Student number n. 0000901857

ALMA MATER STUDIORUM
UNIVERSITY OF BOLOGNA

DEPARTMENT OF INGEGNERIA DELL'INFORMAZIONE

MASTER DEGREE IN ELECTRONIC ENGINEERING

Deep Neural Recovery for Compressed Imaging

Dissertation on Signal Processing

Supervisor

Prof. Riccardo Rovatti

Co-supervisor

Prof. Mauro Mangia
Dottor Alex Marchioni

Presented by

Filippo Martinini

Session 10/3/2021
Academic year 2020/2021

TABLE OF CONTENT

TABLE OF FIGURES

1 Motivations

Since the discovery of the firestone mankind developed fast enough, we invented the wheel, we found a good approximation of π, we created really fast cars, we landed on the moon… We developed fast enough so that we do not feel we actually have so many years of evolution behind us, till nowadays where we have internet, artificial-intelligence-based systems and a too much pollution in the air we cannot continue ignoring it.

At the dawn of civilization man dreamt of mighty gods who could control all the elements and who could interact with death, life and so on. Man created stories and religions based on what they could only imagine, until the day someone did what before was just ‘reasonably impossible'.

That is what always happened: you cannot do something you would really love to, so you make up a tail. The day the tail turns out to be reality, you move on, on something you still cannot do.

Man always chased future by inventing stories. Nowadays we can still hear and read about the long-forgotten stories of the past and laugh about them, but when what is now technology was only magic people were fascinated by it, people imagined the problem, spent hours arguing about the complications and studied it. Stories always gave an input, a kick-off into the minds of great scientists, they gave a target, maybe an impossible one, but they gave a target to the people.

The great sci-fi writer Arthur C. Clarke summarized the topic into 3 laws[1]:

Clarke's First Law: *When a distinguished but elderly scientist states that something is possible, he is almost certainly right. When he states that something is impossible, he is very probably wrong*. (Clarke)

Clarke's Second Law: *The only way of discovering the limits of the possible is to venture a little way past them into the impossible.* (Clarke)

Clarke's Third Law: *Any sufficiently advanced technology is indistinguishable from magic.* (Clarke)

What is going to be discussed in this work is not something concerning the future of humankind, is not something unimaginable, something only Arthur C. Clarke could write, but another little step toward the progress, a thing that had not be created jet and was only waiting for someone to be invented.

What brought Earthmen to we call ‘our society’, is the same that brought me here: an inspiration.

I am convinced that we owe so much, maybe we owe more than what I think, to the imagination of the people who wrote the story, who directed the film, who played the poem, who… that has been inspiring us, from the first to the last human being.

What we, engineers, are, is the link between the present and the future: a future that has already been written by our favourite sci-fi writer. We, engineers, work to hear someone call, one day, ‘reality’, what for us, now, is science fiction.

Without us, engineers, future would be tediously possible, and magic would just remain magic and, at last, Clarke would have never discovered his third law.

Finally, my motivation is a science fiction future where positronic robots live alongside humans and Earth is not the only inhabited world in the universe and people can live up to 400 years.

What this project discusses is not directly linked with robots and extended life, but I like to think that it is another step, even if it is a short one, toward Asimov ‘novels.

2 Introduction

This introduction chapter will be split in the following way:

1. Introduction
2. MRI
3. Compressed Sensing
4. Image Processing
5. Artificial Intelligence
6. Neural Network

This sequence of targets will introduce step by step the state-of-art method that this work is going to improve.

In chapter two the state of the art will be discussed and explained in depth.

In chapter three the new contributions will be discussed.

In chapter four the results of the contributions will be shown.

2.1 Introduction

The story of electrical engineering begins when, around 3700 years ago, someone used to play with eels, to pet a cat's fur and to rub amber. These ancient engineers, having no materials but their dreams used their own body as a conductor. As the work of an engineer has always been a hard one, especially in those times, nobody wanted to apply for it and electricity remained quite a curiosity for many millennia and nobody cared of studying it.

In 1700 glimpses of a resting curiosity emerged, and some bizarre inventions came out of the cylinder appearing as magic to whom didn't know of the third law of Clarke [1]. The century continued flawlessly until a real invention 'shocked' the world: Volta invented the battery, it was 1800 and the world could not discard electricity anymore, it became clear that electricity was useful, and it was in the same way clear, that it was possible to make money out of it and the researchers appeared.

In the nineteenth century the interest on developing new tools using electricity grew rapidly and at the beginning of 1900 incredible discoveries, as the Marconi's radio, were already commercials.

The story of transistors dates to 1947/1948 when Bardeen, Brattain and Shockley invented the first types of transistors. They could probably not imagine, but they moved humanity into the future, probably further than almost everybody body else before them. That day they announced

their discovery the ‘transistor era’ became real. Transistors found, in a short time, an incredible interest. A large group of researchers grew in few years, improvements and applications were close to see the light.

Everything started few lines ago, it seems impossible, but now it is 20 June 1969: and humanity lands on the moon. Every time it had to seem a short step, in particular for the guy who liked to fish and pet his eels. But in the end, a small step became part of a huge jump: such a huge jump that we reached the moon!

Since the world showed increased interest in science, it has never been only technological inventions ending in themself, but it has been a chain of improvements in every aspect. The developing of new technologies created new possibilities for the market, economy systems grew and wealthier states guaranteed safer lives[2]. Medicine, following this trend, engulfed the scientific method and obtained great improvements. With an increasing life expectancy, and a supporting strong economy, always more people gained the possibility to study and to apply new ideas to the world. Market, social life, politics, literature, medicine… everything, received contributions and gave contributions.

Science created a closed loop feedback: a society inside which we still consume, live, study and fuel it.

Of course, there is an important implicit consequence: the degree of improvements must always grow. We live in an old society, a society that has always grown. We reached a high level of complexity in almost every field of research. At the point where we are, innovative ideas and complex schemes must always be tried to solve new problems.

When doctors, at the beginning of 1970, reached a high level of understanding of the human body they faced the problem of: ‘What to do, now, to improve?’. Doctors by themselves could not find out what was the next step, but together with engineers they achieved the construction of the first MRI. It was a huge problem of great complexity and they had to tackle it, they had to improve the state of the art and at the end they succeeded.

It happened, around 2010, that while MRIs were working fine and doctors started using the new devices, that the MRIs themself showed their limitations. Once more the world asked for more developments, and once again engineers replied, with a new revolutionary idea: compressed sensing. Compressed sensing achieved the goal and perfectly adapted.

It is 2021 now and we have been called for another improvement, can we still do better? The answer is Yes, well, the answer is always Yes, it has to be Yes.

This work draws its new key element from a recently developed field of study: Deep Learning, applying it to the well-known compressed sensing method.

This work aims to improve a novel algorithm for accelerating MRI acquisition via compressed sensing using new deep learning model that specializes on the details of the class of the acquired data.

2.2 MRI

Magnetic resonance imaging (MRI) is a medical imaging technique and diagnostic tool in radiology based on the Nuclear Magnetic Resonance (NMR).

MRI is widely used worldwide as one of the main instruments to have accurate images of sections of the body. Relying on MRI doctors can guarantee to the patients, in depth analysis of their health problems.

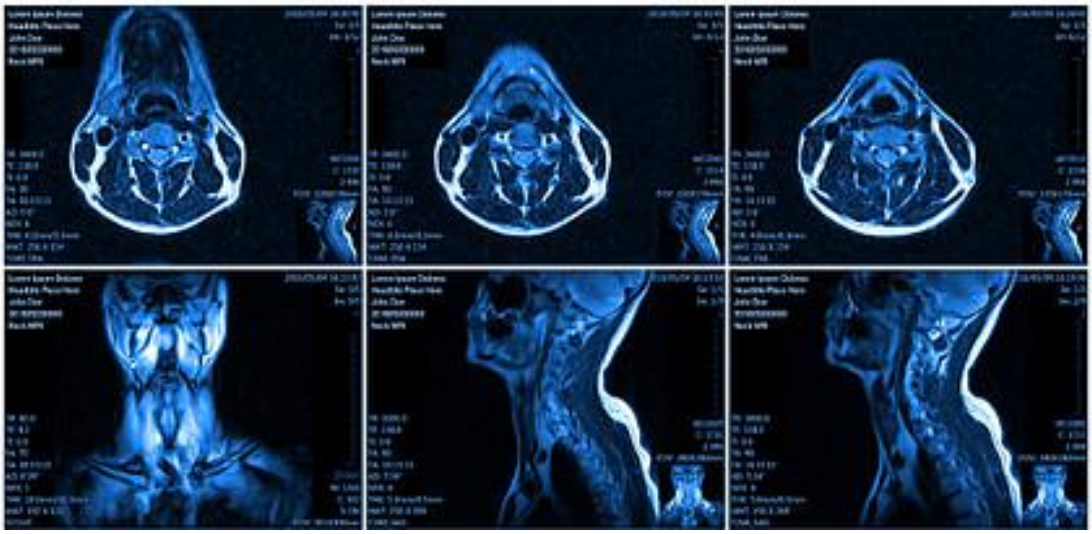


FIGURE 2-1 - EXAMPLE OF MRI SCAN

MRI is a main subject of research in the world, in 2020 7782 paper have been published on 'Scopus' containing the keyword 'MRI', of which 997 are classified as engineering research [3][available at 01/01/2021].

MRI market is constantly increasing in many countries, being MRI a main technology for medical purposes for both public and private health. For example, between 1990 and 2007 number of MRI per one million inhabitants varied from 1.8 to 15.3 in Finland[4].

NMR is the phenomenon that denotes the induced current on a magnetic coil obtained by emission of a magnetic current of an object on which another magnetic field was previously applied.

1. A sample has been placed in an ideal uniform magnetic field produced by a magnetic coil.
2. A current is applied for a short time to the coil, an oscillation of the magnetic field occurs in the sample.
3. Some particles have interesting properties related on absorption of energy given by a magnetic field. If the sample is, for example, hydrogen nuclei, the hydrogen absorbs some energy.
4. If the sample has absorbed energy, some of this energy will be emitted as an electromagnetic wave which frequencies lies in the radiofrequency (RF), in the interval of $[3K, 300G]Hz$.
5. The coil will be affected by the same magnetic field produced by the sample and a current will appear on the coil, oscillating at the same frequency of the RF signal.

The current that is measured is called NMR signal.

More in-depth explanation:

Relaxation is the process by which any excited particles relax back to its equilibrium state, after a magnetic field gave energy to it. It can be divided into two separate processes: Longitudinal relaxation (T1) and Transverse relaxation (T2).

Every tissue has its relaxation times. In MRI is possible to measure both T1 and T2 to understand what element is hidden inside the body that is analysed. For example, it known that: T1 is always longer than T2. Liquids have very long T1 and T2 values while dense solids have very short T2 values.

A spin echo is produced by two successive RF-pulses (usually 90° and 180°) that create a detectable signal called the echo.

Tissues have different T1 and T2. Signal intensity depends on these parameters: one aims to maximize the contrast between tissues focusing on one of these parameters.

To detect the location of NMR, MRI has three gradient coils, one for each orthogonal axis. A gradient coil produces a small magnetic gradient (compared to the main magnetic field). By producing a gradient is possible to produce the exact magnetic field to be absorbed by the

particle only in a small section of the body (such as 1-2 mm thick). By adjusting the intensities along the three axis is possible to point to a region of the body and scan only such a region.

MRI uses a frequency encoding to return the output: MRI generates a precise frequency magnetic signal and reads the relaxation time of the slice of the body which resonates. This means that the output of MRI is not a normal image but is the Fourier transformation of the image. To obtain the final image is necessary to apply FFT algorithm to the MRI output.

MRI has been studied since its first invention with high interest by the scientific community and many improvements have been done. Here is a list of the pros and of the cons that still nowadays require investigations:

Pros:

1. non-invasive and 3D modelling of the body.
2. no ionization radiations.
3. nice spatial resolution.
4. can scan a wide range of internal tissues.

Cons:

1. is an expensive machine.
2. long-time are required to scan, that leads to artifacts in the image.
3. if patient has metals inside his body, MRI cannot be done due to interaction of metal with magnetic field [5].
4. when on is very loud and is not designed to be comfortable for the patient.

Most of the above introduction to MRI has been inspired by [6].

The biggest usage problem is the long scan time. One of the first researcher who studied the problem finding a good algorithm for speeding up the acquisition is Griswold et al. with his paper [7] where he proposed a fast configuration for autocalibration acquisitions.

2.3 Compressed Sensing

Compressed Sensing (CS) [8] is a recent area of study, it aims to shorten the time duration of the acquisition, the power consumption or it aims to overwhelm the hardware limitations of a sensor by reducing the sampled data and by reconstructing the incomplete data in another moment.

A classically correct acquisition of a data requires the sampling of a source of the signal with a sampling frequency f_s at least higher than two times the maximum frequency f_{max} of the signal itself. This rule is known as the Nyquist theorem and is one the first fundamental theorems every electronical engineer studies. Respecting the Nyquist theorem is usually so important that before acquiring the signal, the signal itself is filtered to cut out every frequency that would escape the limit of the $\frac{f_s}{2}$.

The demonstration that $f_s > 2f_{max}$ to correctly reconstruct the signal is intuitive:

Considering a sine wave of frequency $f_{sine} = f_{max}$ and sampling it by a frequency that do not respect the Nyquist theorem will lead to obtain a set of samples that if reconstructed return a different signal than the sine wave: the reconstructed signal will have more and different frequency components than the expected ones.

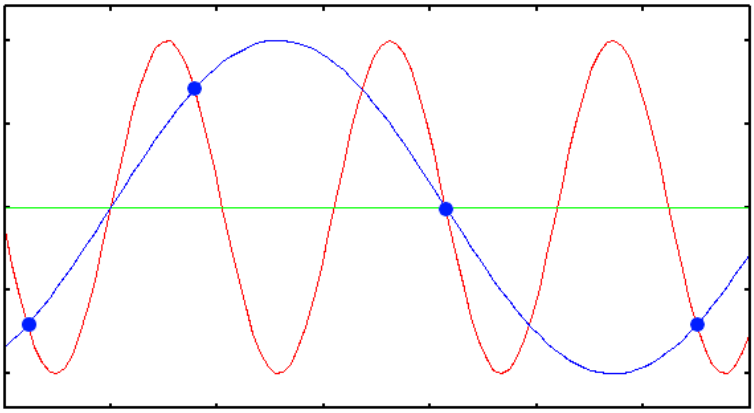


FIGURE 2-2 - EXAMPLE OF UNDER-SAMPLING

When a signal is sampled using a frequency f_s lower than $2f_{max}$, we address the phenomenon as 'under-sampling'. If under-sampling is carried on a complex signal (not just on a sine wave) frequencies will interfere in reconstruction leading to errors known as 'aliasing' errors. The phenomenon itself is called 'aliasing'. Even if 'small number of linear measurements' are available, is possible to reconstruct the data trying to minimize the error[9].

The key idea of CS[10] is to under-sample, helping saving energy and time, obtaining data that cannot be normally reconstructed, but that require more advanced algorithms to be reconstructed.

Mathematically the under-sample operation is a simple linear operation[11], given a vector $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^m$ called 'sample' and 'compressed sample', with $n < m$, the under-sample matrix is a matrix $A \in \mathbb{R}^{n \times m}$:

$$y = Ax$$

This is also called ENCODING. Computational cost and complexity are low, allowing to obtain an easy task. The problem lies in the reconstruction of x given y, that is called DECODING. The reconstruction is critical because A is a rectangular method and A^{-1} allows infinite different solutions.

A naïve way to obtain x could be to compute the pseudo-inverse of A, but this method could not guarantee to retrieve the original sample. Instead of working on the minimization of $\|x\|_{L_2}\ such\ that\ \|y - Ax\|_{L_2} < \eta$, CS works on the hypothesis of sparsity of the signal x and exploits it.

Assuming $x \in \mathbb{R}^n$ is k-sparse, it can be represented with a sparse representation $x = D\xi$ where ξ has at max k non-zero elements and the encoding can be written as:

$$y = B\xi$$

Where $B = AD$.

ξ is DENSE, meaning that even if it is smaller compared to x it contains the same information of x. The problem requires the knowledge of the sparsity k and of the space D (also called 'dictionary') where the signal is sparse, that means that is necessary to study the signal and extract some features out of it. An example could be $x = \sin(2\pi f t)$ that has a sparse representation in the frequency domain: $x = F\xi$ where F is a Fourier operator and ξ is vector with a unique non-zero element that is the frequency f of $\sin(2\pi f t)$.

Given these hypothesis CS searches the sparsest vector $\hat{\xi}$:

$$\hat{\xi} = \text{argmin}\left(\|\xi\|_{L_0}\right)\quad such\ that\text{: } \|y - B\xi\|_{L_2} < \eta$$

This problem is NP-hard [12] add citation on NP-hard, so it is requiring to solve, so the request is lightened by requiring:

$$\hat{\xi} = \text{argmin}\left(\|\xi\|_{L_1}\right)\quad such\ that\text{: } \|y - B\xi\|_{L_2} < \eta$$

This is possible only under certain conditions, called 'Restricted Isometry Constraint' (RIP)[13] [14] that requires the matrix B to be a 'quasi-isometry'. The great power of CS is that it guarantees that matrix B respects the RIP condition whenever:

$$A\ has\ A_{i,j} \sim N(0,1)\ with\ i = 0, \dots, n-1; j = 0, \dots, m-1\ and\ m = O\left(k \log\left(\frac{n}{k}\right)\right)$$

Without any constraint on D. Once obtained $\hat{\xi}$ the original signal is reconstructed by computing $\hat{x} = D\hat{\xi}$.

The given introduction to CS gives the guarantee that CS works fine for a sparse vector and it is robust for generic signals. CS can be further improved for real application by specializing the method and adapting it in two main ways:

1. Adapting CS to a particular class of signal by improving the under-sampling (encoding).
2. Adapting CS to obtain a better reconstruction, given the under-sampled signal (decoding).

It has been demonstrated that *A* can be also a sub-gaussian matrix and respects RIP condition, meaning that the ENCODING procedure can be even easier than it already has been shown. An interesting case is when *A* is a random under-sampling of *I* (identity matrix): *B* is simplified but the generality of the classes of sparse signals on which CS now works is reduced. CS can be adapted to specific classes of signals by the use of different and more convenient *B* that are not, in general, valid.

CS can simplify the acquisition, but at the same time can adds complexity to the reconstruction, making it an ideal method to apply in contests where resources at acquisition time are low while the receiver has computational power.

CS is a huge helper for MRI acquisitions: CS relaxes the acquisitions of MRI to speed up the scans and helps preventing errors caused by accidental movements of the patient and reducing the cost of procedures.

CS algorithm applied to MRI have shown acceleration up to 80% [15] and made feasible new medical treatments such as the first pass cardiac perfusion MRI [16]. The same work demonstrated feasibility of 8-time acceleration in vivo imaging.

MRI related to CS is a main subject of research in the world, in 2020 34 engineering papers have been published on 'Scopus' containing both the keyword 'MRI' and 'compressed sensing'[3].

To under-sample in an MRI machine means to stimulate the region of the body with less frequencies and obtain a less dense *k*-space representation of the body. An under-sampled *k*-space image do not respect the Nyquist theorem: CS must prevent the aliasing phenomenon from reducing the quality of the reconstructed image.

Jong Chul Ye in his paper 'Compressed sensing in MRI: a review from signal processing perspective' [17] delineates the evolution of many algorithms through the last decade. Some of the most important works, who reached interesting results:

LORAKS [18] demonstrates that *k*-space data can also be mapped to low-rank matrices when the image has limited spatial support or slowly varying phase and obtain interesting results. ALOHA [19] is a generalized approach of LORAKS that continues the research and widen the usability of the method.

These two methods reached considerable results and are usually used as benchmarks.

2.4 Image Processing

Image processing is the study of the images from a signal processing point of view. Image processing is a method to perform some operations on an image, in order to get an enhanced image or to extract some useful information from it. Image processing is an incredibly important discipline that is growing fast but it is already a wide area of study. For the sake of clarity in this work only 2D images having 8-bit quality will be treated.

In image processing 2D grayscale images are treated as tensors with $shape = (1, n, m, 1)$ where the first dimension $shape[0]$ means that the image is 2D, the second and the third dimensions, $shape[1,2]$, stand for the $width,\ heigh$ of the image (number of pixels) and the last dimension $shape[3]$ stands for the number of channel of the image: if $shape[3] = 1$ the image is a grayscale image, if $shape[3] = 3$ the image is an RGB image.

Usually every pixel (value of the tensor) of the image is a value that lives inside the natural set $\mathbb{N}$ in the range $[0, 255]$, but the values are usually normalized between $[0,1]$.

Image processing has many and variable targets, two of its major aims are:

1. Image manipulation.
2. Features extraction.

A classical algorithm of image manipulation is the blurring effect, used to reduce details and noise.

The gaussian blur consists in a 'sliding window' of dimension (l, k), which elements are the result of the gaussian function $G(x, y) = \frac{1}{2*\pi*\sigma^2} e^{-\frac{(x^2-y^2)}{2*\sigma^2}}$, that slides all over the image and that returns for every pixel of the image the result of the 2D discrete convolution of the window with the current window of pixels the sliding window is sliding on.

A classical task of feature extraction algorithms is the ‘edge detection’ task.

Edge detection searches the part in the image that a human could consider the edges of shapes. An image with its values in the continuous range $[0,1]$ is taken as input and the output is another image with the same dimensions of the input image but whom pixels could only assume two values: $(0,1)$, where 0 represents the edge and 1 represents the background.

Edge detection[20] is only a branch, one of the first and most important, of the field of image processing, but is a significative example that shows how the subject is constantly growing and updated.

Image processing is used in many applications, also in MRI, to filter and improve the reconstructed image.

It is possible that both a gaussian filter and edge detection will be applied to the MRI image to get rid of some noise and then to extract the position of some interesting portion of the image (e.g., the edges of a broken bone).

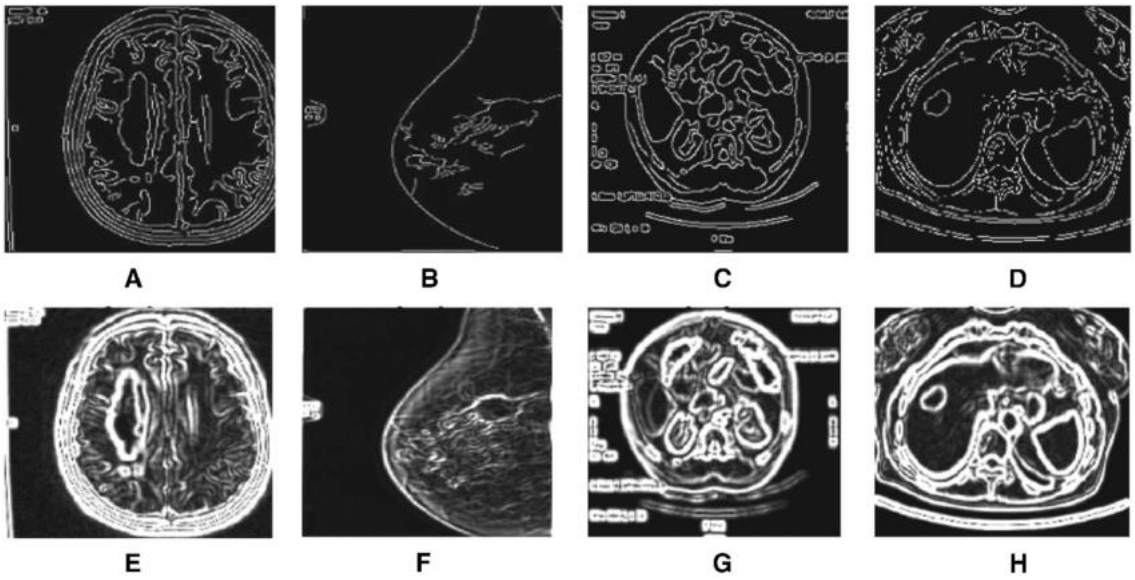


FIGURE 2-3 - EDGE DETECTION ON MRI

CS also makes wide use of image processing algorithms to improve its performances when it aims to reconstruct under-sampled image signals[21].

2.5 Artificial Intelligence, Machine Learning

Artificial intelligence (AI) is a field of research that aims to create ‘intelligent’ algorithms.

Defining what an intelligent machine is, is not an easy task, there exist papers whom purpose is to investigate the complexity of the meaning of AI. A recent published paper that analyses the problem is ‘On Defining Artificial Intelligence’. The paper proposes its definition of AI as “adaptation with insufficient knowledge and resources” asserting that “this definition sheds light on the solution of many existing problems and sets a sound foundation for the field”[22].

For the sake of clarity, we break free from some mind-blowing discussions on the meaning of AI and assume the term ‘intelligent’ wants to classify a set of methods of developing algorithms. ‘Intelligent’ is the method that adapts to the problem, finding the best set of parameters by itself. When applied, an AI algorithm, does not need to be supervised to work at its best, but it automatically finds the optimum.

The real revolution of AI started only recently, when the hype for a particular branch of AI, called Machine Learning (ML), was raised by the new possibilities offered by the development of fast GPUs [23] and new architectures of Neural Networks (NN). The most important competition for AI, a classification task based on the recognition of images, was created and rapidly became a benchmark for researchers to test their new inventions[24].

ML is the field that studies how a model can learn how to specialize itself on the data it is going to work with. ML studies models that need to be trained and only once they are trained can work. We call the first step ‘training’ and the second ‘inference’.

During ‘training’ a model adapts itself to the data (in training we call ‘training set’ the set of data the model uses to train) that must be coherent with the data the model will work with. During training the model modifies itself to adapt to the data. Training can be done in several ways; the easiest way is the ‘unsupervised training’, it consists in giving the dataset to the model without more information. As an alternative to unsupervised, training in a ‘supervised’ fashion means to add information to the samples of the dataset, such as labels that describe what the data represents.

During ‘inference’ the model works with unseen data and returns its outputs based only on what it has already learned during ‘training’. Inference is when the model is used to solve unresolved problems.

An example of unsupervised learning is the training of a model that aims to represent the same data received in input after the model modifies it, for example, after the input has been compressed the model learns how to decode it. These models are called ‘autoencoders’ [25] and they do not need additional information, but data itself is enough.

The key point is that, in this example, the ML model learns by itself how to both encode and decode data, and it searches the encoding/decoding that brings the best performances.

An example of 'supervised' learning: the training of the model that aims to recognize images. If the model aims to learn, for example, what is an MRI 'with tumours' and 'without tumours', data must be labelled to specify if the sample is an 'MRI with tumour' or not[26].

ML models are dependent on what data they see, in fact one of the biggest problems when dealing with ML models is to construct a good dataset containing a good amount of data, good quality data, and good labels too (if needed). If a ML model has been trained with knee MRI images, it will not be able to recognize a brain MRI and its characteristics.

Many different models of ML have been studied, such es 'decision trees', 'Bayesian models' or 'K-nearest neighbour', but the model that, in general, is nowadays more investigated is the 'Neural Network model' (see [24] for some results on ImageNet results).

