POLITECNICO DI MILANO
SCHOOL OF INDUSTRIAL AND INFORMATION ENGINEERING
Master of Science in Telecommunication Engineering

Limited angle computed tomography reconstruction with deep learning enhancement

Supervisor: **Prof. Marcon Marco**
Co. Supervisor: **Ing. Paracchini Marco**

Master's degree thesis of:
Gerace Girolamo, Matr. 898903
Kasenov Erbol, Matr. 916027

Academic year 2019-2020

To my parents,
who have always supported my every choice

Abstract

The Computed tomography (CT) is used in a broad variety of applications nowadays. It is applied to inspect the content of an object without destruction, like searching of dangerous items in airport's public areas, or recognize cancer in the human body.

Computed tomography plays a special role in medicine. Besides the detection of cancer, it is used for the analysis of the organs inside the human body and in the identification of the disease's cause. The CT helps to determine the correct diagnosis and save many human lives.

One of the main problem of CT is the insufficient quality of reconstructed images. Typically, many systems use limitations during the scanning phase in order to speed up the process. Therefore, some errors and artifacts may reduce the quality of reconstructed images.

The aims of this research is to develop methods in order to reduce the noise in tomographic images, which are obtained through limited range angle, exploiting Deep learning technology. Considering tomographic acquisitions obtained in a limited range angle setup, it was possible to enhance the reconstruction by obtaining results very close to the ones that could be obtained without angles measurement restrictions. The effectiveness and endless customization of deep learning networks leads to broadening the horizons of use.

In particular, in the literature, reconstruction denoising is obtained independently considering each slice, besides in this work a multi slice approach is introduced and compared to the single slice one. We have simulated the CT usage analyzing the modern algorithms of reconstruction and apply them in our methods.

Sommario

La Tomografia Computerizzata (TC) oggigiorno viene utilizzata in una vasta varietà di applicazioni. Viene sfruttata per ispezionare il contenuto di un oggetto senza distruggerlo, come la ricerca di materiale pericoloso nelle zone comuni degli aeroporti, oppure nell'individuazione di tumori nel corpo umano.

La Tomografia Computerizzata gioca un ruolo speciale nella medicina. Oltre alla individuazione dei tumori, viene utilizzata per l'analisi degli organi all'interno nel corpo umano e nell'identificazione della causa del malessere. La TC ci aiuta nella corretta determinazione della diagnosi e nel salvataggio di vite umane.

Uno dei maggiori problemi della Tomografia Computerizzata è l'insufficiente qualità delle immagini ricostruite. Tipicamente, molti sistemi utilizzano delle limitazioni durante la fase di scansione in modo da velocizzare il processo. Di conseguenza, alcuni errori ed artefatti compaiono nella ricostruzione riducendo le caratteristiche ricostruite.

Lo scopo di questa ricerca è quello di sviluppare metodi in modo da ridurre il rumore nelle immagini tomografiche, le quali sono ottenute attraverso angoli limitati, sfruttando la tecnologia del Deep Learning. Considerando acquisizioni tomografiche ottenute con una postazione ad angoli limitati è stato possibile migliorare le ricostruzioni ottenendo risultati molto simili alle immagini che possono essere ottenute senza limitazioni negli angoli di misura. L'efficacia e l'infinita personalizzazione delle reti di Deep Learning porta ad un appliamento degli orizzonti di utilizzo.

In particolare, nella letteratura, il denoising della ricostruzione è ottenuto indipendentmente considerando ogni slice, in questo lavoro viene introdotta la trattazione multislice e confrontata con il caso singola slice. Abbiamo simulato l'uso della Tomografia Computerizzata per analizzare i moderni algortmi di ricostruzione ed applicarli ai nostri metodi.

Acknowledgement

Acknowledgement by Kasenov Erbol

I would like to start by acknowledging my professor Marcon Marcon for all guidance, knowledge and experience. During the academic year as a usual student, I become a good technician(at least i hope). Studied an amazing course like video signals provided by the professor at Politecnico di Milano university.

Allowing me to observe the amazing area which is faced everywhere in our modern life. I found the structure and knowledge of the course so interesting. Got a real case: how do the main principles of video signal processing work.

So, I decided to go deeper into that area. Thanks a lot, to Professor Marco Marcon, I got the quite fascinating topic of master's degree research work like industrial computed tomography with limited range angle reconstruction. While learning how does tomography reconstruction work, and the main principles of deep learning. In plain words, our goal was to improve the performance of tomography by applying deep learning.

By the way I am always surprised when look at the professors at the Politecnico di Milano. They are like superheroes who have reached incredible heights in different aspects of life. Although they became professionals, in their scientific field, they also succeeded in business or creativity. They are people without borders, who can colonize Mars and become world-famous rock musicians.

Another pair of honorable mentions goes to research assistant Marco Paracchini. He was our coordinator as an elder brother and led us to the correct path of research work. In difficulties, he showed us the possible solutions and explained our problem when i misunderstood the concepts. Thanks a lot for his quick and efficient support.

I think, there is an infinite list of endless words of my gratitude to everyone which I encountered during these 2.5 years being as a student of master's degree at Politecnico di Milano. I cannot pass my dearest person in my life, a person without whom I would be nothing and would not study here and would not write these words for my final research work. Without whom, my mindset would not have been formed. a person without whom I would not be. This is my mother.

List of Figures

Contents

Chapter 1

Introduction

The problem of inspecting the inaccessible inner content of variety of objects without damaging them is one of the most interesting task that mankind had, and still have to face. Although more than a hundred years have passed since the discovery of X-rays, this scientific breakthrough is still considered one of the most important events in medicine, which made it possible to transfer the process of diagnosing a variety of diseases to a fundamentally new level. An X-ray is a penetrating form of high-energy electromagnetic radiation. It has a wavelength ranging from 10 pm to 10 nm, corresponding to frequencies in the range of 30 PHz to 30 EHz and energies in the range of 124 eV to 124 keV. X-ray wavelengths are shorter than UV rays and typically longer than gamma rays. For example in the fig. (1.1) shown the electromagnetic spectrum.

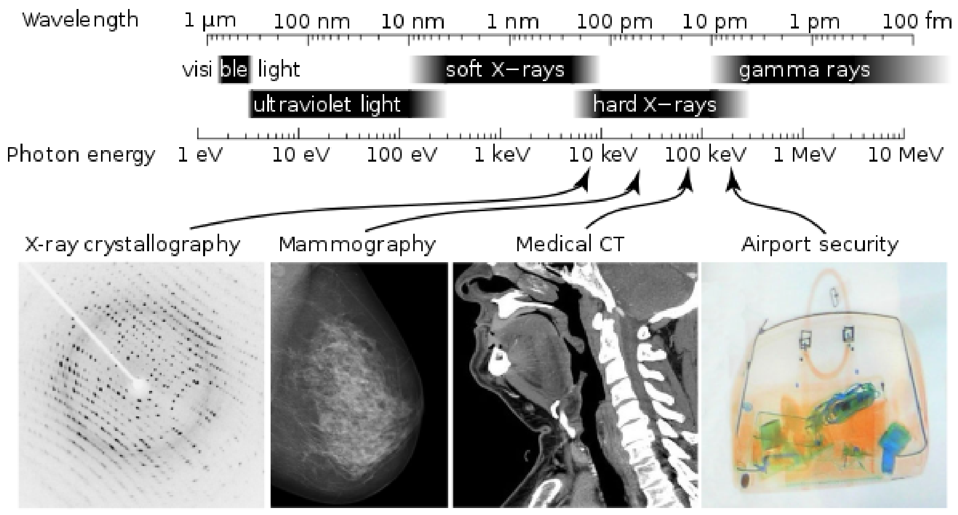


Figure 1.1: Electromagnetic spectrum: different applications use different parts of the X-ray spectrum.

The discovery of X-rays was a powerful impetus for the development of medicine and industry. Based on X-ray research, the science of radiography was born, which is engaged in diagnosing and curing human diseases. Modern applications of radiography include diagnostic and therapeutical medicine. In order to create an image with conventional radiography, a beam of X-rays is produced by an X-ray generator and is projected toward the object. A certain amount of X-rays is absorbed by the object based on its density and structural composition. The X-rays that pass through the object are captured behind the object by a detector. The generation of two-dimensional images by this technique is called projectional radiography. In the computed tomography scanning process, an X-ray source and its associated detectors rotate around the subject, which move through the conical trajectory. Any given point within the subject is crossed from many directions by many different beams at different times. Information regarding the attenuation of these beams is collated and subjected to computation to generate two-dimensional images in three planes (axial, coronal, and sagittal), which can be further processed to produce a three-dimensional image [62]. The discovery of X-rays is so significant and essential nowadays, these rays are used in many areas of life. They are actively used in the jewelry field to determine the authenticity of precious stones; in the art field, to analyze artwork to determine its quality and originality. X-rays play an important role in security issues because, with their help, it has become much easier to analyze the contents of a large amount of luggage for weapons or explosives at client zones and airports. Also, rays are used in different industries and science areas, so that the discovery of Wilhelm Roentgen can deservedly be considered one of the most significant achievements of all time in the field of physics.

1.1 Introduction of research work

The research work is based on improving the performance of computed tomography by developing methods in order to enhance the quality of reconstruction. Usually, an improvement of one of the performances leads to a decrease in other characteristics or an increase in the cost of computed tomography. It is crucial to observe the golden mean between improvement and the cost of the reconstruction process.

Unlike a conventional X-ray which uses a fixed X-ray tube, a CT scanner uses a motorized X-ray source that rotates around the circular opening of a toroid structure called the gantry. CT scanners use particular digital X-ray detectors, which are located directly opposite the X-ray source. As the X-rays leave the object, they are picked up by the detectors and transmitted to a computer. Each time the X-ray source completes one full rotation, the CT computer uses sophisticated mathematical techniques to construct a 2D image slice of the object/patient. The thickness of the tissue represented in each image slice can vary depending on the CT machine but usually ranges from 1-10 mm. When a whole slice is completed, the image is stored, and the motorized bed is moved forward incrementally into the gantry. The X-ray scanning process is then repeated to produce another image slice. This process continues until the desired number of slices is collected.

