to have more control on the exact size of the dataset. 25 patches were cropped from each image for the training dataset and 23 for the validation dataset, for a total of 19000 and 6555 final samples respectively. We will refer to this as Dataset 1.

Next, a second bigger dataset was built for comparison. Indeed, the scans of Kirby21 dataset contain more complex details than the images in T91, and it is common knowledge when dealing with deep learning models that larger dataset add diversity and reduce the risk of overfitting. In this case, extracting more patches from each image increases the chance of more relevant features to be represented in the dataset. For Dataset 2, 56 and 50 patches were cropped for each image of the training and validation set respectively, for a total of

(42560, 14250) samples. Again, all the data are normalized between 0 and 1 before being fed to the network.

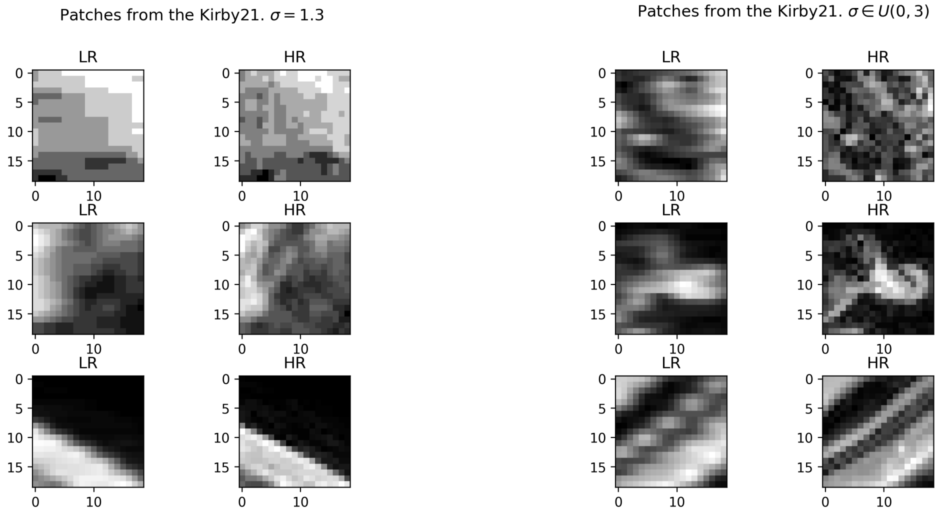

Patches from the Kirby21. $\sigma = 1.3$
Patches from the Kirby21. $\sigma \in U(0,3)$
LR
HR
0
5
10
15
0
10

Figure 9.4: (left) LR patches from slices of the Kirby21 dataset with corresponding ground-truths for fixed $\sigma = 1.3$. (right) same for multiple blurs.

ReCNN

Again, as for T91, the pre-processing of Kirby21 for ReCNN is almost identical to the one for SRCNN-brain. Also this time two datasets are considered: the first equal in size to Dataset 1 for SRCNN-brain, the second having a larger training dataset. Dataset 2 was obtained by cropping each image in the training and validation set 70 and 23 times respectively. These cropping factors were found experimentally to yield very good results. In Table 9.1 the sizes of the different datasets are listed.

Table 9.1: Table of the sizes of the datasets used for training and validation.

SRCNN-nat		
	Train	**Validation**
Dataset 1	18567	6419
SRCNN-brain		
	Train	**Validation**
Dataset 1	19000	6555
Dataset 2	42560	14259
ReCNN-nat		
	Train	**Validation**
Dataset 1	18567	6419
Dataset 2	41725	6419
ReCNN-brain		
	Train	**Validation**
Dataset 1	19000	6555
Dataset 2	53200	6555

9.2. Validation of the Models

As already mentioned in Section 9.1, we used a training dataset and a validation dataset (hold-out split), following the approach of [18]. In Section 7.3.4 we have explained the importance of correctly evaluating a network and guaranteeing unbiased predictions, and talked about the hold-out method as well as k-fold cross validation. The former is usually considered a naïve approach, while the latter is usually the preferred approach. Nevertheless, performing k-fold cross validation with $k = 5$ or 10 (or even 20!) is extremely computationally expensive, and can become even more so depending on the complexity of the network. Indeed, if we consider a network which usually takes around 10 minutes to train, a 5-fold cross-validation will require around 1 hour (50 minutes for the cross-validation and 10 to train the final model). If we contextually perform grid-search with 10 different hyperparameter combinations, the amount of time needed rapidly increases to around 10 hours.

While it would have probably been possible to follow this kind of approach with a shallow network such as SRCNN-nat, in general it is not always so due to hardware and time limitations. Furthermore, in choosing a hold-out split we chose validation sets which we know to be representative of the training data.

Therefore, this kind of split was used to select the optimal hyperparameters for our networks. Then, in order to assess them to the best of our possibilities, k-fold cross validation was performed on the final selected models in order to validate them.

To conclude, the validated models were finalized by training one more time with a hold-out split.

9.3. SRCNN-NAT 2D

The first network which was implemented is SRCNN-nat with fixed standard deviation $\sigma = 1.3$. We varied the number of filters of the convolutional layers and the width of the kernels within the architecture of the SRCNN and compared the results. Furthermore, other hyperparameters such as the optimizer, the learning rate and the weight initialization have been investigated.

9.3.1. MODEL SET-UP

The baseline architecture proposed in [18] is a 2-layer CNN with respectively 64, 32 and 1 filter with kernel sizes 9×9, 1×1 and 5×5. We will refer to this structure as 9-1-5.

The first model was implemented with these settings and a batch size of 256. In order to optimize the remaining hyperparameters, grid-search was performed: a list of possible values for the different hyperparameters was made and all the possible combinations were tried in order to determine the best choices (see Table 9.2).

Table 9.2: Hyperparameters used for different configurations of SRCNN-nat.

SRCNN-nat	
Batch size	256
Epochs	[1000,2000]
Filters	[128-64-1,63-32-1]
Kernels	[9-1-5,9-3-5]
Optimizer	[Adam, SGD]
Learning rate SGD	[0.0001,0.001,0.01]
Momentum SGD	0.9
Learning rate Adam	[0.0001,0.001]
Metric	MSE
Weight initialization	[Xavier, He, RandomNormal($\mu = 0, \sigma = 0.001$)]

We could then observe how for the final model both the loss and the PSNR were in good accordance with the results from the cross validation shown in Table 9.4. This allows us to conclude that our results do not depend on the validation set chosen.

Note that we chose a value $k = 4$ for the k-fold cross-validation in order to keep the same training-to-validation set proportion (75:25) as we have when picking Set14 as a validation set in a hold-out split situation. This will allow us to make a more fair comparison between the different values obtained.

The first observation we were able to make when performing grid-search is that 1000 epochs were always enough to reach convergence, and 2000 might even be detrimental for training (see Figure 9.5). Hence, any value above 1000 for the number of epochs was discarded.

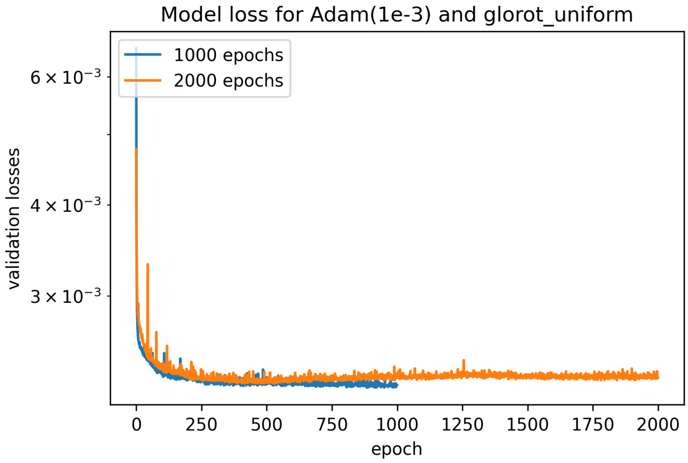


Figure 9.5: Losses plot for SRCNN-nat 64-32-1, 9-1-5.

Secondly, we were able to observe that Adam performs significantly better compared to SGD, as it is evident from Figure 9.6. Concerning the learning rate for Adam, the training curves of 0.0001 and 0.001 are very similar, with 0.0001 yielding a slightly more stable behavior.
Next, we focused on the weight initializers and noticed the clearly better learning curve given by Xavier initialization (see Figure 9.7) in terms of both rapid convergence and (lack of) overfitting. It has been proved [43] that He initialization is more suited for ReLU activation functions than Xavier, but we will see that our network with Xavier initialization performs well and it is able to generalize to new, unseen data.

Note that we will often refer to Xavier and He initialization as glorot_uniform and he_normal respectively, since these are the names of the Keras functions to call such initializers.

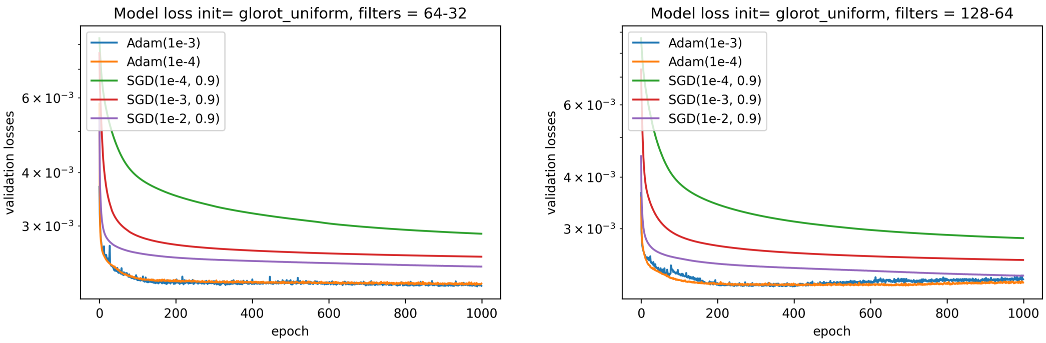


Figure 9.6: Losses plot for SRCNN-nat. Adding more filters does not show any benefit in training and leads to overfitting the dataset.

Lastly, we explored the possibility of a bigger kernel in the central layer of the network. In [18] it was already observed how increasing the filter size could significantly improve the performance of the network and how utilizing neighborhood information in the mapping stage might be beneficial. In Figure 9.8 we show how in our setting we do not observe a marked improvement using a wider kernel.

Increasing the complexity of the network is known to be a possible cause of overfitting, which is what we observe for the 9-3-5 configuration. The loss is slightly lower for the wider network and before overfitting, the PSNR slightly higher, but when testing on the test data there is no actual perceivable difference. Moreover, a wider network comes with a computational cost, which might not be in favor of a good trade-off between performance and speed.

Finally, in order to prevent overfitting and obtain a possibly better network, a possible option could be

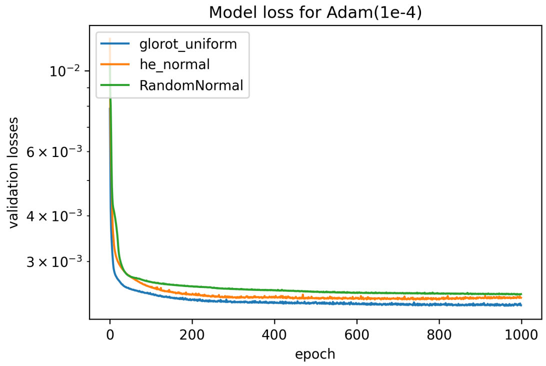


Figure 9.7: Losses plot for SRCNN-nat with optimizer Adam(0.0001). Glorot Uniform outperforms the other initializers.

to increase the number of samples in the dataset. We have observed how 1000 epochs for 9-1-5 SRCNN take approximately 6.5 minutes on a High-Ram Google Colab Pro GPU. Augmenting the data by 10,000 more samples means doubling the training time. After performing several tests in this regard, we were able to conclude that adding more data did not significantly boost the training, while increasing the computational time significantly. Hence, this option was discarded.
From these observations we felt confident in choosing the configuration summarized in Table 9.3 as the final one for SRCNN-nat. It should be noticed that, by performing grid-search to tune the hyperparameters, we have arrived to the same model which in [18] is considered the best trade-off between speed and performance.

Table 9.3: Hyperparameters used for the final configuration of SRCNN-nat.

SRCNN-nat	
Batch size	256
Epochs	1000
Filters	64-32-1
Kernels	9-1-5
Optimizer	Adam
Learning rate	0.0001
Metric	MSE
Weight initialization	Xavier

Table 9.4: 4-fold cross validation for SRCNN-nat with final configuration.

	Fold 1	Fold 2	Fold 3	Fold 4
PSNR	27.33	27.28	27.26	27.26
MSE	$1.85 \cdot 10^{-3}$	$1.87 \cdot 10^{-3}$	$1.88 \cdot 10^{-3}$	$1.88 \cdot 10^{-3}$
Average on folds				
PSNR	27.28			
MSE	$1.87 \cdot 10^{-3}$			

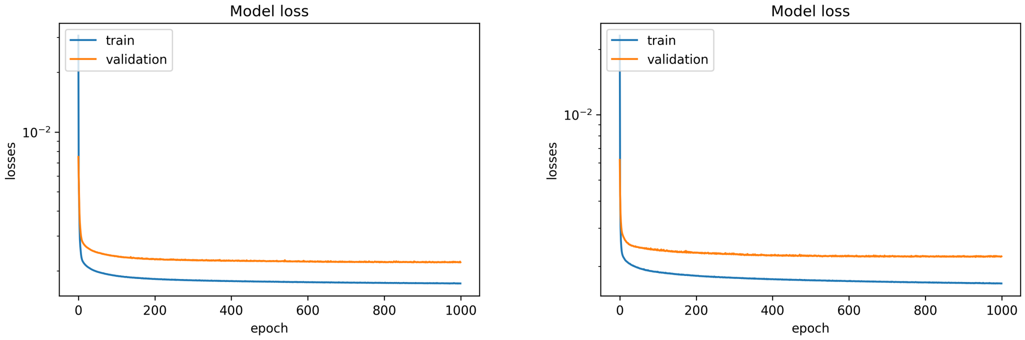


Figure 9.8: Loss plots for SRCNN-nat 9-1-5 (left) and 9-3-5 (right).

9.3.2. RESULTS: TEST DATA & LOW-FIELD DATA

This model was evaluated on a test image from the Set5 dataset, and the result is shown in Figure 9.9 together with the PSNR and the SSIM value measured with respect to the ground-truth image. We use a ModelCheckpoint callback. A callback is a Python object that can perform actions at various stages of training, and in this case it only saves the updated weights of the network if there is an improvement in the validation loss. This way, if the network started overfitting, the affected weights would not be saved. This callback will be used from now on in all our tests.

Clearly we can observe a significant improvement in the quality of the image between simply interpolating and reconstructing the image by feeding it to the network, as testified by both the metrics. The edges are much sharper and the blurring kept to a minimum. There are quite some details missing from the SRCNN reconstruction compared to the ground-truth, but considered how simple our network is and that the image underwent a substantial degradation we are satisfied with the results.

Encouraged by the outcome in Figure 9.9, it was decided to try and apply the SRCNN-nat to the LR apple scans shown in Chapter 5. Indeed, we had the intuition that a network able to reconstruct natural images, including fruit, might be an interesting starting point for the reconstruction of our apple scans. In Figure 9.10 we show the comparison between the bicubic interpolation of the LR images and their SR reconstruction by SRCNN-nat with fixed blur.

It appears evident how the networks managed to deblur and sharpen the edges of the apple substantially. We deem this an impressive result, especially considering that the network was trained with data that differ quite noticeably from the low-field scans.

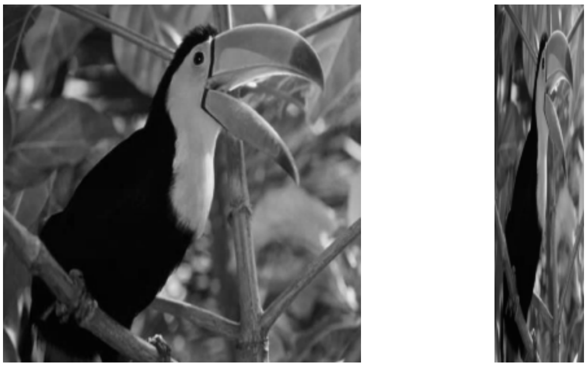
However, as we know from the previous chapters, these networks perform single image super resolution, which does not help when dealing with smaller fields of view. Indeed, as evident from the figures in Chapter 5, none of the LR apple scans manages to capture all the necessary information to make an accurate enough prediction on the reconstructed apple.

In order to combine the benefits of MISR with the potential of deep learning SISR, and hopefully improve the results from the standard SR, it was decided to first combine all the LR scans into a single LR one, and then perform the reconstruction by feeding the LR image to the network. The combined LR scan is obtained as described in Section 6.1.

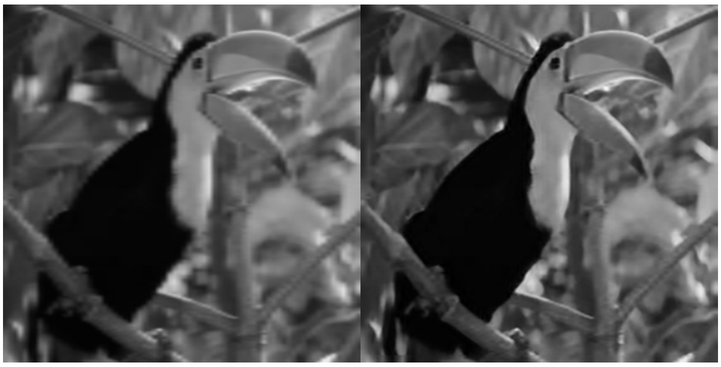
Another option was considered, namely to, instead of enhancing the quality of the combined scan, to combine the 4 enhanced scans shown in Figure 9.10. Nevertheless, the former has proved to be a much better approach. For completeness, the results are shown in Appendix C. In Figure 9.11 a comparison between standard SR, SRCNN, bicubic interpolation and HR apple scan is shown. Regarding standard SR, we decided to compare the deep learning reconstructions with the one in Figure 6.11d, which in Chapter 6 was deemed being superior to the results yielded by Tikhonov regularization.

From a visual point of view, the two reconstructions appear clearly different. The central part of the scan is better preserved and reconstructed slightly better in 9.11b, but the contours on the right side of 9.11c resemble the ones of the HR scan more than 9.11b, where the blurring of the bicubic interpolation 9.11a is accentuated and incorporated as a part of the apple itself. Moreover, the SRCNN-nat reconstructions appear much more realistic than the ones of Chapter 6, resembling the color and “texture” of MRI scans better. The transitions in pixel intensities are smooth, opposed to the staircasing effect caused by the total variation regularization.

A very interesting fact to notice is that the value of σ used during training has a tremendous impact on



(a) Ground-truth. (b) Blurred, downsampled image.



(c) Bicubic interpolation.
PSNR: 29.19, SSIM: 0.91

(d) SRCNN-nat $\sigma = 1.3$.
PSNR: 32.36, SSIM: 0.94

Figure 9.9: Comparison of different reconstructions of a bird from Set5 downsampled and blurred by Gaussian blur with $\sigma = 1.3$.

the quality of the reconstruction. In order to prove this point, we trained SRCNN-nat with the exact same hyperparameters, only changing the value for the standard deviation during the pre-processing. Apparently, a network trained on a smaller Gaussian blur is not able to properly compensate for the blur in the LR scan. The results in Figure 9.12 confirm that $\sigma = 1.3$ is a very good choice for modeling the blur induced by the scanner, and that our analysis in Chapter 6 led to a correct estimation of the standard deviation value.

After training on a fixed blur, it was deemed interesting to train the network with different blurs for each patch. We wanted to investigate whether it would have been possible to improve the result even further by training the network to handle different blurs. Also, such a network would have a wider application and it would be a much more powerful reconstruction tool than both a fixed blur network and standard super resolution. Everything else in the structure of the network was kept the same. In the remaining part of the section, we show the predictions of the multi-blur version of SRCNN-nat when we draw σ from $U(0,3)$.