Returning on MRI approach, now is possible to distinguish between adapted CS and adaptive CS, meaning that adaptation is performed at design-time considering the class of signal to acquire and not at runtime on each signal instance[27]. What AI permits to achieve is CS adaptation and the most powerful weapons it has are the Neural Networks[28].

2.6 Neural Networks

Neural Networks (NN) are nowadays the most studied ML architectures, there has been a constant growing interest on the topic since the last ten years[29].

A neural network, for definition, has at least two layers: one 'Input' layer and one 'Output' layer. A neural network can have more layers in between the input and the output layers, these are called 'hidden' layers.

A layer is a general structure, the most used, and basic layer is the 'Dense' layer: a dense layer is a 1D collection of neurons.

A neuron is a fundamental unit of NNs that takes as input a 1D vector x_{in} of dimension N and returns $x_{out} = \sigma(\sum_{i=0}^{N-1}(x_i * w_i) + b)$, where x_i is the i-th element of the input vector, w_i is the i-th element of the 'weights' of the neuron and b is the bias. A neuron simply multiplies every element of the input vector for a weight, adds a bias element, and finally applies an 'activation function' σ.

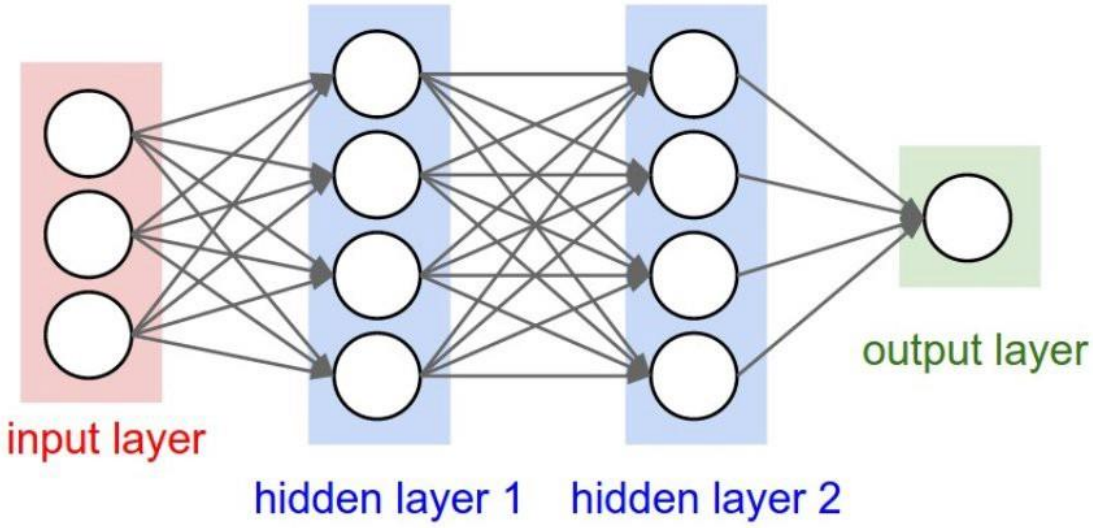


FIGURE 2-4 - SCHEME OF A NEURAL NETWORK MADE WITH DENSE LAYERS

In *Figure 4* is shown a classical 'Sequential' neural network, built up with only Dense layers, where every layer's output is at same time the input of the following layer.

The key characteristic of a neuron is that it can modify its weight and bias values during training time and adapt to solve the task the NN model aims to resolve.

An activation function can be whatever $\mathbb{R}^1 \rightarrow \mathbb{R}^1$ function, usually a non-linear function is used to generalize the behaviour of the NN.

To train, a ML model, exploits the 'backpropagation' and the 'gradient descend' (GD) algorithms. Assuming a simple NN is constructed as a sequence of i Dense layers, the method is divided in the following steps:

1. Initialization of i: $i = number\ of\ layers\ of\ the\ NN$.
2. The NN model takes as input a fixed number of samples from the dataset, the set is called batch. The NN computes the outputs.
3. The NN computes the gradient of the error between the 'outputs' and the 'target outputs' (that must be provided in 'training'). The error is obtained using a 'loss function'.
4. Knowing the gradient, the NN knows the intensity and the direction of the output error: gradient descend is applied to modify all the weights of all the neurons of all layers.

During 'training' this method is repeated for all the elements in the dataset so that the model can see as many provided data as possible, this event is called an 'epoch'. The training is repeated for several epochs on the same dataset, that is shuffled to guarantee generality at the

beginning of every epoch. Usually, the training ends when the error does not descend anymore, meaning that the NN has learned what it could.

Many parameters can be modified to change the training procedure:

1. Loss function: the function used to compute the error. For example, it could be the $L_2\ norm$ of the error.
2. Batch size: number of samples of the dataset used during training at the same time.
3. Callbacks: a set of functions that handle many functionalities during training.
4. Learning Rate (LR): the step the gradient descend takes when modifying the weights, a high LR means a fast learning, a small one means a slow learning.

All the above parameters are in first approximation independent from the dataset and from the NN, but all need to be tuned for a good training.

Here are some examples of what could happen if the parameters are not well tuned:

1. A bad selection of the learning rate could lead to a 'too slow' learning that requires too many epochs to learn, or it could lead to a training that tries to learn 'too fast' and does not learn at all.
2. A bad selection of the batch size could lead to bad generalization, and the gradient computed during backpropagation could have problems.
3. A bad selection of the loss function could lead to a bad training that does not learn the most interesting features or that leads to overfitting.
4. A bad selection of the callbacks leads to uselessly long training, to overfitting and to 'unsafe' training.

'Overfitting' is a phenomenon in which a model does not learn to generalize but can work only with data it has seen during training. An overfitted NN is useless, because at inference time it does not correctly work with unseen data.

In the last decade, scientific community brought to life many different NN architectures, one of the main goals achieved is the invention of a NN called 'convolutional neural network' (CNN) that outperforms all the other models when the data are images.

CNN maintains the same sequential structure of normal NN, but instead of having Dense layers it relies on 'Convolutional', 'pooling' and 'up-sampling' layers.

A convolutional layer [30], instead of having $'N'$ neurons, has $'filt\ number'$ filters. Every filter has the same shape $kernel\ size = (l, k)$, typically $kernel\ size = (3, 3)$. Every filter

works as a sliding filter: the kernel is moved along the two dimensions of the input image and for every movement a dot multiplication is done between the sliding window and the actual window. An activation function is applied at the end of every dot product. This operation returns an image of dimension (n', m'). An image is returned for every filter, at the end, the layer's output has $shape = (n', m', \ filt\ number)$. n' and m' depend on the boundary conditions applied to the image when the window slides near the edges of the image.

Every weight of every filter can be adjusted during training via backpropagation.

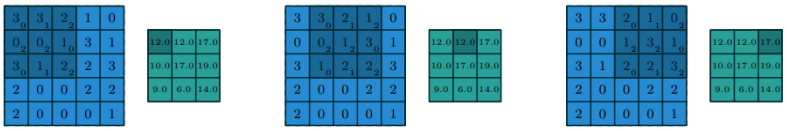


FIGURE 2-5 - EXAMPLE OF CONVOLUTIONAL LAYER

A 'pooling' layer[31] is a non-linear down-sampling function used to reduce the dimension of the input. 'Max pooling' is the most used pooling function: it splits the input image into a set of non-overlapping squares and for each such sub-region, keeps only the maximum value.

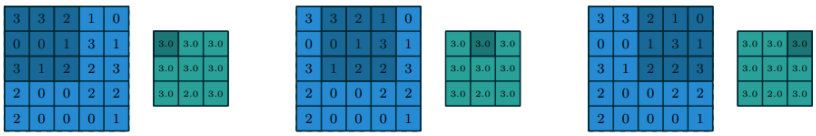


FIGURE 2-6 - EXAMPLE OF POOLING LAYER

An 'up-sampling' layer works in the reverse fashion of the pooling layer: it applies a non-linear dimension augmentation function to the input image. For every pixel, the up-sampling function returns a repeated sequence, of square shape, of the pixel. The section will be combined via a non-overlapping concatenation with the other cells, returned by the function.

CNN is extremely powerful because with its filters it extracts the so called 'features' of the image.

A classic CNN is structured in sequential way where every ‘convolutional’ layer is followed by a ‘pooling’ or by an ‘up-sampling’ and every ‘pooling’ or ‘up-sampling’ is followed by a ‘convolutional’ layer.

CNN has great potentialities. With three or four layers a CNN can recognize handwritten digits, with twenty-five convolutional layers a CNN can distinguish between human faces.

Many competitions born and revealed, during the last decade, what was the best NN. ImageNet, probably the most important, competition[24] revealed what now are considered famous NN, that wrote the story of modelling NN’ architectures.

Here are some of the most important NNs that reinvented the art of building NNs:

1. AlexNet[32] is one the firsts deep CNNs.
2. VGG[33] is the ‘natural evolution’ of CNN, it was deeper, it used more data and it was more organized.
3. GoogleNet [34] is a more generalized and autonomous CNN inspired to VGG.
4. ResNet [35] solves the problem of the ‘vanishing gradient’ encountered in the previous architectures in a smart way: by the introduction of a new special layer ResNet enables CNN to grow deeper and more accurate.

The biggest differences between the cited models is how the smallest unit (which is repeated throughout the network) is designed:

1. In AlexNet there is not any fixed pattern. The convolutions for each layer are decided experimentally.

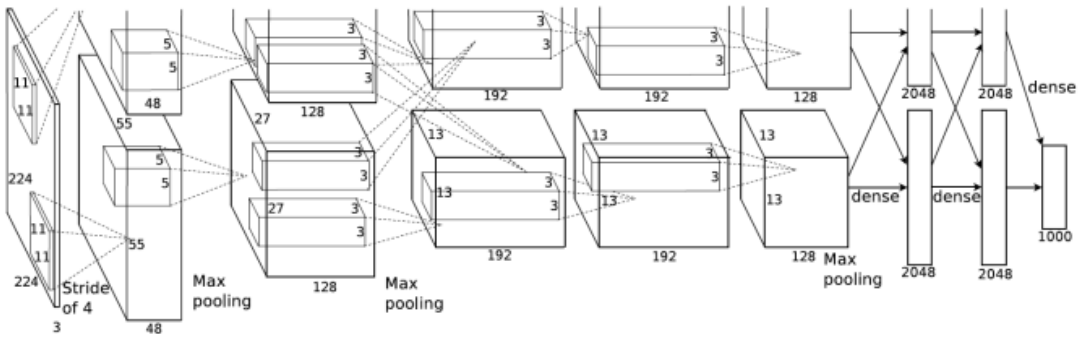


FIGURE 2-7 - ARCHITECTURE OF ALEXNET

2. In VGG the smallest unit is made of two convolutional layers and a pooling layer. The block is repeated multiple times, keeping a constant convolutional kernel of (3,3).

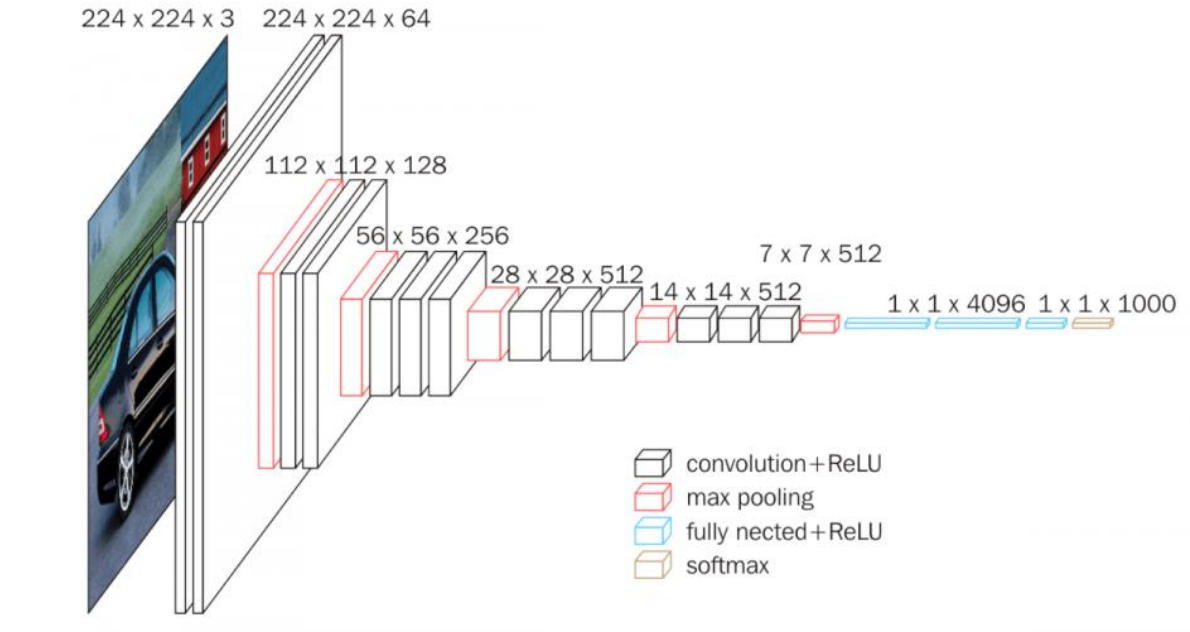


FIGURE 2-8 - ARCHITECTURE OF VGG

3. In GoogleNet the basic structure is called 'inception module'. The inception module repeats convolutions with different kernels and filter sizes ending with pooling layer. GoogleNet automatically figures out what is the best combination of convolutional layers.

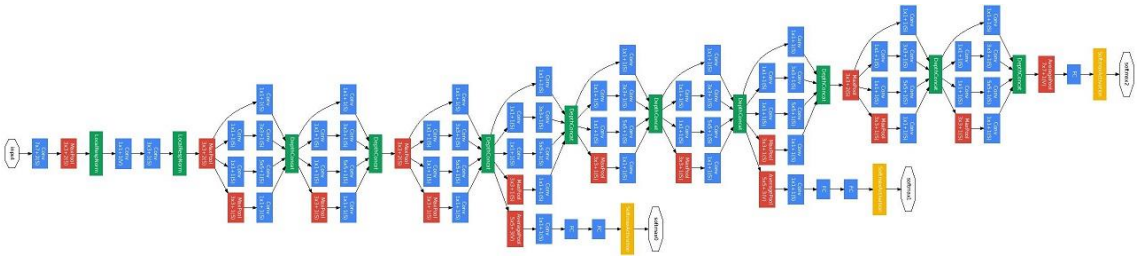


FIGURE 2-9 - ARCHITECTURE OF GOOGLENET

4. ResNet, introducing the 'skip connection' between layers, creates a new concept of NN: the ResNet propagates the residual of the result instead of the result itself.

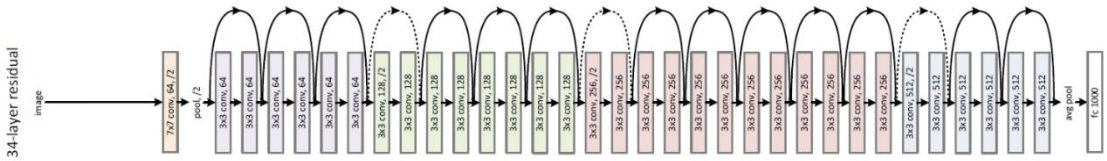


FIGURE 2-10 - ARCHITECTURE OF RESNET

CNN found high interest in the medical community, because of their natural knack on feature extraction, CNN immediately found application in 'biomedical imaging processing'. Interesting results have been reached using AI in medical applications.

In their work Jose Manuel Ortiz-Rodriguez et al. on 'breast cancer detection by means of artificial neural networks' [36] demonstrated how with CNN is possible to outperform the human decision making, scoring more than 95.8% in breast cancer detection.

As NNs today are applied to solve all sorts of applications, NNs have been applied also to MRI image acquisition and reconstruction. CS, that has already shown a strong adaptability for MRI tasks, can be reinterpreted with a machine learning approach[28]. Standard CS approaches guarantee reconstruction on a wide range of signals, that is a fundamental theoretical consideration, but when it comes to implement CS in real word tasks (e.g. MRI) is possible to adapt to specific signals to exploit at best the 'sparsity' of the data.

As already pointed out the CS problem can be split in two parts:

1. Decide how to under-sample: which points to take, and which points to discard.
2. Reconstruct the under-sample image, facing sparse data and aliasing.

Han Y. et al. proposed an interesting approach to improve the CS-MRI reconstruction by training a CNN in their work 'Deep learning with domain adaptation for accelerated projection-reconstruction'[37]. The NN task is to clear the under-sampled image from aliasing problem to augment the image resolution.

It has been shown in the work 'Accelerating magnetic resonance imaging via deep learning'[38] that is possible to speed up an MRI acquisition by the use of an AI system which can understanded how to under-sample at best, somehow mimicking the CS under-sampling task.

Many advanced approaches have been tried, methods that adapt state-of-the-art models to MRI acquisition, like the use of GAN (generative adversarial network) to reconstruct the under-sampled image have been proposed [39].

So far scientific community studied NN approaches dealing with one problem at time, tackling the two parts of the CS problem separately.

A novel approach has been proposed by Cagla D. Bahadir et al. in their 'Deep-Learning-base Optimization of the Under-Sampling Pattern in MRI'[40] (LOUPE), in which they tackle the double problem of CS in MRI at the same time, exploiting the learning potentialities of their NN model.

The smart approach of Cagla D. Bahadir and al. consisted in the hypothesis that every pixel of a *k*-space MRI image has a different degree of importance and ML can:

1. detect what pixels have more importance with respect to the others, finding the ‘best under-sampling pattern’, and
2. learn to reconstruct at best the under-sampled image.

This work starts from the work of Cagla D. Bahadir as a background and implements new solutions to the already existing project to improve the result.

The work is structured as follow:

1. in the next chapter an in-depth description of the work of Cagla D. Bahadir et al. will be presented, with insights on the code and on the mathematical background.
2. In the third chapter the new developed features will be introduced.
3. In the fourth chapter will be presented the results compared with the already known results of the background paper to show the improvements.

3 LOUPE

This work is based on the novel work [40], which proposes a solution integrally based on Neural Network adaptation of CS to MRI scan and reconstruction, called 'LOUPE'. The model that has been developed in [40] will be cited as 'LOUPE' so on.

LOUPE has been entirely implemented on Python using TensorFlow [41].

In this chapter the paper that describes LOUPE will be introduced and explained in an exhaustive way, keeping for granted the application background already given in the introduction (Chapter 1).

LOUPE is an AUTOENCODER. Every autoencoder can be represented as a two part-part model: ENCODER and DECODER.

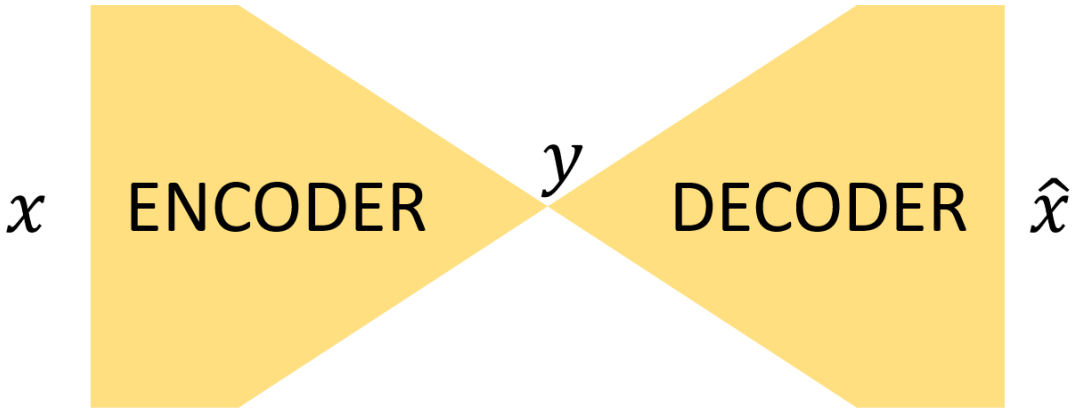


FIGURE 3-1 - AUTOENCODER REPRESENTATION

An autoencoder can be expressed also with two nested functions, one for the encoder and one for the decoder:

$$\hat{x} = decoder(y), \qquad y = encoder(x)$$

x is the input image, the ground truth and full resolution MRI scan, x lives in the Euclidian space. y is the under-sampled data in the *k*-space. y is the raw image given by the MRI after a scan before it is manipulated. $\hat{x}$ is the output image, the image returned after the reconstruction.

This explanation of LOUPE considers the model as a model constructed building two independent parts: an ENCODER and a DECODER. After separate explanations of both the two sections of LOUPE, them will be merged and a justification of the novelty in the construction of the net will be given.

The following chapter will be divided into five parts:

1. DECODER: U-NET
2. ENCODER: ADAPTIVE UNDER-SAMPLING
3. AUTOENCODER: ENCODER + DECODER
4. RESULTS AND CONCLUSIONS
5. PROBLEMS AND HOW TO IMPROVE LOUPE

3.1 Decoder: U-NET

An adaptive model, such as a NN, able to reconstruct the under-sampled *k*-space image to obtain the full resolution Euclidian space images must learn how to reconstruct by looking at the ground truth image.

The ground truth is given as input to the autoencoder but not as input to the DECODER, this means that is necessary to build a dataset containing y as data and x as label. The loss function to minimize, for the DECODER is:

$$\min_{\theta}(\|decoder_{\theta}(y) - \hat{x}\|_2)$$

DECODER takes as input the compressed form y of the ground truth data x and outputs the decoded version $\hat{x}$. It aims to reduce the differences between the original and the reconstructions by adjusting θ, to be intended as adaptive parameters.

LOUPE decoder is divided into two parts:

1. The first part is a non-trainable layer that takes as input the *k*-space raw image given by MRI machine and computes Inverse Fast Fourier Transformation (IFFT), returning the image in its natural domain. This layer does not work for solving aliasing errors, it just introduces a change of domain.
2. The second part is a CNN that takes as input the image returned by the IFFT layer and reconstructs the image by erasing the aliasing errors. The selected CNN is called U-NET [42] and, because of its purpose is also referred to as 'anti-aliasing filter'.

DECODER scheme can be represented as:

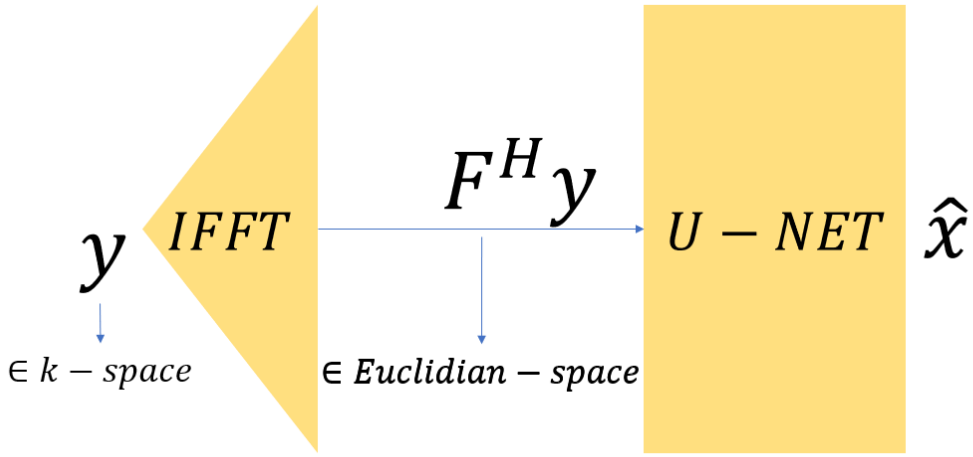


FIGURE 3-2 - DECODER STRUCTURE

The introduction of the IFFT layer is necessary to obtain full performances by the U-NET, in fact U-NET exploits its capacities when it works with 'normal' images. Giving a *k*-space image to the U-NET would have strongly limited the potentialities of it. At least the U-NET would have had to learn the IFFT reconstruction by itself within the antialiasing task. The IFFT layer improves training performances.

The loss function can be rewritten this way:

$$\min_{\theta}(\|A_\theta(F^H y) - \hat{x}\|_2)$$

Where A_θ is the U-NET that takes as input y but translated into the Euclidian space via IFFT $F^H y$. θ are the trainable weights of the U-NET, that are the only adaptive part of the DECODER.

The introduction of U-NET, into the research world, came with the publication of the paper 'U-Net: Convolutional Networks for Biomedical Image Segmentation' [42] in which researchers from Cornell present their new type of CNN to solve segmentation problems.

Segmentation problem refers to the difficulty in identifying the regions of interest inside an image, as a tumour could be in an MRI image. Detection of high importance areas are a key step in identifying disease origins in biomedical imaging. After the detection, the area can be highlighted on the image or isolated so doctors can study the case with more ease.

The aspects that most characterize U-NET are:

1. The U-shape, that also gives the name to the model.
2. The concatenation layers.

The U-shape is given by its scheme representation, where are also visible its concatenation layers (GRAY arrows):

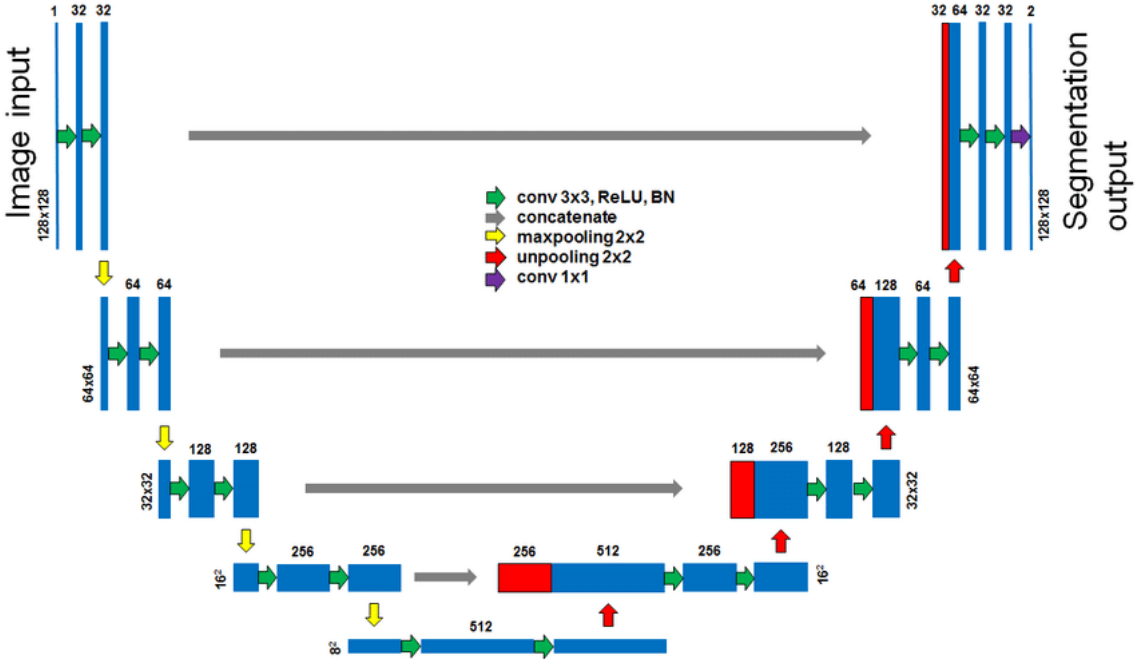


FIGURE 3-3 - U-NET SCHEME

Figure above shows the U-NET first restricts the image dimension with pooling layers and augments the features by using more and more filters for every CNN layer stage, until it comes to a minimum image dimension, then it uses up-sampling to reconstruct the original image dimension following the very same scheme it has already used but in a reverse fashion and with concatenation layers. This trend, when represented, using vertical image length proportional to the image dimension and horizontal length proportional to the number of features, shows the typical U shape.