The technology evolution leads to a new architecture called Translational computed tomography (TCT) as a new low-end CT system, which can obtain the interior image without destroying the scanned object by using the projection data which is obtained from the detector. It utilizes translation to realize linear scanning, where the X-ray source and the flat panel detector are placed face to face with an object between them and are moved in opposite directions during the scanning process. When the acquisition of data from the TCT is complete, the filtered back projection (FBP)-type algorithm can accurately reconstruct some high-quality images. However, in some practical TCT applications, the obtained projection data of the scanned object are usually incomplete, in order to reduce the scan time, to decrease the X-ray radiation, or to avoid the inconsistency of the detector for the large angle scanning in the translational scanning scheme. In this circumstance, some artifacts are presented in the reconstruction by algebraic reconstruction algorithms, such as the simultaneous algebraic reconstruction technique (SART) and the algebraic reconstruction technique (ART), which have a better denoising effect than the FBP method when the projection data are complete. However, if the available projection data are incomplete, these methods cannot obtain a satisfactory reconstructed image. The ideal method of obtaining tomography data consists of a complete rotation of the beam source and detectors. For the motivation seen before, the use of *limited range angle rotation* as a trade-off for the measurement is deployed. The nature of the proposed method degrades the final image quality due to a limitation of the angle. So, there were developed different solutions for resolving these restrictions, like a more optimal reconstruction algorithm, making different assumptions over the methods considered before. In order to improve the image quality at the data processing phase from computed tomography and maintain the same kind of algorithms, deep learning technology was applied, which is a class of machine learning algorithms that uses multiple layers to extract high-level functions from raw input data progressively. In our assumptions was applied a U-Net architecture in deep learning phase, which is a convolutional neural network (CNN) designed to segment biomedical images. In work, there are two methods of denoising: based on single image improvement and multislice array of images which represents the 3d case. For the multislice method, there are used iterative computed tomography reconstruction applied to a bunch of consecutive images. Combining the approaches of limited angle computed tomography and deep learning improvement of reconstruction, we concluded some results of research work. This thesis is divided into:

Chapter 2 - State of the art

In this chapter introduced the brief history of CT, generations of tomographic scanner and reconstruction algorithms of CT from a mathematical point of view. Since a huge number of industries are using deep learning to leverage its benefits, also here explained the deep learning and neural network. Deep learning is used in various industries such as electronics, automated driving and medical research. For example, medical researchers are using it to detect cancer cells.

Basically, deep learning process consists of two key phases — training and inferring. The training phase can be considered a process of labeling vast amounts of data and identifying their matching characteristics. Here, the system compares those characteristics and memorizes them to correct conclusions when it encounters similar data next time. During the inferring phase, the model makes conclusions and labels unexposed data with the help of the knowledge it gained previously. There are three most popular ways to perform object classification: transfer learning, training from scratch and feature extraction.

Chapter 3 - Data generation by computed tomography simulation

This chapter describes the data simulation. Basically in work the data generation procedure is described and discussed highlighting how limited angle range acquisitions are simulated starting from complete reconstructions assumed as ground truth. Here is the goal to regenerate data as reconstruction in simulation. Apply the various modern algorithms of reconstructions and prepare data set for further data processing. The ASTRA toolbox was chosen as a tool for computed tomography simulation.

Chapter 4 - Limited range angle CT reconstruction

This chapter describes the method proposed to enhance and denoise acquisitions which obtained using a limited angle range setting. In particular three different limited angle ranges are considered [0-90], [0-120], [0-150] by proposing a CNN model for each one of them and reporting the training procedure. A multislice denoising approach is also introduced describing its difference with a single slice denoising method.

Chapter 5 - Results

This chapter represents the assessment of reconstruction data showing the result of two different methods based on the usage. The first one based on single image denoising by applying CNN and the second one- multiple slice array, considering in this way a 3d case.

Chapter 6 - Conclusions

In the final part of this work, there is a discussion of the results obtained in the overall process, highlight the performance and the improvement achieved from the reconstruction phase. There is also a discussion of the future directions of research work.

This work has been developed from remote deploy external datasets, from which derived the measurement, and used an Amazon AWS Instance with these characteristics: AMD EPYC 7000, 16 vCpus, and 64 Gb RAM.

Chapter 2

State of the Art

The word tomography derived from Ancient Greek composed by tomos, "slice" and grapho, "to describe", is used to represent the technic developed in the 30s by the Italian radiologist, Alessandro Vallebona [49]. Applying the projective geometry to a circular movement of an X-ray tube and a susceptible film around a patient, an image of a single body part could be obtained.

The evolution of the technologies and the development of the first calculators allowed Sir Godfrey Hounsfield, an English engineer, and Allan Cormack, a Sud African physicist, in 1967 to build the first Computed Axial Tomography. Consequently, four years later, in 1971, the first commercial CT scan was settled in the Atkinson Morley Hospital of Wimbledon. Initially, the system was designed only to analyze the human skull with a precision of one degree and a maximum rotation, around the target, of 180°. They obtaining 160 images in a single scan for every angle position in a range of time-variable between 5 or 10 minutes. Once the all scanning operation was completed they obtained 28800 pictures. These snapshots were then elaborated by a calculator using algebraic reconstruction algorithms, and after two hours and thirty minutes later, provides the brain's final image [4]. After all these operations were obtained the first CT scan on a patient with a suspected brain tumor, the surgeon, after doing the operation, declared: "It looks exactly like the picture." [16].

The researches developed by Godfrey Hounsfield at the Central Research Laboratories of Electric and Musical Industries were awarded the Nobel Prize in Physiology or Medicine in 1979 [35, 39].
After the initial excitement and the firsts discoveries made by the CT scans, an American radiologist, Ralph Alfidi, had the intuition to apply the concept of tomography not only on the human skull but expand it in other body parts. With the approval of the pioneer Sir Godfrey Hounsfield, he started the researches that led to the first abdomen CT scan [55].

The consequent interest of various corporates, like Pfizer and Siemens [21], and the continuous calculator's development brought an exponential growth of CT scans in terms of revelations more precise and detailed, decreasing the time of operations

and technique less intrusive for the human being. With all these upgrades was possible to start scanning in various body zones to analyze multiple concerns like cardiovascular issues, fractures and bone injuries, potential internal damages, tumor detection, and pneumatological analysis.
The modern scanners are based on Hounsfield's fundamentals, but they use different technology. The union of the CT and the contemporary innovation in the fields of rendering and post-production allows us to reach an incredible level of details and quality of the final image, moving on to 3D reconstructions [43]. The future of CT appears to have a strong connection with the field of Artificial Intelligence. Recent studies have shown how well-designed equipment allows finding neurological problems in less than 1.2 seconds, utilizes a neural network trained with 37000 different scans [47].

Moreover, there are studies of 2020 based on classification and diagnosis of pneumatological problems due to the presence of Sars-CoV-2 [41, 22]. This improvement in the recognition process guarantees the quickest response in such a way to anticipate the urgent's cases and having better prevention.

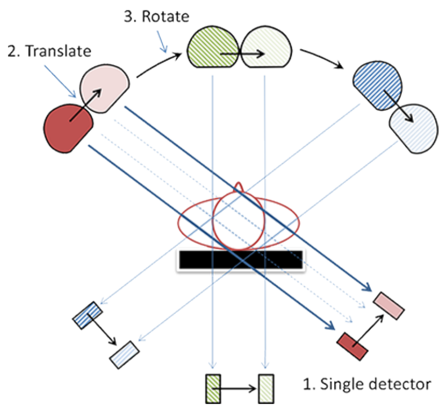
In the last few years the CT is used in different areas like astronomy [15], geophysical imaging [17], industrial imaging [9] and also in gaming area based on discrete tomography [12, 56]. Our interest is to see the CT industrial scan application where is present an exponential growth because it can verify the quality of a product and seek incongruency points between the project and the final product. In the next chapters, the industrial application and the analysis process used in the recognition processes will be described.

2.1 Tomographic scanner's generations

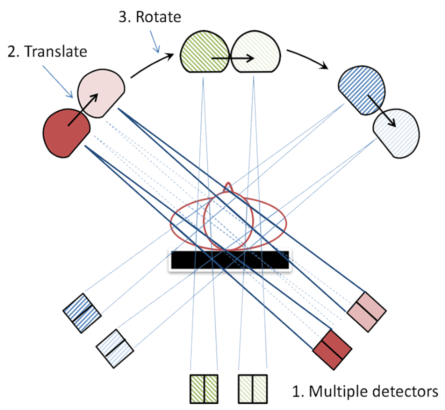
First generation. The first generation of CT scan was composed of a single X-ray tube with a pencil beam and a single small detector, one in front of each other. In this way, it possible to acquire only a single slice of the part. The acquisition was achieved with a translation of the two elements and then a rotation movement for the acquisition at different angles. In this generation, there was the physical movement of the source and the detector, the main problem of this generation is the acquisition time for every slice.

Second generation. The source produces a narrow fan beam with an angle of 10°, the detector is composed of a linear array with 30 elements. This leads to a better acquisition time but, due to the small angle's aperture of the beam, that forces to have the same movement of the first generation, there is anyway a long acquisition's time. Moreover, the narrow beam introduces the scatter's problem that decreases the image's quality.