From a visual assessment there is no noticeable difference in the two reconstructions of the birds. There is though a slight difference in the PSNR, higher for the fixed blur network by 0.44 dB. This result is not surprising, since SRCNN-nat with fixed σ specialized in exactly the kind of blur applied to the bird in Figure 9.13b. When instead we consider the same image, but blur it with a standard deviation of 0.5 or 2, in both cases the multi-blur SRCNN-nat performs better (see Figure 9.14).

If we look at the results on the apple scans we can notice some variations, both in the pixel intensities and in the reconstruction of the overall shape of the apple. In Figure 9.16c the highest intensity parts are smoothed out, and the edges are not as sharp as in 9.11c. We believe that the SRCNN-nat with fixed σ offers a sharper, overall better reconstruction of the apple scan. Following the same reasoning as before, we argue that these last results are in accordance with our estimation of the blur.

For the sake of a more objective comparison, registration has been performed on the HR scan in order to match the reconstructed images, analogously to what was done already in Chapter 6. Then, both the PSNR and the SSIM were computed between the bicubic interpolation, the reconstructions and the HR scan. Nevertheless, the HR scan is not actually the ground truth corresponding to the LR scans. That is why in this case the metrics are not a reliable evaluation tool, and we preferred focusing on a visual assessment of the

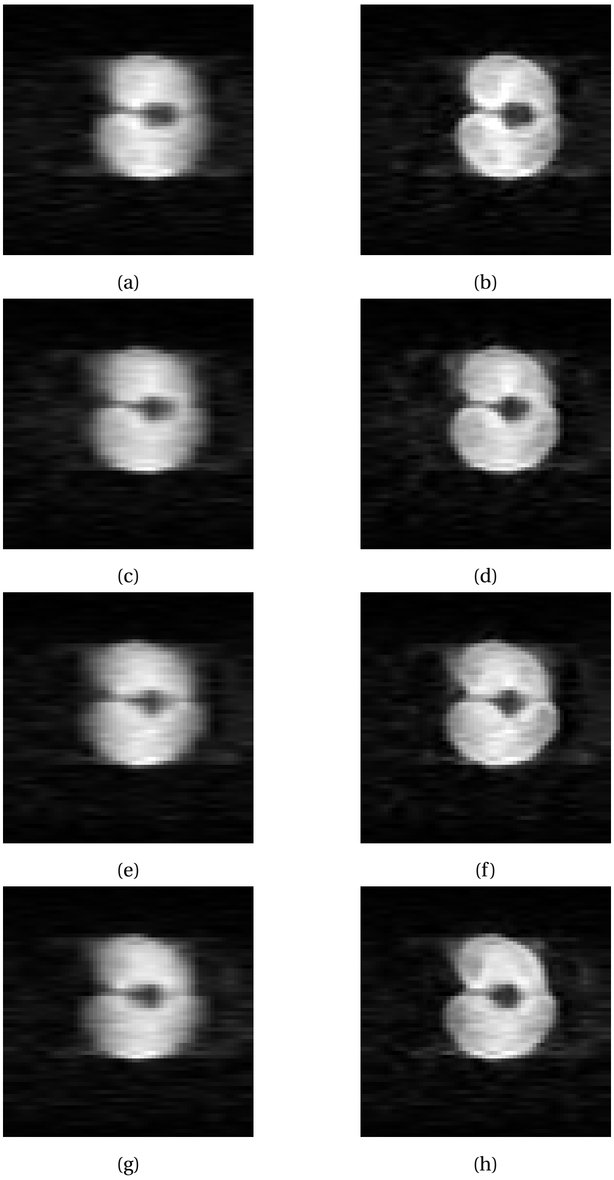


Figure 9.10: (a,c,e,g) bicubic interpolation of LR apple scans, (b,d,f,h) SR reconstruction by SRCNN with fixed blur.

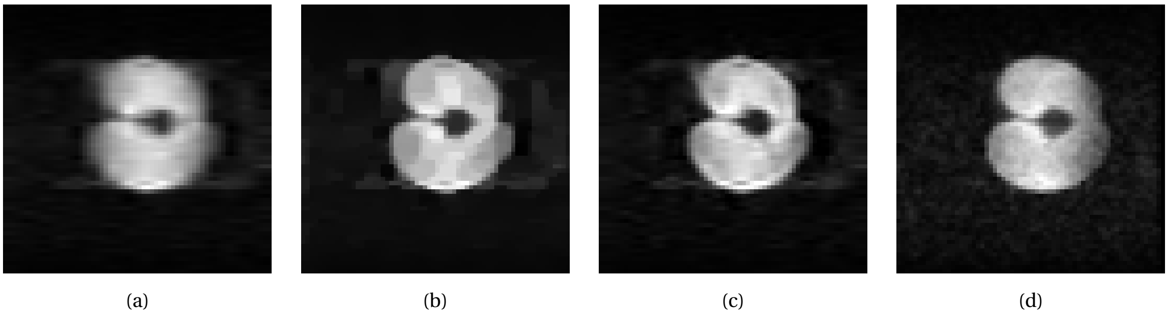
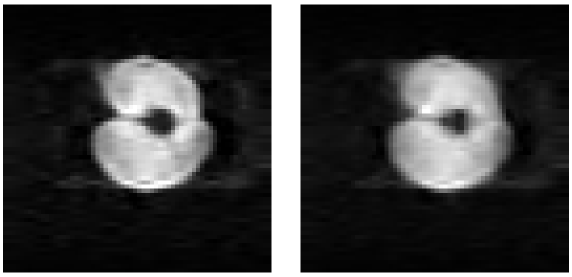


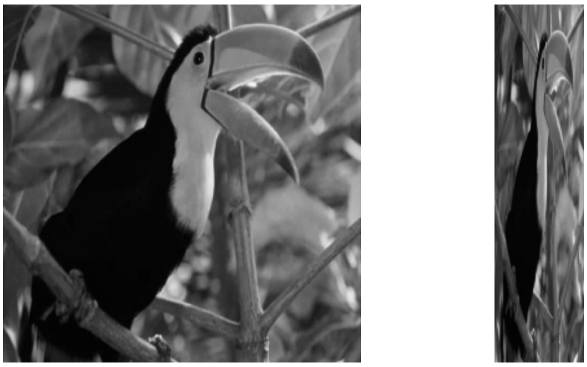
Figure 9.11: (a) bicubic interpolation of LR apple scan, (b) SR reconstruction by standard SR, (c) SR reconstruction by SRCNN-nat with fixed blur $\sigma = 1.3$, (d) HR apple scan.

reconstructions. For completeness, we include the values for these metric in Appendix B.

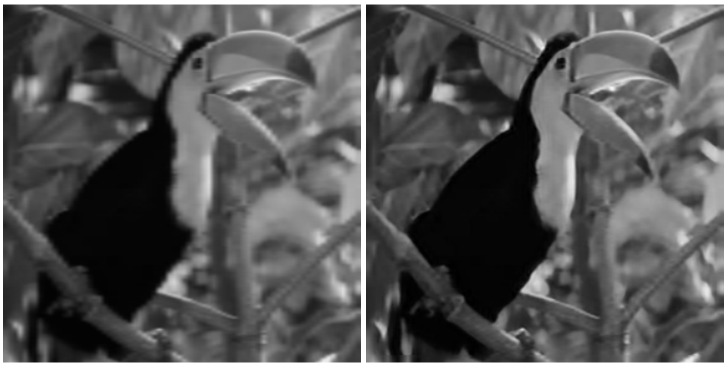


(a) $\sigma = 1.3$ (b) $\sigma = 0.5$

Figure 9.12: Comparison between SRCNN-nat reconstructions of the LR apple scan with fixed $\sigma = 1.3$ and $\sigma = 0.5$.



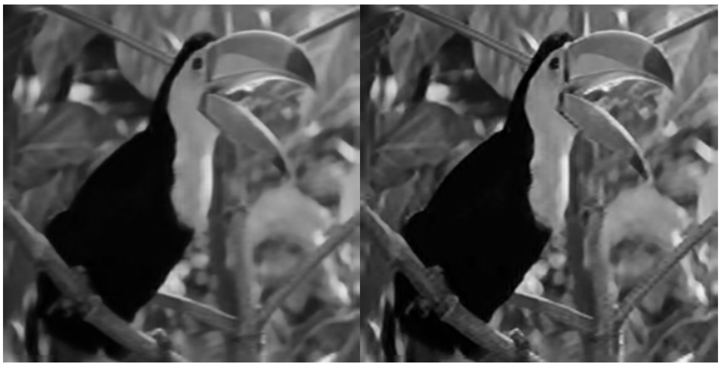
(a) Ground-truth. (b) Blurred, downsampled image.



(c) Bicubic interpolation.
PSNR: 29.19, SSIM: 0.91

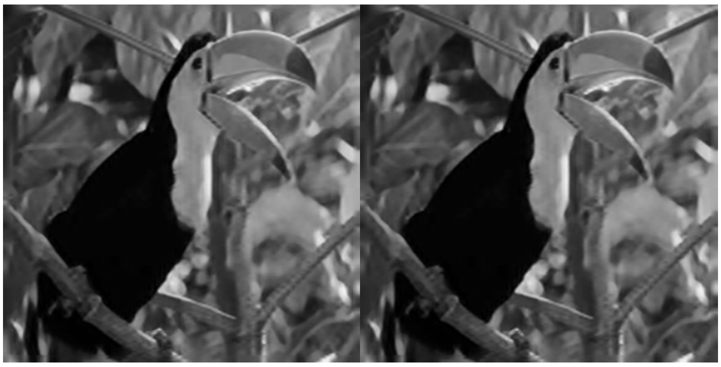
(d) SRCNN-nat $\sigma \in U(0,3)$.
PSNR: 31.92, SSIM: 0.93

Figure 9.13: The reconstruction process of a bird from Set5 downsampled and blurred by Gaussian blur with $\sigma = 1.3$.



(a) SRCNN-nat $\sigma \in U(0,3)$
PSNR: 31.45, SSIM: 0.93.

(b) SRCNN-nat $\sigma = 1.3$
PSNR: 30.92, SSIM: 0.92.



(c) SRCNN-nat $\sigma \in U(0,3)$
PSNR: 30.52, SSIM: 0.91.

(d) SRCNN-nat $\sigma = 1.3$
PSNR: 30.44, SSIM: 0.91.

Figure 9.14: (a,b) bird from Set5 reconstructed from Gaussian blur with $\sigma = 2$ (c,d) bird from Set5 reconstructed from Gaussian blur with $\sigma = 0.5$.

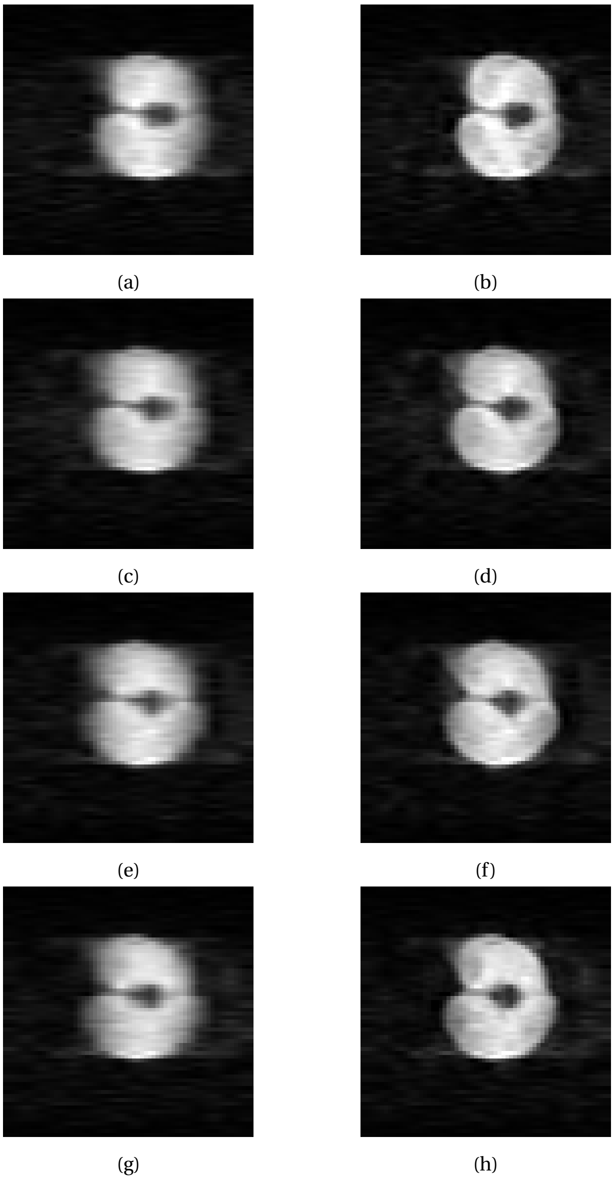

(a) (b) (c) (d) (e) (f) (g) (h)

Figure 9.15: (a,c,e,g) bicubic interpolation of LR apple scans, (b,d,f,h) SR reconstruction by SRCNN-nat with multiple blurs.

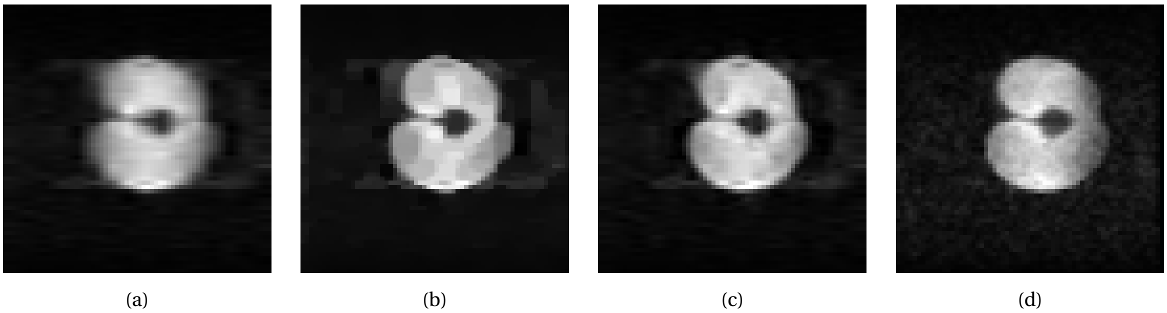

(a) (b) (c) (d)

Figure 9.16: (a) bicubic interpolation of LR apple scan, (b) SR reconstruction by standard SR, (c) SR reconstruction by SRCNN-nat with multiple blurs, (d) HR apple scan.

9.4. SRCNN-BRAIN 2D

As explained in Chapter 1, the final purpose of these SR techniques is to be applied to actual low-field MRI scans of brains. Therefore, after SRCNN-nat it was decided to implement SRCNN-brain by training SRCNN on the Kirby21 dataset with fixed standard deviation $\sigma = 1.3$. As already anticipated in Section 9.1.2, two datasets were considered for the SRCNN-brain: Dataset 1 and 2.

9.4.1. MODEL SET-UP

Analogously to what has been done in Section 9.3 a grid-search was performed on the smaller dataset in order to identify the optimal hyperparameters for the network. Then, after selecting the hyperparameters, the results obtained by training on the two datasets will be compared.
The same configurations as SRCNN-nat were tried, except for the number of epochs, which was kept to 1000.

Again, Adam proved to be the best optimizer for our situation, and both 0.001 and 0.0001 work well as values for the learning rate. From Figure 9.17 we conclude that, also in this case, among the weight initializers RandomNormal gives the worst performance. In Figure 9.18 instead, we show the plots used to select the best configurations for both Adam(0.001) and Adam(0.0001). We found that the model with smaller learning rate performs slightly better when more filters and He initialization are used, while less filter and Xavier initialization seem to be preferable when picking a higher learning rate. After comparing these two configurations (see Figure 9.18a) it was decided to pick a learning rate of 0.0001 and more filters for the final model. We speculate that the choice of considering more filters in the first two layers might be beneficial for the training on such a more complex dataset.

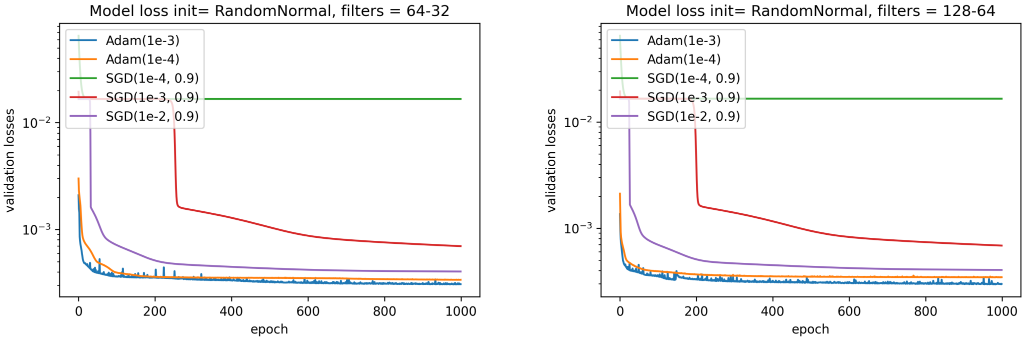
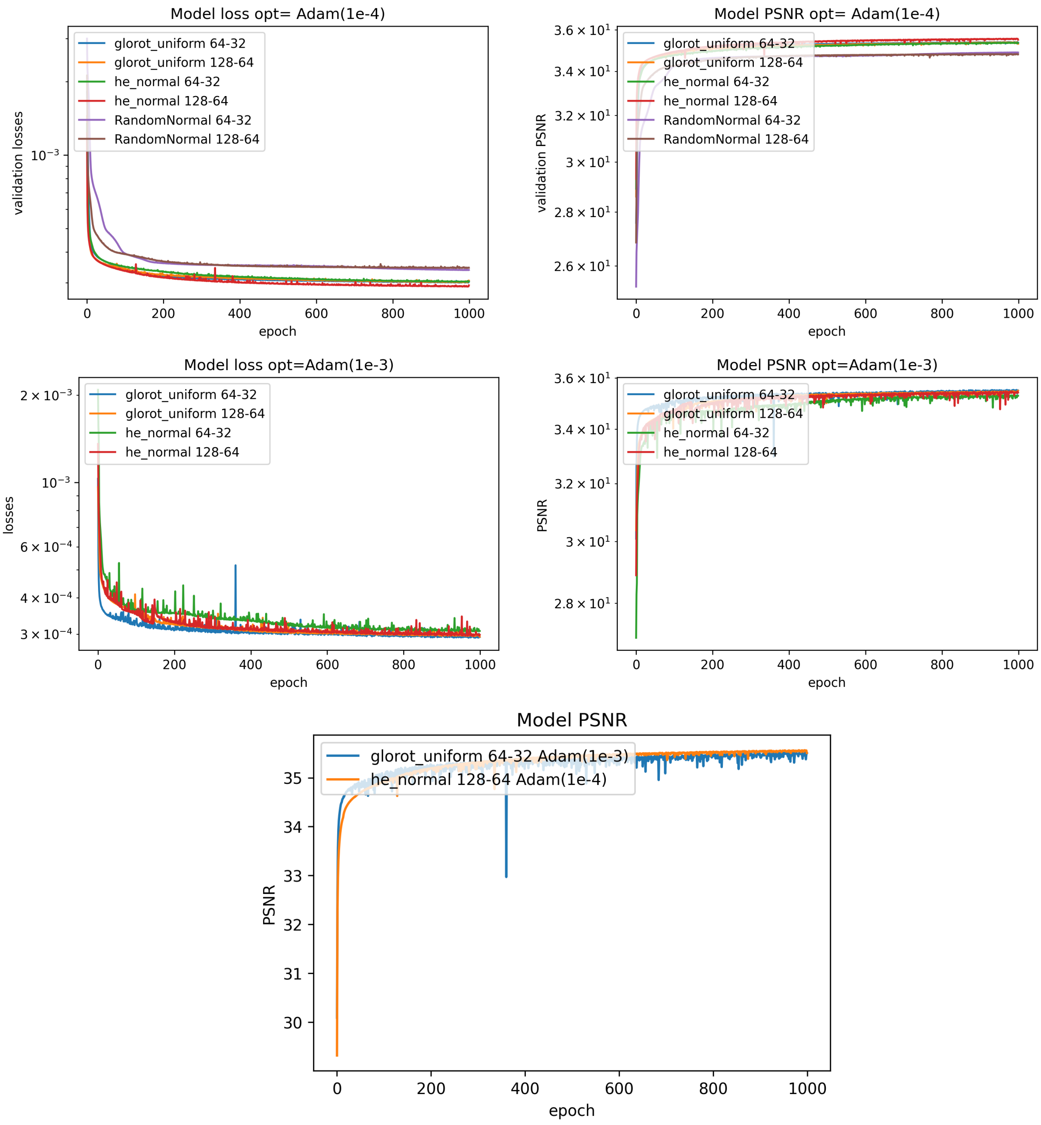


Figure 9.17: Loss plots for SRCNN-brain when picking RandomNormal as weight initializer. Adam shows superior performance as optimizer in this case.