The concatenation layer brings the left element (BLUE) and concatenates it, without overlapping, with the right element (BLUE) to create one bigger element that is represented as a RED and a BLUE rectangle, corresponding to the left-moved-to-right element and to the right element, respectively. Both the samples that are concatenated have the same dimension but not the same number of features.

U-NET was born as a special adapted NN for data segmentation that could work with few instances for training, giving the possibility to train a deep model even if the dataset is rather small. Having a small dataset is a common problem when dealing with medical images because usually data are reserved and are non-homogeneous.

In LOUPE a slightly modified version of the U-NET is used: the core remains the very same, but a layer that computes an addition between the 'pixelwise L2 norm of the input' and the output of the U-NET is added to compute the output. The equation turns out to be: $\hat{x} = UNET(F^H y) + \|F^H y\|^{pixelwise}_{L_2} = A_\theta(F^H y)$.

Another difference is the input image dimension that has two channels instead of one. U-NET must consider the real and the imaginary parts of the input $F^H y = \mathbb{R} + i * \mathbb{I}$ by using one channel for $\mathbb{R}$-valued elements and one for the $\mathbb{I}$-valued ones.

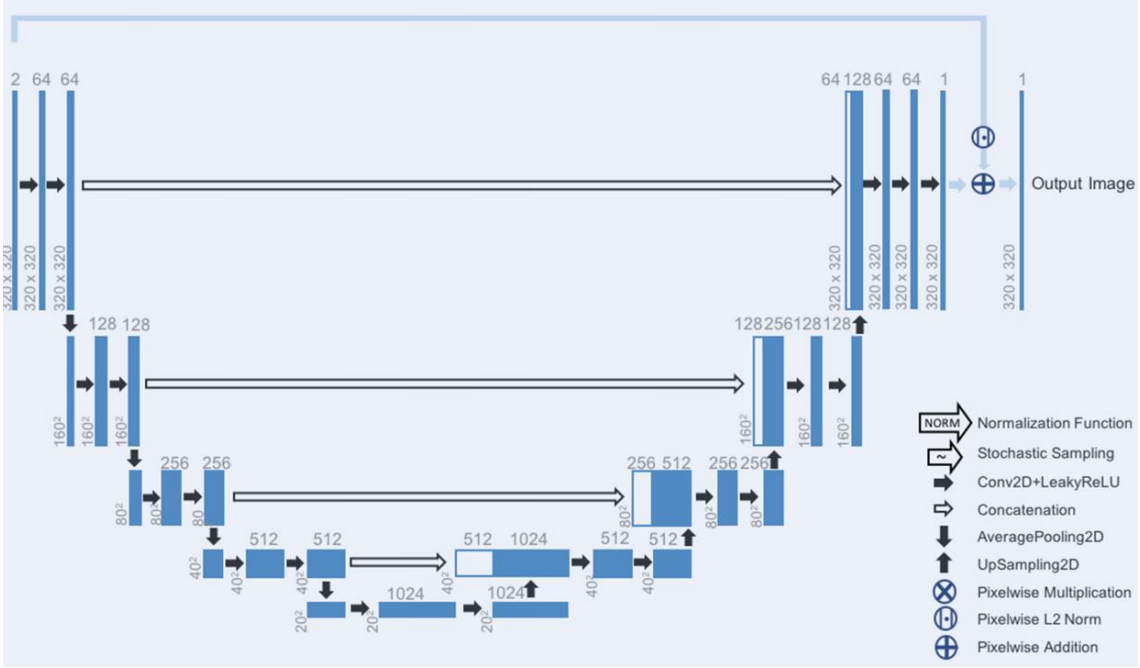


FIGURE 3-4 - U-NET USED IN LOUPE

3.2 Encoder: Adaptive Under-Sampling

The ENCODER is the part of the autoencoder that compresses the signal and, being adaptive, learns how to do it in the best way by keeping the elements with more information and discarding the others.

In first analysis, the encoder mimics the MRI data acquisition reproducing the same images that the MRI would return after its scans. The input of the ENCODER is the ground truth image x and its output is the under-sampled image y.

More in details, the encoder is not only a module that mimics the MRI acquisition, but it adapts the MRI acquisition to the data it reads: it is an adaptive encoder.

ENCODER can be split in three parts:

1. FFT layer that brings the input into *k*-space Fx.
2. A part that learns and returns the under-sampling pattern M.
3. A layer that takes as input the under-sampling pattern and the *k*-space image to perform the under-sampling $y = undersampling(x, M)$.

The second part is the only one that is affected by the training, the other two are static parts. The encoder can be represented this way:

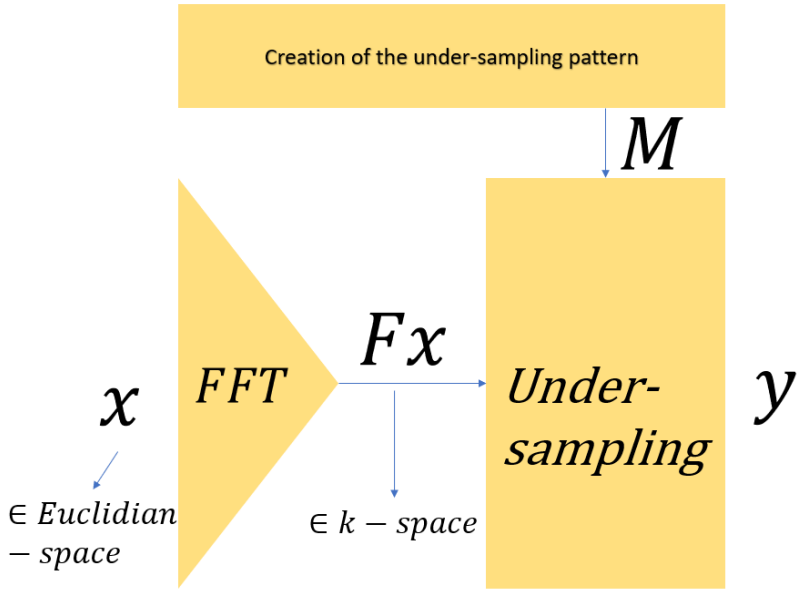


FIGURE 3-5 - DECODER SCHEME

The under-sampling can easily be implemented performing an elementwise multiplication if M is a mask of 0s and 1s having the same shape of Fx: $y = diag(M)Fx$.

$diag(M)$ is the way the elementwise multiplication is written, but no diagonalization is done, it is just notation.

So far, the loss function of the autoencoder can be written as:

$$\min_{\theta}\left(\sum_{j=1}^{N}\left\|A_{\theta}\left(F^{H}diag(M)Fx_{j}\right)-x_{j}\right\|_{L_{2}}\right)$$

Where x_j is one input image and θ contains all the parameter of the neural network.

The FFT and the 'Under-sampling' are standard modules, the heart of the problem lies in the 'creation of the under-sampling pattern' module that implements the function $M = mask_{\theta}(O)$.

LOUPE offers high flexibility when it comes to define the sparsity level of y, LOUPE offers two different implementations to realize the under-sampling pattern.

The first implementation, the one that is going to be explained in more details, defines a parameter called 'sparsity' (α) in a range $[0,1]$ and returns a mask M with mean α. Being M a mask with only 0s and 1s values and having mean $\mu(M) = \alpha$, the normalized sparsity $\frac{\|M\|_{L_0}}{d} = \alpha,\ d = number\ of\ elements(M)$, in other words the number of 1s in M $\#1(M) = \alpha * d$ and all the other elements are null. For this reason, α, even if is not the real sparsity definition $\|\cdot\|_{L_0}$, is called sparsity.

The other implementation does not output a mask M with a given sparsity, but it finds the best M given a penalty on the sparsity of M, that is added to the loss function. This method takes as input a parameter λ that defines how the sparsity constraint weights with respect to the other part of the loss function.

Both the methods require M to be sparse, mathematically it can be called a constraint.

The first implementation is called 'α-method', the other is called 'constraint-method'. Now the α-method is introduced.

After the addition of the constraint of the sparsity the loss function becomes:

$$\min_{\theta}\left(\sum_{j=1}^{N}\left\|A_\theta\left(F^H diag\left(mask(O)\right)Fx_j\right)-x_j\right\|_{L_2}\right),\qquad such\ that\ \frac{\|\sigma_l(O)\|_{L_1}}{d}=\alpha$$

Where $\|\sigma_l\|_{L_1} = \mu(P) = mean(P)$.

The great intuition in LOUPE is to create a custom layer called '*ProbMask'* that returns an element containing its weights. *ProbMask* is a simple layer that takes an input but discards it and only outputs its weights O. The weights O of the layer are TRAINABLE and are the only weights that can be modified during training in the ENCODER.

Because the mask is trainable and O becomes a parameter to minimize the loss function with, the loss function can be written:

$$\min_{\theta,O}\left(\sum_{j=1}^{N}\frac{1}{K}\sum_{k=1}^{K}\left\|A_\theta\left(F^H diag\left(mask(O)^{(k)}\right)Fx_j\right)-x_j\right\|_{L_2}\right),\qquad such\ that\ \frac{\|\sigma_l(O)\|_{L_1}}{d}=\alpha$$

$mask(O)^{(k)}$ represents an instance of M, to find the best possible M is necessary to test over many masks (K). $mask(O)^{(k)}$ are independent realizations of a random process that generates the masks: this is a smart way to introduce randomness and to secure the mask is well-trained.

O is the first and most important element of the block that adapts itself to be minimize the loss function, the remaining layers transform O into M by the addition of some requirements, these are the steps:

1. Bring values of O from $\mathbb{R}$ to $(0,1)$.
2. Initialize O.
3. Apply the α sparsity.
4. Introduce the randomness in the process to create M.
5. Adapt the procedure to be 'trainable'.

The first point is easy to satisfy: a 'sigmoid' function σ can be applied: $P = \sigma_l(O)$.

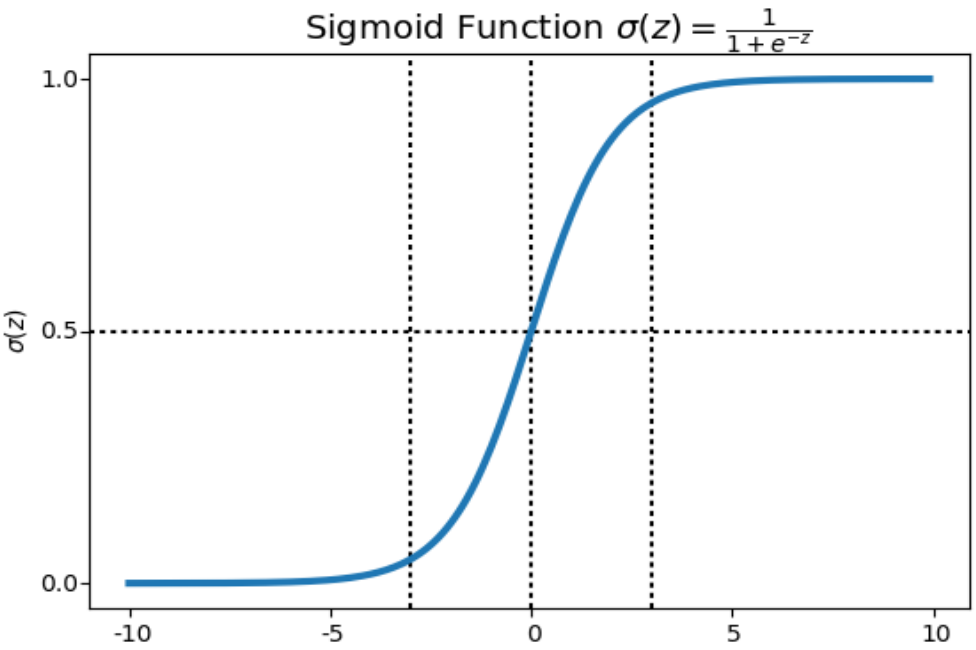


FIGURE 3-6 - SIGMOID FUNCTION

Figure above contains the plot of a standard sigmoid with $l = 1$. In general, $\sigma_l(z) = \frac{1}{1+e^{-l*z}}$, where a higher l means a steeper sigmoid, as l rises σ_l tends to resemble a hard threshold.

In a Dense layer as usually happens, z is assumed to be the typical output of a neuron $z = \sum w_i x_i + bias$ and the sigmoid is its activation function, in the *ProbMask* case the only difference is $z = w_i$.

Sigmoid function brought O from $\mathbb{R}$ to a continuous range of values $[0,1]$. Is important to initialize P with well distributed elements to help the training converge during the first epochs. Because sigmoid is not linear the initialization of O must return the desired distribution once sigmoid is applied. In LOUPE has been found that a good start is provided by $O_0 = \ln\left(\frac{x}{1-x}\right)\frac{1}{l}$, $given\ x \sim U(\xi, 1-\xi),\ \xi\ small \sim 0.01$. The final distribution of P is $P_0 = 1 - x$.

The α-implementation deals with the constraint $\frac{\|\sigma_l(O)\|_{L_1}}{d} = \alpha$, defining a function N_α.

To obtain the final desired sparsity α in M, P needs to have mean of P equal to $\mu(P) = \alpha$. The custom layer '*RescaleProbMask'* in LOUPE achieves this step by rescaling: $P_\alpha = N_\alpha(P)$. The new function N_α is:

$$P_\alpha = \begin{cases} \dfrac{\alpha}{\mu(P)} * P, & if\ \mu(P) \geq \alpha \\ 1 - \dfrac{1-\alpha}{1-\mu(P)} * (1-P), & otherwise \end{cases}$$

N_α turns the mean $\mu(P) = \alpha$, this way ensures the constraint is satisfied and the loss function does not need it to be specified as an explicit constraint anymore. N_α turns an explicit constraint into an implicit one.

To implement the randomness into the scheme, LOUPE inserts two special layers called '*RandomMask'* and '*ThresholdRandomMask'* respectively.

The first layer creates a matrix U with the same dimension of P. Every element of U is a random uniform distributed value in the range $[0,1]$: $U \sim U(0,1)$. This layer takes no inputs.

The output of RandomMask is passed to *ThresholdRandomMask* that takes as input also P and computes the hard threshold $M = P_\alpha > U$. It returns a matrix of 0s and 1s that is used as under-sampling mask. In the location where $P_\alpha > U$ the element of Fx will be taken, otherwise it will be discarded.

At this point many steps can be compressed together and inserted into the loss function:

$$\min_{\theta,O}\left(\sum_{j=1}^{N}\frac{1}{K}\sum_{k=1}^{K}\left\|A_\theta\left(F^H diag\left(N_\alpha\left(\sigma_l(O)\right) > U^{(k)}\right)Fx_j\right) - x_j\right\|_{L_2}\right)$$

Because now the randomness of M has been lifted to the random generation of U, it is U that acquires the $\cdot^{(k)}$ to explicit that the mask is adapted over K generation of U.

As it seems everything is done such that both the encoder and the decoder have been completed, there is still one more adjustment to do: the function as it is written is not trainable. During training is necessary to have all differentiable functions, otherwise backpropagation cannot compute the gradient and cannot work. Inside the loss function the part $\left(N_\alpha\left(\sigma_l(M)\right) > U^{(k)}\right)$ is not differentiable.

To solve this problem is necessary to lighten the assumption that the mask is a binary matrix. As long as M lives into a discrete space it cannot be derivable, the solution is obviously to let elements in M belong to the continuous range $[0,1]$. LOUPE solves the problem using a difference instead of greater operation and to bring back in the $[0,1]$ domain the outcome it uses a sigmoid σ_s with a high steepness (high s) to mimic the threshold but remaining continuous. Mathematically $M = \sigma_s\left(N_\alpha\left(\sigma_l(M)\right) - U^{(k)}\right)$.

The loss function is updated to:

$$\min_{\theta,O}\left(\sum_{j=1}^{N}\frac{1}{K}\sum_{k=1}^{K}\left\|A_\theta\left(F^H diag\left(\sigma_s\left(N_\alpha\left(\sigma_l(O)\right) - U^{(k)}\right)\right)Fx_j\right) - x_j\right\|_{L_2}\right)$$

One last consideration: K in LOUPE is set to $K = 1$ because, as reported in [40], it is a computationally efficient approach that yields an unbiased estimate of the gradient that is used in stochastic gradient descent. The loss function finally is simplified to:

$$\min_{\theta,O}\left(\sum_{j=1}^{N}\left(\left\|A_\theta\left(F^H diag\left(\sigma_s\left(N_\alpha\left(\sigma_l(O)\right) - U^{(k)}\right)\right)Fx_j\right) - x_j\right\|_{L_2}\right)\right)$$

The second implementation, called 'regularization-method', instead of defining a parameter α to force the desired sparsity of M, defined a regularization parameter λ to control the weigh of a regularization factor added to the loss function.

This method is based on many of the considerations that have been done for the α-method. All the layers are the same, expect for RescaleProbMask that is missing. Because *RescaleProbMask* is not used, in the loss function the formula N_α is missing and the sparsity task is carried by the regularization function $\lambda * \left\|M^{(k)}\right\|_{L_1}$. The L_1 norm of M is minimized leading to a higher sparsity of O. The loss function of the regularization-method is:

$$\min_{\theta,O}\left(\sum_{j=1}^{N}\left(\left\|A_\theta\left(F^H diag\left(\sigma_s\left(\sigma_l(O)-U^{(k)}\right)\right)Fx_j\right)-x_j\right\|_{L_2}\right)+\lambda*\left\|M^{(k)}\right\|_{L_1}\right)$$

This method guarantees the mask sparsity, but do not guarantees for a specific value of sparsity. This makes the method not as competitive as the α-method and will not be discussed anymore.

3.3 Autoencoder

From the loss function evident how the approach of LOUPE tackles the two main aspects of CS together:

1. The encoder finds the best under-sampling pattern by training the *ProbMask* layer and modifying O.
2. The decoder adapts to obtain the best possible reconstruction by modifying the weights θ of the U-NET.

The heart of LOUPE is in its double simultaneous adaptiveness. Not only the reconstruction can adapt to the under-sampled image, but the under-sample pattern can adapt itself to the reconstruction.

The implementation of LOUPE can be represented showing its layers:

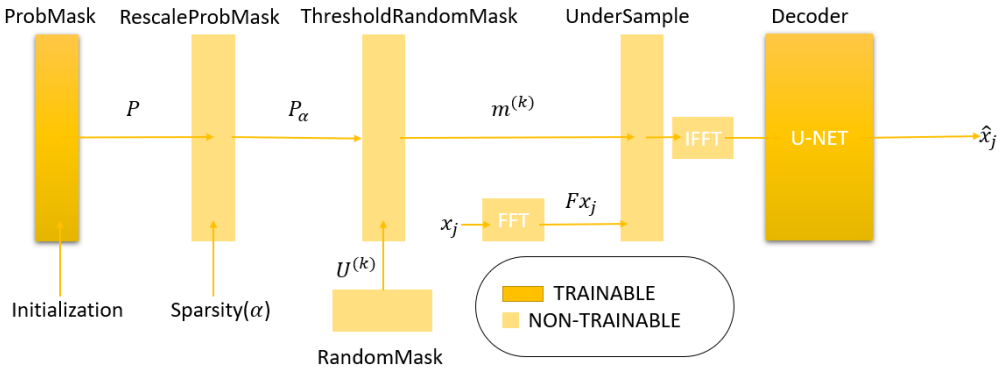


FIGURE 3-7 - LAYERS OF LOUPE

The representation shows all the layers of the encoder, for the decoder the U-NET has been represented as a unique layer, even if as already explained it composed by many different layers.

An important aspect of LOUPE that in the Figure below is highlighted are its trainable and non-trainable parts. It is clear from the figure that for building an adaptive encoder only *ProbMask* needs to be trainable, and for building the decoder only the U-NET needs trainable parameters.

A more detailed representation that shows more mathematical relationships:

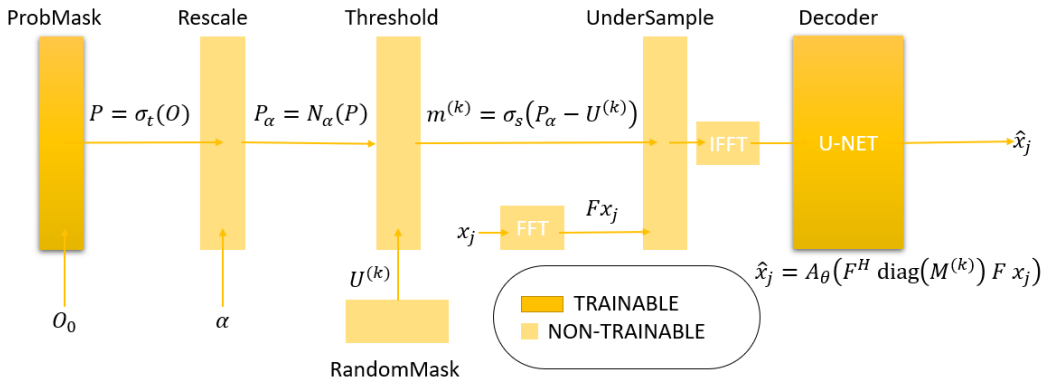


FIGURE 3-8 - LAYERS OF LOUPE WITH FORMULAS

3.4 Results and conclusions

The study [40] trained LOUPE on knee MRI dataset. In particular the considered dataset is the 'NYU fastMRI' dataset [43] (fastmri.med.nyu.edu). [43] is a freely available, large-scale, public set of knee MRI scans.

Validation functions such as MSE, MAE, PSNR, SSIM and HFEN [44] were used to evaluate the results.

More in details the validation function are defined as:

$$PSNR(x, \hat{x}) = 10 * 10 \log\left(\frac{(\max(x))^2 d}{\|x - \hat{x}\|_2^2}\right)$$

$$\mathrm{SSIM}(\mathrm{x}, \hat{x}) = \frac{(2\mu_{\mathrm{x}}\mu_{\hat{\mathrm{x}}} + \mathrm{c1}) + (2\sigma_{x\hat{x}} + \mathrm{c2})}{(\mu_{\hat{x}}^2 + \mu_{\hat{x}}^2 + \mathrm{c1})(\sigma_{\mathrm{x}}^2 + \sigma_{\hat{x}}^2 + \mathrm{c2})}$$

Parameter definition has been provided in [40].

LOUPE has been compared to three different non-ML benchmarks and with one ML benchmark: the three non-ML are BM3D [45], LORAKS [18], TGV [46], the ML one is a U-NET. Every model does not adapt to find the best under-sampling patter but receives under-sampled MRI images whom under-sampling patter was not trained but was user-defined.

The used pattern (for the benchmarks) are Random Uniform pattern [47], [48], parametric variable density [49], [50], data-driven strategy [51], cartesian [52] and line constraint [53].

LOUPE has shown good performances with respect to the other methods, classifying as a novel and functional approach for implementing CS MRI acquisitions.

Based on the 'NYU fastMRI' dataset [43] and on acceleration rates 4 and 8 (corresponding to $\alpha \; in \; [0.25, \; 0.125]$) LOUPE scored:

Acceleration	Method	Masks	MSE PD	MSE PDFS	MAE PD	MAE PDFS	HFEN PD	HFEN PDFS	PSNR PD	PSNR PDFS	SSIM PD	SSIM PDFS
4-fold	BM3D	Cartesian (1D) Skipped Lines	0.00796	0.00629	0.0646	0.0587	0.9484	1.0596	21.1	22.1	0.59	0.63
		LOUPE Line Constrained	0.00023	**0.00089**	0.0105	**0.0234**	0.548	0.7096	36.69	**30.5**	0.95	**0.82**
		Uniform Random	0.01074	0.00902	0.0707	0.0695	0.9467	1.1416	19.95	20.47	0.63	0.58
		Variable Density	0.00032	0.00137	0.0128	0.029	0.5772	0.8288	35.2	28.66	0.93	0.75
		Spectrum	0.00021	0.00135	0.0107	0.0287	0.537	0.8464	37.05	28.72	0.94	0.75
		LOUPE Unconstrained	**0.00014**	0.00094	**0.0088**	0.0241	**0.2972**	**0.5298**	**38.81**	30.28	**0.96**	0.81
	LORAKS	Cartesian (1D) Skipped Lines	0.22717	0.26113	0.1785	0.1896	2.8759	3.5001	8.45	6.93	0.46	0.28
		LOUPE Line Constrained	0.00025	**0.00097**	0.0112	**0.0246**	0.5801	0.7668	36.32	**30.14**	**0.94**	**0.8**
		Uniform Random	0.01692	0.01861	0.0913	0.1052	1.1886	1.2677	17.99	17.3	0.42	0.26
		Variable Density	0.00041	0.00231	0.0152	0.0372	0.6364	1.1334	34.1	26.4	0.89	0.64
		Spectrum	0.00036	0.00208	0.0142	0.0355	0.6691	1.184	34.69	26.85	0.91	0.67
		LOUPE Unconstrained	**0.00019**	0.00113	**0.0105**	0.0266	**0.3754**	**0.6854**	**37.42**	29.48	**0.94**	0.78
	TV	Cartesian (1D) Skipped Lines	0.00789	0.00582	0.0643	0.0564	0.8938	0.9126	21.14	22.46	0.55	0.55
		LOUPE Line Constrained	**0.00089**	**0.0012**	**0.0217**	**0.027**	0.9029	0.7823	**30.57**	**29.22**	**0.8**	**0.73**
		Uniform Random	0.01286	0.01178	0.0755	0.0818	0.9303	0.9184	19.33	19.33	0.62	0.55
		Variable Density	0.003	0.00333	0.0417	0.0454	1.2104	1.0221	25.26	24.78	0.6	0.56
		Spectrum	0.00144	0.00188	0.0289	0.0344	0.8695	0.8492	28.48	27.27	0.71	0.62
		LOUPE Unconstrained	0.00196	0.00218	0.0332	0.0366	**0.8991**	**0.7559**	27.13	26.63	0.71	0.65
	U-Net	Cartesian (1D) Skipped Lines	0.00406	0.00345	0.0381	0.0401	0.8265	0.8648	24.11	24.65	0.83	0.79
		LOUPE Line Constrained	0.00015	0.00058	0.0084	0.0186	0.3563	0.4567	38.53	32.36	0.97	0.89
		Uniform Random	0.0022	0.00204	0.0303	0.0323	0.8342	0.7689	26.89	26.93	0.87	0.85
		Variable Density	0.00014	0.00053	0.0086	0.018	0.4088	0.5186	38.79	32.74	0.97	0.9
		Spectrum	0.00013	0.00053	0.0084	0.0179	0.4266	0.5727	39.05	32.77	0.97	0.9
		LOUPE Unconstrained	**0.00009**	**0.00048**	**0.0071**	**0.0173**	**0.1681**	**0.246**	**40.71**	**33.16**	**0.98**	**0.91**

FIGURE 3-9 RESULTS OF LOUPE TRAINED ON KNEE DATASET

As it is possible to see from the Figure above LOUPE scored always better than the other methods. Figure below shows an example of reconstruction using LOUPE, the dataset consists of knee MRI scans. The image has been taken from [40].