Third generation. This generation was created with the goal to keep an acquisition's time under 20 seconds in order to reduce the artifacts created by the

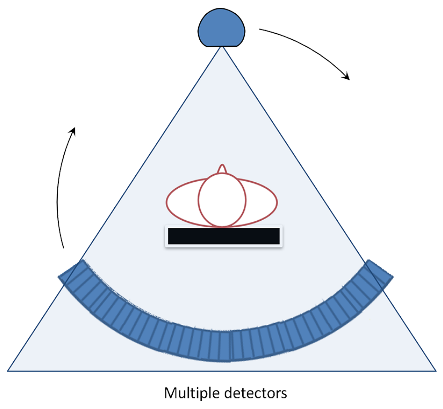


(a) First generation

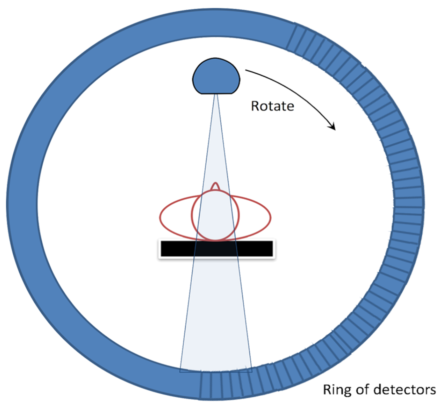


(b) Second generation

Fig. 2.1: Roto-translation scanner



(a) Third generation



(b) Fourth generation

Fig. 2.2: Rotation scanner

breath or the movement of the patient. In this generation, there is a wide fan beam angle, with a value between 40° and 60°. For this reason, there were deployed wider detectors and the source rotates jointly with the detector removing the translation movement, in this way it can produce a faster acquisition. The main drawback of this generation is the creation of *ring artifacts* due to the large number of detectors and the lack of calibration that was often present between the detectors.

Fourth generation. Developed to mitigate the ring artifacts, they are composed of a detector that rotates around the patient and the detector is in a stationary position behind the detector in order to maintain the calibration of the detectors. There are a great number of detectors 5000 and the source can be placed inside or outside the detector's circle, in the second case, the detector's circle must be tilted in order to have a good measurement from only the detector in front of it.

The newest generation of scanner use different technologies like helical movement and multiple detector array. Providing a great improvement in terms of acquired information, in acquisition's time, and in terms of image's quality. In the previous generations, there were different types of images because they are not on a planar section and the final output needs to be computed by a much more sophisticated reconstruction process [13].

2.2 Introducing Formula

A CT scan is composed of an X-ray source and an X-ray detector, these two elements rotate around the object making a series of measurements from different angles. The source creates a photon's beam that impacts the object; the detector placed behind the target detects the number of photons that pass throughout. In this way, are obtained information about the item by exploiting the various measurements' overlapping and achieving the object in 2D.

When all the data are acquired, they are processed using one of the many tomographic reconstruction algorithms available; providing a series of cross-sectional images and then create a 3D reconstruction. Every pixel of the output images shows the radio-density of a specific point, exploiting the attenuation law.

For a better understanding, considering the intensity instead of the number of photons; from quantum mechanics, can be derived the relationship between photons and intensity since the Electromagnetic wave's intensity is equal to the photon's flux flows through a surface S in a time, Δt.

$$E = hf \quad \rightarrow \quad I = \frac{E}{S\Delta t} \tag{2.1}$$

where:

- E: energy of photon;
- I: EM wave's intensity;
- h: Plack's constant;
- f: EM wave's frequency;
- S: surface of interest

The intensity of an X-ray beam over the time, I(t), can be described as follows:

$$I(t) = I_0\, e^{-\mu t} \tag{2.2}$$

Where:

- I_0: Initial intensity;
- μ: Attenuation coefficient which depends from the density ρ and the energy E;

- t: Time;

Note that the intensity has a negative exponential decrease and depends on the material nature. In addition, can be observed a higher attenuation if there is a material with a high atomic number or, if the X-ray beam has a low energy E or if it travels in material with a greater thickness.

For this reason, the pixels in the final output images describe different brightness values, dark pixels represent minimum attenuation; otherwise, brighter pixels describe greater attenuation values. The projection of an object is made up of a set of line integrals; these represent the total attenuation of the beam that moves throughout the object. The collection of different measurements taken at different angles θ is called *sinogram* and collects all the attenuation coefficients.
For the reconstruction must be considered the *Projection slice theorem* defined in 2.2.1.

Definition 2.2.1 *(Projection slice theorem) The Projection-slice theorem, central slice theorem, or Fourier slice theorem in two dimensions states that the results of the following two calculations are equal:*

- *Take a two-dimensional function f(r), project (e.g., using the Radon transform) it onto a (one-dimensional) line, and do a Fourier transform of that projection.*
- *Take that same function, but do a two-dimensional Fourier transform first, and then slice it through its origin, which is parallel to the projection line.*

In operator terms, if

- *F1 and F2 are the 1- and 2-dimensional Fourier transform operators mentioned above,*
- *P1 is the projection operator (which projects a 2-D function onto a 1-D line),*
- *S1 is a slice operator (which extracts a 1-D central slice from a function),*

then: $F_1 P_1 = S_1 F_2$

Considering (2.2), the variable $\mu(x, y)$ can be seen as a distribution of the attenuation coefficient and making our assumption on a series of parallel rays at position r with angle θ, so:

$$I(t) = I_0 \, exp\left(- \int \mu(x, y) \, dS \right) \tag{2.3}$$

The total attenuation p of a ray in position r with an angle of projection θ is

$$p_\theta(r) = \ln\left(\frac{I}{I_0} \right) = - \int \mu(x, y) \, dS \tag{2.4}$$

Considering the coordinate system in 2.3 the value of r can be derived in every point *(x,y)* given the projection angle θ exploiting the trigonometric function:

$$r = x \cos\theta + y \sin\theta \tag{2.5}$$

Rearranging the equation (2.4) as follow:

$$p_\theta(r) = \iint_{-\infty}^{\infty} f(x,y)\delta(x\cos\theta + y\sin\theta - r)dxdy \tag{2.6}$$

The (2.6) is called *Radon Transform* or *sinogram* of the 2D object, where:

- *f(x,y)*: represent the object $\mu(x,y)$, specifically the distribution of the item in the 2D space;
- $\delta()$: Dirac's delta;

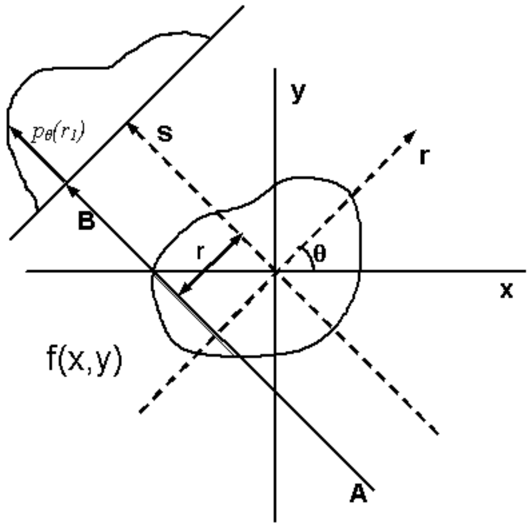


Fig. 2.3: Parallel beam geometry utilized in tomography and tomographic reconstruction

Starting from the sinogram, the target image's reconstruction can de done applying 2.2.1 that correlate the Radon transform (2.6) with the Fourier transform obtaining:

$$P_\theta = \iint_{-\infty}^{\infty} f(x,y)exp[-j\omega(x\cos\theta + y\sin\theta)]dxdy = F(\Omega_1, \Omega_2) \tag{2.7}$$

Where: $\Omega_1 = (\omega\cos\theta)$; $\Omega_2 = (\omega\sin\theta)$.

The result P_θ in (2.7) is a slice of the 2D Fourier transform of *f(x,y)* at angle θ. Exploiting the *Inverse Fourier Transform* the *Inverse Radon Trasform* can easily derived:

$$f(x,y) = \frac{1}{2\pi}\int_0^{\pi} g_\theta(x\cos\theta + y\sin\theta)d\theta \tag{2.8}$$

Where: $g_\theta(x\cos\theta + y\sin\theta)$ is the derivative of the *Hilbert transform* of p_θ(r)

Definition 2.2.2 *(Hilbert Transform) The Hilbert transform is a linear operator which allows the application of function space on another function space, defined on a general signal x(t) as:*

$$\hat{x}(t) \equiv \mathcal{H}\{x(t)\} = x(t) * h_H(t) = \int_{-\infty}^{\infty} x(\tau)h_H(t-\tau)d\tau = \frac{1}{\pi}\int_{-\infty}^{\infty}\frac{x(\tau)}{(t-\tau)}d\tau \tag{2.9}$$

Theoretically, the *Inverse Radon transform* can produce the original image. The Projection-slice theorem declares that if there are an infinite number of projections taken at an infinite number of angles, the original object's reconstruction can be perfectly done.

However, practically, cannot be done infinite measurements, there is a physical limit. Assuming *f(x,y)* had a diameter d and considering a desired resolution R_s, the number of projection N is limited by the general indication:

$$N > \frac{\pi d}{R_s} \tag{2.10}$$

2.3 Reconstruction Algorithms

As a result of the lower bound present in (2.10), there are some practical algorithms based on Radon Transform, statistical knowledge of data acquisition, and geometry of the system, useful for the reconstruction of a 3D object starting on its projections.

2.3.1 Fourier-Domain Reconstruction Algorithm

Starting from (2.2.1) there is a simple way to obtain the reconstruction. Considering the projection of the object at different angles $\theta_1,\dots,\theta_n$. Performing the *Fourier Transform* for each angle are obtained the *2D Fourier Transform* of the object function, F(u,v), on n radial lines over the polar grid. Ideally, there are infinite projections, and every information in frequency could be known using the *Inverse Fourier Transform*, obtaining the original object function f(x,y). Practically, as has been said, there are a finite number of projections, so the F(u,v) is composed of a limited number of samples and, consequently, a restricted number of radial lines in the polar grid. The lacking information needs to be filled using the interpolation, from the radial points to the points on a cartesian grid. After use of *Inverse Fourier Transform* for obtaining the reconstruction. This limit causes artifacts in the reconstructed object and is subject to aliasing because the function object, f(x,y), is not band-limited. Reconstruction performance can improve by modifying the scattering of the information in the polar grid, improving the interpolation's effectiveness; however, this kind of algorithm produces noisy outputs by nature.