After choosing the optimizer, the learning rate, the weight initialization and the filters, we explored the possibility of having a bigger kernel in the second convolutional layer, as for SRCNN-nat. We compared SRCNN-brain 9-1-5 and 9-3-5 with 128-64 filters on the first two layers. We were interested in seeing whether increasing the complexity of the network in such a way would give the model the ability to reconstruct finer details. In Figure 9.19 the losses and PSNR for both architectures are shown.

In Figures 9.19d we do not observe any increase in the PSNR, and only a slight decrease in loss. Nonetheless, it could be easily argued that the dataset used is too small for such a network not to overfit the data, and that adding a significant number of training samples might potentially lead to a much better model. Therefore, both configurations were trained again on Dataset 2. The comparison between the losses on the two datasets is shown in Figure 9.20.

Training on a larger dataset led to a marginally lower loss and higher PSNR and lessened the overfitting to some extent, but it took around 22 minutes to train the network, against the 11 minutes on the smaller dataset. Everything considered, the configuration in Table 9.5 seemed to be the best choice for SRCNN-brain. The selected model was then validated by 4-fold cross-validation (see Table 9.6).

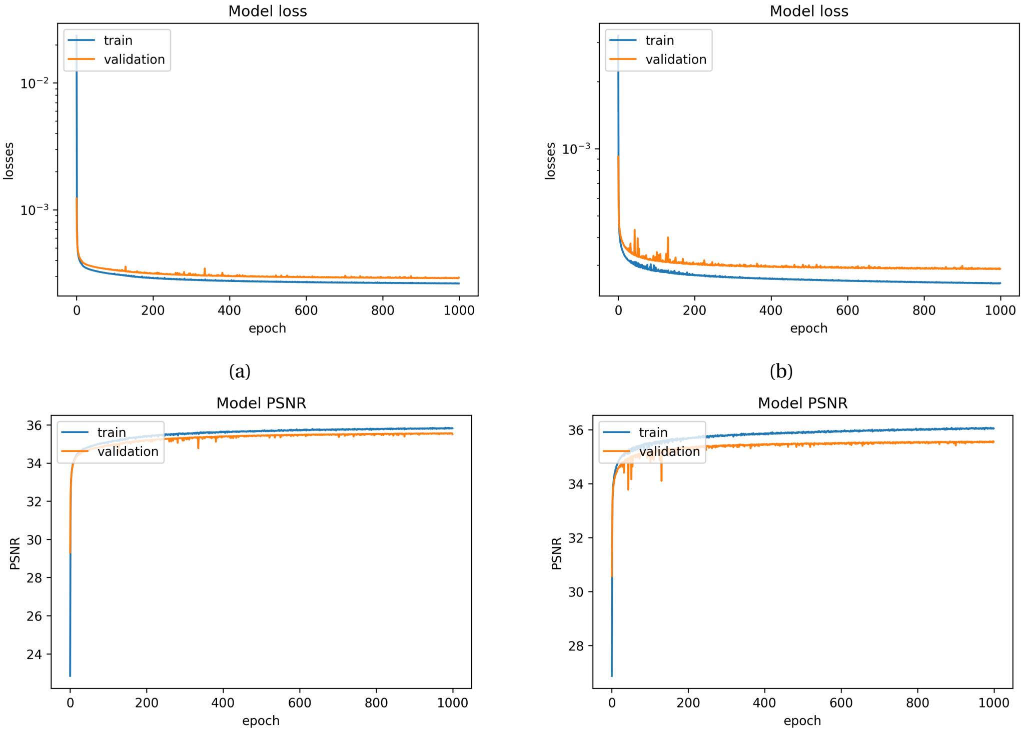


(a)

Figure 9.18: Loss plots for various configurations.

Table 9.5: Hyperparameters used for the final configuration of SRCNN-brain.

SRCNN-brain	
Batch size	256
Epochs	1000
Filters	128-64-1
Kernels	9-1-5
Optimizer	Adam
Learning rate	0.0001
Metric	MSE
Weight initialization	He
Dataset	Dataset 1



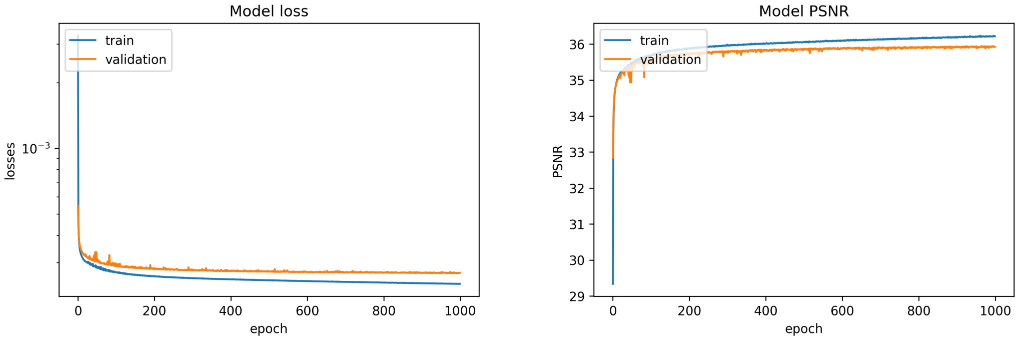
(a) (b)

(c) (d)

Figure 9.19: Losses and PSNR plot for SRCNN-brain 9-1-5 (left) and 9-3-5 (right) and filters 128-64 on Dataset 1.

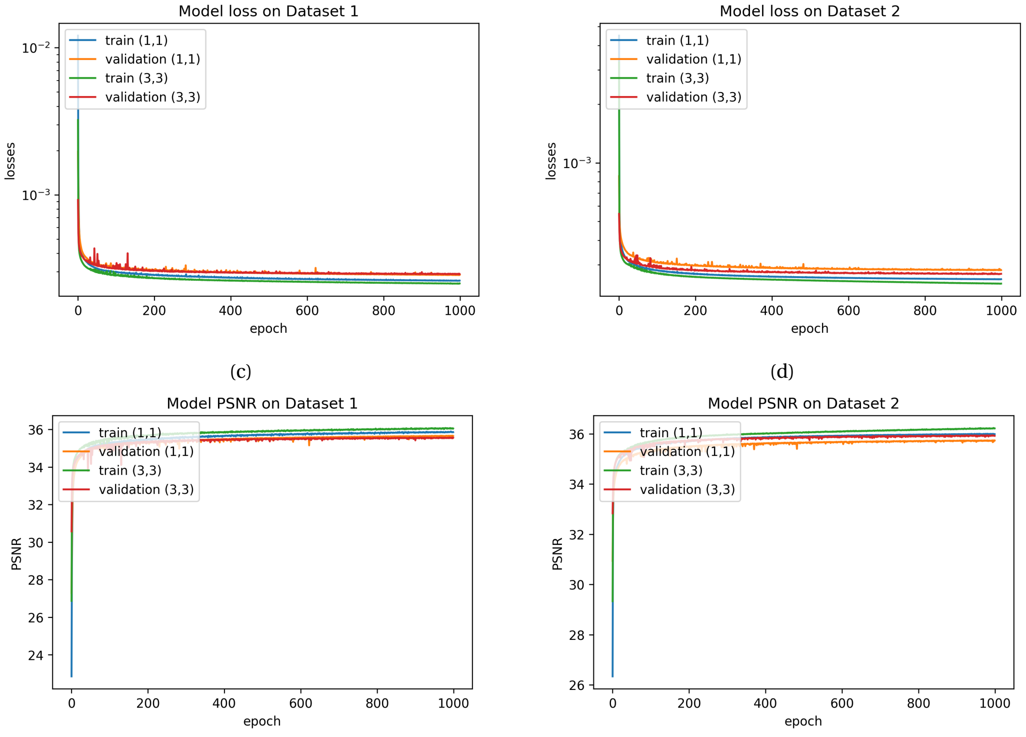
Table 9.6: 4-fold cross validation for SRCNN-brain with final configuration.

	Fold 1	**Fold 2**	**Fold 3**	**Fold 4**
PSNR	35.65	35.64	35.64	35.67
MSE	$2.72 \cdot 10^{-4}$	$2.73 \cdot 10^{-4}$	$2.73 \cdot 10^{-4}$	$2.71 \cdot 10^{-4}$
Average on folds				
PSNR	35.65			
MSE	$2.72 \cdot 10^{-4}$			



(a) 128-64-1, 9-3-5 on Dataset 2.

(b) 128-64-1, 9-3-5 on Dataset 2.



(e)

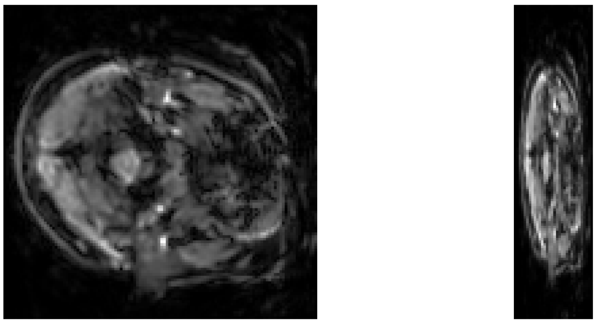
(f)

Figure 9.20: Losses and PSNR for different models.

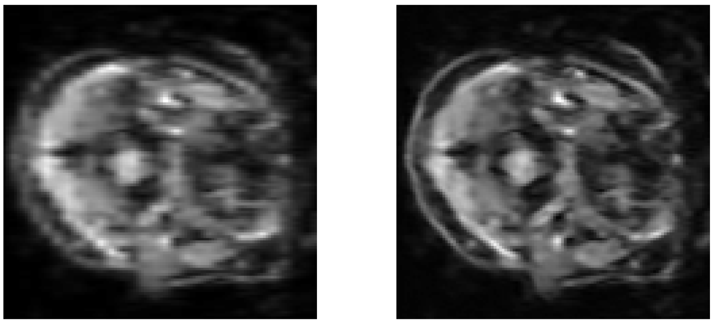
9.4.2. RESULTS: TEST DATA & LOW-FIELD DATA

The finalized network was first tested on a slice of a brain scan from the Kirby21 dataset which the network had never seen during training (see Figure 9.21). Then, SRCNN-brain was tested on the low-field apple scans following the same exact procedure as for SRCNN-nat.

Undoubtedly, the improvement in the quality of the reconstruction compared to the simple interpolation is marked. SRCNN-brain provides a much sharper reconstruction, despite some flaws in the reconstruction of the exact shape. Also, many textural and finer details are lost in the process, but this is perfectly reasonable considering the high level of degradation that the image underwent. Moreover, both metrics are in agreement with the visual perception of the quality of the images.



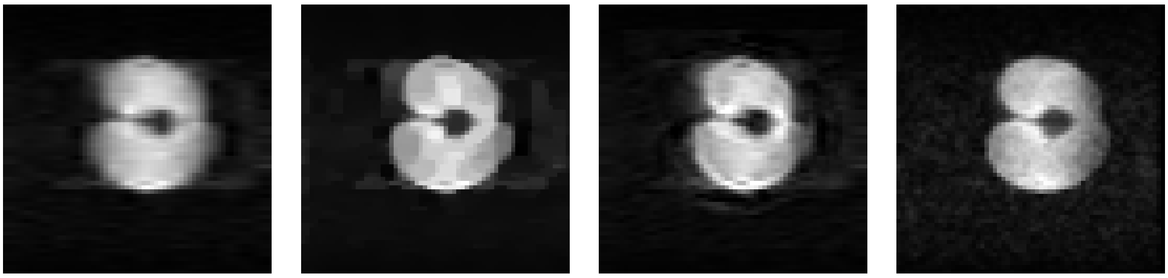
(a) Ground-truth. (b) Blurred, downsampled image.



(c) Bicubic interpolation.
PSNR: 29.36, SSIM: 0.84

(d) SRCNN-brain with $\sigma = 1.3$
PSNR: 31.37, SSIM: 0.88

Figure 9.21: The reconstruction process of a slice of a Kirby21 scan, downsampled and blurred by Gaussian blur with $\sigma = 1.3$.



(a) (b) (c) (d)

Figure 9.22: (a) bicubic interpolation of LR apple scan, (b) SR reconstruction by standard SR, (c) SR reconstruction by SRCNN-brain with fixed blur $\sigma = 1.3$, (d) HR apple scan.

In Figures 9.23 and 9.22 we show the reconstructions for each LR apple scan and for the combined LR respectively. If we compare these predictions with the ones in Figure 9.10 it is apparent which network yields the best reconstructions. SRCNN-brain does indeed improve the quality of the scans considerably, but it also introduces artifacts. The network fails to properly deblur the scans and actually introduces noise on the bottom part of the images and on the edges of the apple. Moreover, the shape itself is not as well preserved as in the case of SRCNN-nat.

It is unclear what should the explanation for this facts be. One possibility is that the dataset used is simply not appropriate for the reconstruction of apple scans. Indeed, the network is trained correctly (as testified by the results in Figures 9.19 and Table 9.6), and yet it fails to give comparable results to the ones of SRCNN-nat. We will see in Section 9.6 that a comparison between SRCNN-brain and ReCNN-brain will allows us to give a more complete answer to this open question.

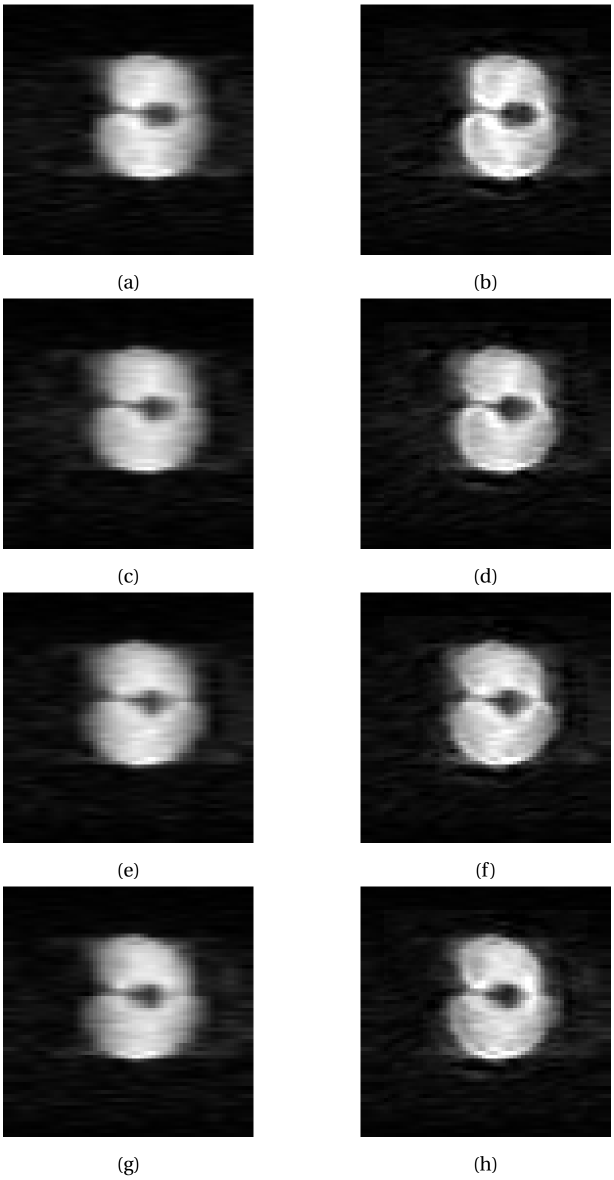
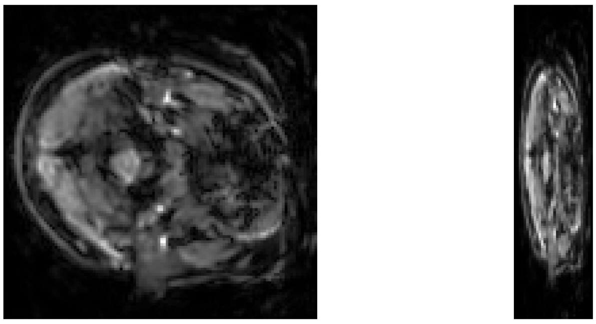


Figure 9.23: (a,c,e,g) bicubic interpolation of LR apple scans, (b,d,f,h) SR reconstruction by SRCNN-brain with fixed blur.

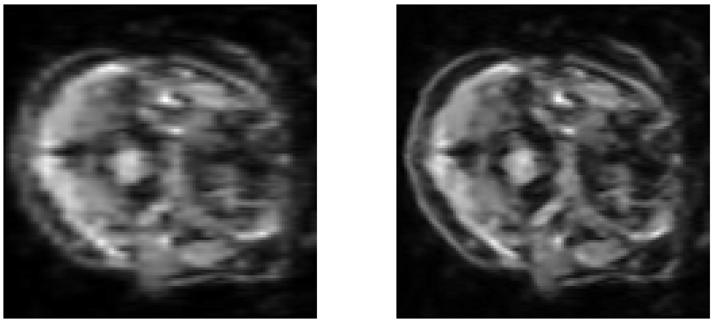
Finally, as for SRCNN-nat, we analyzed the results obtained by training SRCNN-brain on a training set in which each patch was blurred with a different standard deviation. Again, we range for σ was set between 0 and 3.

From a visual point of view, it is hard to tell the difference between reconstruction 9.21d and 9.24d, which is also reflected in the SSIM values. However, the PSNR for 9.21d is slightly higher. This is fully in line with observation already made in the previous section. Indeed, when tested on a specific kind of blur, a network trained on that specific blur factor will always give superior predictions than a model trained on various blurs. Nonetheless, as demonstrated for SRCNN-nat, the multi-blur network is a better choice when having to reconstruct images with unknown blurs, since it overall gives a more than satisfactory performance on various blurs, surpassing the SRCNN-brain trained with fixed σ (see Figure 9.25).



(a) Ground-truth.

(b) Blurred, downsampled image.