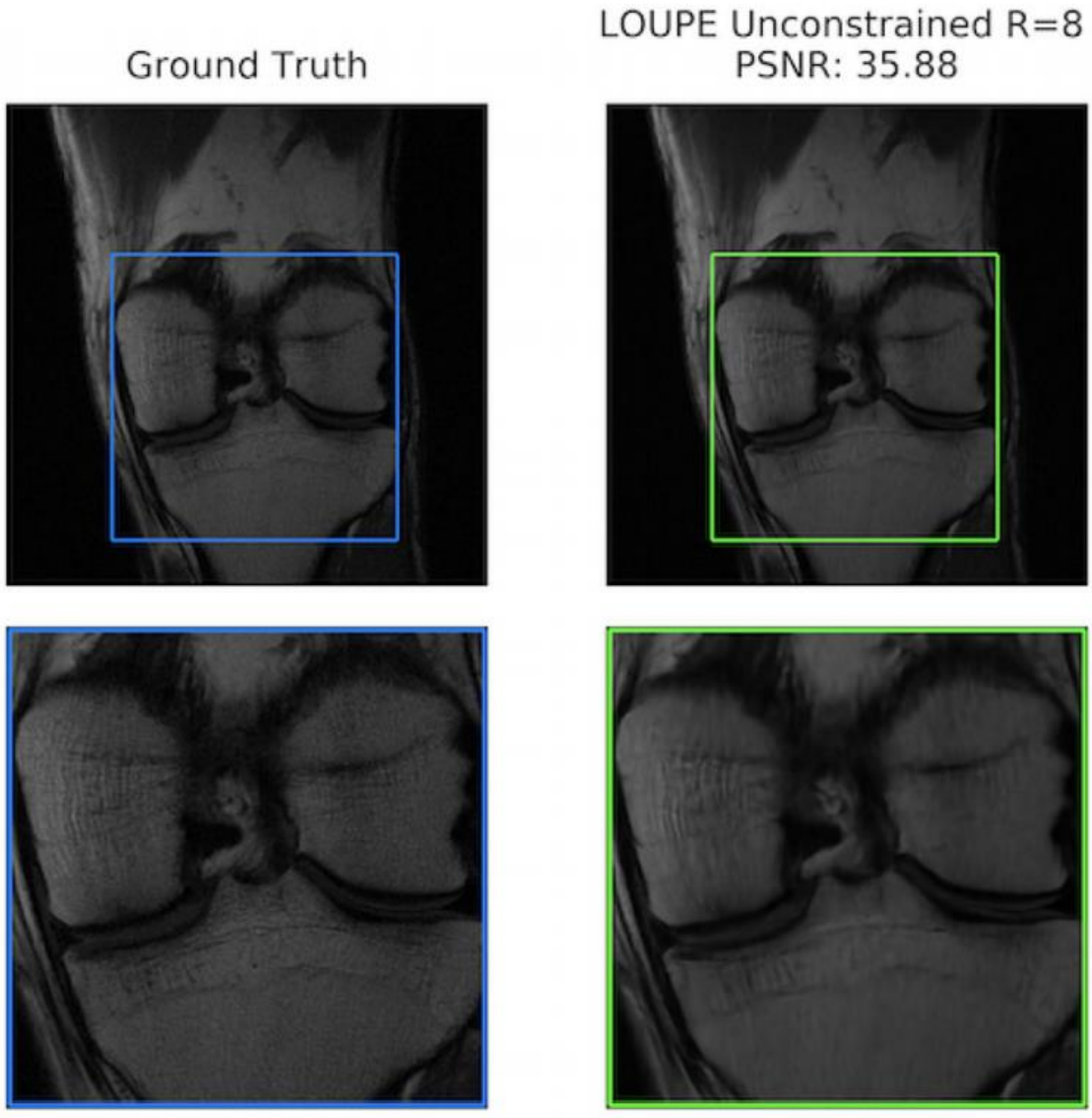


FIGURE 3-10 LOUPE EXAMPLE OF RECONSTRUCTION

3.5 Problems and how to improve LOUPE.

Even if LOUPE is a competitive and well-structured method it still has potentialities to exploit farther.

In this work four main critical points have been highlighted:

1. Regarding the loss function, when the operation $P_\alpha > U^{(k)}$ is modified into $\sigma_s\left(P_\alpha - U^{(k)}\right)$ the original threshold function is approximated into a sigmoid. This approximation must be tuned via the tuning of the slope s of the sigmoid σ_s. The tuning of s can be skipped by the gradual and constant increasing of s during training.
2. The training can be aided by requesting the reconstructed image $\hat{x}$ to have the same known frequency components of y, ensuring that the reconstruction process does not warp the components that are known to be correct.
3. An indicator of the reconstruction success can be found comparing the frequency components of $\hat{x}$ with the known frequency components of y. This result can be used, after training, as a detector for reconstruction failures, or as an indicator that points out if the training is successful or not.
4. In LOUPE, before the under-sampled data y is given as input to the U-NET, y is moved from *k*-space to Euclidian space using the IFFT: $F^H y$. This step can be generalized by the use of a LOUPE-learned transformation.

All these points are ideas to improve LOUPE working on details, that do not modify the kernel of it.

In the next chapter some contributions to LOUPE will be explained and justified, showing in detail how they work and how they improve LOUPE.

4 CONTRIBUTIONS

After having explored in depth LOUPE [40], four different ways to expand and improve it have been proposed:

1. *Prancing pony*: gradual approximation of $\sigma_s\left(P_\alpha - U^{(k)}\right)$.
2. *Flashback*: training minimizing the error between under-sampled *k*-space input and reconstructed.
3. *Flashback* as self-assessment.
4. $TIFT$: *Trainable Discrete Fourier Transform*.
5. *Back to the future*: substituting $\hat{y}$ with y.

In this final chapter all the proposed additions will be better introduced, justified, and implemented one by one in a separate sub-chapter. In the last chapter results and improvements will be shown.

4.1 Prancing pony: gradual approximation of s

In LOUPE the loss function is adapted to be trainable by changing a non-differentiable function (an ideal thresholder) into a continuous and differentiable function σ_s (the sigmoid presented in Chapter 3 with slope s).

σ_s is a good approximation of an ideal thresholder as long as it is steep, but how much should σ_s be steep to accomplish this similarity? The bets pick of s depends on many parameters and a universal acceptable value of s does not exist, for example in [40] the authors suggest to set $s = 200$ because after some experimental trials varying s they found $s = 200$ returned the lowest reconstruction error.

After training is over the use of the model changes, LOUPE is split:

1. From the encoder part of LOUPE only the learned mask is taken and given to the MRI machine that uses it during acquisition to speed up the procedure.
2. The decoder is taken as it is. The acquired under-sampled image, is given as input to the decoder.

At inference time it is necessary to get rid of the simplification introduced to render the encoder trainable: during training the mask is created by using a continuous function $\sigma_s\left(\sigma_l(O) - U^{(k)}\right)$ but at inference time the mask is discrete: $\sigma_l(O) > U^{(k)}$, as it was before the approximation of the threshold. The latter mask is discrete because when MRI is under-sampling it is simply

discarding samples, the approximation introduced for training serves only for training, at inference time its utility is over.

After training the model is evaluated (and used) on data under-sampled with $<$ (hard threshold), but the model has been trained on data under-sampled with something different (soft threshold) and this leads to loss of performances.

To deal with this inconvenience it has been implemented a function that gradually rises the steepness of σ_s by raising s until σ_s is close enough to a threshold. We call this implementation 'prancing pony'.

Prancing pony avoids the issue of tuning s.

The conveniences of Prancing Pony are the possibility to set a high value of s_{max} without a performance degradation, in fact the model will train on various s, as long as $s < s_{max}$, and will automatically find the best one by saving only the model with the lowest loss. A gradual increasing s also means that the model can gradually adapt to data and can explore more during the first training epochs when the image contains more information, specializing more every time σ_s gets steeper, this way helping the convergence.

The idea of Prancing pony is to augment s whenever the model has adapted to data under-sampled with the current s. To detect whenever a function stops improving is necessary to look at the loss function. To operate on function variables throughout the training monitoring the loss, is necessary to use classes of functions called 'callbacks'.

A callback is a function that is automatically called when a specific event occurs during training, for example when an epoch stops or when a batch finishes. The most important and widely used callback functions are:

1. Early stop.
2. Reduce learning rate on plateau (reduce LR).
3. Save checkpoint.

All the three callbacks are called at the beginning of the train for initialization and at the end of an epoch. They work based on a metric loss that has been suggested by the user.

Callback Early stop first looks at the metric loss and if it does not improve after a number of epochs called 'patience' terminates the training. Early stop is usually set to check the loss on the evaluation data to prevent overfitting.

Reduce learning rate on plateau looks at the metric loss and a after a number of epochs (patience) during which it does not descend it lowers the learning rate of the optimizer. Usually reduce learning rate on plateau has a lower patience than Early stop so to continue training until Early stop takes action.

Save checkpoint saves the model after every epoch, so even if the training gets interrupted the work is not lost, but it can be resumed from the last saved checkpoint. Save checkpoint can be adapted to save only the model that gives the best results during training, preventing from saving an overfitted model.

Prancing pony works similarly to the above cited callbacks: it looks at the loss at the end of every epoch, and if after a 'patience' number of epochs the loss does not improve, prancing pony switches the current value of s to a higher one.

The first problem when implementing prancing pony is that it does not mingle well with the other callbacks. In general, after prancing pony triggers, rising s, is possible to predict a worsening of performances due to different data arriving at the decoder that so far was trained on another type of under-sampled data. The net needs time to adapt to this new modification and this does not scale well with the other callbacks because if the evaluation does not improve within a patience number of epochs of the other callbacks, the other callbacks trigger. For example, it could happen that early stop activates itself even if no overfitting point has been reached.

A naïve solution could be to set all the patience values at high values, but this would lead to uselessly long training time, a better solution is to create a unique callback called 'Holistic' that groups all the previous callbacks.

In Holistic two main modifications have been implemented:

1. The creation of a state in which Holistic enters after prancing pony is called and exits only after a new improvement in the evaluation function is registered. As long as Holistic is in this idle state no callback is triggered.
2. The patience of Prancing pony is augmented every time it is triggered.
3. Every time Reduce Learning Rate on Plateau is called its patience is augmented.
4. Every time Prancing Pony activates the Learning Rate is restored to its initial value and patience of Reduce Learning Rate on Plateau is restored to its initial value.

Holistic requires some parameters to be defined:

1. $patiece_x$ is the number of epochs during which the loss function does not improve before x activates. Every $patience_x$ must be defined by the user before the training starts.
2. i is a counter, at the beginning of the training i is set $i = 0$.
3. LR_{min} is the minimum value of LR, decided by the user.
4. s_{max} is the maximum value of s, decided by the user.

Holistic is called at the end of every epoch. Holistic can be explained with the following algorithm:

Holistic algorithm:

1. $If\ (state == idle)$:
    a. $If\ (evaluation\ function\ shows\ improvements)\ or\ (i \geq patience_{Idle})$:
        i. $state = active$
        ii. $i = 0$
    b. $else$:
        i. $i = i + 1$
2. $If\ (state == active)$:
    a. $If\ (evaluation\ function\ shows\ improvements)$:
        i. $i = 0$
    b. $else$:
        i. $i = i + 1$
    c. $if\ \left(i \geq patience_{EarlyStopping}\right)$:
        i. $STOP\ TRAINING$
    d. $if\ (i \geq patience_{ReduceLR})\ and\ (LR > LR_{min})$:
        i. $reduce\ LR$
        ii. $Increase\ patience_{ReduceLR}$
    e. $if\ \left(i \geq patience_{PrancingPony}\right)\ and\ (s < s_{\max})$:
        i. $Increase\ s$
        ii. $state = idle$
        iii. $Increase\ patience_{PrancingPony}$
        iv. $i = 0$
        v. $patience_{ReduceLR} = initial\ patience_{ReduceLR}$
        vi. $LR = initial\ LR$

4.2 Flashback: training on the minimization of the error of the reconstructed *k*-space image.

This contribution is based on the intuition/sensation that there was something in LOUPE that was not exploited as it could have been. Because y is masked in the middle of LOUPE, it could

remain unseen and underrated. So far, y has been treated as the under-sampled signal that the encoder wants to build, and the decoder wants to reconstruct: another brick in the wall.

y is more than house's brick, is at least the door of that house, in fact, at inference time, the first thing the decoder finds is y, is through y that the decoder enters the problem. Once inside the problem LOUPE does its best to reconstruct what is missing, but once finished, does it ask itself, if what it started with, y, has been preserved throughout the process?

Imagine that, during training, after LOUPE has reconstructed x into $\hat{x}$, it looks at its creation $\hat{x}$, and a flashback starts, LOUPE remembers what $\hat{x}$ really was, before all the make-up applied by the decoder. Thanks to Flashback LOUPE remembers the real soul of $\hat{x}$: y, and, regretful, it tries to restore its mistakes by modifying the DECODER to preserve y.

This chapter will introduce and explain a technique called 'Flashback' that emphasis the preservation of the frequencies of y, that are known to be a ground truth.

To check how well the y'components are preserved Flashback is defined as an error function:

$$F_{lash} = \|y - \hat{y}\|_{L_2}$$

Where y is the original input of the DECODER, $y = diag(M)Fx$ and $\hat{y}$ is the matrix containing the reconstructed ground truth frequencies:

$$\hat{y} = diag(M)F\hat{x}$$

Where M is the same matrix learned by the encoder to under-sample x and F is the Fourier operator. Flashback can be rewritten:

$$F_{lash} = \|y - diag(M)F\hat{x}\|_{L_2}$$

Flashback can be used to train LOUPE, in fact it helps the network focusing on the preservation of certain frequency components that are known to be correct, improving performances. It is necessary to weight this contribution with a parameter ϕ that will be tuned. The loss function with the addition of Flashback becomes:

$$\min_{\theta,O}\left(\sum_{j=1}^{N}\left(\phi\left\|A_\theta\left(F^H y_j\right) - x_j\right\|_{L_2} + (1-\phi)\left\|diag\left(M^{(k)}\right)FA_\theta\left(F^H y_j\right) - y_j\right\|_{L_2}\right)\right)$$

Where $y = diag\left(M^{(k)}\right)Fx_j$ and ϕ is the parameter that weights the reconstruction error and $(1-\phi)$ is the parameter that weights Flashback.

The implementation of Flashback can be summarized by the addition of few more blocks inside the scheme of LOUPE:

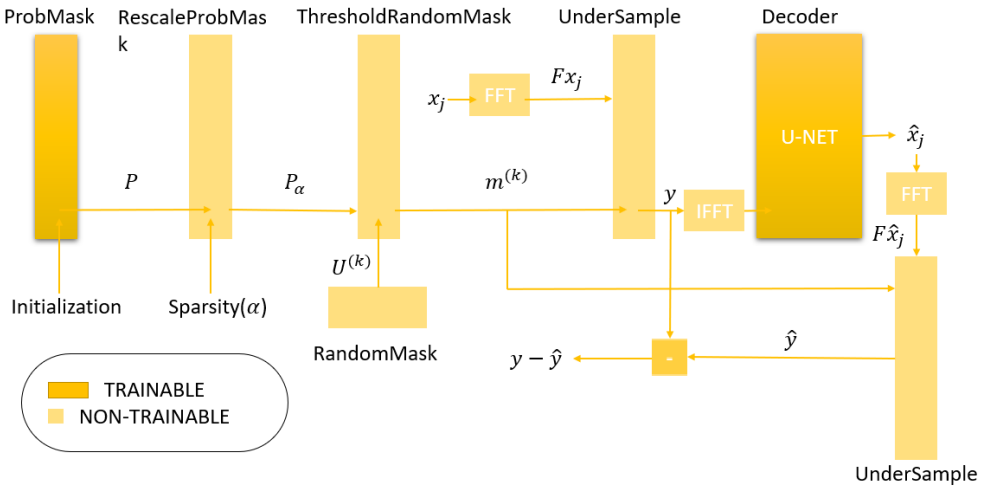


FIGURE 4-1 - SCHEME OF LOUPE WITH FLASHBACK

To implement Flashback is necessary to output the difference $y - \hat{y}$ because during training a loss function can only work with an input, meaning that it is not possible to feed Flashback with y and $\hat{y}$ separately.

All the blocks added for the implementation of Flashback are non-trainable and are copies of the layers already used for the encoder.

4.3 Flashback as self-assessment.

Flashback can be used not only for training but also as self-assessment. It is possible to show experimentally a linear relationship exists between $\|y - \hat{y}\|_{L_2}$ and $\|x - \hat{x}\|_{L_2}$ by plotting them together, one versus the other.

At training time, when both y and x are available, it is possible to map Flashback values versus reconstruction error $\|x - \hat{x}\|_{L_2}$. Using this map is possible to detect bad reconstructions by looking at Flashback in a really fast way.

Plotting $\|x - \hat{x}\|_{L_2}$ versus $\|y - \hat{y}\|_{L_2}$ all points distribute along a line and, for example, if Flashback has a high value (high loss), showing y has not been preserved in $\hat{y}$, the plot shows that similarly x has not been preserved in $\hat{x}$ because also the reconstruction loss is high.

At inference time is not possible to look at $\|x - \hat{x}\|_{L_2}$ because, of course, x is not given. It is possible to look at Flashback, or $\|y - \hat{y}\|_{L_2}$, instead. Because at training time a map has been found to link reconstruction error with Flashback, it is possible at inference time to look at Flashback to retrieve reconstruction errors.

In general, the reconstruction of the whole $\hat{x}$ went bad if the reconstruction of $\hat{y}$ went bad and vice versa, if the reconstruction of $\hat{y}$ is good also the reconstruction of $\hat{x}$ is good.

Supposing that the reconstruction error is uniformly distributed over all pixels composing $\hat{x}$, where $\|x_j - \hat{x_j}\|_{L_2} \sim Uniform(\cdot)$ and x_j is a pixel of x , one can say that the reconstruction error computed on known pixels in the Fourier domain, i.e., on $\|y - \hat{y}\|_{L_2}$ where $y = diag(M)F\hat{x}$, is α times the whole reconstruction error $\|x - \hat{x}\|_{L_2}$.

The prediction of the score $\|x - \hat{x}\|_{L_2}$, returned by Flashback is called self-assessment.

The only implementation that is required is the creation of a model that fits the data, for example using a model that fits a polynomial function such as a Linear Regressor Model.

4.4 TIFT: trainable inverse Fourier transform.

Before giving y to the decoder, LOUPE brings it from *k*-space to Euclidian space using an IFFT layer. It is logical that the image, once in the Euclidian space, can be better analysed by the U-NET, that do not need to learn by itself how to bring the image in the correct representation.

It is not a constraint to use IFFT though, it is only a reasonable choice. Because it is not known if IFFT is the optimal solution, it is possible to reinvent the IFFT layer, building something similar to IFFT but that at the same time could be adapted by the net, because after all we are speaking about neural network: we should stop asking ourselves how to solve problems and the give them directly to the net. The net itself will find the optimum, if there is one.

The idea of this contribution is to modify the IFFT layer, but because IFFT works fine, the idea is to modify it just slightly. Going farther, we are not really modifying it, we are asking the net to modify it by itself. Instead of keeping F^H, that might not be the best transformation, is possible to implement a layer initialized with the Fourier coefficients (F^H) and then let the model train the layer by itself and see if it finds something better.

The new layer is called Trainable Inverse Fourier Transform ($TIFT$). For clarity's sake the output of the new layer (TIFT), that was the IFFT layer, is called y'.

Because to explain $TIFT$ is convenient to reason in terms of 2D and 1D Trainable Inverse Fourier Transform we call $TIFT_{2D}$ the layer that operates a 2D $IDFT$ and $TIFT_{1D}$ the layer that operates a 1D $IDFT$.

The first way IFFT layer could be generalized into $TIFT_{2D}$ is by substituting it with a dense layer that would map every pixel of y into every pixel of y', assuming the image has dimension (n, n), $TIFT_{2D}$: $\mathbb{R}^{n\times n} \rightarrow \mathbb{R}^{n\times n}$. This implementation would work but it would require too many parameters. The number of weights of such a layer would be $n^2(n^2+1)$: too high.

A smarter way to implement $TIFT_{2D}$ is to mimic the inverse discrete Fourier transform (IDFT) exploiting its properties. $IDFT_{2D}$ can be seen as $IDFT_{1D}$ applied two times, such that:

$$y' = IDFT_{2D}(y) = IDFT_{1D}\big(IDFT_{1D}(y)\big)$$

Where the 1D Fourier transform $IDFT_{1D}(y) = F_{1D}^H y^T,\ IDFT_{1D}(y_{m,l}) = \frac{1}{n}\sum_{k=0}^{n-1} y_{k,l} e^{i2\pi\left(\frac{k}{n-1}\right)m}$ for $m, l = 0, .., n-1$ is the 1D Fourier transformation matrix. Writing the formula of y' with matrix notation:

$$y' = F_{1D}^H (F_{1D}^H y^T)^T$$

$TIFT_{2D}$ can be implemented by looking at the formula above by creating a trainable layer that performs $IDFT_{1D}$ and repeating it two times. The layer that implements the matrix multiplication of $IDFT_{1D}$ is called Super Dense. Trainable $IDFT_{1D}$ is implemented by using a layer to transpose the input and then a Super Dense: the whole block is called $TIFT_{1D}$. $TIFT_{2D}$ is simply $TIFT_{1D}$ applied two times to the same input.

Because F_{1D}^H and y have real and imaginary parts is necessary to modify the implementation of Super Dense to deal with it:

$$\begin{aligned} TIFT_{1D}(y) &= \big(\Re(F_{1D}^H) + i * \Im(F_{1D}^H)\big)\big(\Re(y) + i * \Im(y)\big)^T \\ &= \Re(F_{1D}^H)\Re(y)^T - \Im(F_{1D}^H)\Im(y)^T + i * (\Re(F_{1D}^H)\Im(y)^T + \Im(F_{1D}^H)\Re(y)^T) \end{aligned}$$

Where $\Re(F_{1D}^H) = \Re\left(e^{i2\pi\left(\frac{k}{n-1}\right)m}\right) = \cos\left(2\pi\left(\frac{k}{n-1}\right)m\right)$ for $k,\ m = 0, .., n-1$ and $\Im(F_{1D}^H) = \sin\left(2\pi\left(\frac{k}{n-1}\right)m\right)$ for $k,\ m = 0, .., n-1$.

Every Super Dense layer requires four Dense layers to handle the Real and Imaginary parts (one for every matrix multiplication), then it needs a sum layer and a subtraction layer and finally a layer to concatenate the real with the imaginary part to create an output with the same shape of the input. Super Dense can be represented using a simple scheme:

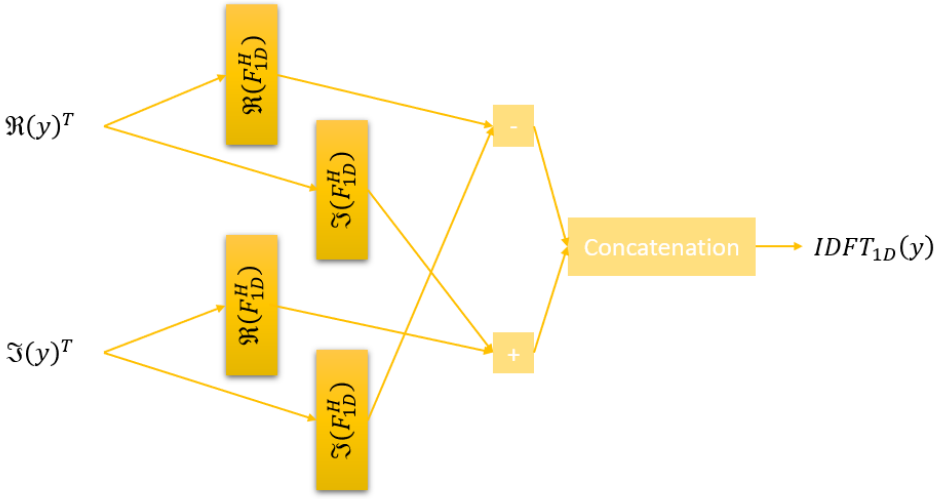


FIGURE 4-2 - SUPER DENSE SCHEME

Finally, Super Dense is concatenated with a transposition layer to create $TIFT_{1D}$ and repeated to implement $TIFT_{2D}$:

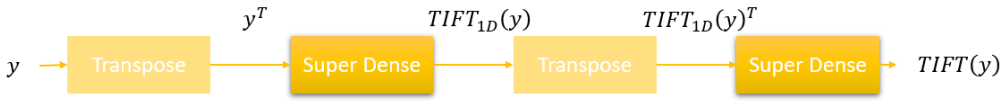


FIGURE 4-3 - TIFT SCHEME

Experimentally this contribution returns good results when training is spit in two steps:

1. $TIFT_{2D}$ is set non trainable so it exactly works as $IDFT_{2D}$ and LOUPE is trained until Early stopping occurs.
2. $TIFT_{2D}$ is set trainable and the training is resumed from where it ended.

The second training starts with the model being already at a good convergence point, so it can focus on improving by modifying $TIFT$.

Moreover, the training can be done using a hard threshold by setting as non-trainable the encoder. This leads to a non-adaptation of the under-sampling pattern of LOUPE, that is assumed is already well adapted, but leads to an adaptation of the net to the hard threshold data.

4.5 Back to the future

y is the ground truth element that is given, at inference time, to the decoder. Being y a ground truth, the model should aim to keep it unchanged throughout the process, but as shown it does not so. If no alternative methods for helping the net to preserve y are implemented, LOUPE learns to reconstruct x in $\hat{x}$, without paying much attention to $\hat{y}$ being as close as possible to y.

A method to correlate $\|y - \hat{y}\|_L$ with $\|x - \hat{x}\|_L$ to esteem one based only on the other (Self-assessment) and a way to help the preservation of y via the use of a regularization method have been proposed (Flashback). These works, in general, are useful and bring improvements, but a better and more direct approach can be applied at the end of the training.

It is simply possible to take y and substitute $\hat{y}$ with it. This method is called *Back to the future*. The idea behind Back to the Future is the possibility to travel in time so that after the reconstruction of the image ($\hat{x}$) is accomplished by the decoder, $\hat{x}$ travels in time, going at beginning of the reconstruction, before IFFT, meeting y and bringing it Back to the Future to save its life, avoiding all the errors committed by the decoder when reconstructing $\hat{y}$ perdure.

Assuming the training is over, and the output of LOUPE is $\hat{x}$, to implement Back to the Future is necessary to build few blocks.