2.3.2 Back Projection Algorithm

The interpolation reconstructions were derived from the *Project Slice Theorem*, (2.2.1). The *Back Projection Algorithm* can be similarly derived by the *discretized form* of the Radon's inversion formula, (2.6), these algorithms are widely used, the main feature is the possibility to work completely in the spatial domain. Assuming different angles of projections $\theta_1,\dots,\theta_N$ the (2.6) can be sampled as follow:

$$f(x,y) = \frac{1}{2\pi}\sum_{i=0}^{N-1} \Delta\theta_i g_{\theta_i}(x\cos\theta_i + y\sin\theta_i) \tag{2.11}$$

where

$$g_{\theta_i}(t) = p_{\theta_i}(t) * k(t) \tag{2.12}$$

With:

- $\Delta\theta_i$, is the angular spacing between the projections;
- k(t), is the radon kernel with $\mathcal{F}\{\text{k(t)}\} = \mid \omega \mid$;

Passing all the projection through a l-D filter with an impulsive response *k(t)* and frequency reponse $\mid \omega \mid$, the output's filter is (2.12). The signal in (2.11) is interpreted as a 2D signal that is filtered in $\hat{x}$ variable and uniform in $\hat{y}$ variable starting from a 1D function to obtain a 2D one, this operation is called *back-projection.*

The function $g_{\theta_i}(x \cos\theta_i + y \sin\theta_i)$ is obtained from $g_{\theta_i}(t)$ by back-projecting that function in the $\hat{y}$ direction. Since the orientation of the ($\hat{x}$, $\hat{y}$) coordinate system is different for each projection angle, each of the back-projected, filtered projections will have a different orientation [7]. Implementing this algorithm creates two main problems:

1. The filter *k(t)* does not have DC gain, so the mean gray level in the reconstruction is zero. Adding a DC bias can resolve this issue.
2. The choice of the projection filter *k(t)* is crucial since the filter gain increases with the frequency, the high-frequency noise is amplified. There needs to select a filter with a linear response and, out of a cut-off frequency, having a response equal to zero.

The FBP method is widely used for computational simplicity, the data from each view is convoluted and backprojected independently from the other views and this characteristic permits the FBP to be the standard reconstruction method for the CT scan.

2.3.3 Iterative Reconstruction Algorithm

In the first implementations of CT in the late 1960s, an Iterative Reconstruction (IR) algorithm was adopted. Since, using IR, the limited computing power of the time required up to 45 minutes to compute the reconstruction of a single slice, the FBP method, see (2.3.2), was introduced and adopted instead of reducing the single slice reconstruction time to just 30 seconds. With the exponential technology evolution in the lasts few decades, the IR method came back in the spotlight. This comeback is due to two fundamental problem that afflicts the FBP reconstruction:

1. The high dose of X-ray, required for the process that creates side effects for lengthy exposition.
2. The high noise present in the final image. This depends on the general scenario, for example, data acquisition with reduced tube output or CT imaging of obese patients is often compromised by high image noise; high-density structures,

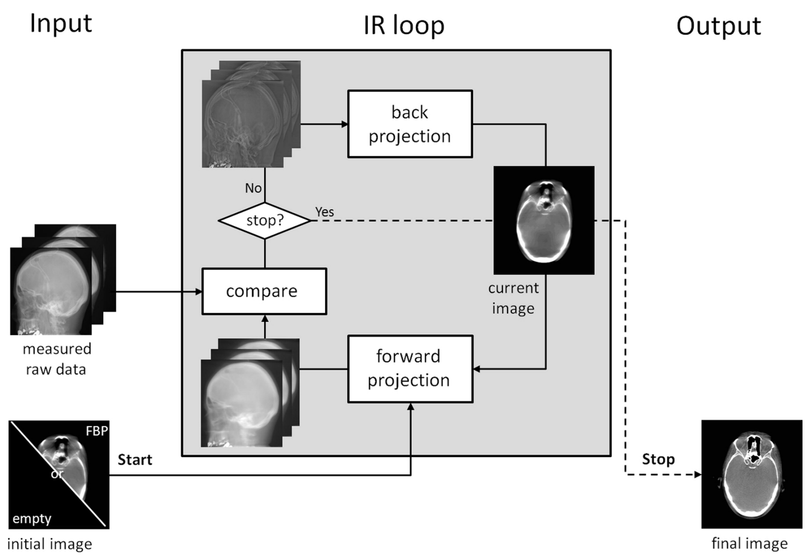


Fig. 2.4: IR algorithm steps, in input there is the comparison between artificial data and the measured data, in output the final image

such as calcifications or stents, result in blooming artifacts; metallic implants or bone structures might lead to severe streak artifacts

FBP can produce a high spatial resolution or a high contrast resolution. The IR technique can achieve the two requirements with a lower dose of X-ray, and for this reason, this method was adopted again [14].

All the IR algorithms are based on three fundamental steps which are iteratively repeated. In the first step, there is the initialization process given by a priori information provided, for example, by a standard FBP reconstruction or an empty image, in this way the *artificial raw data* are obtained. In the second step, the artificial raw data, are compared with the real measured data; their difference is useful for understanding the error level. In the last step, the decision occurs. If the error level or, iteration's number is satisfied, the final output is provided. Otherwise, if the requirements are not satisfied, after the comparison, a back projection and then another forward projection is performed, in this way is obtained the updated sinogram, and the loop restart. [5] In figure, 2.4 there is the graphical representation of the algorithm's loop.

The next pages illustrate the various iterative reconstruction methods.

Algebraic reconstruction

The *Algebraic Reconstruction Technique (ART)*, is the simplest iterative method, based on *Kaczmarz's method* [44], where the cross-section can be calculated by resolving the system of linear equation $A\vec{x} = \vec{b}$ in a iterative way where:

- A is the system m x n matrix used to generate the raw data, it contains the weight of every pixel in the projection. The weight indicates the pixel's value in a particular ray;

- x represents a column vector which contains the values of all the pixels in the output image
- b is a column vector where the results of all projections for every ray per angle are stored. The projection at a given angle is the sum of non-overlapping, equally wide rays covering the figure

Given the matrix A and the vector b $\in \mathbb{R}$ or $\mathbb{C}$, the ART and consequentelly the Kaczmarz's method produce an approximation for the solution of the linear equation's system for k = 0,1,... as follows:

$$x^{k+1} = x^k + \lambda^k \frac{b_i - \langle a_i, x^k \rangle}{\|a_i\|^2} \bar{a}_i \tag{2.13}$$

The equation (2.13) is characterized by:

- i = k mod m+1;
- a$_i$ is the i-th row of matrix A;
- b$_i$ is the i-th component of vector b;
- λ^k is a relaxation parameter with range $0 < \lambda^k \leq 1$, used to slow the system's convergence. This slow down the computation time but improves the Signal-to-Noise Ratio. This value can be reduced each iterative step.
- $\bar{a}_i$ denotes complex conjugation of a_i

Every single entry of matrix A corresponds to a single ray, starting from the source, passing through the object volume, and arriving at the detector,which is the same idea of the integral's line of the attenuation coefficient.

The ART method is simple: Every density is spread over the reconstructed space, each of them is modified after each iteration, changing the grayness of the pixel on the intersections of the rays in order to make the ray sum correspond to the measured projection [45].

The quality of the ART algorithm is limited due to the correspondence between one pixel and one ray. It creates a high noise level due to the multiple ray's crossing on one pixel that produces a discrepancy every time the pixel grid's upgrade occurs after one ray.

To solve this problem the update of the pixel grid is done after going through all the equations, after all the iteration the pixel value's calculating are changed with the average of every value computed for that specific pixel.

This solution is called *Simultaneous Iterative Reconstruction Technique (SIRT)*, developed for the medical's field it not have a large application due to the long time taken for create the reconstructed image.

For this reason, the *Simultaneous Algebraic Reconstruction Technique (SART)*, was developed in 1984 by Anders Andersen and Avinash Kak [3], it generates a good reconstruction in one iteration with better quality withrespect to the ART and SIRT methods, due to this improvement, a lot of different configuration was developed with implementation on parallel processing structures.

The pixel update in the SART method is performed after the computation of the whole projection at a specific angle θ. In this way, the projection's error is compensated over all the rays in a certain projection. In such a way, there is a huge improvement because only one iteration is needed for obtaining a good reconstruction in comparison to the ART, which needs more iteration, and of the SIRT that it is much slower [46].

Implementation of SIRT The implementation, as discussed before, is derived from the ART, the difference is in the pixel's update made after a whole iteration in the SIRT, referring to (2.13) ca be derived the equation for the SIRT:

$$x_j^{k+1} = x_j^k + \lambda \frac{\sum_{b_i \in B} \frac{b_i - \sum_{n=1}^{N} a_{in} \cdot x_n^k}{\sum_{n=1}^{N} a_{in}} \cdot a_{ij}}{\sum_{b_i \in B} a_{ij}} \tag{2.14}$$

In the (2.14) can be seen clearly the difference between SIRT and ART, it require the computing of nested loops in the numerator and in the denominator over all the B space before updating the pixel, this is not acceptable because the computational time is very large

Implementation of SART As in the ART,the scope is to guess the pixel's initial value and modify it until the correct one is reached. The difference respect the ART and the SIRT is the calculation's space. In the SART, the pixel's update is doing taken all the equation over one angle, the order of the angle has great importance choose random angles, can create an unstable reconstruction.

$$x_j^{k+1} = x_j^k + \lambda \frac{\sum_{b_i \in B_\varphi} \frac{b_i - \sum_{n=1}^{N} a_{in} \cdot x_n^k}{\sum_{n=1}^{N} a_{in}} \cdot a_{ij}}{\sum_{b_i \in B_\varphi} a_{ij}} \tag{2.15}$$

2.3.4 Fan-Beam Reconstruction

To this point, the projections are considered as a set of line's integral making the calculations using integrals over straight lines, this means there is a collimated beam from the source. In a practice view, obtain collimated beams means increasing the time exposure for acquiring a long collection of data, this procedure creates a *motion artifacts.*

The solution used in fig. (2.5) is a *non collimated Fan Beam*, it is an useful approximation. In this way, the projection can still be modeled over straight lines but the parallelism is lost.