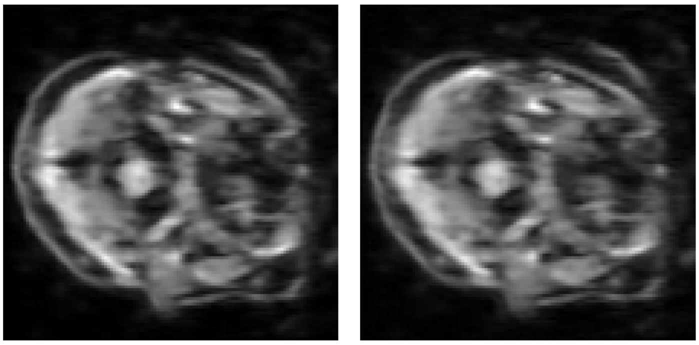


(c) Bicubic interpolation.
PSNR: 29.36, SSIM: 0.84

(d) SRCNN-brain with $\sigma = 1.3$
PSNR: 31.06, SSIM: 0.88

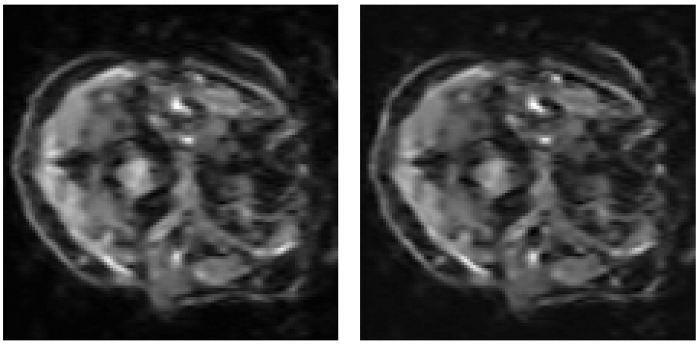
Figure 9.24: The reconstruction process of a slice of a Kirby21 scan, downsampled and blurred by Gaussian blur with $\sigma \in U(0,3)$.

Concerning the apple scans, SRCNN-brain with multiple blurs gives a blurry, unclear prediction



(a) SRCNN-brain $\sigma \in U(0,3)$
PSNR: 30.63, SSIM: 0.87.

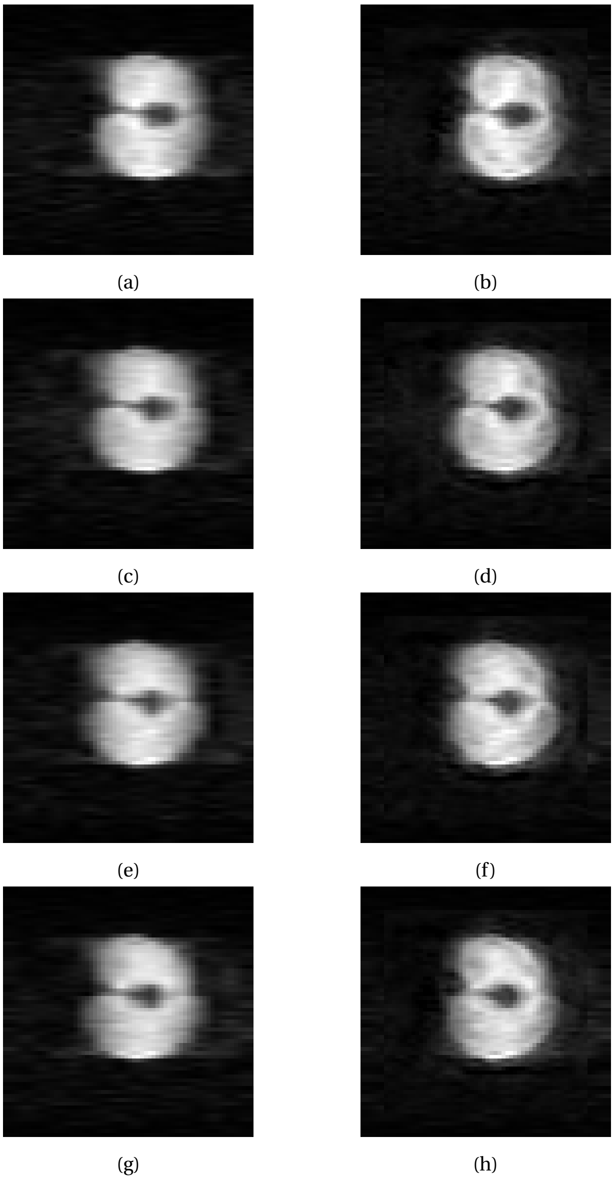
(b) SRCNN-brain $\sigma = 1.3$
PSNR: 30.49, SSIM: 0.86.



(c) SRCNN-brain $\sigma \in U(0,3)$
PSNR: 30.26, SSIM: 0.86.

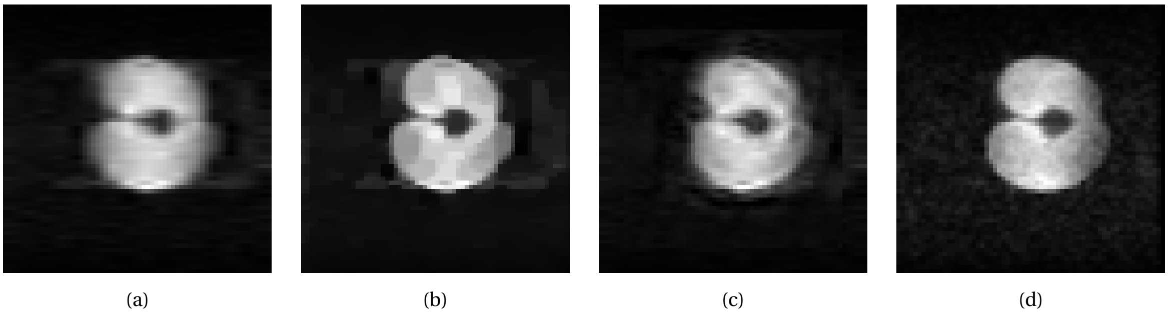
(d) SRCNN-brain $\sigma = 1.3$
PSNR: 29.64, SSIM: 0.85.

Figure 9.25: (a,b) slice from Kirby21 scan reconstructed from Gaussian blur with $\sigma = 2$ (c,d) slice from Kirby21 scan reconstructed from Gaussian blur with $\sigma = 0.5$.



(a) (b) (c) (d) (e) (f) (g) (h)

Figure 9.26: (a,c,e,g) bicubic interpolation of LR apple scans, (b,d,f,h) SR reconstruction by SRCNN-brain with multiple blurs.



(a) (b) (c) (d)

Figure 9.27: (a) bicubic interpolation of LR apple scan, (b) SR reconstruction by standard SR, (c) SR reconstruction by SRCNN-brain with multiple blurs, (d) HR apple scan.

9.5. ReCNN-nat 2D

In this section we will first make an analysis of our choices for the hyperparameters of the ReCNN-nat network and later present results on the test dataset Set5 and on actual low-field MRI apple scans, analogously to what has been done for SRCNN-nat.

9.5.1. Model Set-up

In [56, 60] a thorough investigation of possible hyperparameters has been made, and both papers agree in selecting He as the optimal weight initialization. As already mentioned in Section 7.3.3, this is indeed a theoretically sound choice for networks utilizing ReLU activation functions. Furthermore, by comparing several commonly used initializers, it was proved that ReCNN might struggle and even fail to converge when using a different value for this hyperparameter.[56] It was therefore decided to comply with this choice also for our tests.

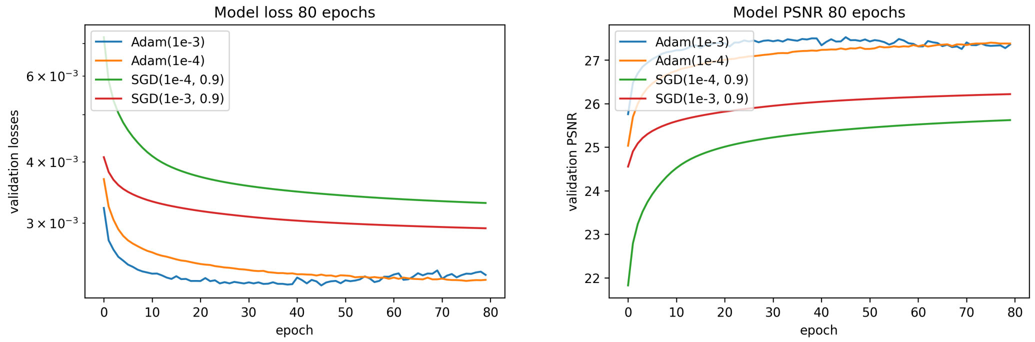


Figure 9.28: Loss and PSNR plots for ReCNN-nat.

Both the optimizer, the learning rate and the number of epochs where investigated on Dataset 1 by means of grid-search, and the outcome is shown in Figure 9.28. Clearly Adam outperforms SGD. Before overfitting between epoch 30 and 40, Adam with learning rate 0.001 reaches a lower loss and higher PSNR than its counterpart with smaller learning rate. As it was already anticipated in Section 9.1, we are not surprised to see the network overfitting on such a “small” dataset.

In order to face this problem, two strategies were used: increasing the training dataset and using early-stopping. ReCNN with Adam(0.001) was trained several times with EarlyStopping as a callback with a patience of 10, and in Figure 9.29 we show the loss and PSNR for the training compared to Adam(0.0001). The use of this callback allowed us to determine that for this model 70 is a number of epochs which does not cause overfitting. Despite the losses and PSNR achieving similar values in both cases, the smaller learning rate does not allow the training loss to go below a certain level. Increasing the number of epochs does not yield an improvement in the training of the model with smaller learning rate, but instead leads to overfitting.

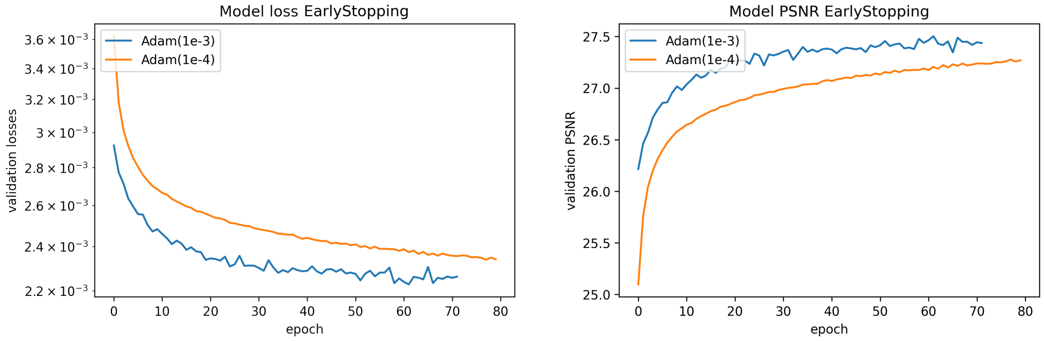


Figure 9.29: Loss plot and PSNR for ReCNN-nat with optimizer Adam(0.001) and EarlyStopping on Dataset 1.

The other option which was explored was to increase the size of the dataset. We trained ReCNN-nat on Dataset 2 with Adam(0.001) and showed the corresponding plots in Figure 9.30. Contrarily to our expectations, adding more training data did not help the training. In fact, it actually led to a worse training behavior, with the network overfitting after only 15 epochs. Adding one or more Dropout layers is a possibility which might be worth exploring in the future, in order to tackle this problem.

It is still unusual that adding more data would cause overfitting instead of preventing it. An explanation for this might be found in [65]: "Using a larger database does not necessarily generate better results, on the contrary, a larger number of irrelevant examples not only increase the computational time of searching for the best matches, but also disturb this search". It might be that simply adding more patches creates a repetitive dataset. An option might be to perform other, more complex data augmentation procedures, and train the network on this augmented dataset, and maybe then be able to truly exploit the benefits of a larger dataset.

All considered, it was decided to test the network with the configuration in Table 9.8. In the following pages we show the results on the Se5 dataset and on the MRI apple scans.
Note that in order to compare the IQA metrics values for SRCNN-nat and ReCNN-nat, it was necessary to crop the results of the latter to match the results of the former. The PSNR is known to vary based on the size of the image. Indeed, PSNR represents the relation between maximum value of signal and variance of noise, and adding more pixels might have an influence on one or both of these quantities. Hence, it would not be a fair comparison if we considered different sizes for the predictions of the two networks.

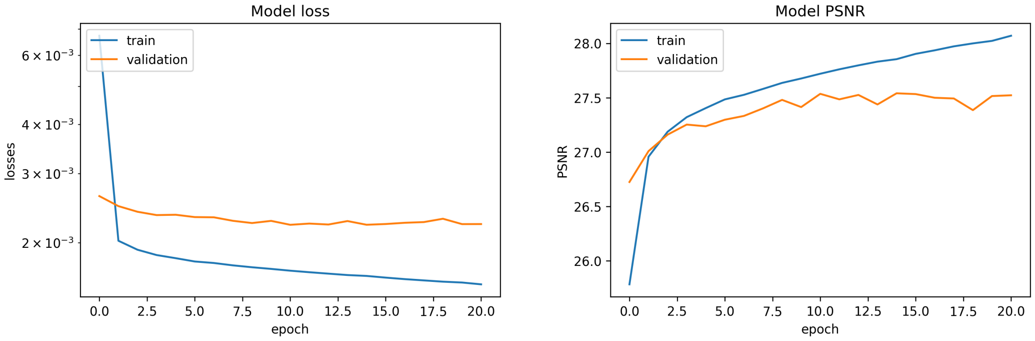


Figure 9.30: Loss plot and PSNR for ReCNN-nat with optimizer Adam(0.001) and EarlyStopping on Dataset 2.

Table 9.7: 4-fold cross validation for ReCNN-nat with final configuration. The values on the different folds are very similar, confirming that the

	Fold 1	Fold 2	Fold 3	Fold 4
PSNR	27.64	27.54	27.83	27.64
MSE	$1.72 \cdot 10^{-3}$	$1.76 \cdot 10^{-3}$	$1.65 \cdot 10^{-3}$	$1.72 \cdot 10^{-3}$
Average on folds				
PSNR	27.3			
MSE	$1.91 \cdot 10^{-3}$			

Table 9.8: Hyperparameters used for the final configuration of ReCNN-nat.

ReCNN-nat	
Batch size	256
Epochs	80
Optimizer	Adam
Learning rate	0.001
Metric	MSE
Weight initialization	He
Dataset	Dataset 1

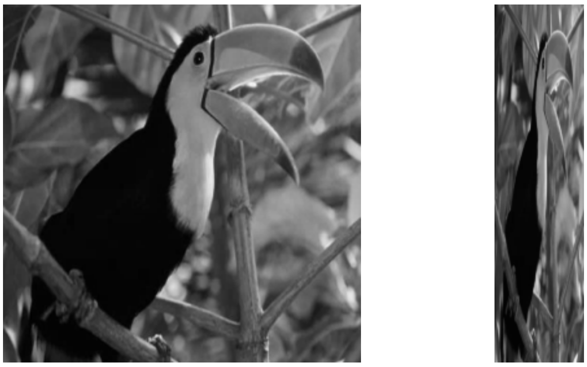
9.5.2. Results: Test Data & Low-Field Data

Looking at Figure 9.31d and comparing it with 9.9d, we immediately notice that the lines and edges are sharper for ReCNN-nat, especially around the beak. This is also reflected in the IQA metrics, were we have the same SSIM for both reconstructions but a higher PSNR for 9.31d.

The reconstruction of the LR apple scans is comparable to the one of SRCNN-nat for both the ReCNNs trained on one or multiple blurs. Weirdly, the edges of the apple are highlighted for Figure 9.32f and much brighter than they should be. Also, this only happens for one of the scans, while for the others we observe a nice, faithful reconstruction of the different intensities in the LR scans. Overall, we are fully satisfied with both reconstructions and in general find very little differences.

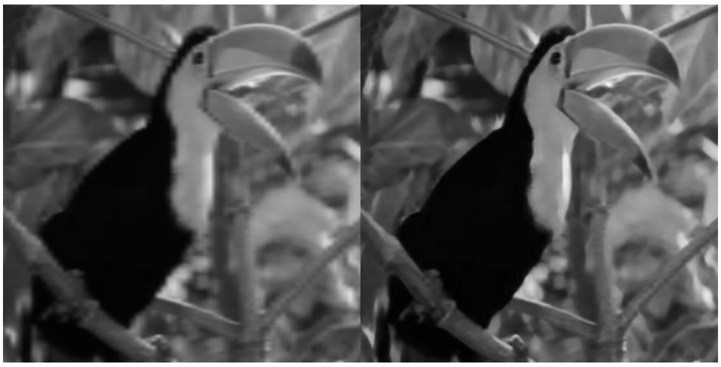
ReCNN-nat provides an extremely sharp reconstruction for the combined apple scan (see Figure 9.33 and 9.37), although it is hard to judge whether the results yielded are superior to the ones of SRCNN-nat. Moreover, the shapes are reconstructed differently by the two ReCNN-nat models, with 9.33c resembling the standard SR reconstruction more than 9.37c, but with the right side of the latter being visually very similar to the ground-truth.

Again, also this model provides clearly better results than the ones yielded by standard super resolution. Compared to Figure 9.33b, the pixel intensity variations are much smoother, the staircasing effect is absent and the shape itself of ReCNN-nat reconstruction resembles the one of the ground-truth more.



(a) Ground-truth.

(b) Blurred, downsampled image.



(c) Bicubic interpolation.
PSNR: 29.19, SSIM: 0.91

(d) ReCNN-nat $\sigma = 1.3$.
PSNR: 32.61, SSIM: 0.94

Figure 9.31: Comparison of different reconstructions of a bird from Set5 downsampled and blurred by Gaussian blur with $\sigma = 1.3$.

Considering again the ReCNN-nat trained with standard deviation $\sigma \in U(0,3)$, we notice again a lower PSNR for the bird reconstruction. More interestingly, while the network trained on multiple blurs seems to handle a standard deviation of 2 much better than the model trained on $\sigma = 1.3$, when the bird is blurred with a standard deviation of 0.5, this is not anymore true. This is in contrast with what was found for SRCNN-nat. Furthermore, the contour delimiting the “back” of the animal is evidently blurred for Figure 9.35d compared to 9.35c, which goes against the values for PSNR and SSIM. This outcome reinforces our belief that these metrics are not fully reliable, certainly not as much as human perception.

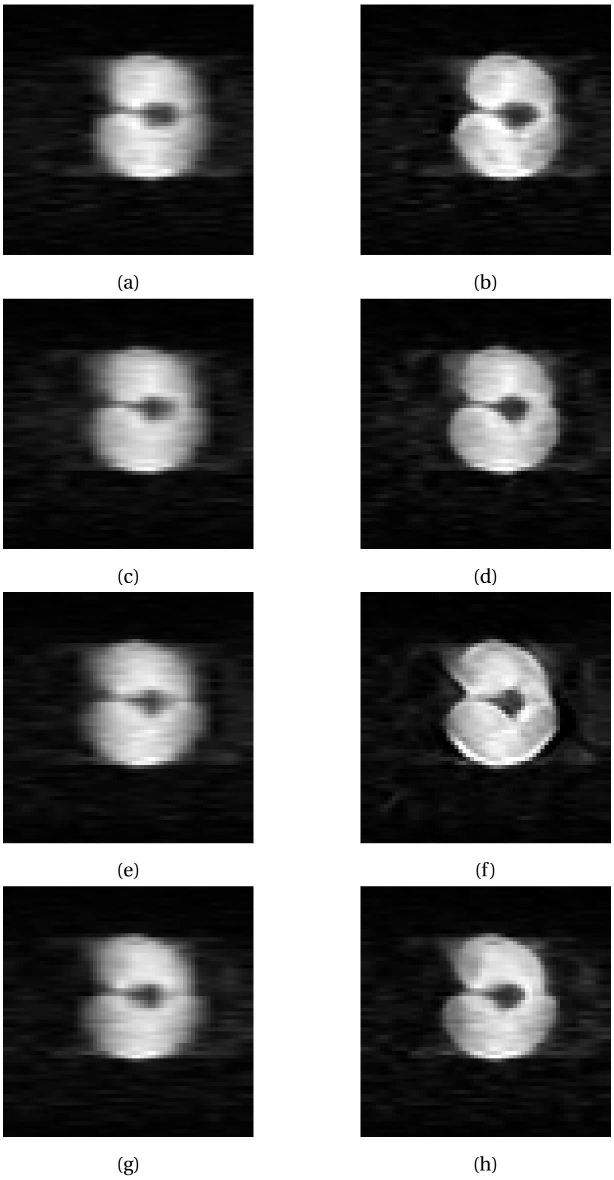


Figure 9.32: (a,c,e,g) bicubic interpolation of LR apple scans, (b,d,f,h) SR reconstruction by ReCNN-nat with fixed blur.