It is possible to add these blocks to the net and obtain a new version of LOUPE that automatically outputs the new x, called $\hat{\hat{x}}$ ($x\ double\ hat$), or it is possible to forget about LOUPE and only take its outputs. The description of back to the future is given considering only the outputs of LOUPE, in particular considering $\hat{x}$, M and y, where for every $\hat{x_j}$ and y_j, M_j is the associated random realization.

$\hat{y}$ is the under-sampled k-space representation of $\hat{x}$ that can simply be obtained by applying a Fourier transform to $\hat{x}$ ($F\hat{x}$) and under-sampling it using M ($\hat{y} = diag(M)F\hat{x}$). $\hat{y}$ is the part of $F\hat{x}$ can be written as:

$$F\hat{x} = \hat{y} + diag\big(NOT(M)\big)F\hat{x}$$

Where $NOT(M)$ is M but with 1s swop with 0s and vice versa: $NOT(M)$ simply discards the points that M takes and keeps the points that M discards.

$\hat{y}$ is the k-space part of $F\hat{x}$ that back to the future wants to substitute with y, so $\hat{y}$ is simply excluded and y is simply added:

$$\hat{\hat{x}} = F^H\big(y + diag\big(NOT(M)\big)F\hat{x}\big)$$

This formula is implemented this way:

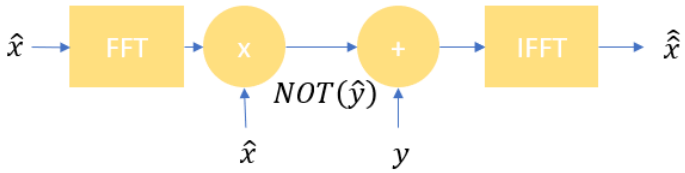


FIGURE 4-4 SCHEME OF BACK TO THE FUTURE

The scheme can also be generalized considering the whole LOUPE:

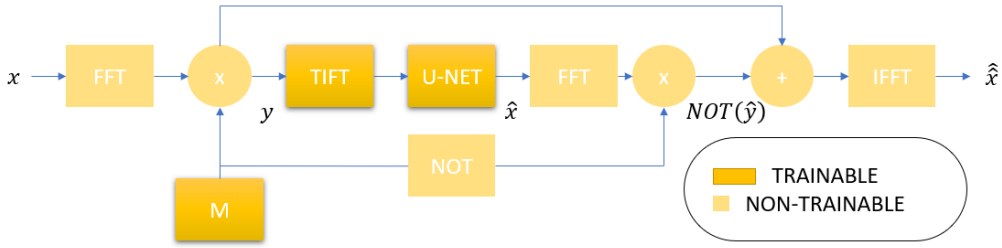


FIGURE 4-5 SCHEME OF BACK TO THE FUTURE GENERALIZED

This implementation can be further improved by using the already discussed blocks TIFT (trainable inverse Fourier transform) and TFT (trainable Fourier transform) instead of IFFT and FFT to give the model more adaptability. The scheme changes into:

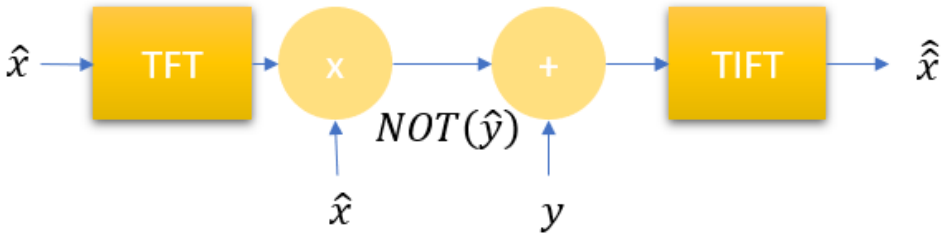


FIGURE 4-6 SCHEME OF BACK TO THE FUTURE WITH TRAINABLE FOURIER LAYERS

Back to the future leads to an improvement in performances when looking at PSNR, it is demonstrated that back to the future improves the loss, but it could introduce artifacts from a visual point of view on the reconstructed image.

The introduction of new different frequencies could lead to anomalies even if Flashback gets better, so it is necessary to measure the PSNR of the reconstruction and visually evaluate.

5 RESULTS

In this chapter the results related to the contributions presented in chapter 4 will be presented in the following order:

1. Presentation of the Dataset.
2. Results of LOUPE and norm tuning.
3. Tuning of slope s in σ_s and Prancing Pony.
4. Flashback used as training regularization term.
5. TIFT.
6. Back to the future.
7. Back to the future built with TFT and TIFT.
8. Flashback used as self-assessment.
9. Results of LOUPE with all contributions.

The evaluation of the model on the test set has been done always using a hard under-sampling threshold, avoiding the use of the soft under-sample σ_s the results are as close as possible to a real use situation.

Throughout the following sub-chapters PSNR will be used as evaluation metric.

5.1 Presentation of the Dataset

The dataset we used to train and evaluate the model [54] is composed by 110 patients' brain MRI, with every brain having from 20 to 60 MRI scans. We removed from the dataset all the images containing the last part of the head so to have all image containing informative scans. We reshaped every image to match the dimension $(128,128)$ and rescaled the pixel values of the images into the range $[0,1]$.

After pruning the inconsistent and low-informative images, the dataset contains 3441 images.

The dataset has been split into training set, containing 2688 images (80%) and test set containing 688 images (20%). For evaluation during training 15% of the training set has been used.

Figure below shows some scans randomly taken from the dataset.

Further on we will show a visualization example of how the trained models operate only showing an image taken from the dataset having a high number of details that, for this reason, we consider representative.

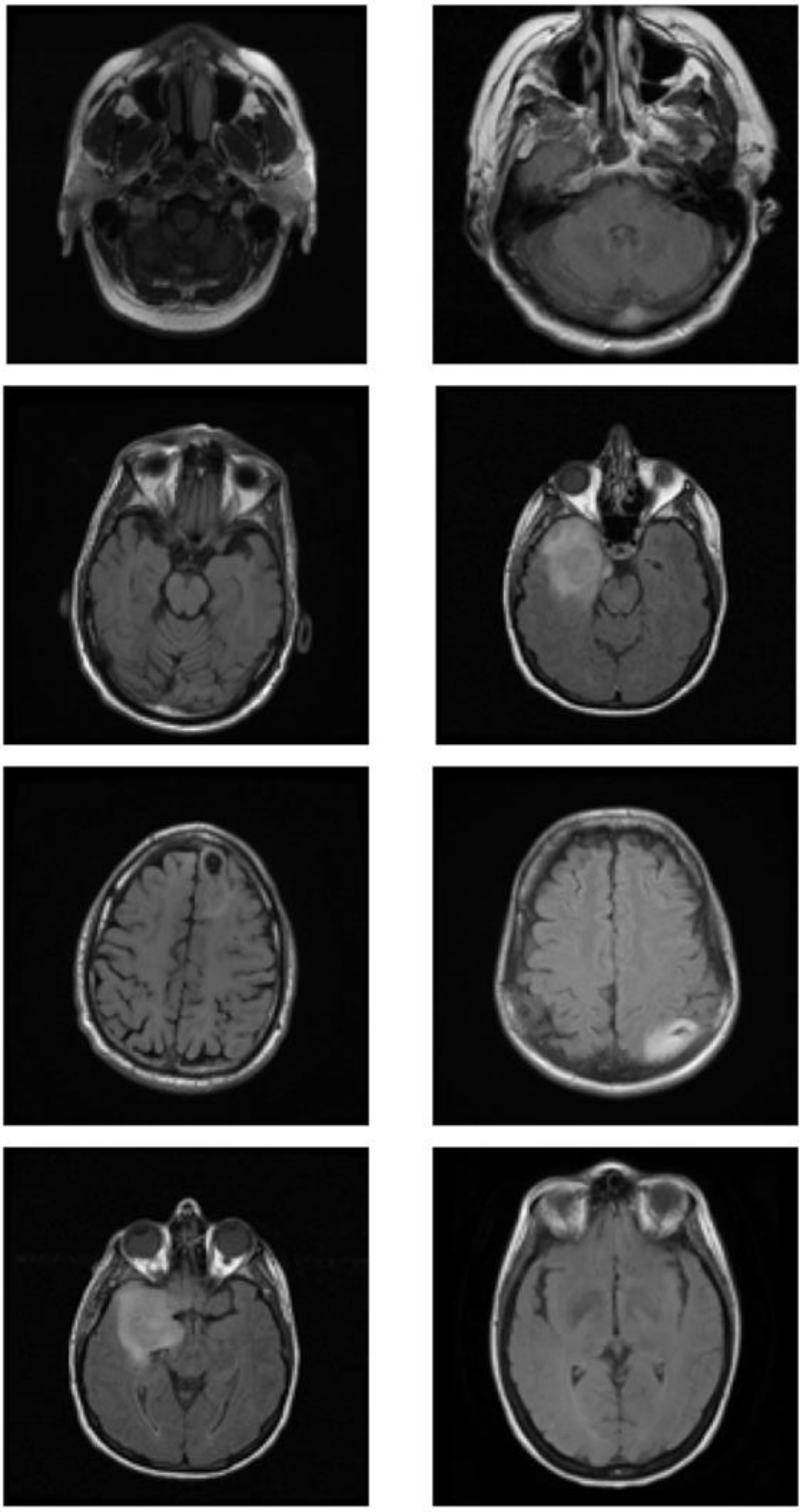


FIGURE 5-1 EXAMPLE OF MRI SCANS FROM THE DATA SET

5.2 Results of original LOUPE and norm tuning.

In [55] [40] LOUPE has been presented both with a loss function that used norm L_1 and a loss function that used norm L_2, in the last paper [40] the authors suggest to use L_1 norm because they found it to return the best results.

In this work both L_1 and L_2 have been evaluated on the presented dataset using different sparsity levels.

$$\left\| A_\theta \left(F^H diag\left(\sigma_s\left(N_\alpha\left(\sigma_l(O) \right) - U^{(k)} \right) \right) F x_j \right) - x_j \right\|_{L_2, L_1}$$

We evaluated the model on 7 different values of sparsity α, where α, as already described, is the mean of the mask M used to under-sample. A higher level of sparsity means a higher number of non-null elements.

The tuning has been done for every $\alpha \; in$ [0.05,0.07,0.1,0.125,0.15,0.2,0.25].

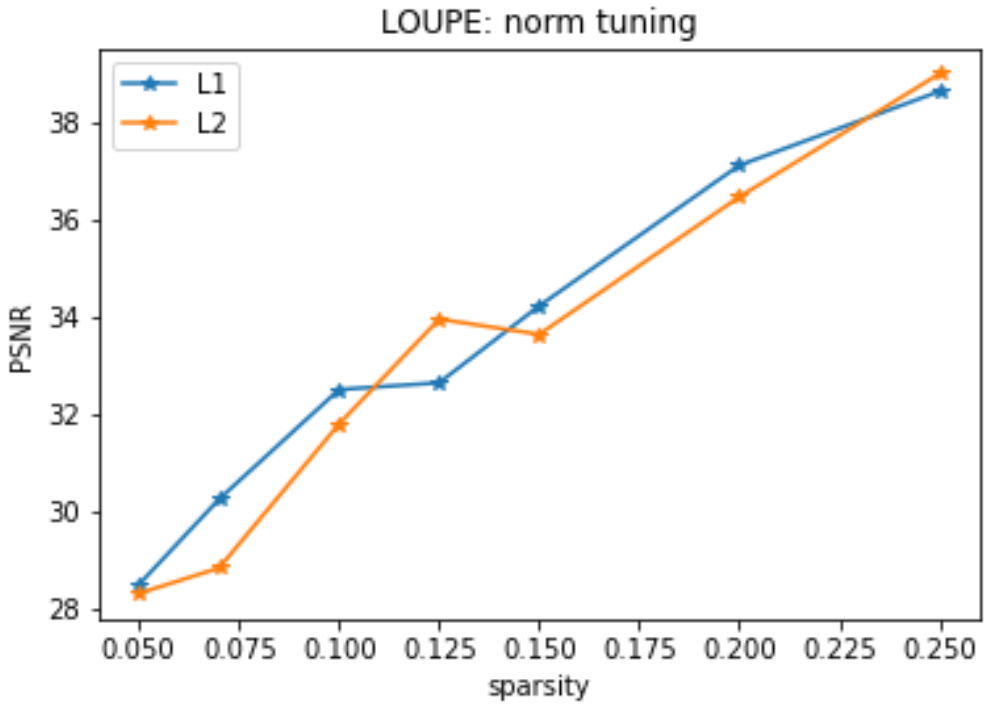
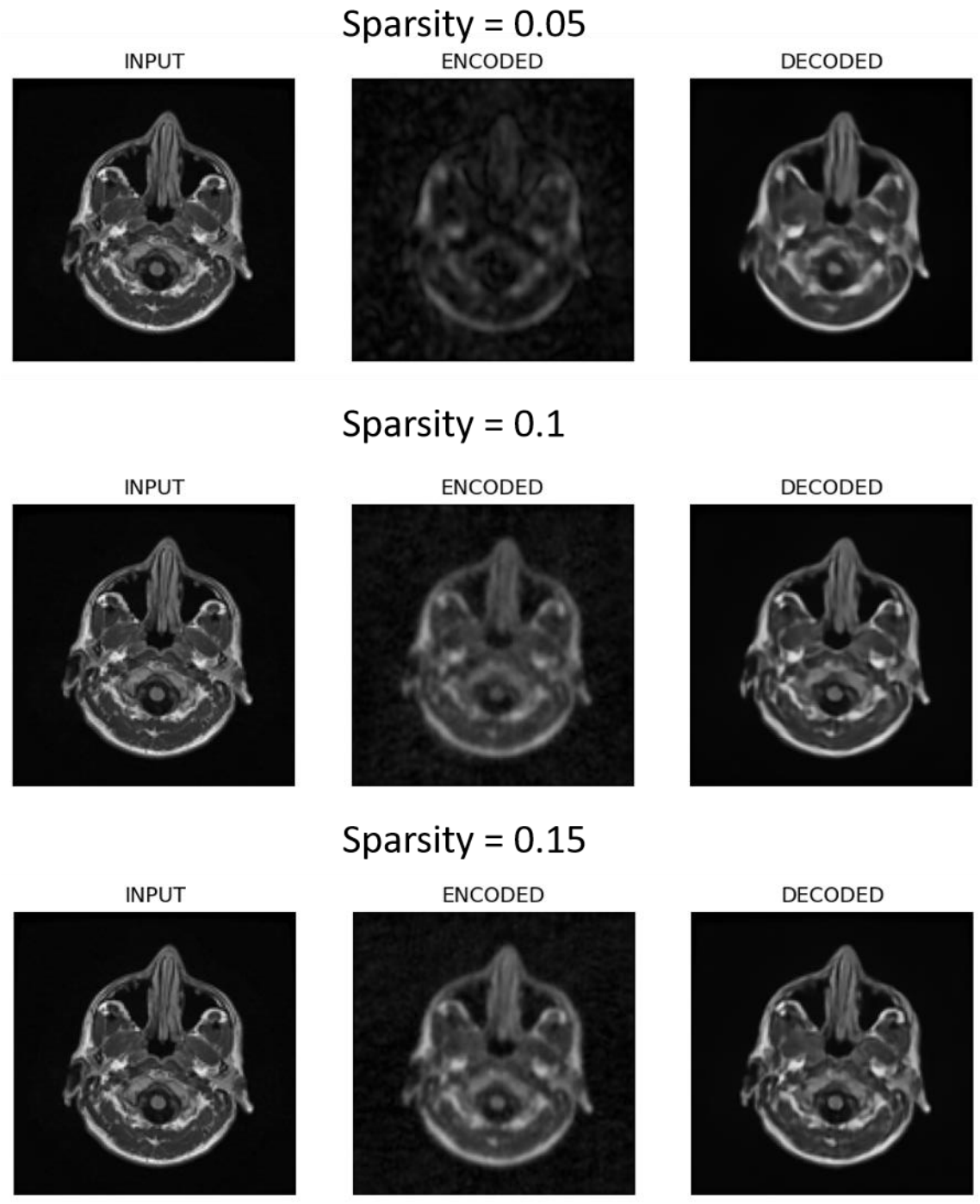


FIGURE 5-2 LOUPE L1 VS LOUPE L2

The graph shows different values of the reconstruction error (PSNR) for many sparsity values. The blue line represents the model trained using L_1 norm meanwhile the orange line represents the model trained using L_2 norm.

Following the analysis and the already presented considerations of [40] we confirm L_1 norm achieves the best results, for this reason we will focus on improving the performance of LOUPE keeping the L_1 version of loss function.

We set the obtained PSNR as the performance baseline we aim to surpass.



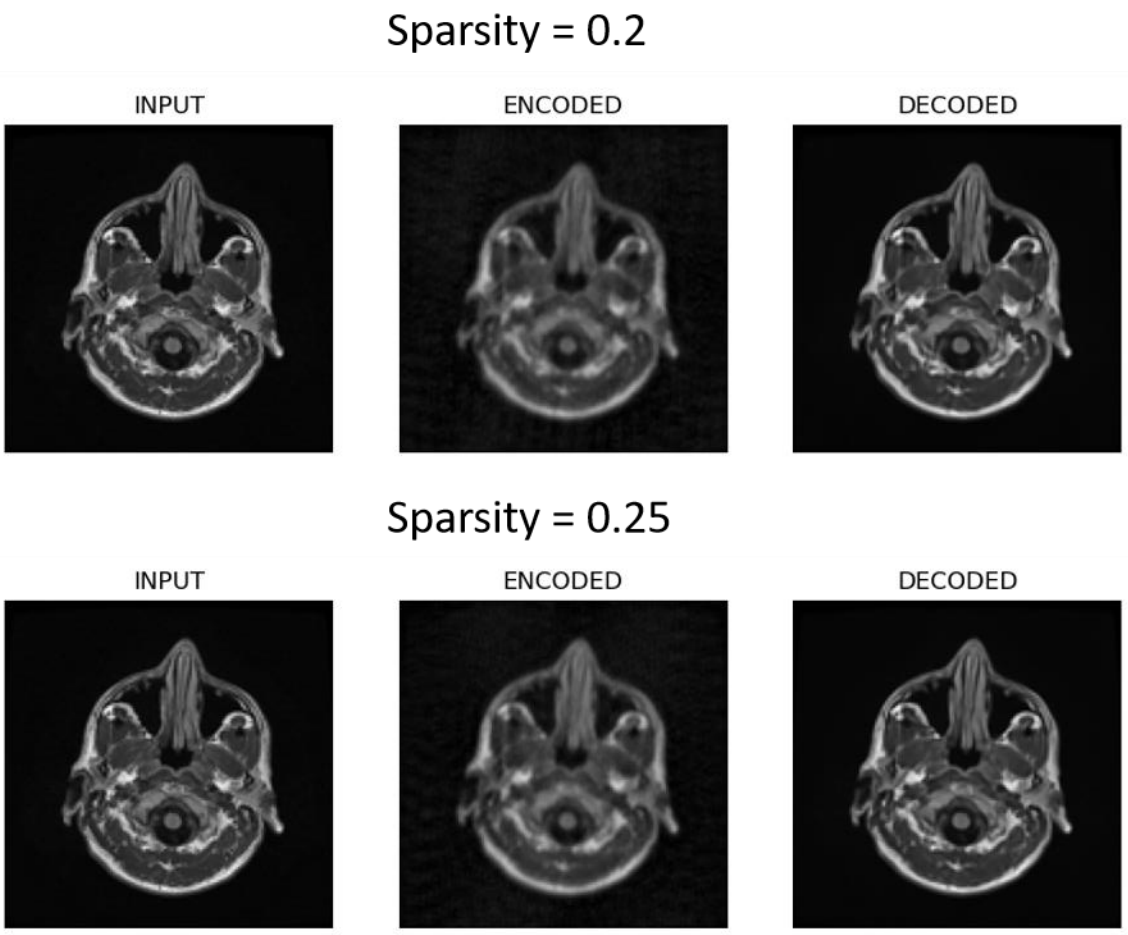


FIGURE 5-3 MRI SCANS ENCODED AND DECODED BY LOUPE

The Figure above represent in vertical order the models trained with $\alpha \in [0.05,0.1,0.15,0.2,0.25]$. For every row, the corresponds to a different model, are shown x: the ground truth (INPUT); $F^H y$: the under-sampled image in Euclidian domain before reconstruction (ENCODED); and $\hat{x}$: the image reconstructed by the decoder (DECODED).

5.3 Tuning of slope s in σ_s and Prancing Pony.

To have a trainable encoder is necessary to lighten the assumption of the under-sampling threshold. To achieve trainability, threshold is substituted by σ_s whom parameter s has to be tuned.

A high s means having a steep sigmoid, it improves similarity of σ_s with the threshold but at the same time leads to a difficult training. A low value of s improves trainability but it does not

guarantee the final mask will correctly under-sample. Setting s too low leads to a high number of pixels that are not as close as necessary to 0 or to 1.

If the mask contains too soft values (values that are not close enough to 0 or 1 but are in between them) the under-sampling procedure preserves more points than it should. Even if the mask multiplies the elements of y for a coefficient that is really small (close to 0 but not a true 0) the decoder can learn how to use that fragment of information. This turns out to be a problem when at inference time a hard threshold is used instead of σ_s and the decoder does not encounter the values it was used to but encounters true 0s and this leads to a loss of performances.

To achieve a good training s has to be well tuned to avoid complications.

We used 3 values of s: $s\ in\ [20,200,2000]$ where 200 is the value found in [40] that returns best performances. Also, we used the model with Prancing Pony that automatically selects the best s, without requiring to tune it.

The following graphs show the results of LOUPE for the three values of s and for $\alpha\ in\ [0.05,0.07,0.1,0.125,0.15,0.2,0.25]$. For every graph both the results on test-set using σ_s and threshold will be displayed, where the first is the result obtained at training time and the second is the result obtained at inference time.

The only model that from training time to inference time visibly drops its performances is the model $s = 20$. Models $s = 2000$ and Prancing Pony does not have difference from training to inference. Model $s = 200$ shows a really slight drop of performances around low sparsity values that are negligible.

The best performances, in terms of PSNR, are returned by $s = 200$ and by Prancing Pony.

Prancing Pony returns almost the same performances of $s = 200$ but it does not require the tuning of s. Prancing Pony automatically finds the best s by rising it until s_{max} and by keeping only the model that returns the best results.

It is evident from the Figure below that $s = 20$ achieves bad results at inference time because the model is not trained congruently: the decoder at inference sees different data then the data it has seen at training time.

We also notice how $s = 2000$ achieves bad results, we think because it tends to work with an almost vanishing gradient.

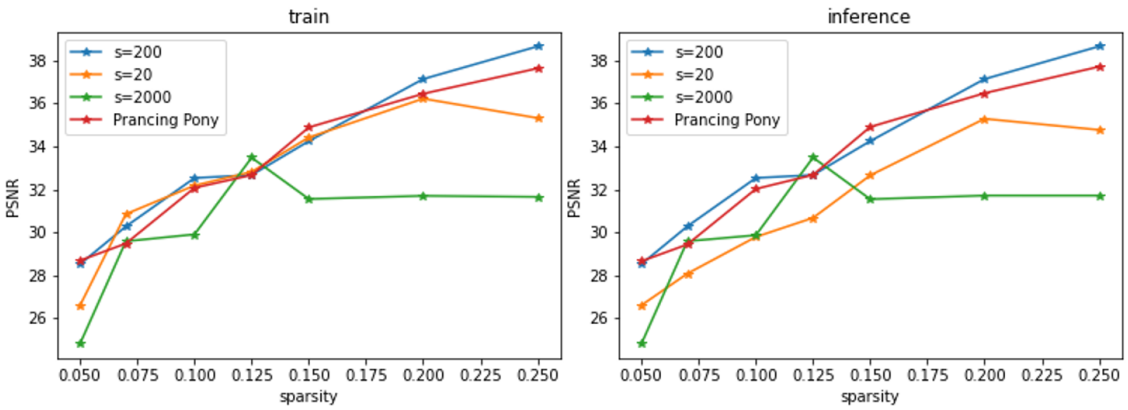


FIGURE 5-4 S TUNING: TRAIN

To explain the meaning of these results is necessary to look at the produced masks. In the following figures we will plot the histograms of the pixel values of all the masks produced to predict the test-set.

For every image of the test-set a random realization of mask m is created and what we plot is the distribution in bins of all the pixels. For a hard threshold mask the distribution results in being only a 2-valued graph at $pixel = 0$ and $pixel = 1$. Substituting the threshold with σ_s leads to a distribution with values in between 0 and 1.

In the following figures will be presented the distribution of the pixels of the masks for four models trained respectively with $s\ in$ [20,200,2000] and Prancing Pony.

The figure below shows how Prancing Pony achieves almost the same distribution of $s = 2000$, the distribution that best mimics the distribution of the threshold, but at the same time gets the same PSNR of $s = 200$ that gets the best score on the reconstruction error.

LOUPE plus Prancing Pony obtains the same results of the model with the best s that has to be found by tuning, but it works without asking for an s.

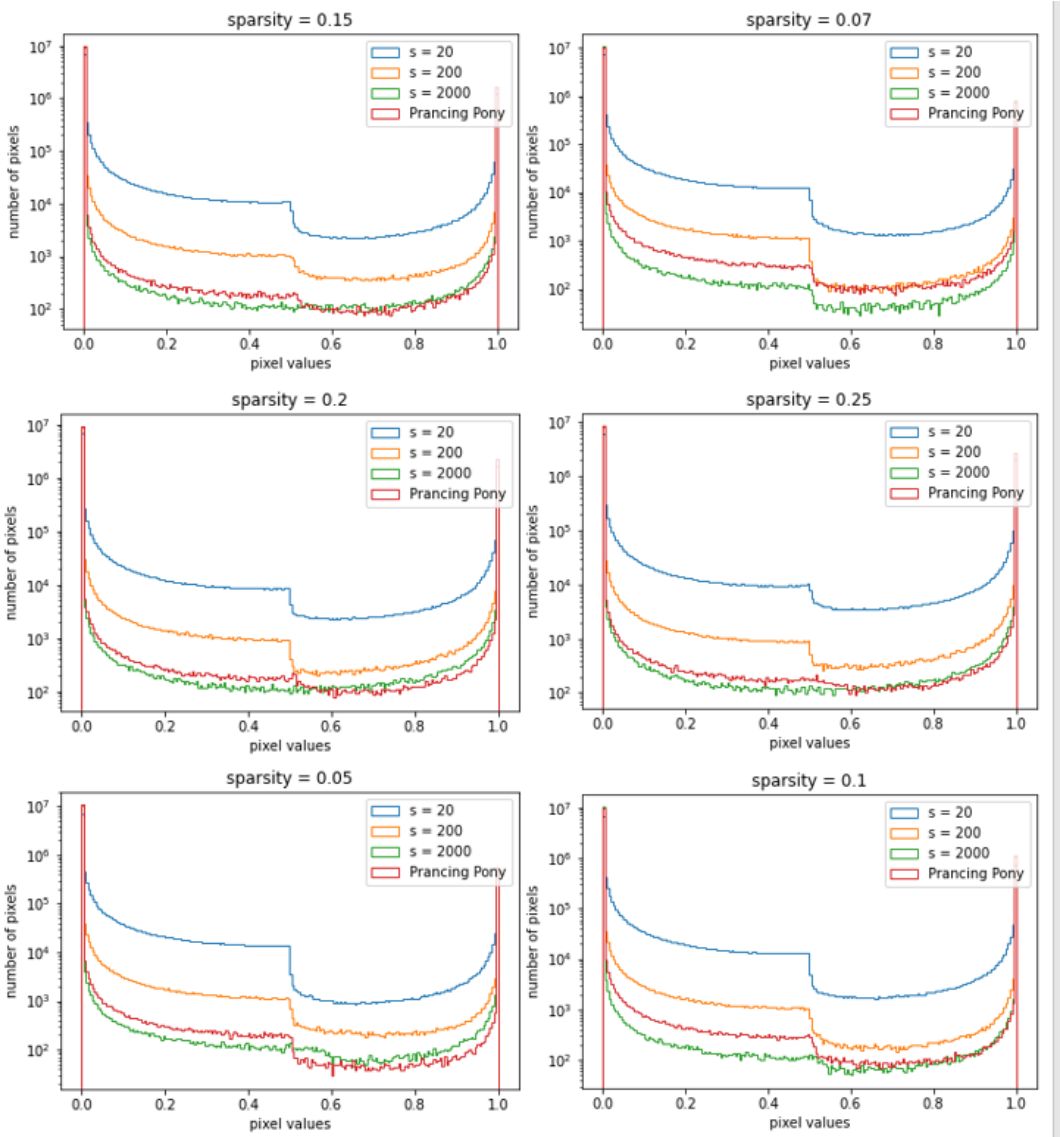


FIGURE 5-5 DISTRIBUTION OF THE MASK PIXELS

5.4 Flashback used as training regularization term.

Flashback looks at the reconstruction error between y, the under-sampled input image in k-space and $\hat{y}$ the under-sampled reconstructed image in k-space:

$$F_{lash} = \|y - \hat{y}\|_{L.}$$

Where $L.$ points out that Flashback can be done using whatever norm. In this work we focus on the use of norm L_1 and L_2 and we keep the one that returns the best performances.