Using a fan-beam instead of a parallel beam, creates some reconstruction issues. The fan-beam needs a range of 360° angles for a complete reconstruction and the project slice theorem (2.2.1) can not be directly applied. Some arrangements in the (2.11) are needed, changing the variables with the *Jacobian operator* introducing differential coefficients, to modify the reference system.

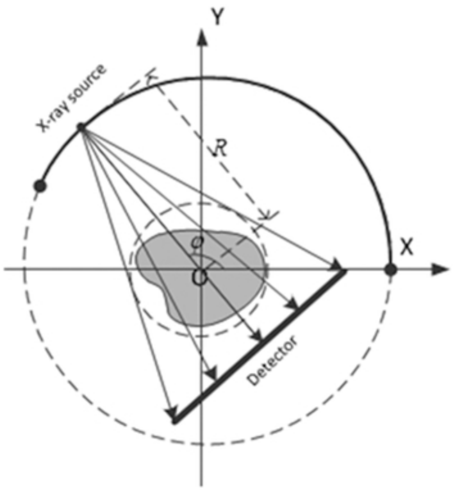


Fig. 2.5: General setup of a CT scanner

Considering the setting in fig. (2.6) there is a density function provided by Randon's integral [38]:

$$f(r,\phi) = \frac{1}{4\pi^2}\int_0^{2\pi}\int_{-\infty}^{\infty}(-\frac{1}{t})\frac{\partial}{\partial l}p(l,\theta)dld\theta \tag{2.16}$$

where, $f(r,\phi)$ is the density with polar coordinates (r,ϕ), t is the perpendicoular distance between the point and the reference ray, $p(l,\theta)$ is the density integral w.r.t vertical axis l with angle θ Applying the *Jacobian operator* to the (2.16) and making some arrangement:

$$f(r,\phi) = \frac{1}{4\pi^2}\int g(\varepsilon',\eta)\frac{\partial\theta}{\partial\eta}d\eta \tag{2.17}$$

$$g(\varepsilon',\eta) = -\int \frac{1}{l(\varepsilon)-l(\varepsilon')}\frac{\partial}{\partial\varepsilon}p(\varepsilon,\eta)d\varepsilon \tag{2.18}$$

The (2.18) is the integral form of the reconstruction considering the distance difference, $l(\varepsilon)-l(\varepsilon')$, between the reference ray from the origin in the two reference systems.

In these assumptions is considered a continuous model, in the real case can not be produced an infinite number of X-rays, also the detector's number is finite. A discrete model is useful for the computation. Considering from (2.17)-(2.18) the approximation:

$$f(r,\phi) \backsimeq \frac{1}{4\pi^2}\sum_j g_j(\varepsilon')\delta\theta_j \tag{2.19}$$

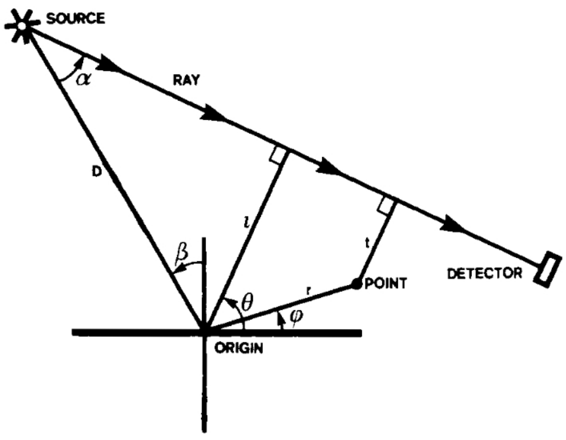


Fig. 2.6: Geometric approach of an incident ray

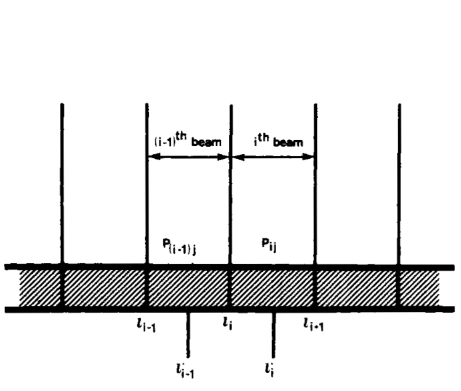
$$g_{i'j} = -\sum_{j} \frac{(p_{ij} - p_{(i-1)j})}{(l_i - l'_{i'})} \tag{2.20}$$

In the (2.19),(2.20), are introduced:

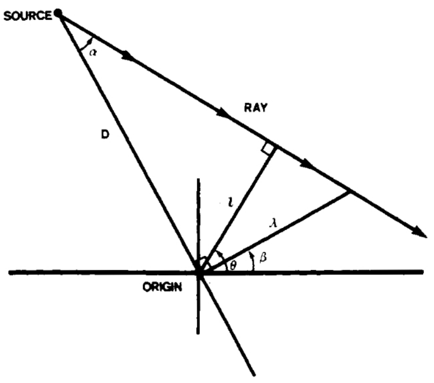
- $\delta\theta_j = \dfrac{(\theta_{j+1} - \theta_{j-1})}{2}$, is angular interval associated with the jth projection;
- $l'_{i'} = \dfrac{(l_i + l_{i+1})}{2}$, is the centre of the i-th beam;
- p_{ij}, is the density integral obtained form the i-th detector in the j-th projection.

In fig.(2.7(a)) are shown the discreate representation of the detector and the incident beams.

For the generation of the density function $f(r,\phi)$, must be found the value of $g_j(\varepsilon')$ making an interpolation operation over the discrete set $\{g_{i',j}\}$.
The discrete set $\{g_{i',j}\}$ is achieved from the sequence $\{p_{i,j}\}$ by a linear operator, this can be assumed as a *convolution problem*, considering a space-variant filter with coefficients that depends on the width of the detector. In this way we can relate the FBP obtaining the filter function F_k.



(a) Discrete fan beam



(b) Treatise of a single ray

Fig. 2.7: Details of the detector and incident beams

Using the Jacobian operator implies a change also in the distances and angles variables used to characterize the position between the point of interest and the reference ray.

The calculations for the change of variables are not trivial. By making some arrangements is possible to create a situation where the desired density function depends only on geometric variables with relate the distances and the angles between the origin and the reference ray, see fig.(2.7(a)). The discrete approximation is:

$$f(r,\phi) \backsimeq \frac{1}{4\pi^2}\sum_j g_j(\lambda')\frac{D^2}{D + r\sin(\beta_j - \phi)} \tag{2.21}$$

$$g_{i'j} = \sum_j F_{i-i'}\frac{D}{\sqrt{D^2+\lambda^2}}P_{ij}\delta\lambda \tag{2.22}$$

With:

- $\lambda' = D\tan\alpha'$, is the sampled value of the distance between the reference ray and the origin;
- p_{ij}, is the projection of the i-th ray in the j-th fan;
- $\delta\lambda$, is the fixed interval between the intersection of the ray with the line λ
- $\delta\beta_j = \dfrac{\beta_{j+1}-\beta_{j-1}}{2}$, is the angolar interval associated with a specific fan.
- $F_{i-i'}$, is the sampled version of the filter that contains geometrical coefficients which depends only from the difference between two ray.

The last step is the interpolation to obtain $g_j(\lambda')$ from the set $g_{i'j}$. Sampling N rays uniformly along a segment of length L, the intersection's interval is:

$$\delta\lambda = \frac{L}{N-1} \tag{2.23}$$

The ith ray corresponds to:

$$\lambda_i = -\frac{L}{2} + i\delta\lambda \tag{2.24}$$

Consequently, $g_j(\lambda')$ is obtained by interpolation from $g_{i'j}$ and $g_{(i'+1)j}$ where:

$$i' = \left\lfloor \frac{(\lambda' + \frac{L}{2})}{\delta\lambda} \right\rfloor \tag{2.25}$$

In practice, a fixed array of detectors is deployed, placed behind the object in the same circumference of the source. All the previous assumptions are now made concerning the angle because the center's angle is two times the angle at the source so there is an equal angular space between source and detectors [33]. The previous set of equation, (2.21) and (2.22) are modified:

$$f(r,\phi) \backsimeq \frac{1}{4\pi^2}\sum_j g_j(\alpha')\frac{1}{r^2 + D^2 + 2rD\sin(\beta_j - \phi)}\delta\beta_j \tag{2.26}$$

$$g_{i'j} = \sum_j F_{i-i'} D\cos\alpha_i p_{ij}\delta\alpha \tag{2.27}$$

Recalling the previous steps, the interpolation's computation is required based now on the angle α angle, considering a sampling of N rays over an arc of angle A:

$$\delta\alpha = \frac{A}{N-1} \tag{2.28}$$

So, the i-th ray corresponds to:

$$\alpha_i = -\frac{A}{2} + i\delta\alpha \tag{2.29}$$

Consequently, $g_j(\alpha')$ is obtained by interpolation from $g_{i'j}$ and $g_{(i'+1)}j$ where:

$$i' = \left\lfloor \frac{(\alpha' + \frac{A}{2})}{\delta\alpha} \right\rfloor \tag{2.30}$$

The equations above can be seen as a pre-multiplication of the ray by $D\cos\alpha_i$ followed by a convolution and another multiplication by:

$$\frac{1}{K^2} = \frac{1}{r^2 + D^2 + 2rD\sin(\beta_j - \phi)} \tag{2.31}$$

In the calculations made in this last section, a linear operator is used and not all the geometry lead to this result. However, what was seen, can be adapted and used for an arbitrary Fan-beam geometry [19].