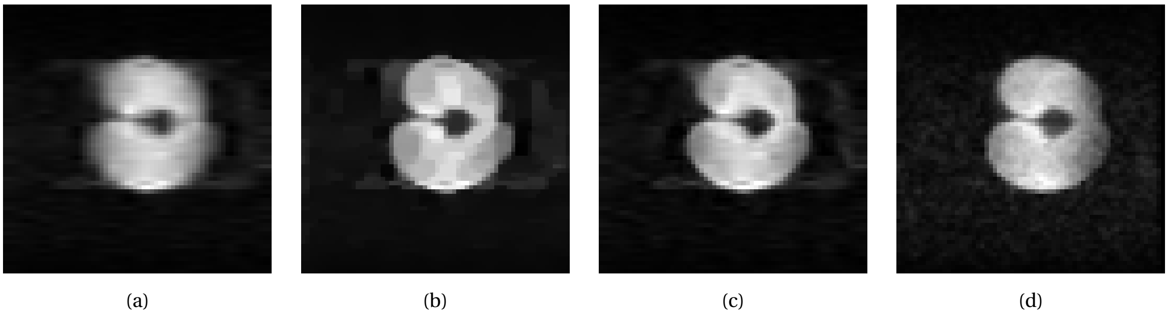
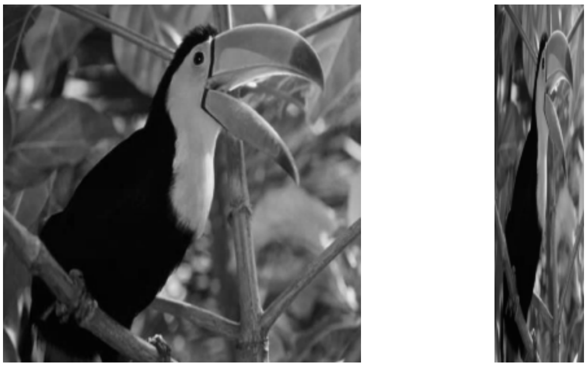
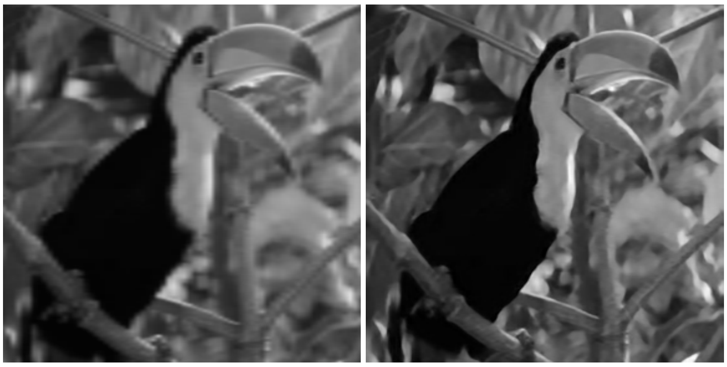


Figure 9.33: (a) bicubic interpolation of LR apple scan, (b) SR reconstruction by standard SR, (c) SR reconstruction by ReCNN-nat with fixed blur $\sigma = 1.3$, (d) HR apple scan.



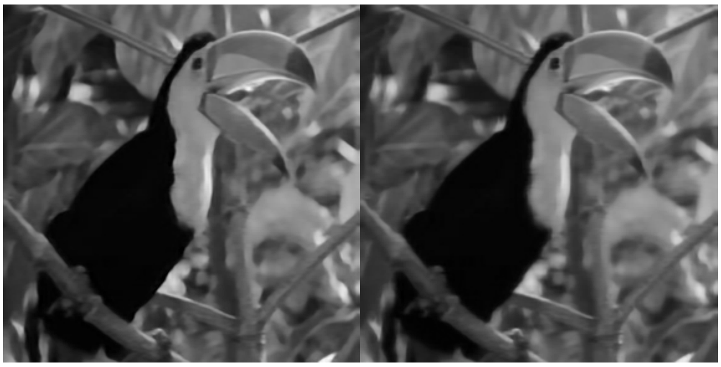
(a) Ground-truth. (b) Blurred, downsampled image.



(c) Bicubic interpolation.
PSNR: 29.19, SSIM: 0.91

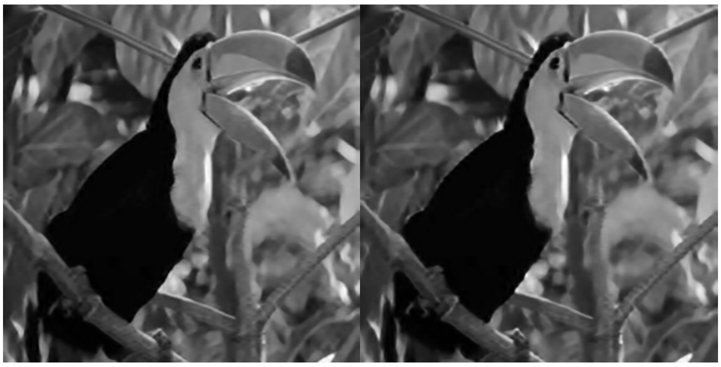
(d) ReCNN-nat $\sigma \in U(0,3)$.
PSNR: 31.58, SSIM: 0.93

Figure 9.34: The reconstruction process of a bird from Set5 downsampled and blurred by Gaussian blur with $\sigma = 1.3$.



(a) ReCNN-nat $\sigma \in U(0,3)$
PSNR: 31.48, SSIM: 0.93.

(b) ReCNN-nat $\sigma = 1.3$
PSNR: 30.57, SSIM: 0.92.



(c) ReCNN-nat $\sigma \in U(0,3)$
PSNR: 30.14, SSIM: 0.92.

(d) ReCNN-nat $\sigma = 1.3$
PSNR: 30.27, SSIM: 0.93.

Figure 9.35: (a,b) bird from Set5 reconstructed from Gaussian blur with $\sigma = 2$ (c,d) bird from Set5 reconstructed from Gaussian blur with $\sigma = 0.5$.

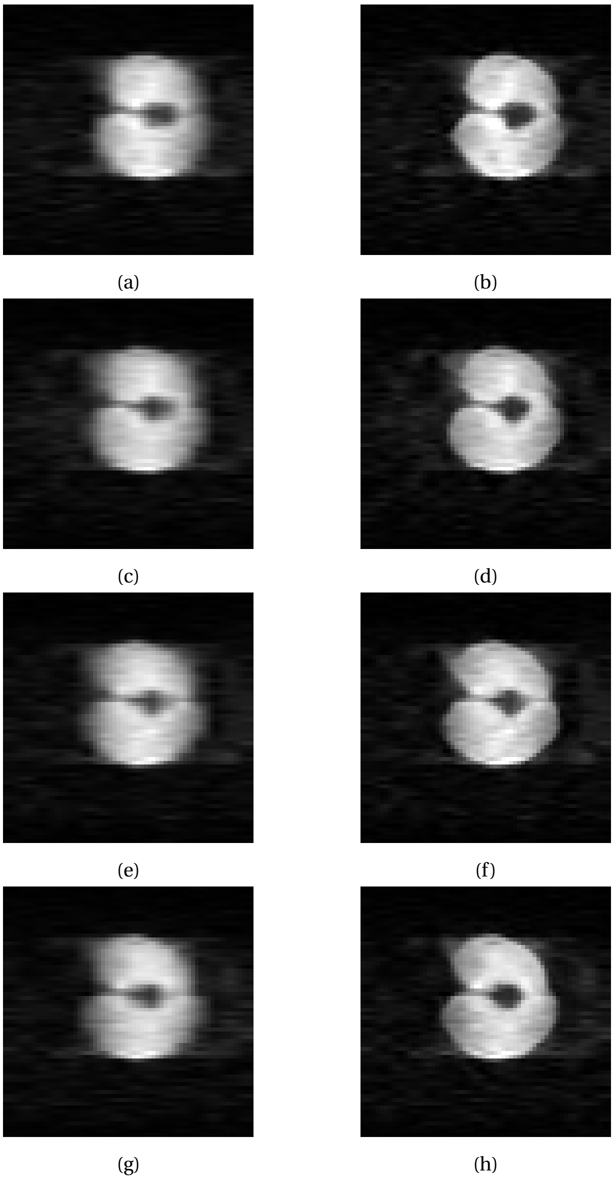


Figure 9.36: (a,c,e,g) bicubic interpolation of LR apple scans, (b,d,f,h) SR reconstruction by ReCNN-nat with multiple blurs.

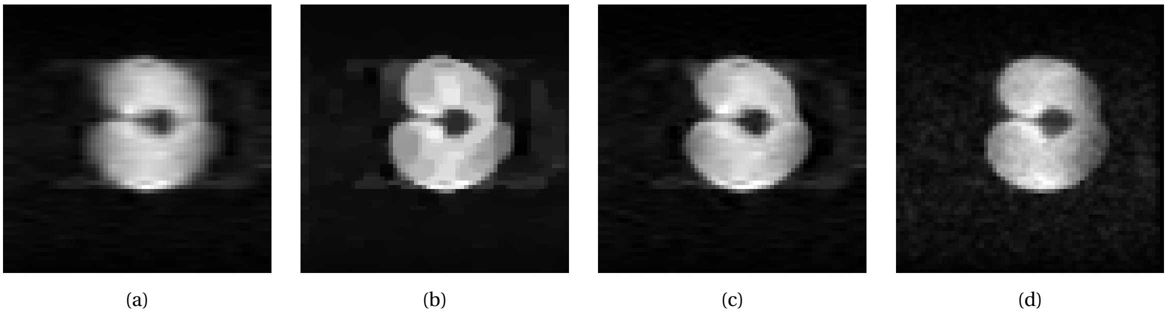


Figure 9.37: (a) bicubic interpolation of LR apple scan, (b) SR reconstruction by ReCNN-nat with multiple blurs, (c) SR reconstruction by standard SR, (d) HR apple scan.

9.6. ReCNN-brain 2D

In this section ReCNN will be trained on the Kirby21 2D dataset described in Section 9.1.2.

9.6.1. Model Set-up

Since Adam proved to be the best choice in terms of optimizers for all the networks trained until now, SGD will not be considered anymore in our investigation of the hyperparameters. Concerning the learning rate, the learning curves for both 0.001 and 0.0001 were compared (see Figure 9.38). Until epoch 100 the model trained with higher learning rate undoubtedly surpasses the other. After epoch 100, it starts overfitting, but despite this, the loss of the model trained with lower learning rate always stays above the one for Adam(0.001). In order to reduce or even prevent overfitting of the model, we trained it several times using EarlyStopping as a callback. This way we were able to see that 100 is indeed the appropriate amount of epochs for the model. By training models with the two learning rates for 100 epochs we were able to make a final comparison and draw the conclusion that a learning rate of 0.001 is indeed the best choice.

As foretold in Section 9.1.2, ReCNN-brain was also trained on the bigger Dataset 2. It appears from Figure 9.40 that training on Dataset 2 would improve the generalization of the model, yielding a lower validation loss and higher PSNR. Furthermore, we are confident that adding even more data might further boost the performance of the model. Nevertheless, it is worth noting that the training time on Dataset 1 was of around 7 minutes, while it took double of this time to train the model on the bigger dataset. Therefore, depending on the time and computational power availability, we deem that the option of a larger dataset might be worth exploring. Since our goal is to be able to obtain the best image quality possible, from now on in our predictions we will then consider ReCNN-brain trained on Dataset 2.

The final model was then cross-validated (see Figure 9.10). As observed in Section 9.3 the number of folds was chosen in order to keep the same proportion training-to-validation that we have when using a simple hold-out split.

Lastly, we remind the reader that the average training time for SRCNN-brain on Dataset 1 was of 11 minutes, very close to the 15 minutes needed to train ReCNN-brain on a much larger dataset (a smaller number of epochs is needed for ReCNN to converge).

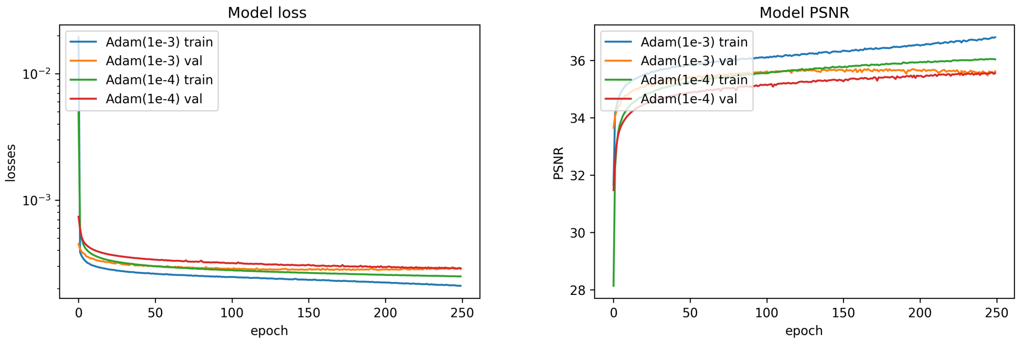


Figure 9.38: Loss plot and PSNR for ReCNN-brain. Comparison between Adam(0.001) and Adam(0.0001) on Dataset 1.

Table 9.9: Hyperparameters used for the final configuration of ReCNN-brain.

ReCNN-brain	
Batch size	256
Epochs	80
Optimizer	Adam
Learning rate	0.001
Metric	MSE
Weight initialization	He
Dataset	Dataset 2

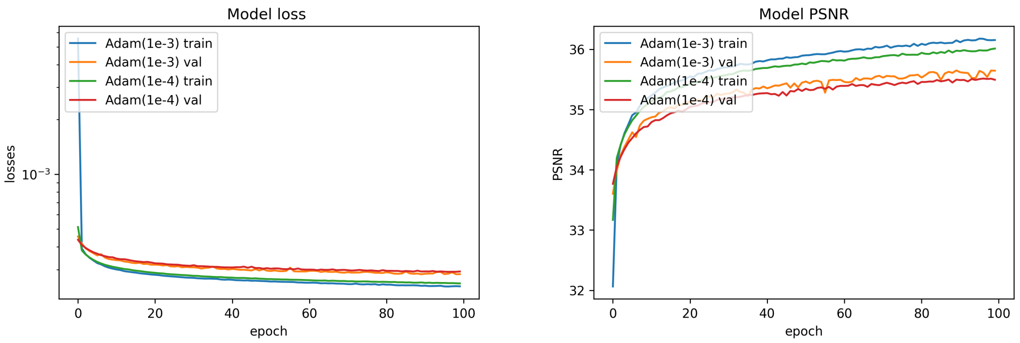


Figure 9.39: Loss plot and PSNR for ReCNN-brain on 100 epochs. Comparison between Adam(0.001) and Adam(0.0001) on Dataset 1.

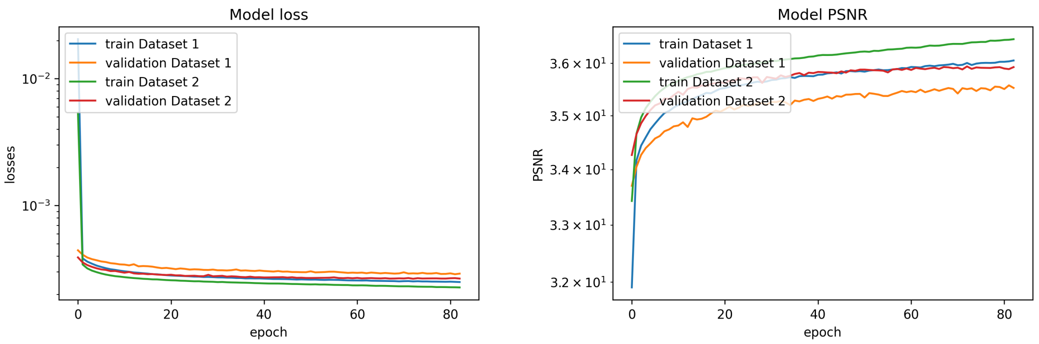


Figure 9.40: Loss plot and PSNR for ReCNN-brain on 80 epochs. Comparison between training on Dataset 1 and 2.

Table 9.10: 10-fold cross validation for ReCNN-brain with final configuration.

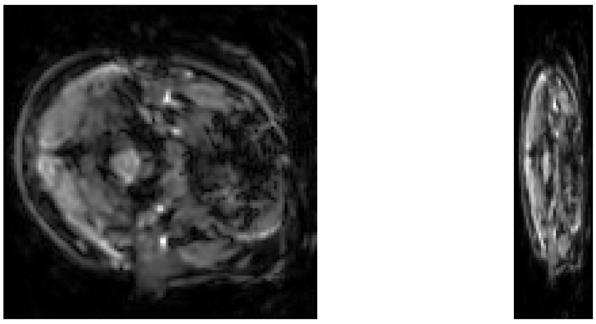
	Fold 1	**Fold 2**	**Fold 3**	**Fold 4**	**Fold 5**	**Fold 6**	**Fold 7**	**Fold 8**	**Fold 9**
PSNR	36.18	36.16	36.16	36.27	36.20	36.29	36.25	36.20	36.16
MSE $\cdot 10^4$	2.41	2.42	2.42	2.36	2.40	2.35	2.37	2.40	2.42
Average on folds									
PSNR	36.22								
MSE	$2.39 \cdot 10^{-4}$								

9.6.2. Results: Test Data & Low-Field Data

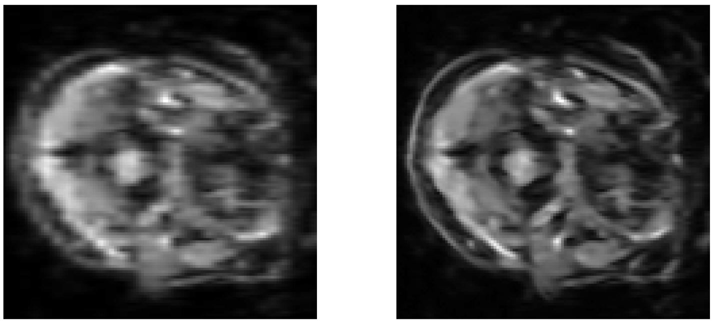
The first fact that can be observed is that both the PSNR and SSIM for the brain MRI scans are always higher for ReCNN-brain compared to SRCNN-brain. From a visual point of view, Figure 9.41d and 9.21d are very similar. ReCNN probably manages slightly better than SRCNN to reconstruct the outer shape of the cranium.

It is very surprising that the quality of the apple reconstruction is extremely high even when training on Kirby21, in opposition to what was observed for SRCNN-brain. This goes also against the hypothesis mustered in Section 9.4 of the Kirby21 dataset not being suited for the reconstruct of our apple scans. ReCNN-brain is evidently capable of generalizing very well on new data, so well that it manages to reconstruct even data which differ greatly form the ones it has been trained on. Again, also this time we deem the network reconstruction to be more realistic, and overall superior to the standard one

Concerning ReCNN-brain trained on multiple blurs, for the test brain scan we identify the same trend as for the models previously trained: as we would expect, the model trained on $\sigma = 1.3$ performs better on data degraded with the same exact blur, but it is less able to generalize to other blurs (see Figure 9.45).
The predictions on the apple scans are considerably more blurred for the model trained on multiple blurs, even though they still represent an improvement compared to the SRCNN-brain counterparts.