In this chapter we use Flashback as a training regularization terms to induce LOUPE focus on the reconstruction of y. The loss function becomes:

$$\phi\left\|A_\theta\left(F^H y_j\right) - x_j\right\|_{L_1} + (1-\phi)\left\|diag\left(M^{(k)}\right)FA_\theta\left(F^H y_j\right) - y_j\right\|_{L_\cdot}$$

Where $y = diag\left(M^{(k)}\right)Fx_j$, and j is the index for the input images, and ϕ is the parameter that weights the reconstruction error and $(1-\phi)$ is the parameter that weights Flashback.

Flashback introduces complexity because ϕ is a parameter to tune. In this chapter we present the results obtained by LOUPE plus Flashback on different values of ϕ.

The following graphs show the results of LOUPE for $L_\cdot = L_1, L_2$ and for $\phi\ in$ [0.99,0.9925,0.995,0.9975,0.999,0.99925,0.9995,0.99975,0.9999].

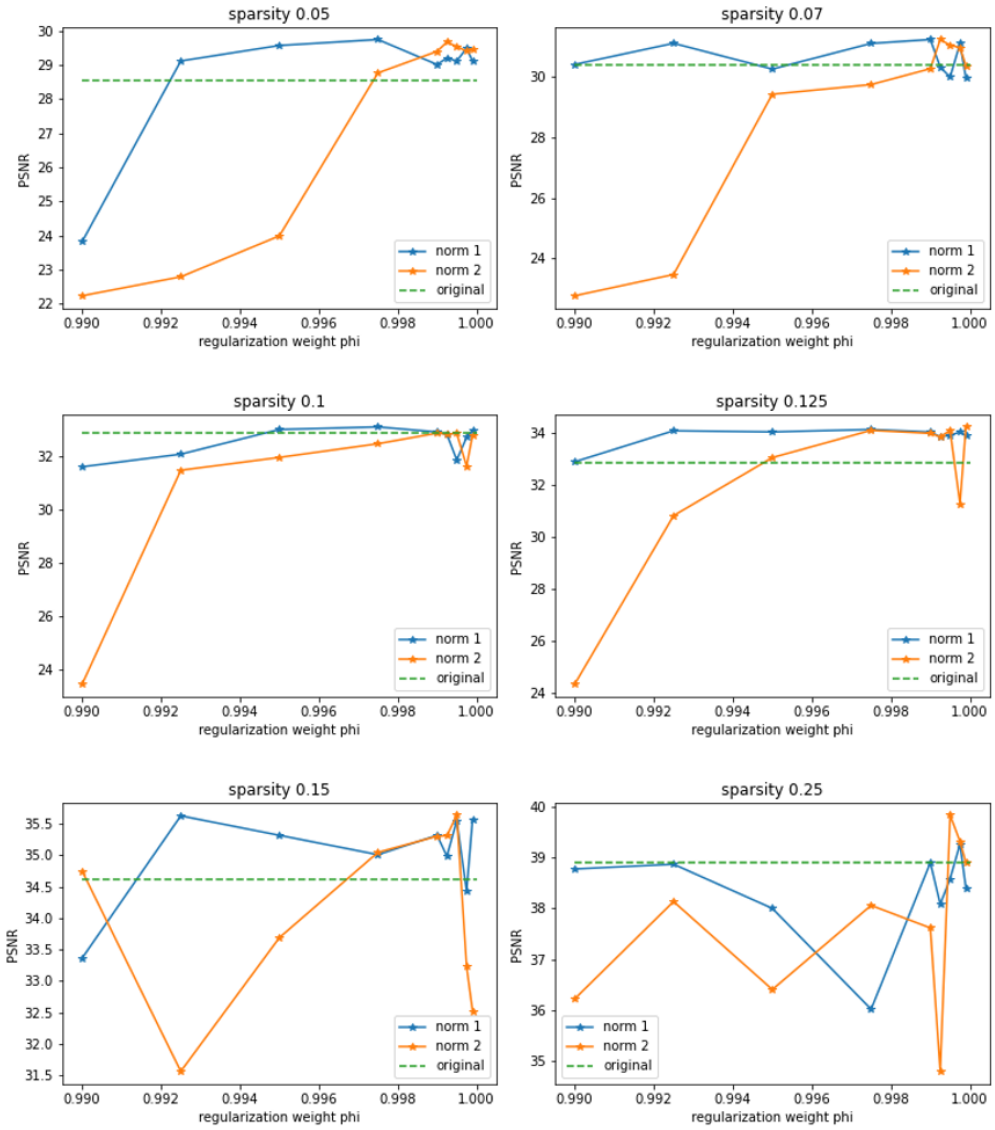


FIGURE 5-6 FLASHBACK PSNR OVER PHI FOR A SET OF ALPHAS

The plots highlight that when a correct value of ϕ is given to Flashback, LOUPE improves its performances.

In particular we can visualize what is the best ϕ by looking at the following figure where respectively for every α (sparsity) is shown what ϕ returns the best $PSNR$.

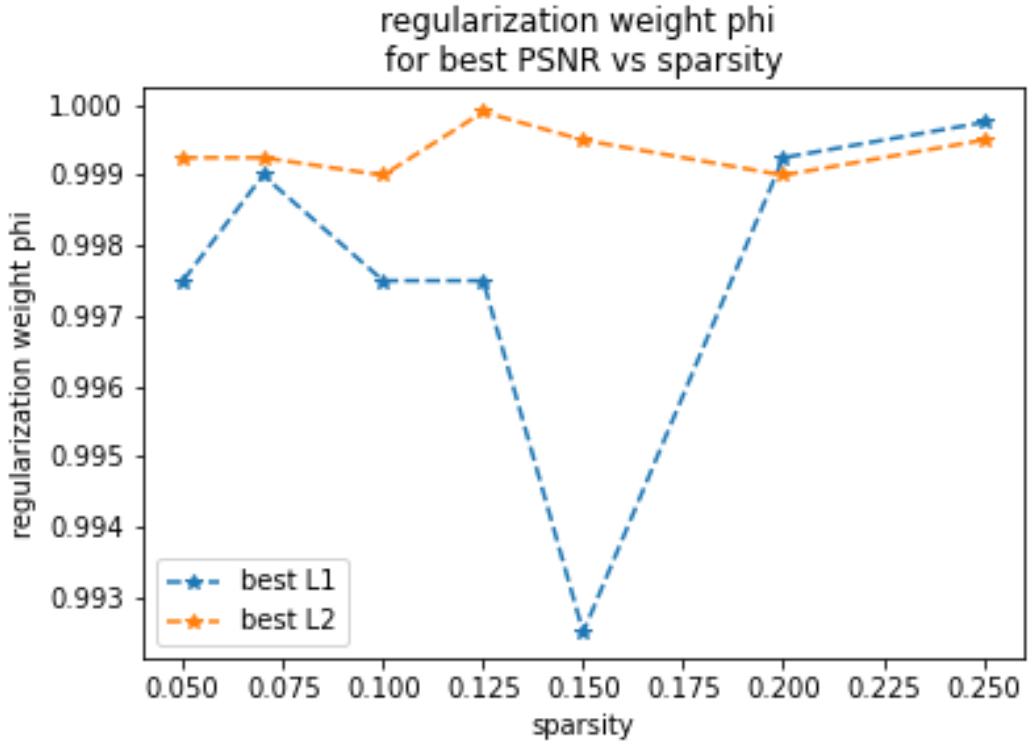


FIGURE 5-7 FLASHBACK: PHI TUNING

It is also possible to tune $L_.$: from the following graph we can identify what norm L_1 or L_2 returns the best PSNR.

Both Flashback with L_1 and L_2 return similar PSNR values but L_2 presents advantages for high α values with respect to the other norm.

From the Figure above we can also consider L_2 easier to tune because best PSNR is returned by a set of ϕ with a smaller variance with respect to the ϕ that return the best PSNR for L_1 norm Flashback.

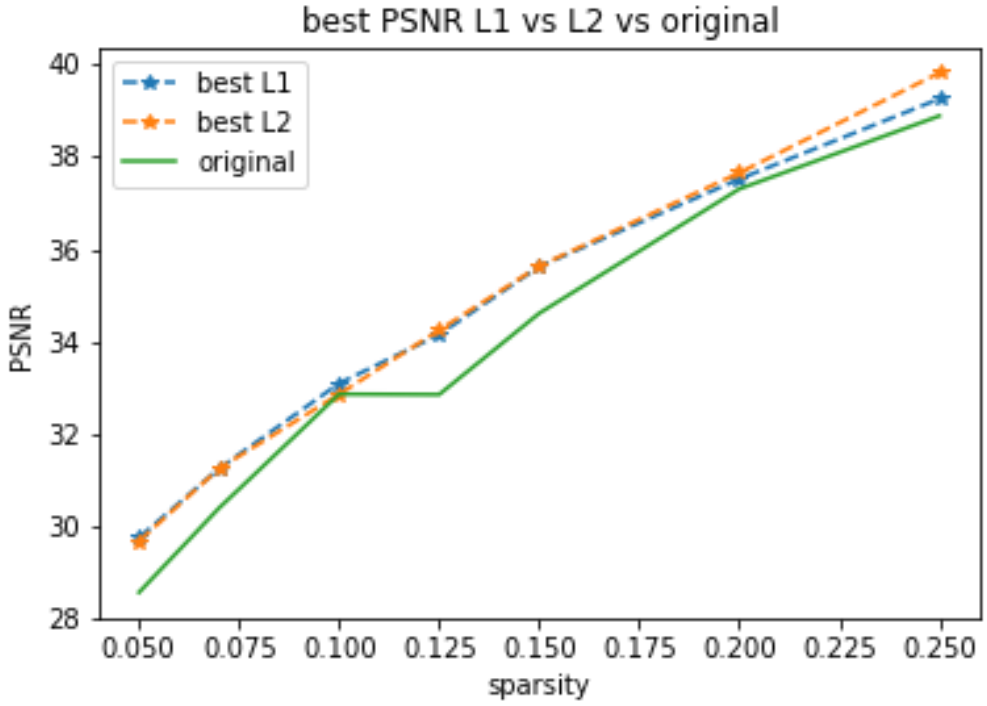


FIGURE 5-8 FLASHBACK: TUNING OF THE NORM

5.5 TIFT.

TIFT stands for trainable inverse Fourier transform, it is a layer used instead of *IFFT* that mimics the behaviour of *IFFT* as long as it is set non-trainable. After *TIFT*'s attribute of trainability is set True, starting from the initial *IFFT* configuration, the net can adapt its weights trying to improve performances.

We tried many training procedures:

1. Training LOUPE starting with the trainable property of *TIFT* set as True since initial epoch.
2. Training LOUPE, first with the trainable property of *TIFT* set as False until Early Stop occurs, and secondly with trainable property of *TIFT* set as True.
3. Training LOUPE, first the with trainable property of *TIFT* set as False until Early Stop occurs, and secondly with the trainable property of *TIFT* set as True, using the hard threshold instead of sigmoid to under-sample and freezing the mask that is set as non-trainable.

The first method allows the model to train mask, U-NET and TIFT at the same time since the first epoch.

The second method guarantees a regular training for LOUPE that first lets the mask and U-NET converge. Once LOUPE can start from a good convergence point, complexity is raised by setting the trainability of $TIFT$ True.

Finally, the third method, as the second one, ensures the training of $TIFT$ starts only with an already trained LOUPE, but instead of training the model in the same way, it tries to adapt to the condition imposed at inference time when the under-sampling is done using a hard threshold. Because of a hard threshold is used, the gradient cannot be computed, and the encoder is set non-trainable.

The model trained using the first method has convergence problems and performs quite bad. We assume that starting from a LOUPE model whom complexity is increased by $TIFT$ leads to difficult trainings and the other two methods solves this problem by first setting $TIFT$ non-trainable. Performances of this method are not displayed because are low and not interesting.

Results of the second method are plotted in the Figure below.

The third method did not lead to improvements nor worsening of the performances, it resulted in being ineffective.

For simplicity's sake in the following figure only method two is considered and compared to LOUPE trained without $TIFT$.

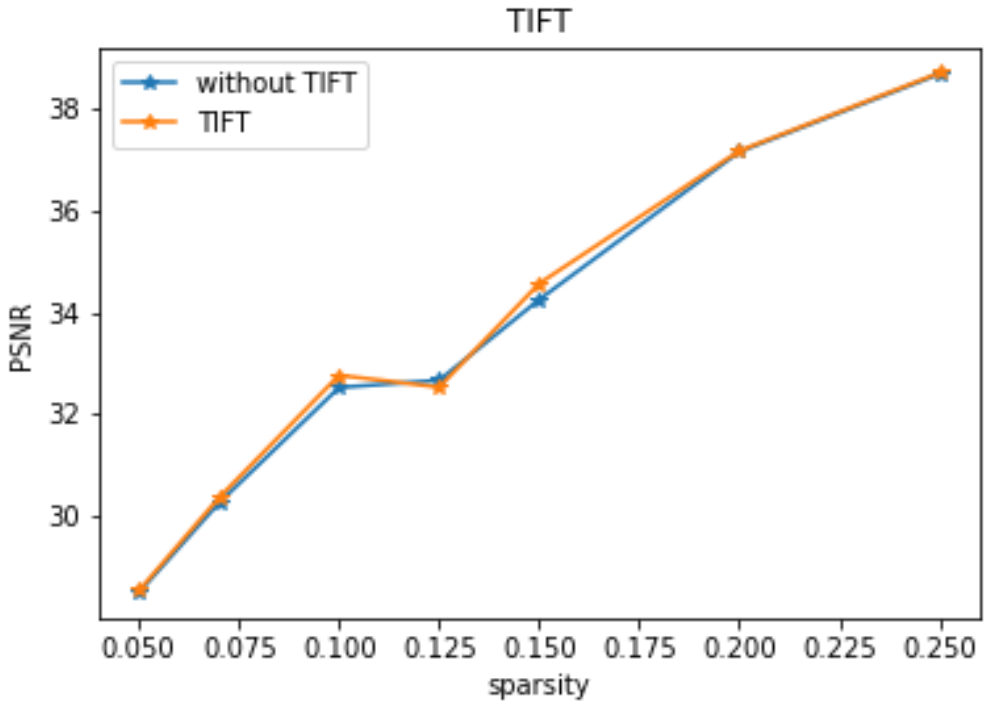


FIGURE 5-9 WITHOUT TIFT VS TIFT

Results suggest $TIFT$ does not improve the net considerably, only in few points $TIFT$ gets a small fraction of dB more with respect to the original.

5.6 Back to the future

As Flashback underlines the importance of the problem of reconstruction of the original components y, known to be ground truth, it shows that if guided through the training by a regularization factor, LOUPE exhibits an improvement of performances.

Flashback let LOUPE focus, during training, on the preservation of y by the use of a regularization term, but a more direct way to solve the problem is to substitute the reconstructed $\hat{y}$ with the original one (y). Back to the future does not limit the efficiency or the validity of Flashback because it is implemented as a separate block that operates only after the reconstruction of $\hat{x}$ (after Flashback), ensuring to correct all the errors done only on y. Flashback helps LOUPE training in general because it points out what is a better road to follow helping all the image reconstruction.

Back to the future in this implementation does not require any training and its usage is immediate.

Results for LOUPE plus back to the future are displayed in the following figure.

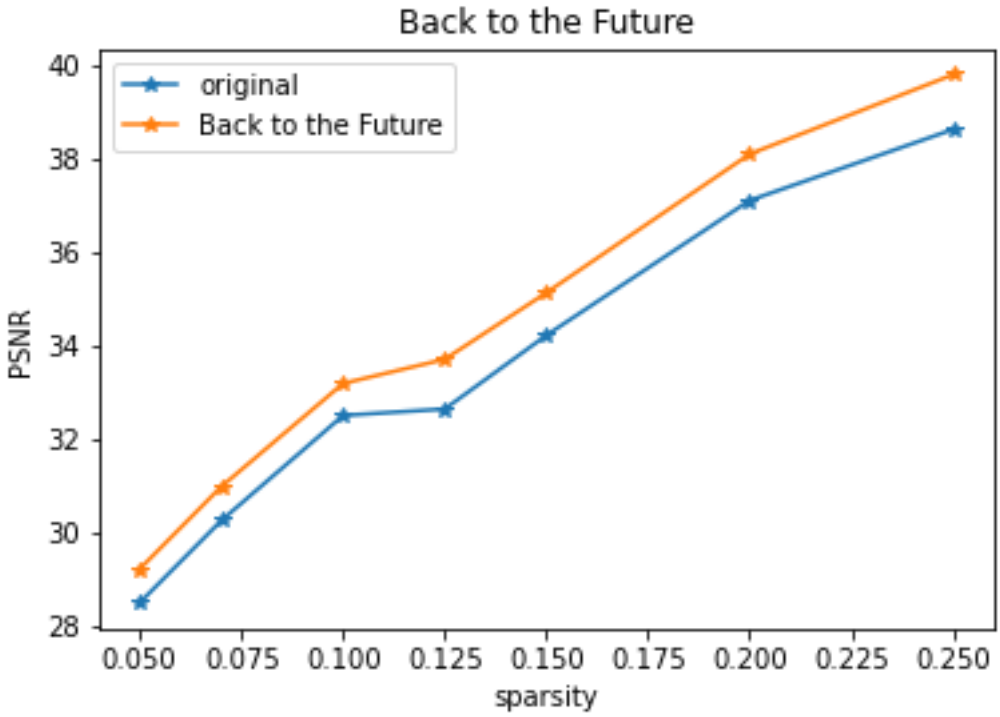


FIGURE 5-10 LOUPE VS LOUPE AND BACK TO THE FUTURE

From the plot we notice how back to the future always brings an improvement to the PSNR that is around 1dB. In particular we notice a greater improvement for the high-α values, that is explainable considering the higher portion of image that is reconstructed with respect to the low-α values models that are less interested by the method.

5.7 Back to the future built with TFT and TIFT.

As back to the future is implement using FFT and $IFFT$, it is possible to convert the two layers (FFT and $IFFT$ layers) with their trainable versions, called Trainable Fourier Transform (TFT) and Trainable Inverse Fourier Transform ($TIFT$).

$TIFT$ has already been described in chapter 4.4, and used, results have been shown in 5.5. In this chapter we also use TFT: the non-inverse version of $TIFT$. The only difference between these two layers is the initialization of the imaginary part of the Super Dense layers, that in TFT requires negated values with respect to $TIFT$ values.

Differently from the previous version of back to the future this implementation requires to be trained. Training starts only after LOUPE has already been trained until convergence, back to the future trains over the prediction of LOUPE $\hat{x}$, producing $\hat{\hat{x}}$ and minimizing $\left\|\hat{\hat{x}} - x\right\|_{L_1}$

Results are displayed in the following Figure:

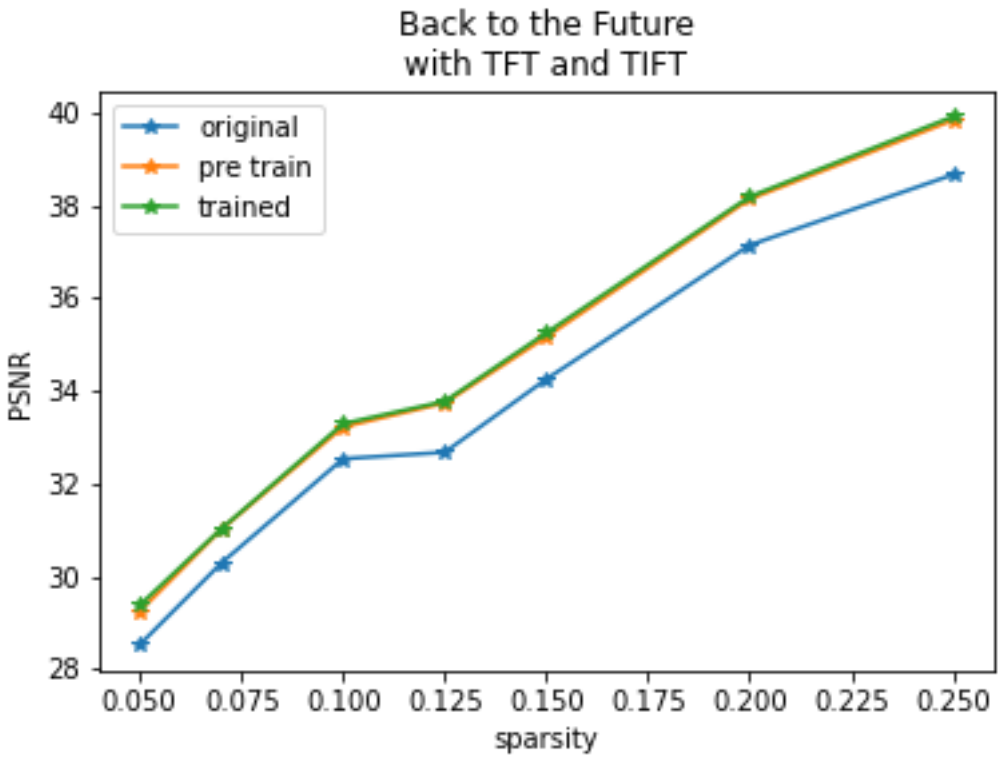


FIGURE 5-11 LOUPE WITH BACK TO THE FUTURE BUILT USING TFT AND TIFT

The introduction of TFT and $TIFT$ introduces no relevant results to the already developed back to the future. In the figure above the two lines are almost overlapped meaning the new version does not perform sensibly better.

5.8 Flashback used as self-assessment.

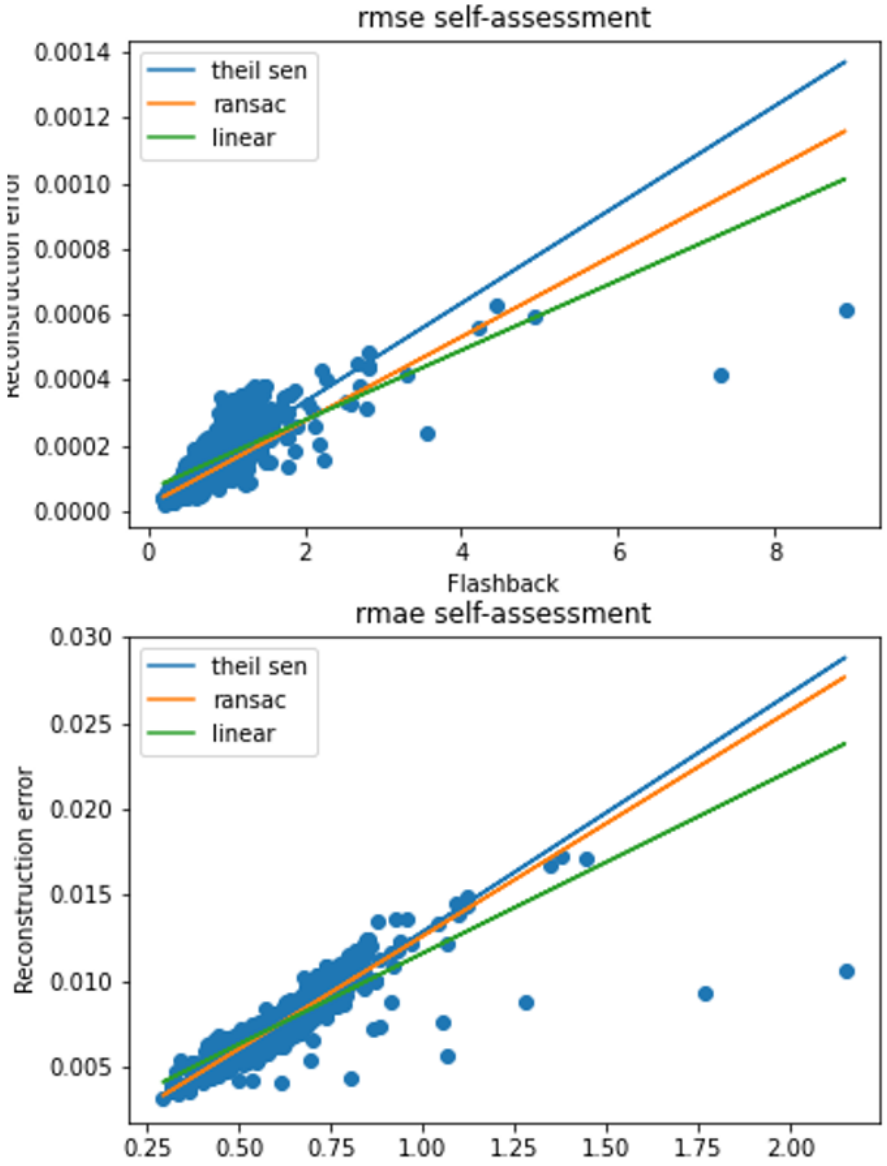
It can be shown that a relationship between Flashback and the reconstruction error exists, mathematically speaking, between $\|y - \hat{y}\|_{L_2}$ and $\|x - \hat{x}\|_{L_1}$, we say, we have a dependency.