2.4 Deep learning

Deep Learning(DL) is a subfield of Artificial Intelligence that is nowadays used as a synonym of Neural Networks. This particular technique was conceive inspired by neurobiology, in 1944, by Warren McCullough and Walter Pitts, in University of Chicago. Using a combination of algorithms and mathematics, they proposed the McCulloch-Pitts Neuron Model [30], also known as the linear threshold gate model that can mimic human thought processes and it is still the standard for all the applications.

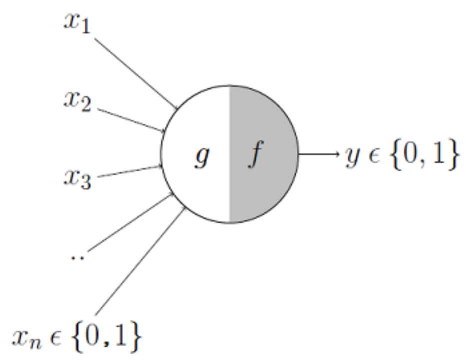


Fig. 2.8: McCulloch-Pitts model

Based on the neuron's function, this particular model takes some binary inputs and produces a single binary output related to the threshold's value. In 1957, thanks to the computational improvement and the new medicine's research of the brain's neurons, the psychologist Frank Rosenblatt designed the Perceptron model, a major improvement over the MCP neuron model [40].

Rosenblatt demonstrates that artificial neurons can learn from data; he came up with a supervised learning algorithm for this modified MCP neuron model that enabled the artificial neuron to figure out the correct weights directly from training data by itself.

Rosenblatt perceptron is a binary single neuron model. The inputs integration is implemented by adding the weighted inputs with fixed weights obtained during the training stage if the result of this addition is larger than a given threshold θ, the neuron fires.
When the neuron fires, its output is set to 1. Otherwise, it is set to 0.

In the 60s, Alexey Ivakhnenko, a soviet mathematician, created the first working deep learning network based on a set of algorithms that are settled the foundation of this particular branch of computer science. He developed the *Group Method of Data Handling* [23]. GMDH is used in data mining, knowledge discovery, prediction, complex systems modeling, optimization, and pattern recognition. GMDH algorithms are characterized by an inductive procedure that performs sorting-out of gradually complicated polynomial models and selects the best solution through the external

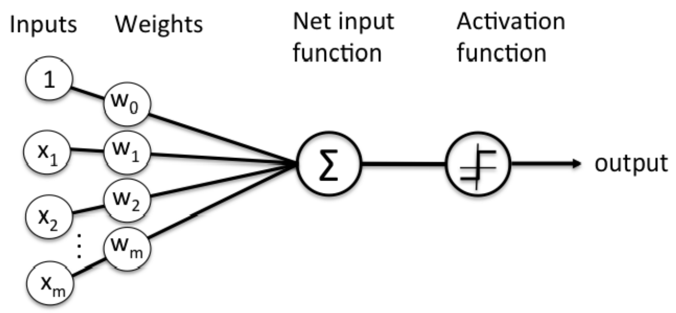


Fig. 2.9: Perceptron model

criterion.

In 1971 Ivakhnenko and his team, using the GMDH, were able to create an 8-layer deep network, and they successfully demonstrated the learning process in a computer identification system called Alpha [2].

In the same year, Henry J. Kelley, a Professor of Aerospace and Ocean Engineering at the Virginia Polytechnic Institute, derives the basis for the *Back Propagation Model* where the error is used to train the network efficiently and allow the neurons to learn and update their decision based on it [25].

Another forerunner in the deep learning history is Kunihiko Fukushima, a Japanese computer scientist. In late 1979, he creates an ANN called *neocognitron* used for handwritten character recognition and other pattern recognition tasks and served as the inspiration for *convolutional neural networks* [10].

He creates this cascade structure using two different layers, called S-Cell and C-Cell, alternately arranged in the hierarchical network. These cells resemble human brain-behavior; the S-Cells work like cells in the primary visual cortex useful for feature extraction. The features extracted by S-cells are determined during the learning process. Otherwise, C-cells resemble the visual cortex's complex cells and allow positional errors in the features. C-cells' input connections, which come from the preceding layer's S-cells, are fixed and invariable. Each C-cell receives excitatory input connections from a group of S-cells that extract the same feature but from slightly different positions.

The recognition is obtained by combining all the local and global features extracted from the cells; in the lower stages, the local features are extracted, i.e., edges or lines in particular orientations; in the higher stages, the global features are considered. The specific network's design allowed the computer the way to recognize visual patterns [11]. Starting from this fundamental architecture, an acceleration in this field is possible thanks to the *backpropagation* and better CPUs and GPUs that allow the DL to resolve the difficult problem and attract the attention of everyone for its versatility in every kind of field.

2.4.1 Neural Network

The main goal of DL is to resolve problems finding automatically a solution. During a training phase, the so-called training data set is preprocessed and meaningful features are extracted. Based on the feature vector $\mathbf{x} \in \mathbb{R}^n$ the classifier has to predict the class y, estimated by:

$$\hat{y} = \hat{f}(\mathbf{x}) \tag{2.32}$$

The classifier uses a specific parameters vector $\boldsymbol{\theta}$ estimated during the *training phase* and then evaluated on a *test data-set.*
The fundamental and simplest unit of DL, like in the human brain, is the *neuron* and consider two types of parameters:

- Bias value, b_0;
- Weight vector, $\boldsymbol{\omega} = (\omega_1, \ldots, \omega_n)$;

Every neuron can be seen as a single classifier that takes the vector $\boldsymbol{\theta} = (b_0, \omega_1, \ldots, \omega_n)$ to model a decision:

$$\hat{f}(\mathbf{x}) = h(\mathbf{w}^\top \mathbf{x} + b_0) \tag{2.33}$$

The function h(x) is a non-linear activation function that need to be chosen based on the desired result. Typical function, considering them as monotonic, bounded and continuous, are:

- Sign function: $sgn(x) = \begin{cases} -1 & \text{if } x > 0, \\ 0 & \text{if } x = 0, \\ 1 & \text{if } x > 0. \end{cases}$
- Sigmond's function: $\sigma(x) = \dfrac{1}{1 - e^{-x}}$
- Hyperbolic tangent function: $tanh(x) = \dfrac{e^x - e^{-x}}{e^x + e^{-x}}$

Increasing the number of neuron causing a better modelling capabilities, the (2.33) can be seen as a linear combination [20]:

$$\hat{f}(\mathbf{x}) = \sum_{i=0}^{N-1} v_i h(\mathbf{w}_i^\top \mathbf{x} + b_{0,i}) \tag{2.34}$$

The term v_i represent the combinations of weight of a single neuron. The difference between the true function f($\mathbf{x}$) and the decision function $\hat{f}(\mathbf{x})$ is one of the main parameters of the decision process and it is bounded by:

$$|f(\mathbf{x}) - \hat{f}(\mathbf{x})| < \varepsilon \tag{2.35}$$

Note that the error ε decrease if the neuron's number N increases. Hence, with a large number of neurons can be approximated every kind of function with a single layer network.

Gradient Descent The main problem now is to determine the parameter vector $\boldsymbol{\theta}$, the solution is obtained using the *Gradient Descent Algorithm* that is an *optimization algorithm* for finding a local minimum of a differentiable function. Gradient descent is simply used to find the values of a function's parameters, the coefficients $\boldsymbol{\theta}$, that minimize iteratively an *Objective function* finding the lowest possible point.

There is the necessity to define the group of functions called *Loss Functions* $\mathrm{L}(\boldsymbol{\theta})$ used to evaluate the quality of our model with respect to the dataset, in this way, the error of every single training example can be computed. Considering all the training datasets, the error is defined by *Cost Function.* Generally cost and loss functions are synonymous but cost function can contain regularization terms in addition to the loss function.

The loss function is a useful tool for a better understanding of the model and for the evaluation in case of modifying applied to the network. It helps also to understand how much the predicted value differs from the actual value. The loss functions are divide into three sections, every section contains many functions, the most widely used [18, 32] will be described:

- *Regression Loss Functions*: It models a linear relationship between a dependent variable, Y, and several independent variables, X's

  1. *Mean Squared Error*: One of the most commonly used and takes the average squared difference between the predictions $\hat{x}$ and expected results y over the entire dataset N. It takes the distances, and consequently the error, from the point of the regression line.

  $$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{x}_i)^2 \tag{2.36}$$

  2. *Mean Absolute Error*: measures the average magnitude of errors in a group of predictions, without considering their directions. In other words, it's a mean of absolute differences among predictions and expected results where all individual deviations have even importance.

  $$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{x}_i| \tag{2.37}$$

- *Binary Classification Loss Functions*: is a prediction algorithm where the output can be either one of two items, indicated by 0 or 1. The classification happens based on a threshold's value.

  1. *Binary Cross-Entropy*: it measures how far away from the true value, which is either 0 or 1, the prediction is for each of the classes and then averages these class-wise errors to obtain the final loss.

The cross entropy is defined as follows, considering y_i as a label with value 0 or 1 and $p(\hat{x_i})$ is the probability of the predicted point:

$$BCS = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(\hat{x_i})) + (1-y_i)\cdot log(1-p(\hat{x_i})) \tag{2.38}$$

- *Multi-Class Classification Loss Functions*: is an extension of binary classification, an object can be classified over more classes respect the previous one.