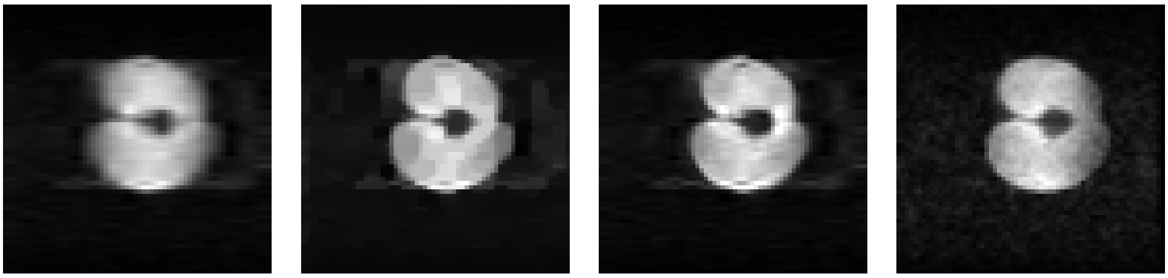
(a) Ground-truth. (b) Blurred, downsampled image.



(c) Bicubic interpolation.
PSNR: 29.36, SSIM: 0.84

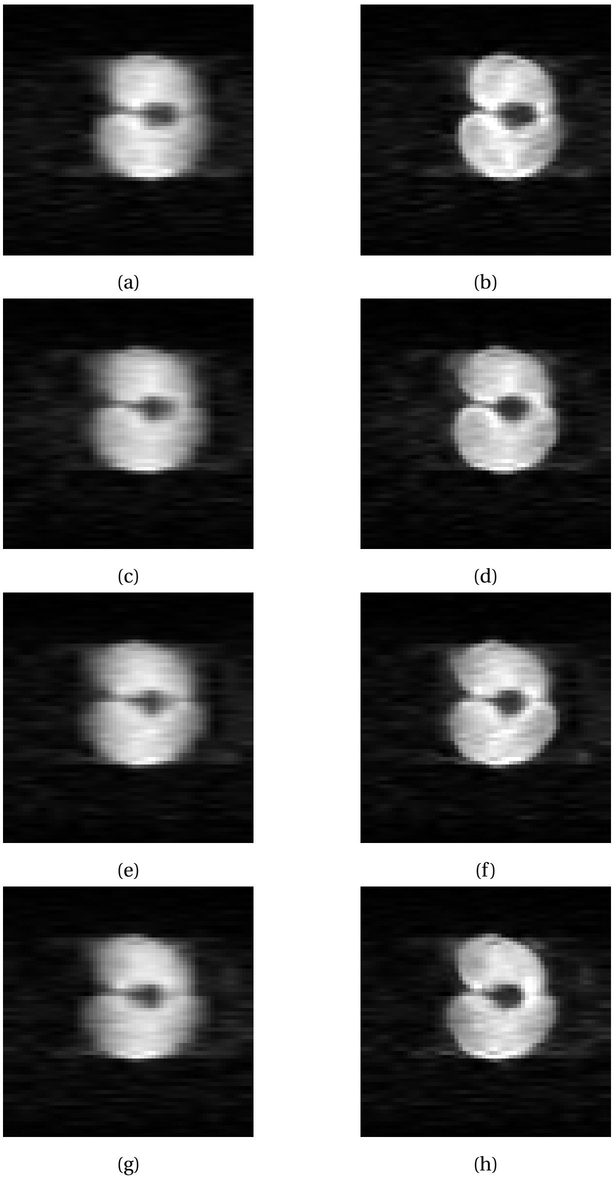
(d) ReCNN-brain with $\sigma = 1.3$
PSNR: 31.52, SSIM: 0.89

Figure 9.41: The reconstruction process of a slice of a Kirby21 scan, downsampled and blurred by Gaussian blur with $\sigma = 1.3$.



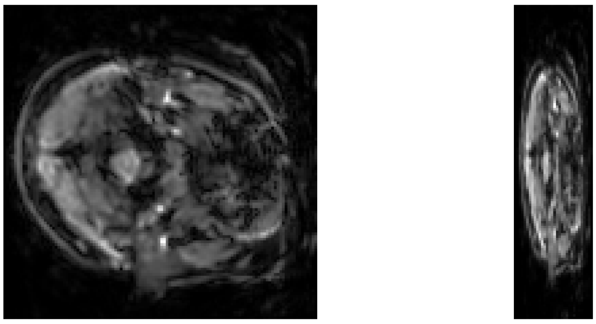
(a) (b) (c) (d)

Figure 9.42: (a) bicubic interpolation of LR apple scan, (b) SR reconstruction by standard SR, (c) SR reconstruction by ReCNN-brain with fixed blur $\sigma = 1.3$, (d) HR apple scan.



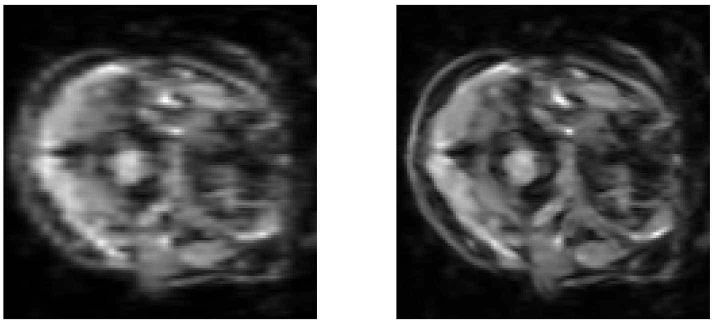
(a) (b) (c) (d) (e) (f) (g) (h)

Figure 9.43: (a,c,e,g) bicubic interpolation of LR apple scans, (b,d,f,h) SR reconstruction by ReCNN-brain with fixed blur.



(a) Ground-truth.

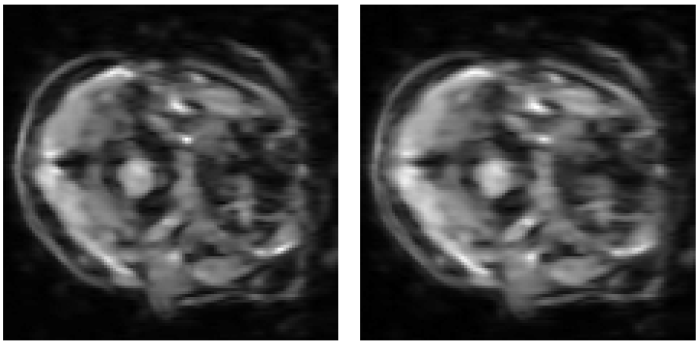
(b) Blurred, downsampled image.



(c) Bicubic interpolation.
PSNR: 29.36, SSIM: 0.84

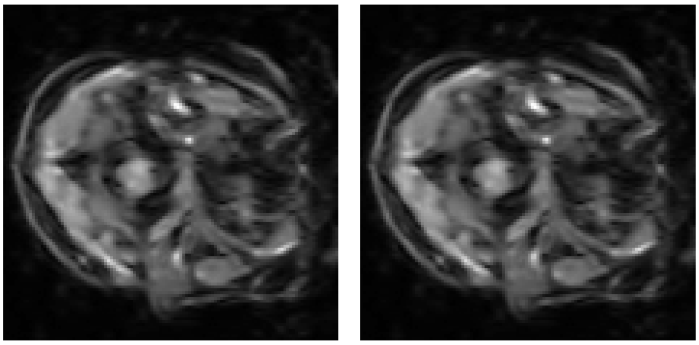
(d) ReCNN-brain with $\sigma \in U(0,3)$
PSNR: 31.03, SSIM: 0.88

Figure 9.44: The reconstruction process of a slice of a Kirby21 scan, downsampled and blurred by Gaussian blur with $\sigma \in U(0,3)$.



(a) ReCNN-brain $\sigma \in U(0,3)$
PSNR: 31.00, SSIM: 0.88.

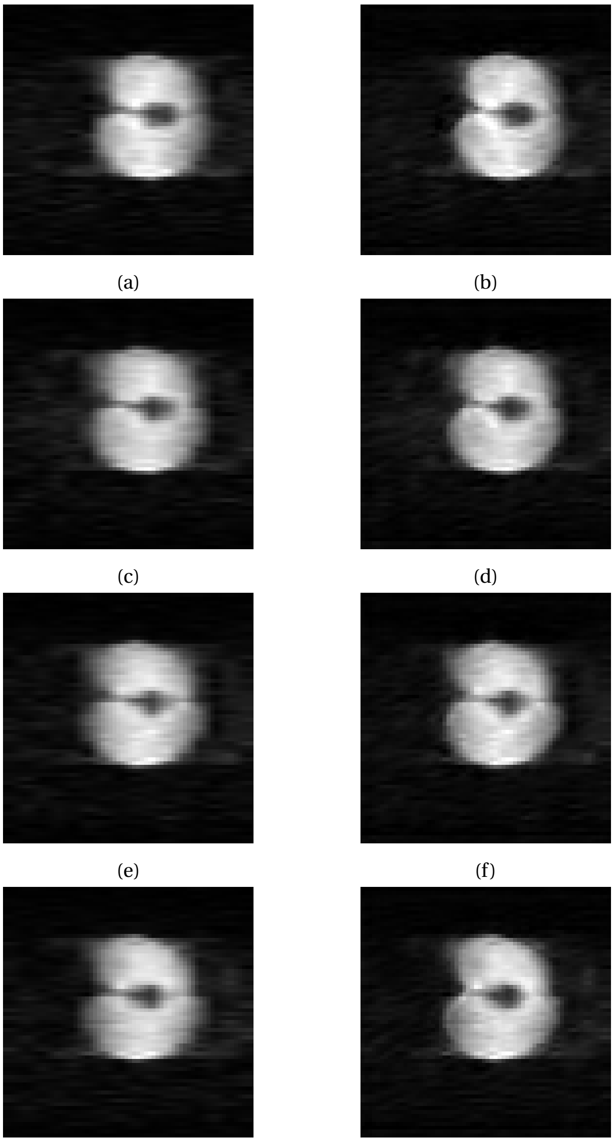
(b) ReCNN-brain $\sigma = 1.3$
PSNR: 30.63, SSIM: 0.87.



(c) ReCNN-brain $\sigma \in U(0,3)$
PSNR: 30.30, SSIM: 0.87.

(d) ReCNN-brain $\sigma = 1.3$
PSNR: 29.28, SSIM: 0.85.

Figure 9.45: (a,b) slice from Kirby21 scan reconstructed from Gaussian blur with $\sigma = 2$ (c,d) slice from Kirby21 scan reconstructed from Gaussian blur with $\sigma = 0.5$.



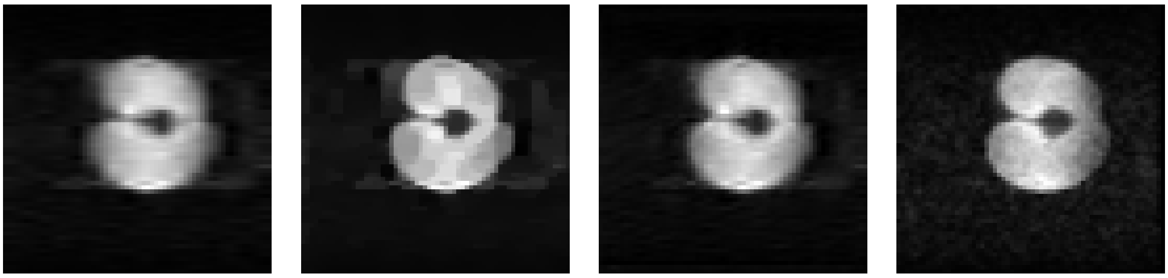
(a) (b)

(c) (d)

(e) (f)

(g) (h)

Figure 9.46: (a,c,e,g) bicubic interpolation of LR apple scans, (b,d,f,h) SR reconstruction by ReCNN-brain with multiple blurs.



(a) (b) (c) (d)

Figure 9.47: (a) bicubic interpolation of LR apple scan, (b) SR reconstruction by standard SR, (c) SR reconstruction by ReCNN-brain with multiple blurs, (d) HR apple scan.

10

CONCLUSIONS AND RECOMMENDATIONS

In the introductory chapter of this thesis, two research questions have been formulated:

1. Can Multi Image Super Resolution (MISR) yield comparable results to images directly acquired in higher resolution?
2. Can super resolution deep learning techniques yield better results than standard ones?

After the research conducted during this project, we believe we are able to give a positive answer to both questions. It has been shown that deep learning techniques yield results which are superior to the ones of standard techniques. Furthermore, the visual quality of these reconstructions is comparable to the one of the HR scan taken as ground-truth. We deem this result to be extremely promising and to be able to have direct application in the framework of the low-field MRI project.

Multiple super resolution algorithms have been compared.

First, the observation model for standard Multi Image Super Resolution (MISR) has been presented, where multiple low-resolution (LR) observations of the same scene are linked to a high-resolution (HR) image of the original object. The LR images are obtained from their HR counterpart by means of shifting, Gaussian blurring and downsampling in the z-direction only. The goal of Super Resolution Reconstruction (SRR) is to invert this process and retrieve the original HR object. The degradation operations mentioned earlier were modeled into matrix form, leading to the observation model in Eq. 3.1.4. This system is ill-posed, and a multiplicity of solutions exist for this inverse problem. To overcome this issue, the concept of regularization was introduced as a way to enforce prior knowledge on the system.

Two types of regularization were introduced: Tikhonov and Total Variation (TV), leading to minimization problems 3.3.1 and 3.3.4. The added regularization terms in both cases penalize the jumps between pixels, although in a different way. Indeed, the Tikhonov penalty term grows quadratically with the difference between pixels, while for the TV regularization the penalty is linear. CGLS was then adapted to the new Tikhonov minimization problem through the introduction of a regularization matrix F, modeled as a second-order difference matrix. ADMM was instead used as a solver for minimization problem with TV regularization. This time, a first-order difference regularization matrix F was considered.

These algorithms were first applied to a Shepp-Logan phantom and then to actual low-field MRI scans. Concerning the latter, two approaches were followed. Firstly, the scans were sliced along the sagittal plane and 2D standard SR applied on the slices. Then, 3D SR was applied to the whole 3D data, and some preliminary results were obtained. The results yielded by Tikhonov and Total Variation were compared, and various values for the regularization parameters λ and a were used. It was observed that, choosing the appropriate parameters, Total Variation outperformed Tikhonov regularization. Despite the staircasing effect typical of TV, the reconstructions of the low-field scans were much sharper for this kind of regularization and overall of better quality. Furthermore, these reconstructions considerably improved the resolution of the LR scans. Nevertheless, none of the regularization yielded results comparable to the scan directly acquired in a higher resolution.

In the second phase of the project, the focus was shifted to Deep Learning (DL) techniques. DL-based Single Image Super Resolution (SISR) methods have attracted the interest of many researcher in recent years and have been proved to often achieve state-of-the-art performance on various benchmarks. These networks aim to reconstruct an HR image from a single LR version of that image, contrarily to the standard SR method. It was decided to explore the possibilities of two of these neural networks: SRCNN and ReCNN. For each of those, two 2D and one 3D version were implemented. 2D SRCNN-nat and ReCNN-nat have been trained on the T91 dataset of natural images, while 2D SRCNN-brain and ReCNN-brain on slices of 3D data coming from the Kirby 21 dataset of MRI brains. The samples are pre-processed by means of bicubic interpolation and Gaussian blurring with standard deviation σ, both in the z-direction. Moreover, each of these network was trained in two ways: first, with a fixed standard deviation of 1.3, next with a different value for σ for each training sample, drawn from $U(0,3)$.

The two deep learning models differ greatly in complexity. Indeed, SRCNN is quite a shallow fully convolutional network with 2 hidden layers. On the other hand, ReCNN is much deeper, having 10 layers. The number of parameters is then around 8.000 for SRCNN with filters 64-32-1 (20.000 for architecture 128-64-1) and 300.000 for ReCNN. While some hyperparameters were set based on available literature, most of them were fine-tuned for both networks, performing grid-search on different possible combinations. It was found that Adam consistently outperforms SGD as an optimizer, with an optimal learning rate of 0.001. Moreover, He initialization was preferred as a weight initialization approach for most of the models due to both its theoretical soundness and the good training behavior yielded.

The quality of the neural network reconstructions was first assessed on test data drawn from the same distribution as the training data, by means of both a simple visual judgment and two well-known Image Quality Assessment (IQA) metrics: PSNR and SSIM. The positive outcome of this assessment testifies that the networks were trained correctly, being able to fit the training data well, as well as to generalize to new, unseen data. On these test data, ReCNN-nat and -brain yield marginally better results than their SRCNN counterpart, confirmed by a higher PSNR and a higher or equal SSIM. Furthermore, the training time of ReCNN-nat and SRCNN-nat is approximately 7 minutes on Google Colab Pro GPU for both models. Similarly, the training time of the two final models for SRCNN-brain and ReCNN-brain is approximately 15 minutes.

Next, the 2D models were applied to slices of low-field MRI scans. Both SRCNN and ReCNN-nat successfully managed to reconstruct the LR apple scans, clearly outperforming standard SR. As already explained in the Introduction, because of the smaller Field of View (FoV) of the low-field MRI scanner, we are in need of a SR technique which would allow us to combine different LR scans shifted of a sub-pixel quantity one from the other. Hence, we are more interested in MISR methods than SISR. Following this line of thought, it was decided to combine the LR MRI scans into one, with the same low-resolution, but with additional information compared to each scan taken separately. The networks were then applied to these combined scans. The results yielded by both SRCNN-nat and ReCNN-nat were extremely satisfactory, with ReCNN-nat performing marginally better. The reconstructions are sharp and evidently outperform standard SR, and are comparable in visual quality to the HR scan taken as ground-truth. Concerning SRCNN-brain, training on MRI brains took its toll on the reconstruction of the apple scan, introducing artifacts and leading to blurred images. It is clear that SRCNN-brain was not capable to generalize to images which differ substantially from the training data. Lastly, we were impressed by the performance of ReCNN-brain. Indeed, ReCNN-brain with $\sigma = 1.3$ was able not only to predict the test data correctly, but also to generalize to the apple scan surprisingly well.
As a last observation, models trained with differently blurred samples generally yielded worse results on the apple scans, compared to their counterparts with fixed blur. This is because the latter were trained with $\sigma = 1.3$, which was proved to be the same standard deviation of the LR scans. Hence, these models were specialized exactly for the right kind of degradation. However, when the value for the standard deviation is unknown, multi-blur networks have proved to be a good alternative.

We conclude that ReCNN models should be preferred, as they yield better results than SRCNN in the same amount of time. In particular, ReCNN-brain with fixed sigma has shown the best generalization abilities. Furthermore, almost all the DL models analyzed yielded a better performance than standard SR. Therefore, we believe that in the future the best choice would be to focus on SR networks.
The next step would be to apply the algorithm on brain data obtained with the low-field MRI scanner, when they become available.

In addition to the 2D models, a 3D extension of SRCNN and ReCNN on Kirby21 was considered. Nevertheless, we lacked the time to fine-tune and properly test such networks. Therefore, the results obtained were not satisfactory and were not included in the report.

10.1. RECOMMENDATIONS FOR FUTURE RESEARCH

There are still many options that can be explored further to improve the super resolution networks in the framework of low-field MRI. In this section we give some recommendations for future work.

- **Low-field MRI dataset**
  The neural networks tested during the course of this project were trained on commonly used datasets for super resolution. Although we were already successful in reconstructing MRI scans by training the model in such a way, we are confident that building a low-field MRI dataset and training the networks on such a dataset might be beneficial. The models would then learn the mapping between LR and HR scans coming from our machine, thus leading to more detailed reconstructions and possibly being applicable also to brain scans. As seen with SRCNN-nat and ReCNN-nat, in order to train a successfully working network, a dataset made of 91 samples was sufficient. In our case, the number of samples should be doubled, because we would need the LR scans and their HR counterparts. Nevertheless, we believe then that creating such a dataset is an achievable goal.