This relationship can be used at inference time to have an estimation of the reconstruction error even if the original x is not provided. Analysing y, the physical output returned by the MRI machine that is always available, we can evaluate the reconstruction of it by looking at $\hat{y}$ by using Flashback, whose work is to compute $\|y - \hat{y}\|_{L_2}$. Because a relationship between

Flashback and the reconstruction error is known, is possible to use Flashback to estimate the reconstruction error, or in other word as a self-assessment.

The relationship has to be found at training time when x is available. In the Figure below reconstruction error is plotted against Flashback, so to see how Flashback and reconstruction error are linked together.

We call the following plots Squiddy plots.



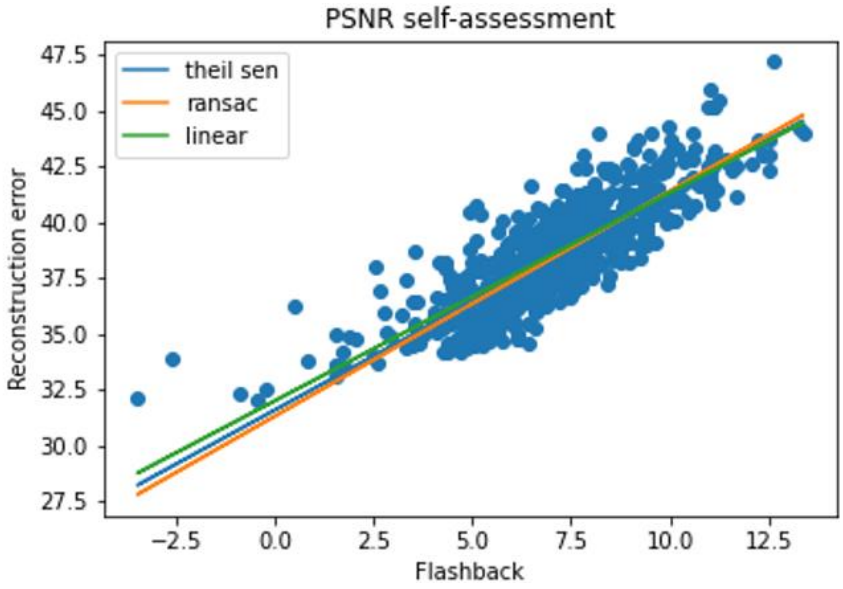


FIGURE 5-12 SQUIDDY PLOT: RECONSTRUCTION ERROR VS FLASHBACK

The blue points are the values of respectively MSE, MAE and PSNR obtained by LOUPE, trained with sparsity $\alpha = 0.25$, on the whole test set.

The Squiddy plot shows the polynomial functions used to fit the data. To robustly fit a line we tried 3 different models implemented in scikit-learn [56]: Linear Regressor, Theil Sen estimator and Ransac estimator [57].

Ransac method has been forced to have a small y-intercept to fit at best MSE and MAE points.

The plotted metrics are, in order, MAE, PSNR and MSE. All of the three metrics can be used with the purpose of self-assessment: one can use a metric returned by Flashback to predict the metric that would have been returned by the loss function.

In the next Figure fitting polynomials with order greater than 1 are used as self-assessment on the test data for model trained with sparsity $\alpha = 0.2$.

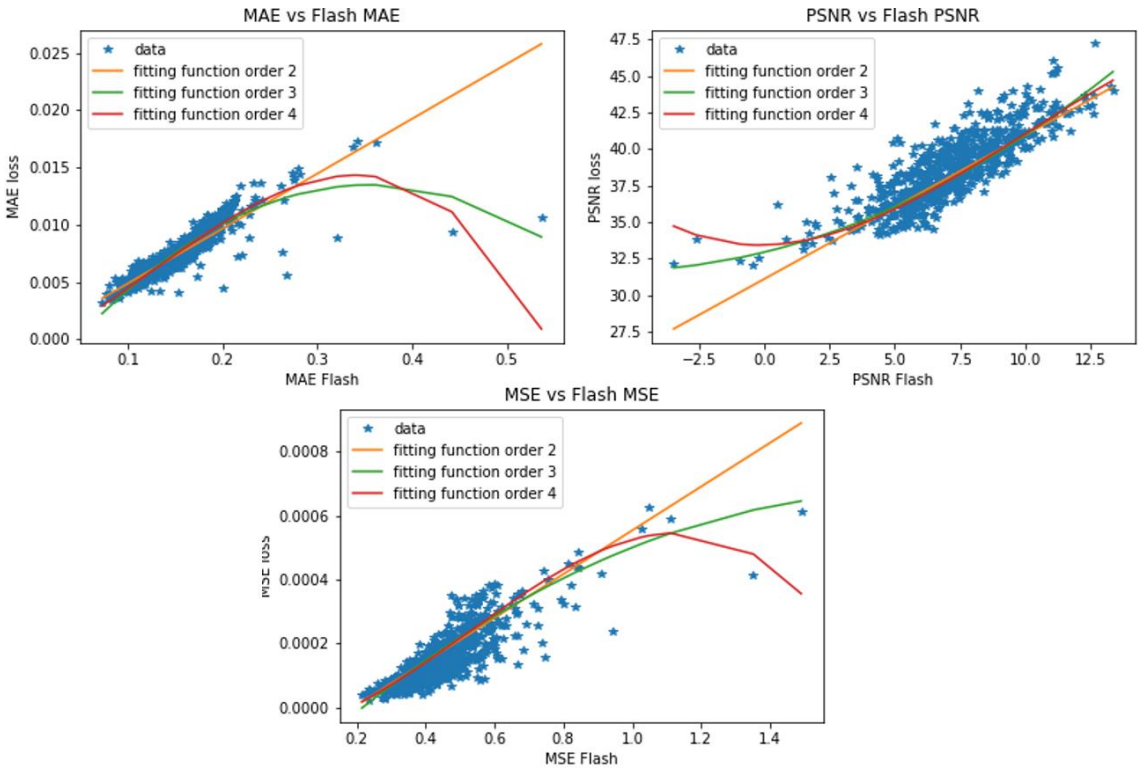


FIGURE 5-13 SQUIDDY PLOT: FITTING WITH POLYNOMIAL OF DEGREE GREATER THAN 1

We maintain the use of the first order polynomial to fit the data because it gives good approximations and, in general, a better generalization with respect to the higher order polynomials.

Is possible to notice from Figure 5.10 how Ransac and Theil Sen estimators return robust line fittings, discarding the points considered outliers.

Even if Ransac usually returns the best line, it also commonly returns a different line at every iteration, so that it is not a stable algorithm, but it has to be tuned by the addition of some constraints such as the norm of the *y-intercept* to converge to a low variance result over many iterations, to return a convergent line.

To better evaluate the prediction of the reconstruction error returned by Flashback as self-assessment is possible to visualize the distribution of the error on the predictions for all the metrics and for all the fitting models.

In the following figures the errors on the predictions are shown in the following order: MSE, MAE and PSNR. In every Figure the fitting models are presented in the following order: Linear Regressor, Theil Sen and Ransac.

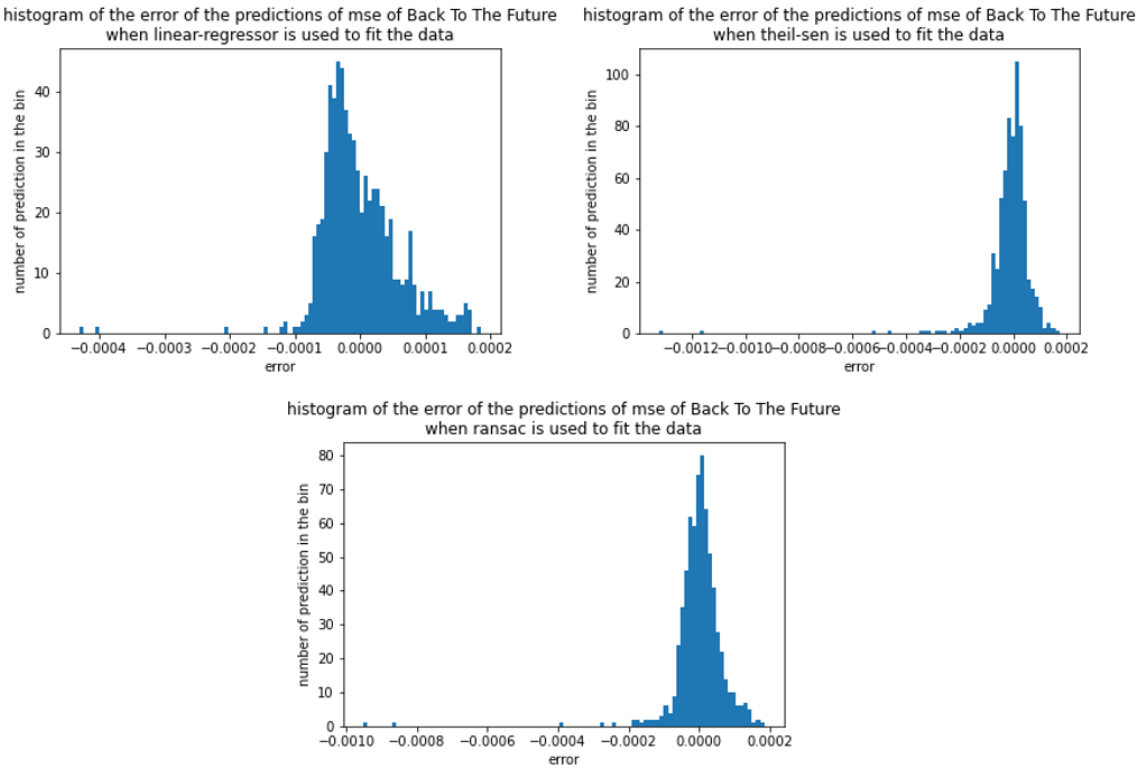


FIGURE 5-14 HISTOGRAM OF THE PREDICTED MSE ERROR

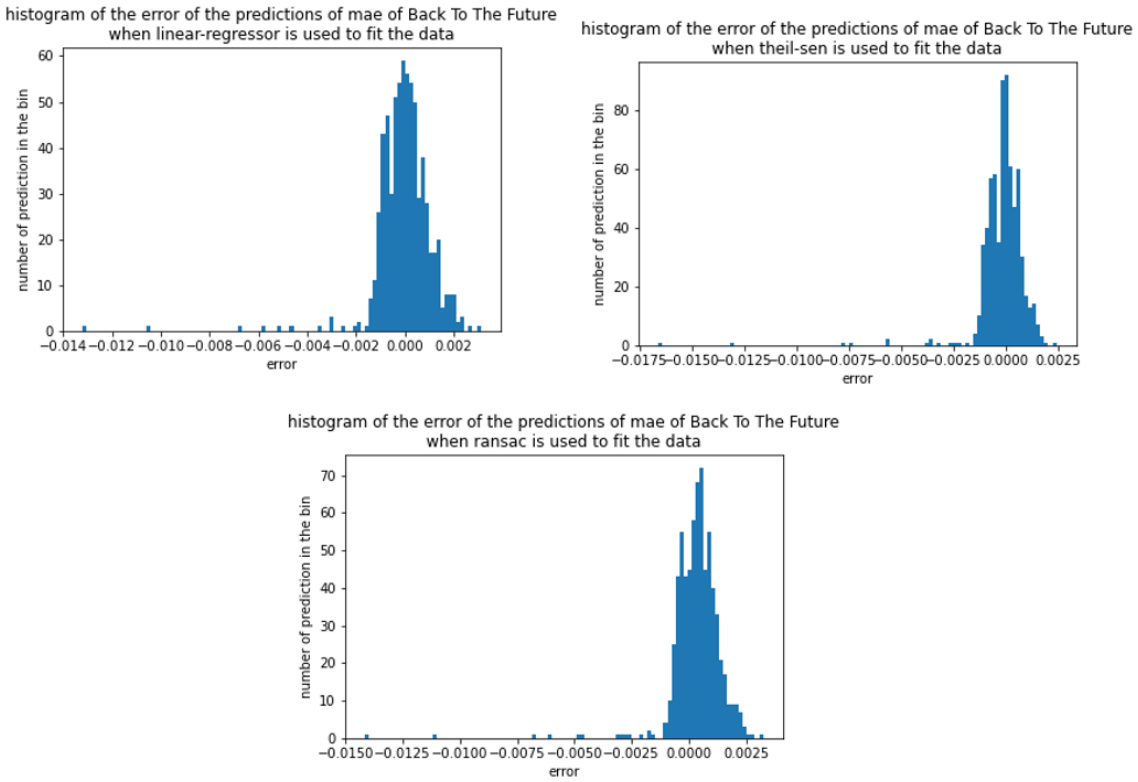


FIGURE 5-15 HISTOGRAM OF THE PREDICTED MAE ERROR

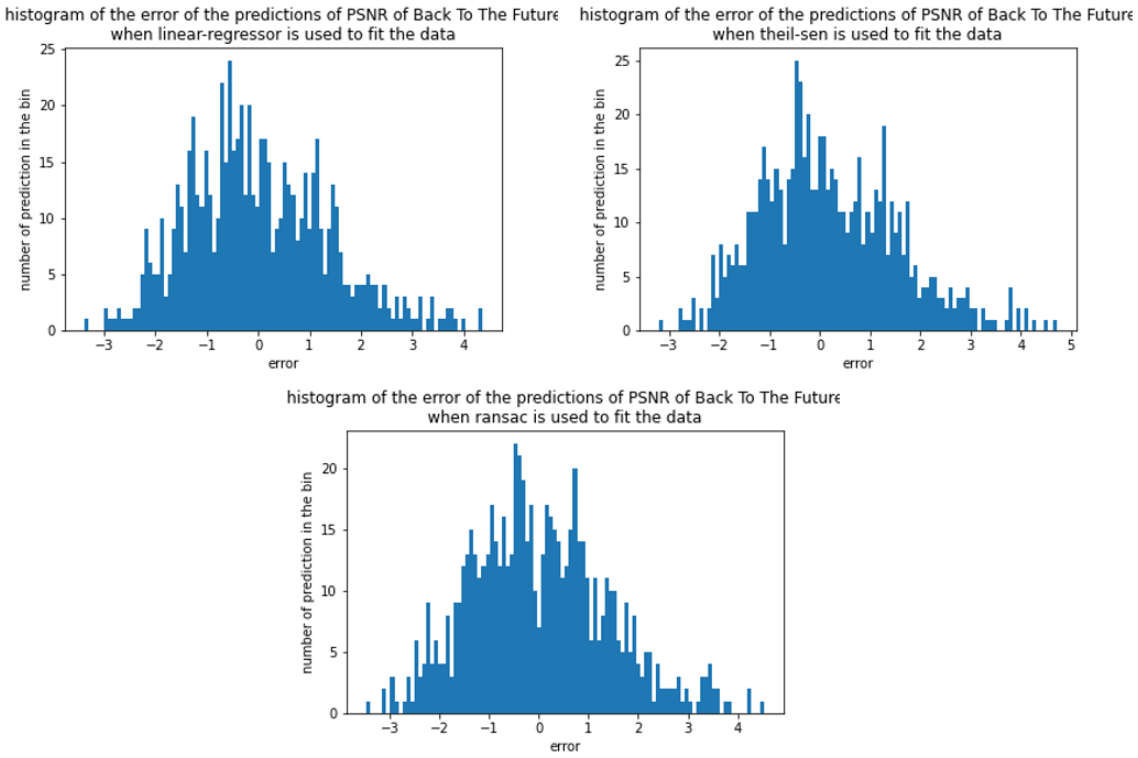


FIGURE 5-16 HISTOGRAM OF THE PREDICTED PSNR ERROR

To study which of the self-assessment combinations works best, returning the most reliable predictions we can look at the mean, variance, and relative mean absolute error of the error on the predictions of all the combinations.

The following table shows the mean value of the errors:

mean	Linear Regressor	Theil Sen	Ransac
MSE	1.76e-20	-2.38e-05	-7.91e-06
MAE	8.27e-19	-2.65e-05	-2.40e-05
PSNR	8.26e-17	0.15	0.36

The following table shows the variance of the errors:

variance	Linear Regressor	Theil Sen	Ransac
MSE	3.55e-09	1.61e-08	6.68e-09
MAE	1.30e-06	1.36e-06	1.45e-06
PSNR	1.84	1.85	1.86

The following table shows the mean relative absolute error $\frac{(real - prediction)}{real}$ of the errors:

Mean absolute error	Linear Regressor	Theil Sen	Ransac
MSE	0.38	0.45	0.37
MAE	0.099	0.091	0.089
PSNR	0.028	0.029	0.028

The values reported in the tables and the distributions plotted in Figures 5.12, 5.13 and 5.14 do not express an absolute advantage of one model with respect to the others for many reasons, for example because the metrics are only valid for a model built with $\alpha = 0.25$ and because Theil Sen and Ransac return fitting lines that are the output of a random process and at every iteration the two models produce different lines.

It is however clear that predicting the reconstruction PSNR, looking at the Flashback PSNR assures the lowest Mean Absolute Error. At the same time working on MAE guarantees around 9% of Mean Absolute Error and a lower variance.

Linear Regressor always has the lower mean on the errors, but the model is not robust to outliers and in this leads to a loss of performances at inference time.

The best self-assessment fitting lines are Theil Sen and Ransac used on MAE and PSNR.

Finally, we show that the hypothesis of having a distributed uniform reconstruction error over the pixels of $\hat{x}$ is not valid and a fitting line can only be estimated because an analytical solution is not known.

Assuming x_j is a generic pixel of a generic image of the whole dataset, in the following figure we show:

$$\left\|diag(M)Fx_j - diag(M)F\hat{x}_j\right\|_{L_2}^2 and \left\|diag(\bar{M})Fx_j - diag(\bar{M})F\hat{x}_j\right\|_{L_2}^2$$

distributions for every pixel in every image of the test set. M is a hard threshold under-sampling mask and $\bar{M}$ is the result of $M < 0.5$, $\bar{M}$ simply keeps the elements discarded by M and discards the element kept by M.

The first equation returns the reconstruction error of an input pixel of y (in the Fourier domain), in other words, it is a version of Flashback that does not work on the whole y but on a pixel of it. The second equation returns the reconstruction error of a pixel that was discarded during the under-sampling.

In the Figure below we present the distributions of the errors on the pixels for the model trained using sparsity $\alpha \in [0.25,0.2,0.1,0.05]$ following the decreasing sparsity order using a log scale for both the x-axis and the y-axis.

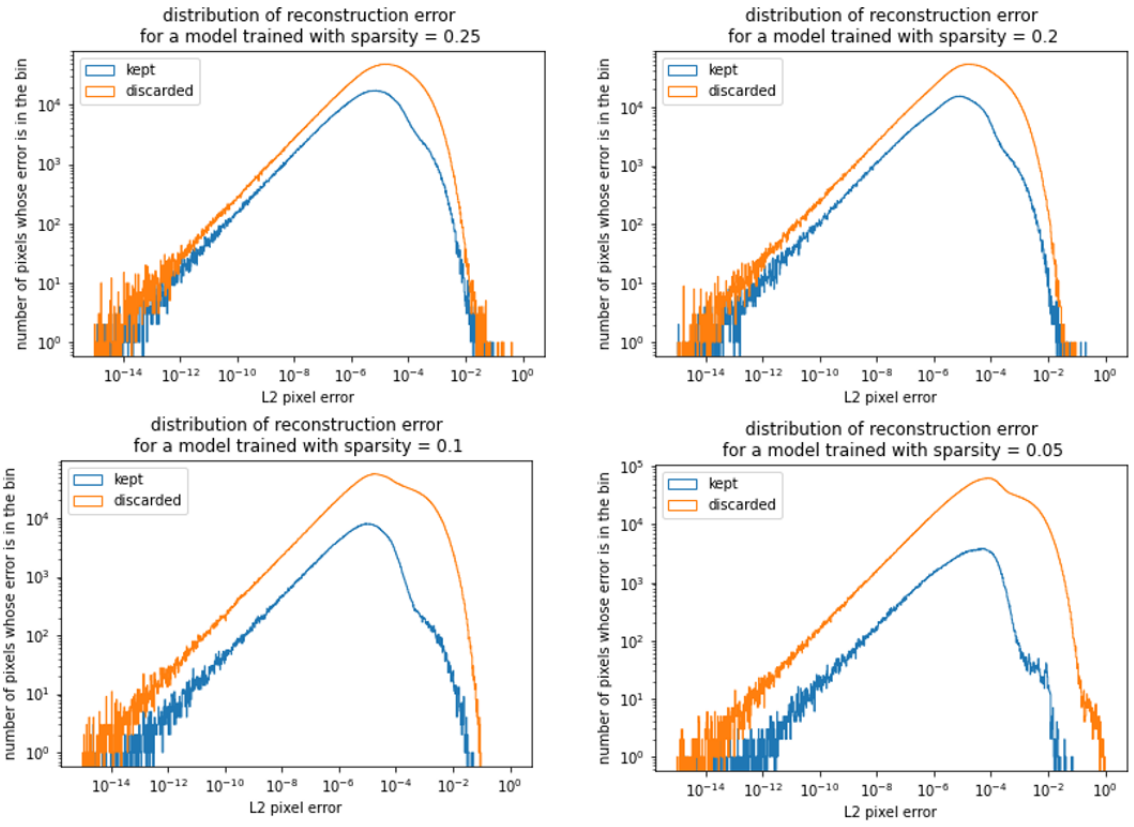


FIGURE 5-17 DISTRIBUTION OF THE ERROR ON RECONSTRUCTED PIXELS

5.9 Results of LOUPE with all contributions.

Finally, we present LOUPE trained with all the contributions and we compare the new results with the results returned by original LOUPE.

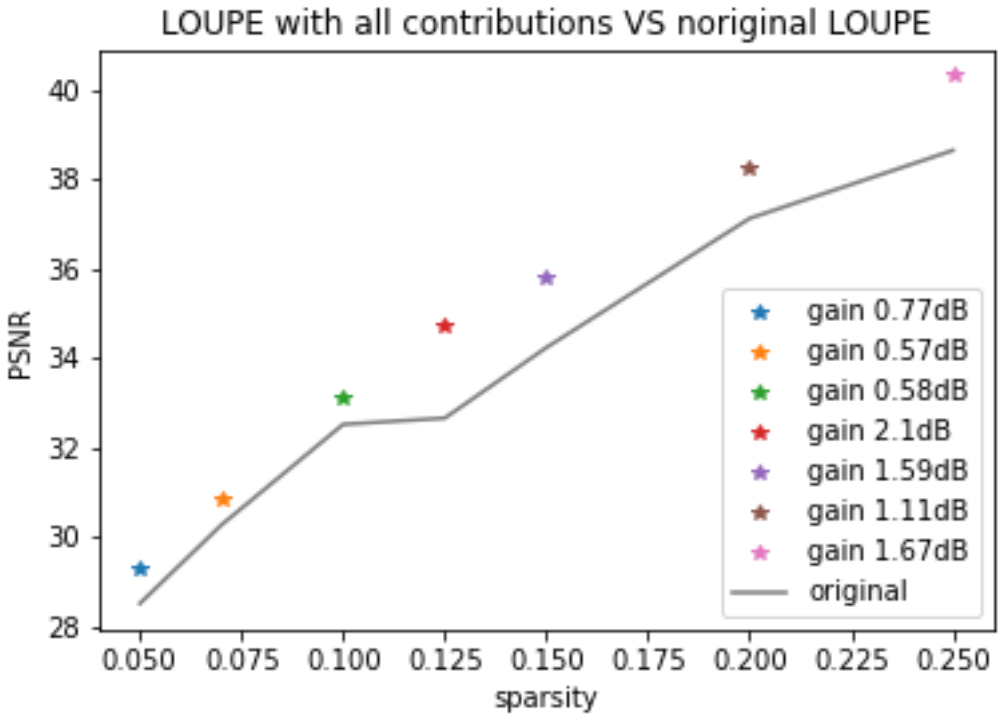


FIGURE 5-18 LOUPE WITH ALL CONTRIBUTIONS VS ORIGINAL LOUPE

The Figure shows the improvements gained for every tested sparsity. For high values of sparsity our contributions let LOUPE gain more dB, with respect to the improvements for low sparsity values. This is interesting from an implementation point of view because doctors rely on high quality images, with PSNR assuming values. In other words, we gain more where the goal of having high performance is close to be reached.

It is possible to see the above Figure and our contributions not as a method for improving the quality of the image, given a sparsity, but as a method to speed up an acquisition given a target PSNR. For example, if $PSNR = 38$ is at least needed for the reconstructed image, using the original LOUPE it is necessary to use at least a sparsity $\alpha = 0.25$, but using LOUPE plus all our contributions $\alpha = 0.2$ is enough. Following this example and the results plotted in the Figure above, from a speed up factor ($\frac{1}{\alpha}$) of 4 is possible to reach a speed up factor of 5, that corresponds to an increase of 20%.

We notice how Prancing Pony and Flashback have some problems working together, in fact using the two methods at the same time we do not get the same improvements we expect by looking at the methods separately.

Now we compare the results of all the contributions but Prancing Pony. We start considering for every sparsity the best model obtained in Chapter 5.4, trained only with Flashback (and without Prancing Pony) and by applying to them the remaining contributions (TIFT and Back to the Future built with TFT and TIFT).

What we obtain is displayed in the following Figure:

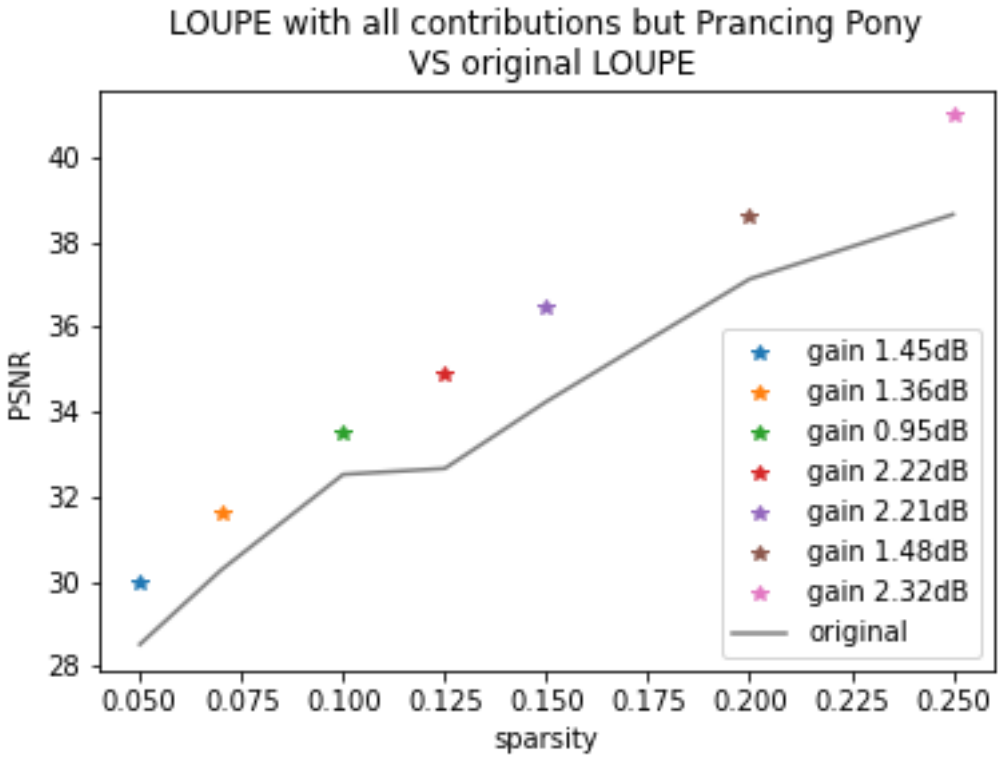


FIGURE 5-19 LOUPE TRAINED WITH ALL CONTRIBUTIONS BUT PRANCING PONY

The obtained results visibly outperform the results obtained by LOUPE trained with all contributions, confirming that Prancing Pony and Flashback have to be improved before they can work simultaneously without loss of performances.