    1. *Multi-Class Cross-Entropy*: extending the (2.38) to a non binary label k, obtaining:

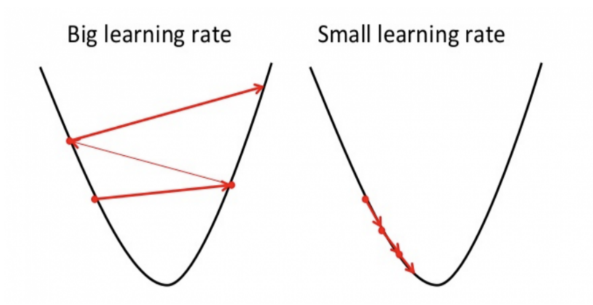
$$MCS = -\frac{1}{N}\sum_{i=1}^{N} y_i^{(k)} \cdot log(p(\hat{x_i}^{(k)})) + (1-y_i^{(k)})\cdot log(1-p(\hat{x_i}^{(k)})) \tag{2.39}$$

Considering one of the loss functions seen before, the gradient descent is computed by setting randomly weights values of vector parameters $\boldsymbol{\theta}$ and calculating the gradient on that particular point to provide the slope of the tangent line. To identify how much the slope is changing the gradient must be computed, consequently estimate the partial derivatives of the loss function with respect to $\boldsymbol{\theta}$. The changing value's is multiplied by a coefficient η called *Learning Rate* and then update the new weight's value, $\boldsymbol{\theta^{(i+1)}}$, making the difference between the previous weight's value $\boldsymbol{\theta^{(i)}}$ and η

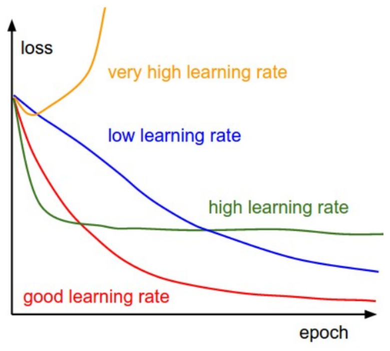
$$\theta^{(i+1)} = \theta^{(\mathbf{i})} - \eta \cdot \nabla_\theta L(\theta) \tag{2.40}$$

If there is a positive changing value the direction is towards the minimum and the iteration will go on until the global minimum L_min($\boldsymbol{\theta}$) is reached.

The value of η must be chosen wisely, it establishes how big is the step size of the movement towards the function. If the learning rate is too large the model can't converge to a minimum, if it is too small, the model needs more time to learn and to converge [65, 57].



(a) Impact of the LR on the gradient



(b) Evolution of LR over the epoch

Fig. 2.10: Learning rate

There are three different types of gradient and based on the dimension of the dataset can be used one of these establish a trade-off between the accuracy of the parameter's update and the time it takes to compute the update [58, 59, 36]:

1. Batch gradient descent: basic and simplest form of gradient seen in (2.40). The calculation are made over the whole dataset in one update and, if it is large, the computational time can be huge and potentially additional memory is necessary to complete the entire process due to redundant calculations

2. Stochastic gradient descent: the SGD performs a parameter update for each training example $x^{(i)}$ and label $y^{(i)}$, based on (2.40) obtaining:

$$\theta^{(i+1)} = \theta^{(\mathbf{i})} - \eta \cdot \nabla_\theta L(\theta, x^{(i)}, y^{(i)}) \tag{2.41}$$

   The training dataset must be partitioned into m examples and shuffle to avoid every preexisting order. It is much faster but the continuous update with high variance can create convergence's problem towards the minimum due to the noise, if the learning rate reduces slowly, it leads back to the basic *batch gradient* converging to a local or global minimum.

3. Mini-batch gradient descent: is an approach that combines the two version seen before, it performs an update every mini-batch of n training examples having better computational efficiency and memory fitting:

$$\theta^{(i+1)} = \theta^{(\mathbf{i})} - \eta \cdot \nabla_\theta L(\theta, x^{(i:i+n)}, y^{(i:i+n)}) \tag{2.42}$$

   This method reduces the variance of parameter updates lead to a more stable convergence. Common mini-batch sizes range between 50 and 256, but can vary for different applications. Mini-batch gradient descent is typically the algorithm of choice when training a neural network.

Momentum One optimization used to improve the gradient's performance is the *Momentum*, it is useful because when the minimum is near, the field lines are close to each other, so we can have steeply surface in one direction than in another, with this method the oscillations can be damped by adding fraction γ of the update vector of the past time step to the current update vector:

$$\theta^{(i+1)} = \gamma\theta^{(\mathbf{i})} - \eta \cdot \nabla_\theta L(\theta) \tag{2.43}$$

The momentum term increases for dimensions whose gradients point in the same directions and reduces updates for dimensions whose gradients change directions. As a result, faster convergence and reduced oscillation are achieved [42, 37].

Back propagation The concept of *back propagation* can be exposed, it is commonly used to efficiently compute gradients for neural network training. Back-propagation evaluates the expression for the derivative of the loss function as a product of derivatives also called *Chain rule.* These derivatives describe the sensibility of the loss function to variation of the weight's values and the activation layer's values, in such a way there is the *backwards propagated error.* Considering a single layer fully-connected network with linear activation as $\hat{\boldsymbol{x}} = \hat{\boldsymbol{f}}(\mathbf{x}) = \mathbf{W}\mathbf{x}$. Using a MSE loss is obtained the following loss function:

$$L(\theta) = \frac{1}{2}||\mathbf{y} - \hat{\mathbf{x}}||_2^2 = \frac{1}{2}||\mathbf{y} - \mathbf{W}\mathbf{x}||_2^2 \tag{2.44}$$

In order to update the weight's parameter the resolution of partial derivative is required:

$$\frac{\partial L}{\partial \mathbf{W}} = \frac{\partial L}{\partial \hat{\mathbf{f}}}\frac{\partial \hat{\mathbf{f}}}{\partial \mathbf{W}} = (\mathbf{y} - \mathbf{W}\mathbf{x})(\mathbf{x}^\top) \tag{2.45}$$

Referring to (2.40) can be achieved the vector gradient form:

$$\mathbf{W}^{j+1} = \mathbf{W}^j + \eta(\mathbf{W}^j\mathbf{x} - \mathbf{y})\mathbf{x}^\top \tag{2.46}$$

If these assumptions are extended on a more complex network structure with 3-layers $\hat{\mathbf{x}} = \hat{\mathbf{f}}_3(\hat{\mathbf{f}}_2(\hat{\mathbf{f}}_1(\mathbf{x}))) = \mathbf{W}_3\mathbf{W}_2\mathbf{W}_1\mathbf{x}$, based on (2.45) obtaining:

$$\frac{\partial L}{\partial \mathbf{W}_3} = \frac{\partial L}{\partial \hat{\mathbf{f}}_3}\frac{\partial \hat{\mathbf{f}}_3}{\partial \mathbf{W}_3} = (\mathbf{y} - \mathbf{W}_3\mathbf{W}_2\mathbf{W}_1\mathbf{x})(\mathbf{W}_2\mathbf{W}_1\mathbf{x})^\top \tag{2.47}$$

$$\frac{\partial L}{\partial \mathbf{W}_2} = \frac{\partial L}{\partial \hat{\mathbf{f}}_3}\frac{\partial \hat{\mathbf{f}}_3}{\partial \mathbf{W}_2} = \frac{\partial L}{\partial \hat{\mathbf{f}}_3}\frac{\partial \hat{\mathbf{f}}_3}{\partial \hat{\mathbf{f}}_2}\frac{\partial \hat{\mathbf{f}}_2}{\partial \mathbf{W}_2} = \mathbf{W}_3^\top(\mathbf{y} - \mathbf{W}_3\mathbf{W}_2\mathbf{W}_1\mathbf{x})(\mathbf{W}_1\mathbf{x})^\top \tag{2.48}$$

$$\frac{\partial L}{\partial \mathbf{W}_1} = \frac{\partial L}{\partial \hat{\mathbf{f}}_3}\frac{\partial \hat{\mathbf{f}}_3}{\partial \mathbf{W}_1} = \frac{\partial L}{\partial \hat{\mathbf{f}}_3}\frac{\partial \hat{\mathbf{f}}_3}{\partial \hat{\mathbf{f}}_2}\frac{\partial \hat{\mathbf{f}}_2}{\partial \hat{\mathbf{f}}_1}\frac{\partial \hat{\mathbf{f}}_1}{\partial \mathbf{W}_1} = \mathbf{W}_3^\top\mathbf{W}_2^\top(\mathbf{y} - \mathbf{W}_3\mathbf{W}_2\mathbf{W}_1\mathbf{x})(\mathbf{x})^\top \tag{2.49}$$

Note that many intermediate results can be reused during the computation of the gradient,which is one of the reasons why back-propagation is efficient in computing updates.

The activation function has a huge impact on the computation of the loss and cost functions, taking the output of the previous neuron and convert it to a useful form for the next neuron, also confining the value preventing computational issue due to high magnitude values that can be obtained in a massive neural network with a lot of parameters.

A precise selection of activation function can resolve the *vanishing gradient problem* in fact, with a classical non-linear function, like the hyperbolic tangent function, a gradient value in the range (0,1) is achieved. Since the gradient descent works with the chain rule, an exponentially decreasing gradient is present due to the product of n number in range (0,1) for a network with n layers. Using a non-bounded activation functions the solution is the deployment of *Rectified Linear Unit*:

- Rectified Linear Unit: $\mathit{ReLU(x)} = \begin{cases} x & \text{if } x \geq 0, \\ 0 & \text{elsewhere,} \end{cases}$

- Leaky Rectified Linear Unit: $LReLU(x) = \begin{cases} x & \text{if } x \geq 0, \\ & \text{elsewhere,} \end{cases}$

There is various activation function that is convex and with non zero derivative regions, with these characteristic, the vanishing gradient and non-learning neurons are avoided. Focusing on ReLU there are some computational problems due to the presence of a zero in the function that causes a kink in the derivative, causing non linearity in this point.

The main goal of the gradient is to reach the minimum in the function, and having non-linearity in 0, the slope of the line could be 0, for negative values, or either 1 for positive value. So must be considered a limited problem of non linearity that can be solved using the constant bias value present in every neuron, the sign of the bias can determine the direction of the slope. In real cases, there are a great number of neurons and every neuron has a different bias' value, so each node can change the slope at different value. When training on a reasonable-sized batch, there will usually be some data points giving positive values to any given node. So the average derivative is rarely close to 0, which allows gradient descent to keep progressing [28].