- **3D neural network**
  It was already mentioned that a 3D extension to the network models was implemented. Sadly, due to time limitations it has not been feasible to fully explore the possibilities provided by such architectures. However, we are confident that it is absolutely feasible to train 3D models yielding superior results on low-field data. We refer the reader to [56, 66], where possible 3D super resolution models are thoroughly described and tested on MRI data. Here (and in most of the literature currently available), an isotropic scale factor of 2 is considered. It would be interesting to extend such models to our case, were we have an anistropic factor of 4. As a consequence, the training might be more difficult and an accurate fine-tuning of the hyperparameters particularly necessary.

- **MISR neural network**
  In this thesis, multi image super resolution using deep learning was achieved following two distinct processes: first, combining MRI scans, then applying neural networks. The inverse approach was also considered, but the results yielded were not comparable.
  Until 2019, deep learning models had almost only being developed for SISR. In [53] an approach to a MISR neural network is presented, and in [17] the idea of such a network is resumed. In would be compelling to work on this kind of possibility, in order to fill the gap between standard SR and the deep learning models presented in this report.

- **Assumptions**
  The only geometric transformations considered were shifts in the z-direction. Since the final goal of the low-field MRI project is of course to image living beings, clearly it would not be possible to make such assumptions anymore. Thus, rotations and shifts in other directions would need to be included in a future SR model. Furthermore, image registration between low-field MRI samples was performed under the same assumptions, while we know that some small rotation might have actually occurred during the scanning of the objects. We would recommend performing a more complete registration, including also other kinds of transformations, since it might have a relevant impact on the result.

- **No-reference image quality metric**
  Two IQA metrics were considered during this research: PSNR and SSIM. Both these metrics require a reference image to assess a reconstruction, thus were deemed unreliable in our case (the available HR scan is not exactly the ground-truth). Implementing a no-reference image quality metric would then allow to objectively estimate the quality of a reconstruction also in such cases (see for instance [67]).

BIBLIOGRAPHY

[1] M. Sarracanie, C. D. LaPierre, N. Salameh, D. E. Waddington, T. Witzel, and M. S. Rosen, *Low-cost high-performance mri,* Scientific reports **5**, 1 (2015).

[2] D. B. Gecmen, *Deep Learning Techniques for Low-Field MRI,* Master's thesis, Delft University of Technology (2020).

[3] T. O'Reilly, W. Teeuwisse, and A. Webb, *Three-dimensional MRI in a homogenous 27 cm diameter bore Halbach array magnet,* Journal of Magnetic Resonance **307**, 106578 (2019).

[4] NINDS, *Hydrocephalus Fact Sheet,* Accessed: 24-03-2020.

[5] M. Wijchers, *Image reconstuction in MRI: The possibilities of portable low-cost MRI scanners,* Master's thesis, Delft University of Technology (2016).

[6] M. L. de Leeuw den Bouter, *Image reconstruction in low-field MRI: a super-resolution approach,* Master's thesis, Delft University of Technology (2017).

[7] B. de Vos, P. Fuchs, T. O'Reilly, A. Webb, and R. Remis, *Gradient Coil Design and Realization for a Halbach-Based MRI System,* IEEE Transactions on Magnetics **56**, 1 (2020).

[8] D. H. Poot, V. Van Meir, and J. Sijbers, *General and efficient super-resolution method for multi-slice mri,* in *International Conference on Medical Image Computing and Computer-Assisted Intervention* (Springer, 2010) pp. 615–622.

[9] E. Van Reeth, I. W. Tham, C. H. Tan, and C. L. Poh, *Super-resolution in magnetic resonance imaging: a review,* Concepts in Magnetic Resonance Part A **40**, 306 (2012).

[10] F. Chollet, *Keras,* (2015).

[11] A. Kazemnejad, *How to do Deep Learning research with absolutely no GPUs - Part 2,* Accessed: 22-09-2020.

[12] Z.-P. Liang and P. C. Lauterbur, *Principles of magnetic resonance imaging: a signal processing perspective* (SPIE Optical Engineering Press, 2000).

[13] A. Berger, *How does it work?: Magnetic resonance imaging,* BMJ: British Medical Journal **324**, 35 (2002).

[14] D. Moratal, A. Vallés-Luch, L. Martí-Bonmatí, and M. E. Brummer, *k-Space tutorial: an MRI educational tool for a better understanding of k-space,* (2008).

[15] J. P. Marques, F. F. Simonis, and A. G. Webb, *Low-field mri: An mr physics perspective,* Journal of magnetic resonance imaging **49**, 1528 (2019).

[16] S. C. Park, M. K. Park, and M. G. Kang, *Super-resolution image reconstruction: a technical overview,* Signal Processing Magazine, IEEE **20**, 21 (2003).

[17] F. Salvetti, V. Mazzia, A. Khaliq, and M. Chiaberge, *Multi-image Super Resolution of Remotely Sensed Images using Residual Feature Attention Deep Neural Networks,* arXiv preprint arXiv:2007.03107 (2020).

[18] C. Dong, C. C. Loy, K. He, and X. Tang, *Image Super-resolution using Deep Convolutional Networks,* IEEE transactions on pattern analysis and machine intelligence **38**, 295 (2015).

[19] J. Yang and T. Huang, *Image super-resolution: Historical overview and future challenges,* in *Super-Resolution Imaging* (CRC Press, 2017) pp. 1–33.

[20] P. C. Hansen, *Analysis of discrete ill-posed problems by means of the L-curve,* SIAM review **34**, 561 (1992).

[21] *Methods of Conjugate Gradients for Solving Linear Systems,* Journal of Research of the National Bureau of Standards **49**, 409 (1952).

[22] C. A. Bouman, *Model based image processing,* Purdue University (2013).

[23] C. B. Clement, M. Bierbaum, and J. P. Sethna, *Image registration and super resolution from first principles,* arXiv preprint arXiv:1809.05583 (2018).

[24] M. Guizar-Sicairos, S. T. Thurman, and J. R. Fienup, *Efficient subpixel image registration algorithms,* Optics letters **33**, 156 (2008).

[25] P. C. Hansen, J. G. Nagy, and D. P. O'leary, *Deblurring images: matrices, spectra, and filtering,* Vol. 3 (Siam, 2006).

[26] C. C. Paige and M. A. Saunders, *LSQR: An algorithm for sparse linear equations and sparse least squares,* ACM Transactions on Mathematical Software (TOMS) **8**, 43 (1982).

[27] S. Nagashima, T. Aoki, T. Higuchi, and K. Kobayashi, *A subpixel image matching technique using phase-only correlation,* in *2006 International Symposium on Intelligent Signal Processing and Communications* (IEEE, 2006) pp. 701–704.

[28] U. of Stanford, *CS231n: Convolutional Neural Networks for Visual Recognition,* Accessed: 04-08-2020.

[29] R. Chalapathy and S. Chawla, *Deep learning for anomaly detection: A survey,* arXiv preprint arXiv:1901.03407 (2019).

[30] J. Brownlee, *What is deep learning?* Accessed: 24-07-2020.

[31] R. Chalapathy and S. Chawla, *Deep learning for anomaly detection: A survey,* (2019), cite arxiv:1901.03407.

[32] J. Kaur Gill, *Automatic Log Analysis using Deep Learning and AI,* Accessed: 10-08-2020.

[33] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning* (MITP, 2018).

[34] A. Mohanty, *Multi layer Perceptron (MLP) Models on Real World Banking Data,* Accessed: 30-08-20.

[35] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, *Activation functions: Comparison of trends in practice and research for deep learning,* arXiv preprint arXiv:1811.03378 (2018).

[36] V. Nair and G. E. Hinton, *Rectified linear units improve restricted boltzmann machines,* in *Proceedings of the 27th International Conference on International Conference on Machine Learning,* ICML'10 (Omnipress, Madison, WI, USA, 2010) pp. 807—-814.

[37] S. Ruder, *An overview of gradient descent optimization algorithms,* Accessed: 15-09-2020.

[38] S. Raschka, *Gradient Descent and Stochastic Gradient Descent,* Accessed: 24-09-2020.

[39] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization,* arXiv preprint arXiv:1412.6980 (2014).

[40] C. M. Bishop, *Pattern recognition and machine learning* (Springer, 2006).

[41] S. K. Kumar, *On weight initialization in deep neural networks,* arXiv preprint arXiv:1704.08863 (2017).

[42] X. Glorot and Y. Bengio, *Understanding the difficulty of training deep feedforward neural networks,* in *Proceedings of the thirteenth international conference on artificial intelligence and statistics* (2010) pp. 249–256.

[43] K. He, X. Zhang, S. Ren, and J. Sun, *Delving deep into rectifiers: Surpassing human-level performance on imagenet classification,* in *Proceedings of the IEEE international conference on computer vision* (2015) pp. 1026–1034.

[44] C. Francois, *Deep learning with Python* (Manning Publications Company, 2017).

[45] A. Burkov, *The Hundred-Page Machine Learning Book,* Expert Systems **5**, 132 (2019).

[46] S. Klosterman, *Overfitting, underfitting, and the bias-variance tradeoff,* Accessed: 04-10-2020.

[47] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, *Dropout: a simple way to prevent neural networks from overfitting,* The journal of machine learning research **15**, 1929 (2014).

[48] S. Varma and S. Das, *Deep learning,* Accessed: 10-09-20.

[49] A. Murphy, *Fully connected neural network,* Accessed: 26-08-2020.

[50] S. Albawi, T. A. Mohammed, and S. Al-Zawi, *Understanding of a convolutional neural network,* in *2017 International Conference on Engineering and Technology (ICET)* (IEEE, 2017) pp. 1–6.

[51] S. Saha, *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way,* Accessed: 09-09-2020.

[52] A. Dertat, *Applied Deep Learning - Part 4: Convolutional Neural Networks,* Accessed: 09-09-2020.

[53] M. Kawulok, P. Benecki, S. Piechaczek, K. Hrynczenko, D. Kostrzewa, and J. Nalepa, *Deep Learning for Multiple-Image Super-Resolution,* IEEE Geoscience and Remote Sensing Letters **17**, 1062 (2019).

[54] V. Cherukuri, T. Guo, S. J. Schiff, and V. Monga, *Deep mr brain image super-resolution using spatio-structural priors,* IEEE Transactions on Image Processing **29**, 1368 (2019).

[55] C. Dong, C. C. Loy, K. He, and X. Tang, *Learning a Deep Convolutional Network for Image Super-Resolution,* in *European conference on computer vision* (Springer, 2014) pp. 184–199.

[56] C.-H. Pham, C. Tor-Díez, H. Meunier, N. Bednarek, R. Fablet, N. Passat, and F. Rousseau, *Multiscale brain MRI super-resolution using deep 3D convolutional networks,* Computerized Medical Imaging and Graphics **77**, 101647 (2019).

[57] J. Yang, J. Wright, T. S. Huang, and Y. Ma, *Image Super-Resolution via Sparse Representation,* IEEE transactions on image processing **19**, 2861 (2010).

[58] J. Yang, J. Wright, T. Huang, and Y. Ma, *Image Super-Resolution as Sparse Representation of Raw Image Patches,* in *2008 IEEE conference on computer vision and pattern recognition* (IEEE, 2008) pp. 1–8.

[59] A. Hore and D. Ziou, *Image quality metrics: Psnr vs. ssim,* in *2010 20th international conference on pattern recognition* (IEEE, 2010) pp. 2366–2369.

[60] J. Kim, J. Kwon Lee, and K. Mu Lee, *Accurate Image Super-Resolution Using Very Deep Convolutional Networks,* in *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016) pp. 1646–1654.

[61] DeepAI, *Exploding Gradient Problem,* Accessed: 27-09-2020.

[62] R. Pascanu, T. Mikolov, and Y. Bengio, *On the difficulty of training recurrent neural networks,* in *International conference on machine learning* (2013) pp. 1310–1318.

[63] C.-H. Pham, A. Ducournau, R. Fablet, and F. Rousseau, *Brain MRI super-resolution using deep 3D convolutional networks,* in *2017 IEEE 14th International Symposium on Biomedical Imaging (ISBI 2017)* (IEEE, 2017) pp. 197–200.

[64] B. A. Landman, A. J. Huang, A. Gifford, D. S. Vikram, I. A. L. Lim, J. A. Farrell, J. A. Bogovic, J. Hua, M. Chen, S. Jarso, *et al.*, *Multi-parametric neuroimaging reproducibility: a 3-t resource study,* Neuroimage **54**, 2854 (2011).

[65] K. Nasrollahi and T. B. Moeslund, *Super-resolution: a comprehensive survey,* Machine vision and applications **25**, 1423 (2014).

[66] Y. Chen, Y. Xie, Z. Zhou, F. Shi, A. G. Christodoulou, and D. Li, *Brain mri super resolution using 3d deep densely connected neural networks,* in *2018 IEEE 15th International Symposium on Biomedical Imaging (ISBI 2018)* (IEEE, 2018) pp. 739–742.

[67] C. Ma, C. Yang, X. Yang, and M. Yang, *Learning a no-reference quality metric for single-image super-resolution,* CoRR **abs/1612.05890** (2016), arXiv:1612.05890 .

A

ACQUISITION PARAMETERS - MRI SCAN

Table A.1: Acquisition parameters for one LR scan of the apple.

Acquisition parameters	
90Amplitude	-28
180Amplitude	-22
accumulate	"yes"
acqTime	3.2
autoPhase	"no"
b1Freq	2.14882d
bandwidth	20
dwellTime	50.0
echoTime	20e3
etLength	32
experiment	"TSE3D_V2_1"
expName	"TSE3D_v2.1"
expNr	16
filter	"no"
filterType	"none"
flatFilter	"yes"
FOVphase1	128
FOVphase2	128
FOVread	128
gradCorr	480
incExpNr	"yes"
kTraject	"In-out"
nPhase1	64
nPhase2	16
nrPnts	64
nrScans	1
plane	"yz"
position	[48,143]
pulseLength	100
repTime	2e3
rxGain	31
rxPhase	0
saveData	"true"
timeMag	"yes"
usePhaseCycle	"yes"
windowSize	"large"

B

PSNR AND SSIM - MRI SCANS

In this appendix we include the PSNR and SSIM for the apple scan reconstructions in Chapter 9. Unexpectedly, the SSIM for standard SR is always higher than for the network reconstructions, while for the latter the PSNR varies. Depending on the particular network, it is higher or slightly lower than the one for TV. Nevertheless, it is true that the models were trained to maximize the PSNR, thus this result is not unreasonable. Interestingly, what is probably the worse reconstruction (Figure B.4c) yields the highest PSNR and SSIM.

Furthermore, both the PSNR and SSIM for the bicubic interpolation are usually higher than for the network reconstructions. This is truly surprising, since all the network reconstructions are undoubtedly of better quality with respect to the simple interpolations. We interpret this fact as a confirm that a visual assessment is more appropriate in this situation.

B.1. SRCNN-NAT

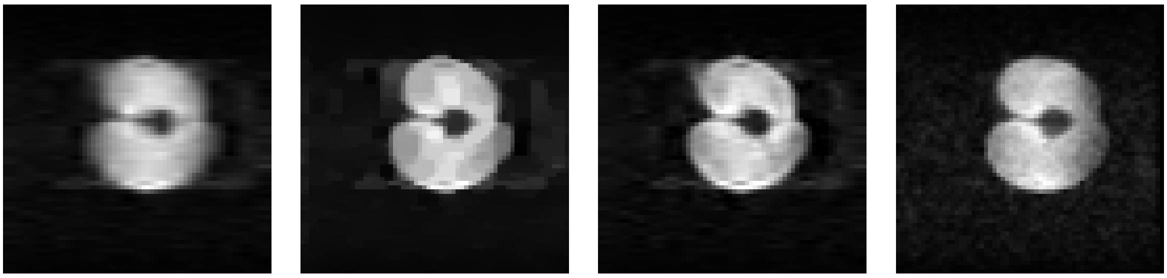

(a) PSNR: 25.06, SSIM: 0.59 (b) PSNR: 24.24, SSIM: 0.62 (c) PSNR: 24.38, SSIM: 0.57 (d)

Figure B.1: (a) bicubic interpolation of LR apple scan, (b) SR reconstruction by standard SR, (c) SR reconstruction by SRCNN-nat with fixed blur $\sigma = 1.3$, (d) HR apple scan.

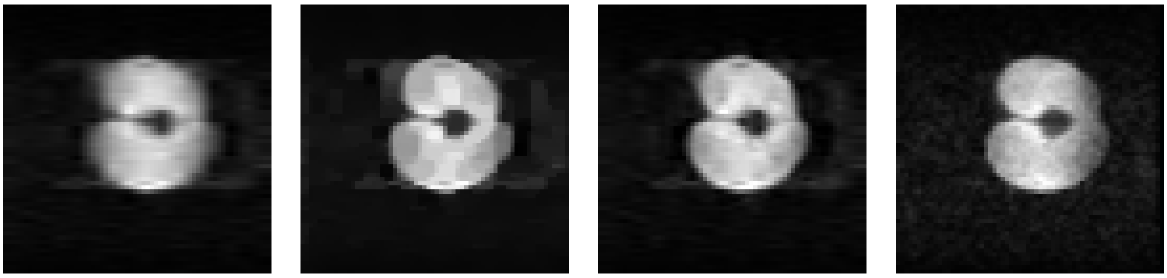
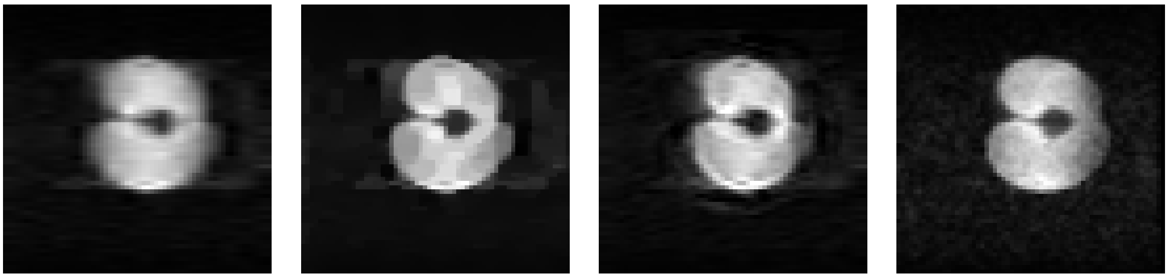

(a) PSNR: 25.06, SSIM: 0.59 (b) PSNR: 24.24, SSIM: 0.62 (c) PSNR: 23.30, SSIM: 0.56 (d)

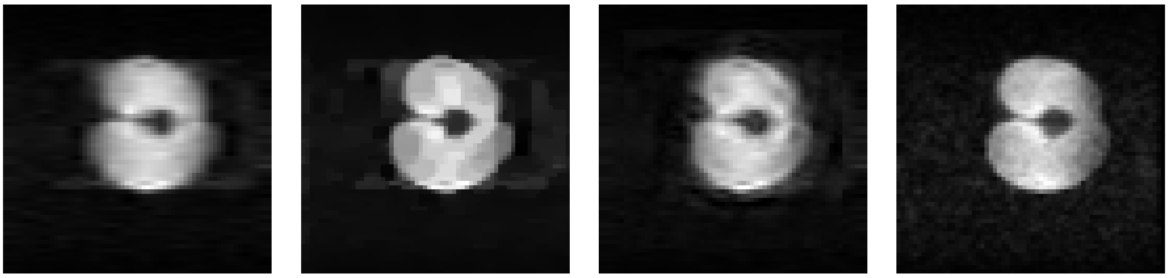
Figure B.2: (a) bicubic interpolation of LR apple scan, (b) SR reconstruction by standard SR, (c) SR reconstruction by SRCNN-nat with multiple blurs, (d) HR apple scan.