To obtain high PSNR is more convenient to tune s without the use of Prancing Pony.

At the same time, we show the results obtained by adding the only contribution Back to the Future at the considered models from Chapter 5.4. We get almost identical results compared to the once shown in the Figure above, meaning that the addition of TFT and TIFT does not lead to a significant improvement.

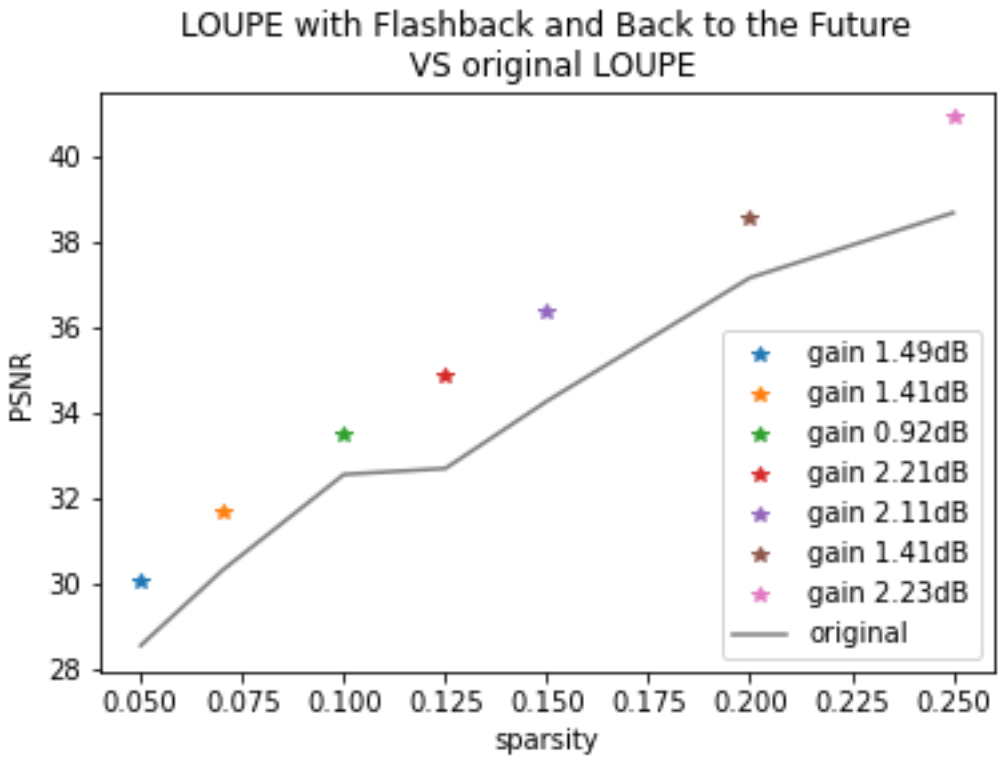
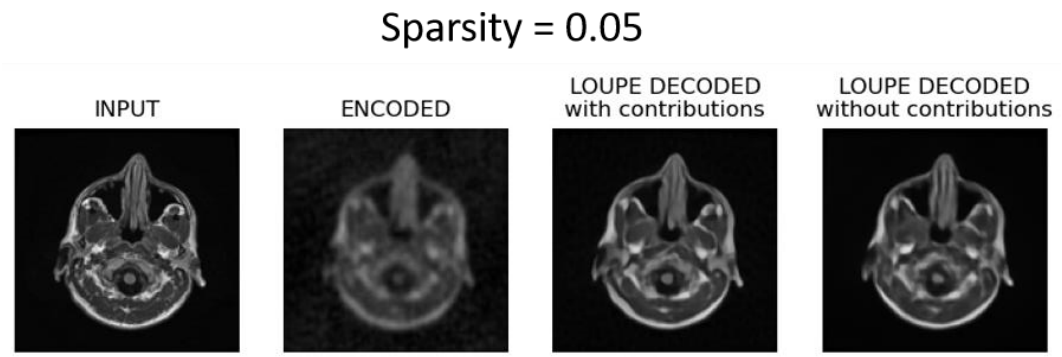


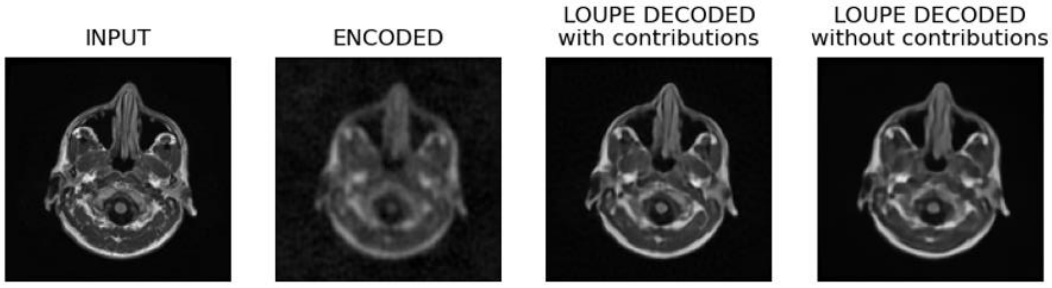
FIGURE 5-20 LOUPE TRAINED WITH FLASHBACK AND BACK TO THE FUTURE

Finally, we conclude saying the best performances are returned by a model whose slope s has been tuned without the use of Prancing Pony, and by the use of both Flashback and Back to the Future.

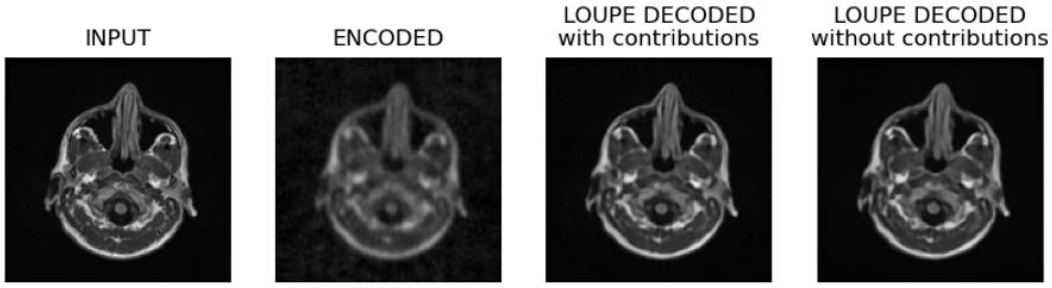
The Figure below contains in vertical order the models trained with $\alpha \in [0.05,0.07,0.1,0.125,0.15,0.2,0.25]$. For every row, the corresponds to a different model, are shown x: the ground truth (INPUT); $F^H y$: the under-sampled image in Euclidian domain before reconstruction (ENCODED); and $\hat{x}$: the image reconstructed by the decoder (DECODED).



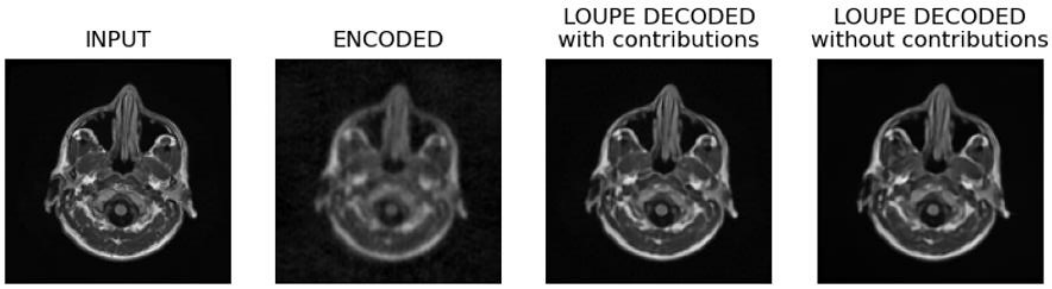
Sparsity = 0.07



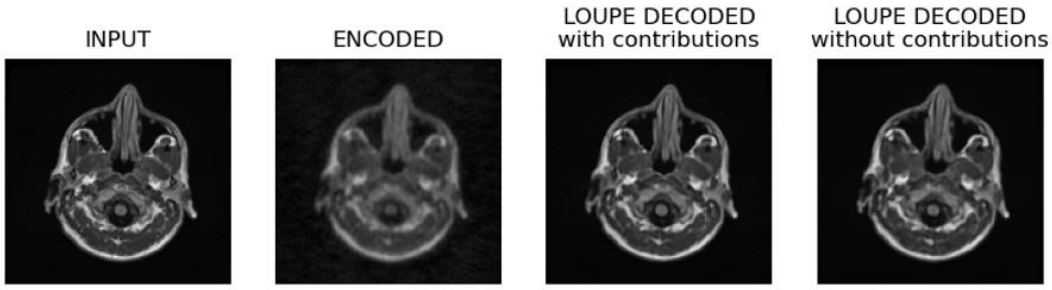
Sparsity = 0.1



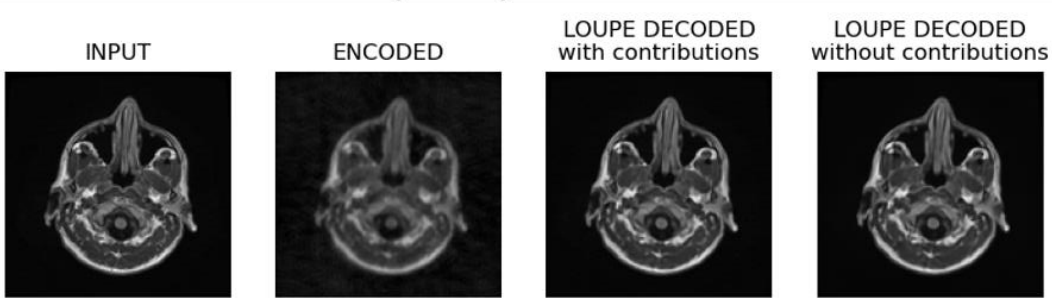
Sparsity = 0.125



Sparsity = 0.15



Sparsity = 0.2



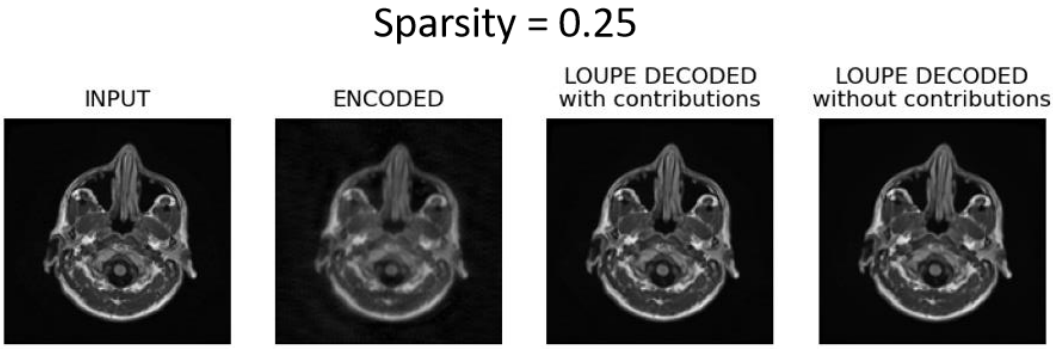


FIGURE 5-21 MRI SCANS ENCODED AND DECODED BY LOUPE WITH CONTRIBUTIONS

5.10 Future Research

This work focused on the implementation of new techniques for exploiting at best the possibilities offered by LOUPE. In this final chapter we highlight some of the aspects, we think, are worth of being further studied.

One of the most important tests is trying LOUPE with our contributions on a wider and different dataset (for example on [43]). The analysis of LOUPE has been limited by the use of a dataset containing brain MRI and on the reduced dimension of the images. We think is necessary to evaluate our model on a wider set of MRI scans with higher dimensionalities to fully understand the possibilities offered by it.

Another essential key point of this work is its adaptability, in fact, even if we focused on the reconstruction of MRI scans all the assumptions and ideas can prove valid if adapted for another general task. In particular, we aim to use LOUPE with all those types of sparse data, that are not necessarily acquired in the frequency domain. We want to explore different uses of our model by changing the Fourier-related layers and see how it performs when it works with different data.

Another important path to follow to further improve LOUPE is the use of new state of the art neural network models. [58], [59] are some of the new networks that promise advantages compared to the original U-NET, that can be easily implemented.

A modern approach, that found high relevancy in the field of Machine Learning since few years, is the Generative Adversarial Network (GAN) [60]. Following [39] as an example of GAN applied to MRI reconstruction problem, we think our model can be adapted to a GAN network structure.

As we presented the results obtained by TIFT we noticed they were not really relevant. A future research could consist in finding a more general adaptable trainable Fourier layer. Is possible to argue that in TIFT the small number of parameters leads to a learning bottleneck that do not have as many parameters as are needed to adapt and improve performances. In future new architectures having more adaptability can be tried to see if better results are achieved.

When training LOUPE with both Prancing Pony and Flashback we encountered some problems and we had to discard their simultaneous use to find the model that returned the best results. In future it will be interesting to explore more in depth how Prancing Pony and Flashback interact, searching for a more elegant and tuning free implementation of both.

We tuned Flashback by selecting what norm between L_1 and L_2 norm returned best results. At the end we selected L_2 norm, but more trials with different norms can be done and a more detailed tuning can be operated.

6 REFERENCES

[1] A. C. Clarke, “Hazards of Prophecy: The Failure of Imagination,” *Profiles Futur. An Enq. into Limits Possible*, pp. 14, 21, 36, 1973.

[2] H. R. Max Roser, Esteban Ortiz-Ospina, “Life Expectancy,” *Publ. online OurWorldInData.org*, 2013, [Online]. Available: https://ourworldindata.org/life-expectancy.

[3] “Scopus,” 2021. https://www.scopus.com/search/form.uri?display=basic&zone=header&origin=.

[4] OECD, “Number of magnetic resonance imaging (MRI) units and computed tomography (CT) scanners.,” *Health (Irvine. Calif).*, vol. 120, pp. 363–364, 2010, [Online]. Available: https://www.cdc.gov/nchs/data/hus/2010/120.pdf%0Ahttp://www.cdc.gov/nchs/data/hus/2010/120.pdf.

[5] A. France-Presse, “Man dies after being sucked into MRI scanner at Indian hospital,” *Agence France-Presse*. https://www.theguardian.com/world/2018/jan/30/mri-scanner-india-death.

[6] S. L. Talagala and I. J. Lowe, “Introduction to magnetic resonance imaging,” *Concepts Magn. Reson.*, vol. 3, no. 3, pp. 145–159, 1991, doi: 10.1002/cmr.1820030303.

[7] M. A. Griswold *et al.*, “Generalized Autocalibrating Partially Parallel Acquisitions (GRAPPA),” *Magn. Reson. Med.*, vol. 47, no. 6, pp. 1202–1210, 2002, doi: 10.1002/mrm.10171.

[8] D. L. Donoho, “Compressed Sensing,” *IEEE Trans. Inf. THEORY*, vol. 52, no. 1, pp. 137–147, 2006, doi: 10.1109/TAES.2017.2649698.

[9] I. F. Gorodnitsky and B. D. Rao, “Sparse signal reconstruction from limited data using FOCUSS: A re-weighted minimum norm algorithm,” *IEEE Trans. Signal Process.*, vol. 45, no. 3, pp. 600–616, 1997, doi: 10.1109/78.558475.

[10] E. J. Candes and T. Tao, “Near-optimal signal recovery from random projections: Universal encoding strategies?,” *IEEE Trans. Inf. Theory*, vol. 52, no. 12, pp. 5406–5425, 2006, doi: 10.1109/TIT.2006.885507.

[11] M. Mangia, F. Pareschi, R. Rovatti, and G. Setti, “Adapted Compressed Sensing: A Game Worth Playing,” *IEEE Circuits Syst. Mag.*, vol. 20, no. 1, pp. 40–60, 2020, doi: 10.1109/MCAS.2019.2961727.

[12] Wikipedia, “NP-hardness.” https://en.wikipedia.org/wiki/NP-hardness.

[13] E. J. Candès, “The restricted isometry property and its implications for compressed sensing,” *Comptes Rendus Math.*, vol. 346, no. 9–10, pp. 589–592, 2008.

[14] E. Candès and J. Romberg, “Robust signal recovery from incomplete observations,” *Proc. - Int. Conf. Image Process. ICIP*, pp. 1281–1284, 2006, doi: 10.1109/ICIP.2006.312579.

[15] V. D. Geethanath, S., Baek, H. M., Ganji, S. K., Ding, Y., Maher, E. A., Sims, R. D., Choi, C., Lewis, M. A., & Kodibagkar, “Compressive sensing could accelerate 1H MR metabolic imaging in the clinic,” *Radiology*, vol. 262, no. 3, pp. 985–994, 2012, [Online]. Available: https://doi.org/10.1148/radiol.11111098.

[16] R. Otazo, D. Kim, L. Axel, and D. K. Sodickson, “Combination of compressed sensing and parallel imaging for highly accelerated first-pass cardiac perfusion MRI,” *Magn. Reson. Med.*, vol. 64, no. 3, pp. 767–776, 2010, doi: 10.1002/mrm.22463.

[17] J. C. Ye, “Compressed sensing MRI: a review from signal processing perspective,” *BMC Biomed. Eng.*, vol. 1, no. 1, pp. 1–17, 2019, doi: 10.1186/s42490-019-0006-z.

[18] Justin P. Haldar [Member, “Low-Rank Modeling of Local k-Space Neighborhoods (LORAKS) for Constrained MRI,” *IEEE Trans Med Imaging*, vol. 33, no. 1, pp. 668–681, 2014, doi: 10.1109/TMI.2013.2293974.Low-Rank.

[19] J. C. Y. Dongwook Lee, Kyong Hwan Jin, Eung Yeop Kim, Sung-Hong Park, “Acceleration of MR parameter mapping using annihilating filter-based low rank hankel matrix (ALOHA),” *Int. Soc. Magn. Reson. Med.*, 2016.

[20] A. M. A. G. M. H. Amer, “Edge detection methods,” *2015 2nd World Symp. Web Appl. Netw.*, pp. 1–7, 2015, [Online]. Available: 10.1109/WSWAN.2015.7210349.

[21] A. Hladnik and P. Saksida, “Compressed Sensing and Some Image Processing Applications,” pp. 567–572, 2018, doi: 10.24867/grid-2018-p68.

[22] P. Wang, “On Defining Artificial Intelligence,” *J. Artif. Gen. Intell.*, vol. 10, no. 2, pp. 1–37, 2019, doi: 10.2478/jagi-2019-0002.

[23] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips, “GPU computing,” *Proc. IEEE*, vol. 96, no. 5, pp. 879–899, 2008, doi: 10.1109/JPROC.2008.917757.

[24] A. C. B. and L. F.-F. Olga Russakovsky*, Jia Deng*, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, “ImageNet Large Scale Visual Recognition Challenge.,” [Online]. Available: http://image-net.org/.

[25] P. Baldi, “Autoencoders, Unsupervised Learning, and Deep Architectures,” *ICML Unsupervised Transf. Learn.*, pp. 37–50, 2012, doi: 10.1561/2200000006.

[26] S. Handore and D. Kokare, “Performance analysis of various methods of tumour detection,” *2015 Int. Conf. Pervasive Comput. Adv. Commun. Technol. Appl. Soc. ICPC 2015*, vol. 00, no. c, pp. 4–7, 2015, doi: 10.1109/PERVASIVE.2015.7087002.

[27] and G. S. M. Mangia, F. Pareschi, V. Cambareri, R. Rovatti, “Adapted Compressed Sensing for Effective Hardware Implementations: A Design Flow for Signal-Level Optimization of Compressed Sensing Stages,” *Springer-Verlag*, 2018.

[28] M. Mangia, L. Prono, A. Marchioni, F. Pareschi, R. Rovatti, and G. Setti, “Deep Neural Oracles for Short-Window Optimized Compressed Sensing of Biosignals,” *IEEE Trans. Biomed. Circuits Syst.*, vol. 14, no. 3, pp. 545–557, 2020, doi: 10.1109/TBCAS.2020.2982824.

[29] “Brief History of Machine Learning,” *Machine Learns*. https://codetain.com/blog/a-brief-history-of-machine-learning/#:~:text= A brief history of machine learning ,trend raised after 1990— kernel methods. More.

[30] V. Dumoulin and F. Visin, “A guide to convolution arithmetic for deep learning,” pp. 1–28, 2016, [Online]. Available: http://arxiv.org/abs/1603.07285.

[31] D. Scherer, A. Müller, and S. Behnke, “Evaluation of pooling operations in convolutional

architectures for object recognition," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 6354 LNCS, no. PART 3, pp. 92–101, 2010, doi: 10.1007/978-3-642-15825-4_10.

[32] G. E. H. Alex Krizhevsky, Ilya Sutskever, "ImageNet Classification with Deep Convolutional Neural Networks Alex," *Adv. Neural Inf. Process. Syst.*, pp. 1–1432, 2012, doi: 10.1201/9781420010749.

[33] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *3rd Int. Conf. Learn. Represent. ICLR 2015 - Conf. Track Proc.*, pp. 1–14, 2015.

[34] C. Szegedy *et al.*, "Going deeper with convolutions," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 07-12-June, pp. 1–9, 2015, doi: 10.1109/CVPR.2015.7298594.

[35] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2016-Decem, pp. 770–778, 2016, doi: 10.1109/CVPR.2016.90.

[36] J. C. M. G. and J. A. B. G. Jose Manuel Ortiz-Rodriguez, Carlos Guerrero-Mendez, Maria del Rosario Martinez-Blanco, Salvador Castro-Tapia, Mireya Moreno-Lucio, Ramon Jaramillo-Martinez, Luis Octavio Solis-Sanchez, Margarita de la Luz Martinez-Fierro, Idalia Garza-Veloz, "Breast Cancer Detection by Means of Artificial Neural Networks," *Intech*, no. tourism, p. 13, 2018, [Online]. Available: http://dx.doi.org/10.5772/intechopen.71256.

[37] Y. J. Han Y, Yoo J, Kim HH, Shin HJ, Sung K, "Deep learning with domain adaptation for accelerated projection-reconstruction," *mr. Magn Reson Med*, vol. 80, no. 3, pp. 1189–1205, 2018.

[38] D. Wang, S., Su, Z., Ying, L., Peng, X., Zhu, S., Liang, F., Feng, D., Liang, "Accelerating magnetic resonance imaging via deep learning," *Biomed. Imaging (ISBI), 2016 IEEE 13th Int. Symp.*, pp. 514–517, 2016.

[39] T. M. Quan, T. Nguyen-Duc, and W. K. Jeong, "Compressed sensing MRI reconstruction using a generative adversarial network with a cyclic loss," *arXiv*, vol. 37, no. 6, pp. 1488–1497, 2017.

[40] C. D. Bahadir, A. V. Dalca, and M. R. Sabuncu, "Learning-based optimization of the under-sampling pattern in MRI," *arXiv*, vol. 6, pp. 1139–1152, 2019.

[41] C. C. Mart´ın Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen *et al.*, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems," *Prelim. white Pap.*, 2015, [Online]. Available: www.tensorflow.org.

[42] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 9351, pp. 234–241, 2015, doi: 10.1007/978-3-319-24574-4_28.

[43] J. Zbontar *et al.*, "fastMRI : An Open Dataset and Benchmarks for Accelerated MRI," pp. 1–35.

[44] R. and Y. Bresler, "MR Image Reconstruction From Highly Undersampled k-Space Data by Dictionary Learning," *IEEE Trans. Med. Imaging*, vol. 30, no. 5, pp. 1028–1041, 2011.

[45] E. M. Eksioglu, “Decoupled Algorithm for MRI Reconstruction Using Nonlocal Block Matching Model : BM3D-MRI,” *J. Math. Imaging Vis.*, vol. 56, no. 3, pp. 430–440, 2016, doi: 10.1007/s10851-016-0647-7.

[46] W. Guo, J. Qin, W. Yin, and L. Angeles, “A New Detail-Preserving Regularization Scheme A New Detail-Preserving Regularization Scheme ∗,” no. January 2016, 2014, doi: 10.1137/120904263.

[47] D. L. Donoho, J. M. Santos, and J. M. Pauly, “Compressed Sensing MRI [,” no. March 2008, pp. 72–82.

[48] U. Gamper, P. Boesiger, and S. Kozerke, “Compressed Sensing in Dynamic MRI,” vol. 373, pp. 365–373, 2008, doi: 10.1002/mrm.21477.

[49] P. J. S. Lustig M, Donoho D, “MRI: The application of compressed sensing for rapid MR imaging.,” *Magn Reson Med*, vol. 58, no. 6, pp. 1182–95, 2007.

[50] Z. W. and G. R. Arce, “Variable density compressed image sampling,” *IEEE Trans. Image Process*, vol. 19, no. 1, pp. 264–270, 2010.

[51] J. V. and R. R. Machireddy, “A robust adaptive sampling method for faster acquisition of MR images,” *Mag. Reson. Imag*, vol. 33, no. 5, pp. 635–643, 2015.

[52] & L. Haldar, J. P., Hernando, D., “Compressed-sensing MRI with random encoding,” *IEEE Trans. Med. Imaging*, vol. 30, no. 4, pp. 893–903, 2011.

[53] T. Weiss, S. Vedula, O. Senouf, A. Bronstein, O. Michailovich, and M. Zibulevsky, “Learning Fast Magnetic Resonance Imaging,” pp. 1–12.

[54] M. Buda, “Brain MRI segmentation,” 2019. https://www.kaggle.com/mateuszbuda/lgg-mri-segmentation.

[55] C. D. Bahadir, A. V. Dalca, and M. R. Sabuncu, “Learning-Based Optimization of the Under-Sampling Pattern in MRI,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 11492 LNCS, pp. 780–792, 2019, doi: 10.1007/978-3-030-20351-1_61.

[56] É. D. Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, “Scikit-learn: Machine Learning in Python,” vol. 12, no. 85, p. 2825−2830, 2011.

[57] R. B. M. Fisher, “Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography.”

[58] Z. Zhou and M. R. Siddiquee, “UNet++: A Nested U-Net Architecture for Medical Image Segmentation.”

[59] O. Oktay *et al.*, “Attention U-Net : Learning Where to Look for the Pancreas,” no. Midl, 2018.

[60] Y. Goodfellow, I.J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, “Generative adversarial nets,” *Adv. Neural Inf. Process. Syst.*, pp. 2672–2680.