B.2. SRCNN-brain



(a) PSNR: 25.06, SSIM: 0.59 (b) PSNR: 24.24, SSIM: 0.62 (c) PSNR: 23.97, SSIM: 0.56 (d)

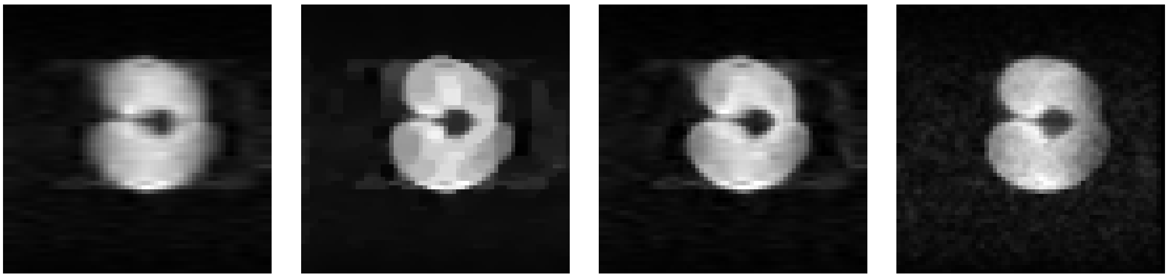
Figure B.3: (a) bicubic interpolation of LR apple scan, (b) SR reconstruction by standard SR, (c) SR reconstruction by SRCNN-brain with fixed blur $\sigma = 1.3$, (d) HR apple scan.



(a) PSNR: 25.06, SSIM: 0.59 (b) PSNR: 24.24, SSIM: 0.62 (c) PSNR: 25.01, SSIM: 0.57 (d)

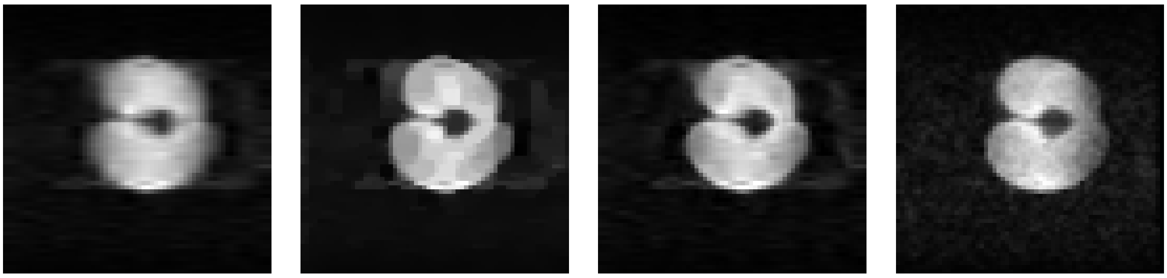
Figure B.4: (a) bicubic interpolation of LR apple scan, (b) SR reconstruction by standard SR, (c) SR reconstruction by SRCNN-brain with multiple blurs, (d) HR apple scan.

B.3. ReCNN-nat



(a) PSNR: 25.06, SSIM: 0.59 (b) PSNR: 24.24, SSIM: 0.62 (c) PSNR: 24.22, SSIM: 0.55 (d)

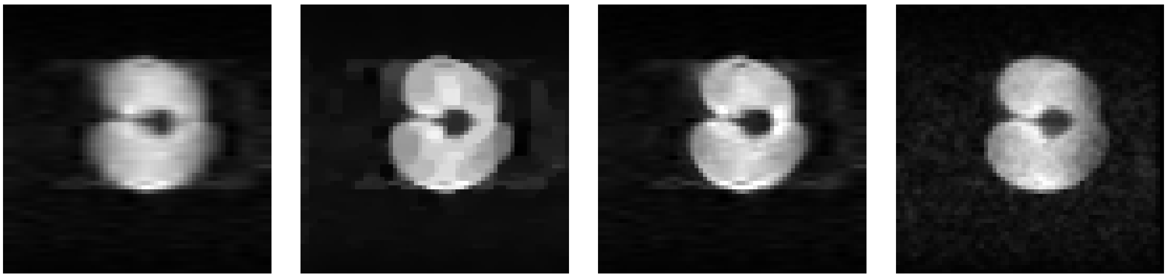
Figure B.5: (a) bicubic interpolation of LR apple scan, (b) SR reconstruction by standard SR, (c) SR reconstruction by ReCNN-nat with fixed blur $\sigma = 1.3$, (d) HR apple scan.



(a) PSNR: 25.06, SSIM: 0.59 (b) PSNR: 24.24, SSIM: 0.62 (c) PSNR: 24.43, SSIM: 0.56 (d)

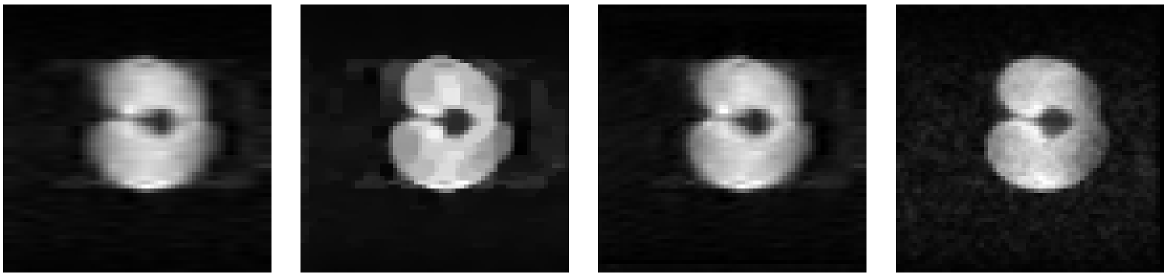
Figure B.6: (a) bicubic interpolation of LR apple scan, (b) SR reconstruction by standard SR, (c) SR reconstruction by ReCNN-nat with fixed blur $\sigma = 1.3$, (d) HR apple scan.

B.4. ReCNN-brain



(a) PSNR: 25.06, SSIM: 0.59 (b) PSNR: 24.24, SSIM: 0.62 (c) PSNR: 23.83, SSIM: 0.56 (d)

Figure B.7: (a) bicubic interpolation of LR apple scan, (b) SR reconstruction by standard SR, (c) SR reconstruction by ReCNN-brain with fixed blur $\sigma = 1.3$, (d) HR apple scan.



(a) PSNR: 25.06, SSIM: 0.59 (b) PSNR: 24.24, SSIM: 0.62 (c) PSNR: 25.01, SSIM: 0.57 (d)

Figure B.8: (a) bicubic interpolation of LR apple scan, (b) SR reconstruction by standard SR, (c) SR reconstruction by ReCNN-brain with multiple blurs, (d) HR apple scan.

C

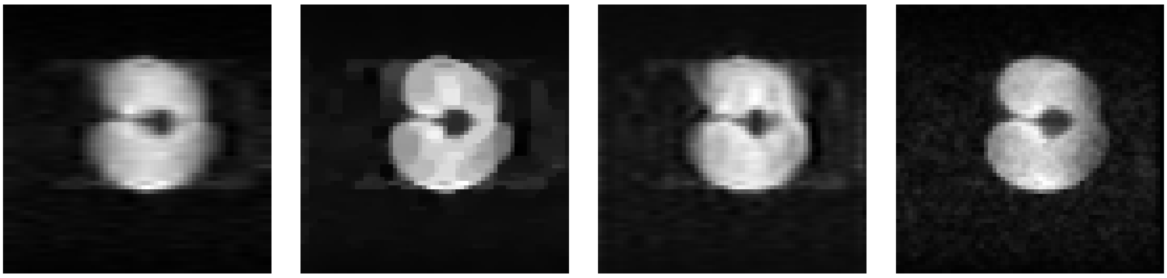
ADDITIONAL RESULTS ON MRI SCANS WITH DEEP LEARNING

In this appendix we present additional results on the apple MRI scans, obtained by first enhancing the LR scans and then combining them together, as anticipated in Section 9.3.2. After experimentally trying different configurations, it was decided to obtain the reconstructions by using TV regularization with $\lambda = a10 = 6.7 \cdot 10^{-9}$.

The reconstructions yielded in this setting are clearly of poor quality compared to both standard SR and the HR scan. Despite this, we notice a trend similar to the one of Chapter 9, with ReCNN-nat and ReCNN-brain with fixed $\sigma = 1.3$ yielding better, but still very blurry and imprecise, reconstructions.

A possible explanation for the failure of this approach is that the networks modify the images by potentially eliminating or transforming some of those details which actually connect the scans between themselves, not allowing standard SR to work properly.

C.1. SRCNN-NAT



(a) (b) (c) (d)

Figure C.1: (a) bicubic interpolation of LR apple scan, (b) SR reconstruction by standard SR, (c) SR reconstruction by SRCNN-nat with fixed blur $\sigma = 1.3$, (d) HR apple scan.

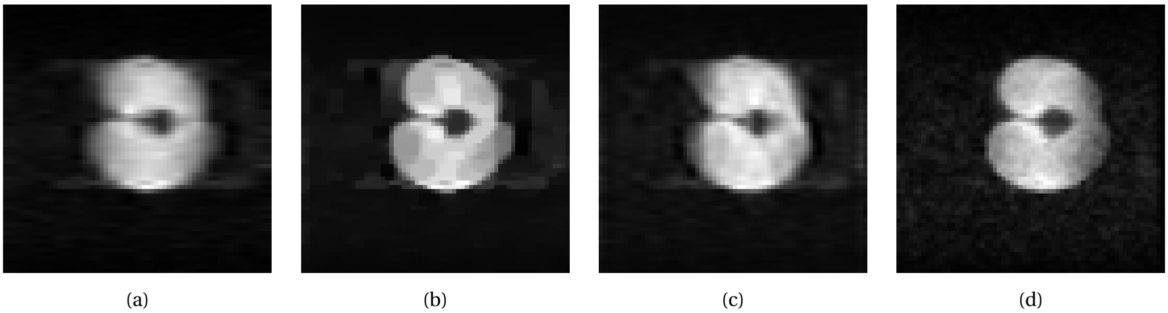

(a) (b) (c) (d)

Figure C.2: (a) bicubic interpolation of LR apple scan, (b) SR reconstruction by standard SR, (c) SR reconstruction by SRCNN-nat with multiple blurs, (d) HR apple scan.

C.2. SRCNN-BRAIN

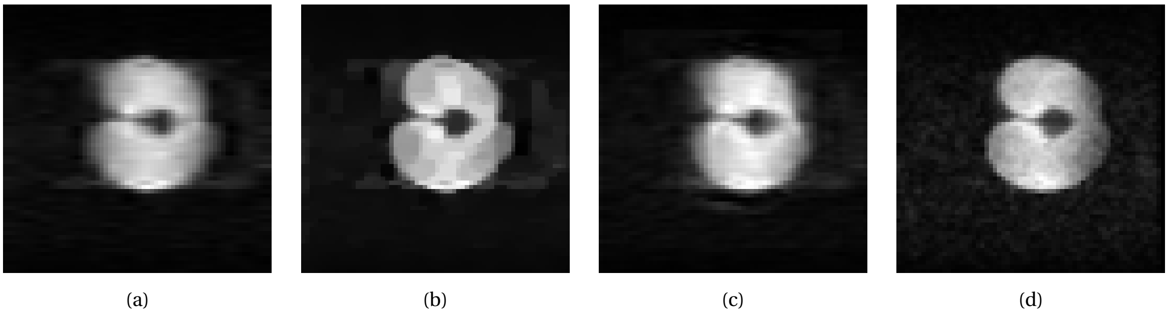

(a) (b) (c) (d)

Figure C.3: (a) bicubic interpolation of LR apple scan, (b) SR reconstruction by standard SR, (c) SR reconstruction by SRCNN-brain with fixed blur, (d) HR apple scan.

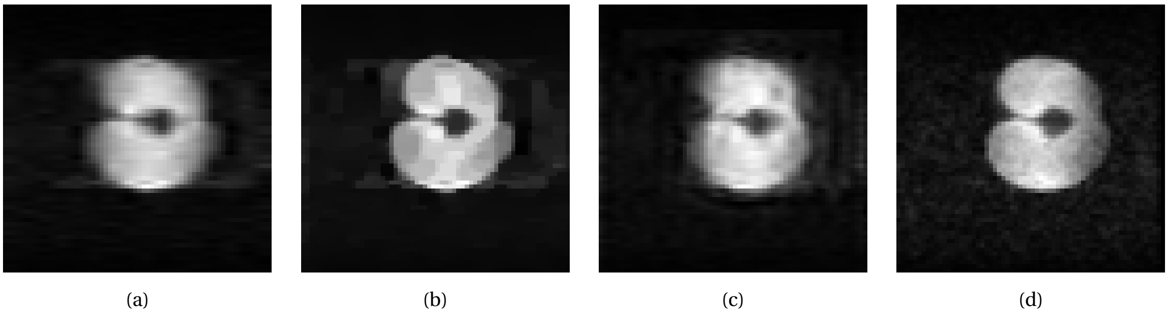

(a) (b) (c) (d)

Figure C.4: (a) bicubic interpolation of LR apple scan, (b) SR reconstruction by standard SR, (c) SR reconstruction by SRCNN-brain with multiple blurs, (d) HR apple scan.

C.3. RECNN-NAT

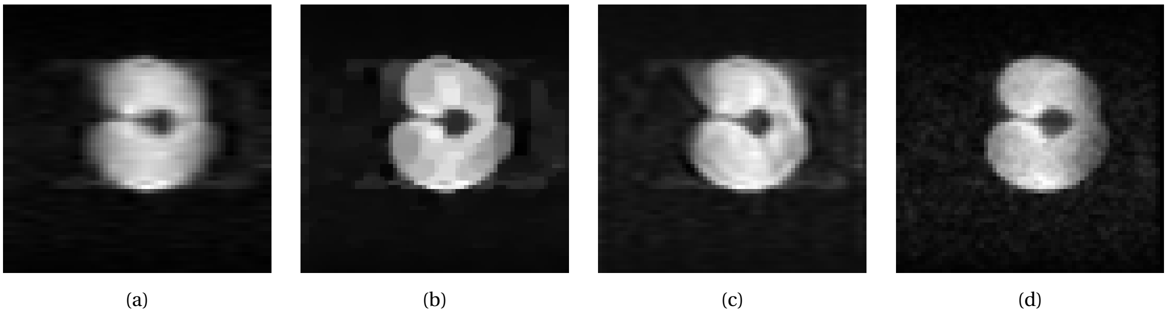


Figure C.5: (a) bicubic interpolation of LR apple scan, (b) SR reconstruction by standard SR, (c) SR reconstruction by ReCNN-nat with fixed blur $\sigma = 1.3$, (d) HR apple scan.

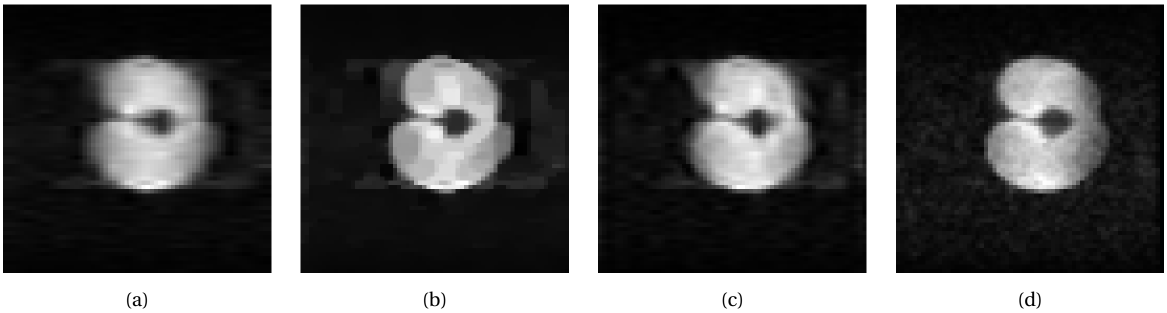


Figure C.6: (a) bicubic interpolation of LR apple scan, (b) SR reconstruction by standard SR, (c) SR reconstruction by ReCNN-nat with multiple blurs, (d) HR apple scan.

C.4. ReCNN-brain

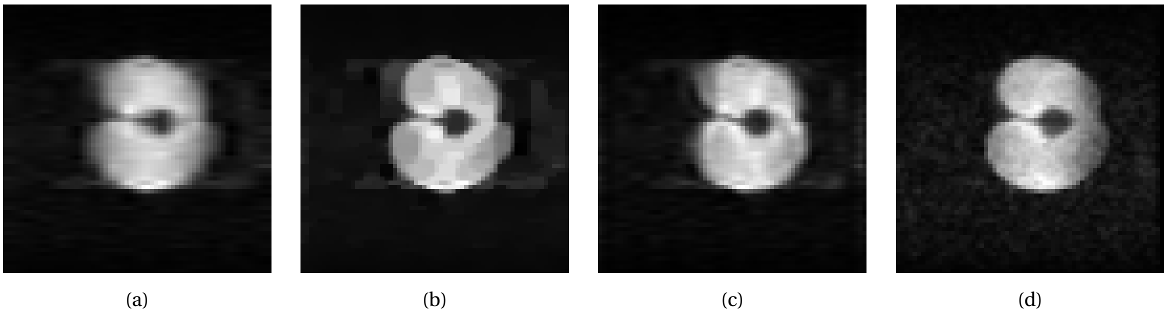


Figure C.7: (a) bicubic interpolation of LR apple scan, (b) SR reconstruction by standard SR, (c) SR reconstruction by ReCNN-brain with fixed blur $\sigma = 1.3$, (d) HR apple scan.

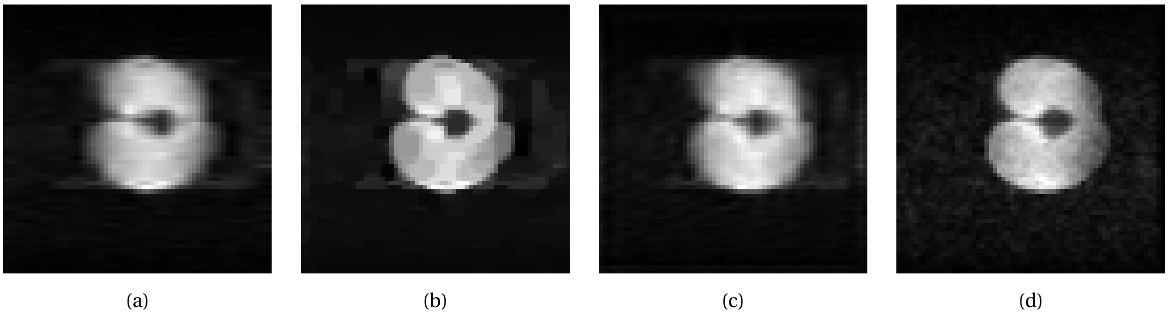

(a) (b) (c) (d)

Figure C.8: (a) bicubic interpolation of LR apple scan, (b) SR reconstruction by standard SR, (c) SR reconstruction by ReCNN-brain with multiple blurs, (d) HR apple scan.

D

ACRONYMS AND ABBREVIATIONS

SNR	Signal-to-Noise Ratio
SR	Super Resolution
SRR	Super Resolution Reconstrucion
DL	Deep Learning
MISR	Multi-Image Super Resolution
SISR	Single-Image Super Resolution
PSF	Point Spread Function
TV	Total Variation
HPD	Hermitian Positive Definite
CG	Conjugate Gradient
CGLS	Conjugate Gradient Least Squares
ADMM	Alternating Directions Method of Multipliers
FFT	Fast Fourier Transform
PSNR	Peak-signal-to-noise-ratio
SSIM	Structural Similarity Index Metric
ANN	Artificial Neural Network
MLP	Multilayer Perceptron
AF	Activation Function
FNN	Feedforward Neural Network
ReLU	Rectified Linear Unit
MSE	Mean Squared Error
GD	Gradient Descent
SGD	Stochastic Gradient Descent
FCNN	Fully Connected Neural Network
SRCNN	Super Resolution Convolutional Neural Network
ReCNN	Residual Convolutional Neural Network
IQA	Image Quality Assessment