Convolutional Neural Networks Deep learning had a huge improvement exploiting specialized layers called *convolutional* and *pooling* layers. The continuous improvement in the quality, and consequently, the increasing definition and dimensions of the images, can create some problems of computation [8]. The main role of *convolutional layers* is to reduce the images providing the easiest process of extraction of features e.g., edges, circle, line, without loss, in this way some prediction's critical issue can be limited [60]. This particular layer generates an *activation map* as a result of a convolutional operation between the image matrix and a filter called *kernel*, it's useful because the nearby pixel is more correlated respect The activation map, also the output of the convolution has a dimension of:

$$W_{out} = \frac{W - F + 2P}{S} + 1 \tag{2.50}$$

with:

- W = input matrix dimension
- F = filter dimension
- P = padding size, number of pixel around the element
- S = stride, sliding size of the filter

if the image input is composed of multiple channels, the values of the activation function are the sum of the convolutional result in the position (x,y) of every channel adding also the bias term. Increasing the number of convolutional layer can analyze more deeply the images and extract high level features giving a wholesome understanding of the dataset. The application of convolutional layers in computer vision is based on three fundamentals pillar [31]:

1. *Sparse interaction*: achieved by using a smaller dimension of kernel with respect the input dimension, in this way can be extracted the important features from few number of pixel instead of considering the entire image and have an improvement in storage and computational management.

2. *Shared parameters*: in a convolutional network, the same value of the weights are used in input of different layers that can produce different results based on what feature they extract from the images.

3. *Equivariance to translation*: due to parameter sharing, the layers have a this particular characteristic if the input is changed is a way the output will change in the same way

Typically, in a deep learning network, after some convolutional layer, there is another important layer called *pooling layer.* The function of a pooling layer is to reduce the spatial dimension of the output matrix to reduce the number of parameters and the computational power, this operation is done, independently, on each feature map. There are several pooling functions such as: *average of the rectangular neighborhood, L2 norm of the rectangular neighborhood, and a weighted average based on the distance from the central pixel.* However, the most popular process is *max pooling*, which reports the maximum output from the neighborhood. It is present similarity to (2.50) where the dimension of the pooling's output is [31]:

$$W_{out} = \frac{W - F}{S} + 1 \tag{2.51}$$

With all these information, we can now build up our example of *Convolutional Neural Network.* As said before, a deep learning network takes some inputs and produce one output, for reaching this goal all these methods must be put together. The architecture can be divided in two main part:

1. *Feature extraction*, composed by convolutional and pooling layers. They are piled up is this way can be created different feature map,if the network's deep increases an high level features can be extracted. Moreover, *non linear layers* with function, i.e ReLU, Sigmoind, are added after the convolutional one to introduce a non linearty in the feature map ;

2. *Classification*, composed by a *fully connected network* that takes the output of the previous layers and flatten creating a single vector. Then, it makes the classification applying the weights and making decision based on the activation function of the neurons

In the fully connected part of the network, the input layer's value and also the output layer one are known, in between, there are the so called *hidden layers* where there is the computing of the decision based on the weights and the activation functions, in fig. (2.11) can be seen the complete architecture of a *Convolutional Neural Network* [60].

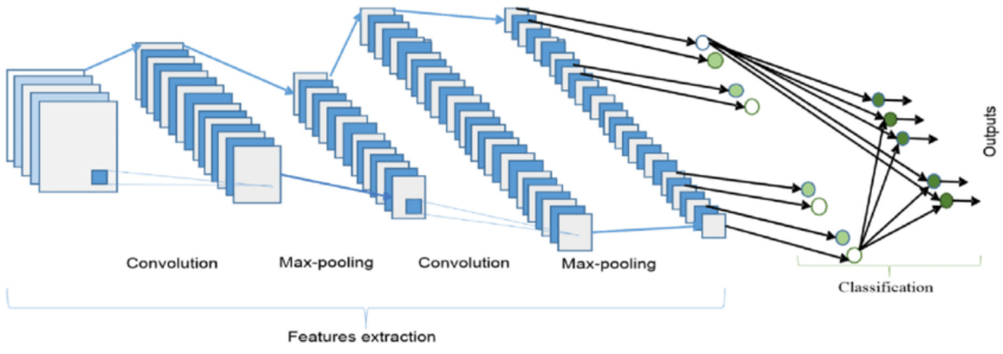


Fig. 2.11: CNN architecture

After the architecture's design, we need to setup all the networks allowing the *model* to fit and improve its results. The model and, consequently, the weights are constantly updated every *Epoch* which is completed when the entire dataset is passed forward and backward through the neural network. Sometimes, the sizes of the datasets are very large and that requires a great usage of computational resources and time for completing an epoch. One solution is to apply the *batch data processing*, an iterative approach on a portion of the dataset. Usually, it is implemented when the memory's machine doesn't fit the entire dataset, the *batch size* define the portion of data that the CNN takes to train the model, in such a way the weights' update is complete after a number of data corresponding to the batch size. Every time a batch size cross, forward and backward, an *iteration* is completed [29, 61]:

$$Epoch = Batchsize * Iteration \tag{2.52}$$

The dataset selected for training the model is split into three main categories and everyone treats differently the data:

1. *Training data*: data used to train and adjust the network's parameters to minimize the gradient of the objective function allows the CNN to learn and extract new features.

2. *Validation data*: data used to evaluate the integrity of the training procedure, the loss's results obtained on this particular set provide how accurate is the training phase. Subsequently, the model will tune its parameters based on the frequent evaluation results on the validation set

3. *Testing data*: data that are never seen by the CNN and are useful for the testing phase, after the entire training phase with the train and validation sets, in order to verify the accuracy of the overall procedure.

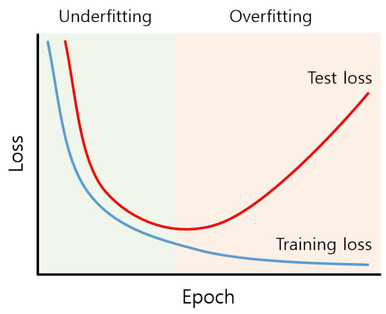
Respecting all the features that compose a CNN, it can still have poor performance due to *overfitting* and *underfitting*, before defining those fundamentals there is the necessity to introduce different concepts. A particular design can produce an higher accuracy over a defined train set, but applying the test set over the model, it can not achieved the same one. It means that the model does not have a *good generalization*, it does not have the ability to complete accurately new tasks never seen after the learning phase. Generalization, in that sense, refers to the abstract feature of intelligence which allows us to be effective across thousands of disciplines at once. In addition to generalization, to understand the problem of overfitting and underfitting, the *bias* is introduced, that is the systematic error provided by the difference between the average prediction and the correct value that lead us to a great error's value in the training phase, and the *variance*, that provide the prevision's fluctuations if the model is trained with different dataset, so it determines the sensibility of the network due to the data's randomness.

The *underfitting condition* occurs when the network is too simple and it can not accurately capture relationships between a dataset's features and a target variable. It happens when a high bias and a low value of variance are present. The solution is to extract more parameters expanding the hypothesis space. Usually, it is easy to detect the underfitting condition through the evaluation of the metrics' performance.

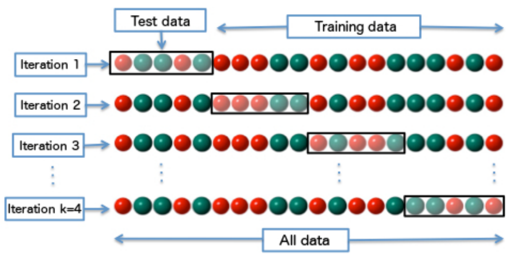
The entire dataset must be divided and the *overfitting condition* can not be detected until the *test set* is used. If the performance is better in the *train set*, maybe, the network is afflicted by *overfitting*.

This condition occurs when the model has a low bias and a high variance, so it trains over a noisy dataset. There are some solutions to mitigate the overfitting and the most popular are [24]:

- *Train with more data*: it can help the algorithm for a better detection but it can not work any time because the newly added data can be very noisy, it need also to provide clean data.
- *Early stopping*: is a method that provides an arbitrary number of epoch and stops the training when the model test's metrics, including the validation set, stops improving. See fig.2.12(a).



(a) Early stop



(b) Cross validation

Fig. 2.12: Example of overfitting restrictions

- *Cross-validation*: it considers k iteration and divides the data in the subset for the test, validation and the training, for each iteration the subset is changed so other data are considered for the analysis. See fig. 2.12(b).

Chapter 3

Data generation by computed tomography simulation

In this chapter, described the simulation process of computed tomography by considering the powerful ASTRA toolbox. It provides 2d, 3d case and supports MATLAB or Python. It handles various reconstruction algorithms and different beam geometries [1]. All functionalities of the Toolbox are available from MATLAB. ASTRA gives quick and adaptable building squares for 2D/3D tomographic remaking, pointed at calculation designers and researchers. It was started at the Vision Lab of the College of Antwerp in Belgium by PhD understudies and post-docs [48]. In toolbox also used CUDA for computed tomography simulation. CUDA (Compute Unified Device Architecture) is a parallel computing platform. The main advantage of GPU is the ability to compute parallel tasks which makes the ASTRA toolbox so powerful. The main advantages of ASTRA toolbox are:

- *Flexibility*: is possible to custom the algorithms and geometries,
- *Powerful*, because based on C++ and CUDA,
- *Intuitive to use*, can execute on MATLAB or Python interface,
- *Free*, open source and cross platform.

On fig. (3.1) shown the main features of ASTRA Toolbox.

The toolbox is represented as a software package that provides tools, samples, functions, building blocks for tomographic reconstruction and algorithm design. Moreover, provides almost all popular reconstruction algorithms.

Since ASTRA is based on Object-oriented programming languages the creation of geometric feature is simple and do not require a high level of resources for the computation.For example, the **astra_create_example_cone.m** file creates a standard cone beam geometry with parameters such as:

- *normal*: creates a non-vector geometry,

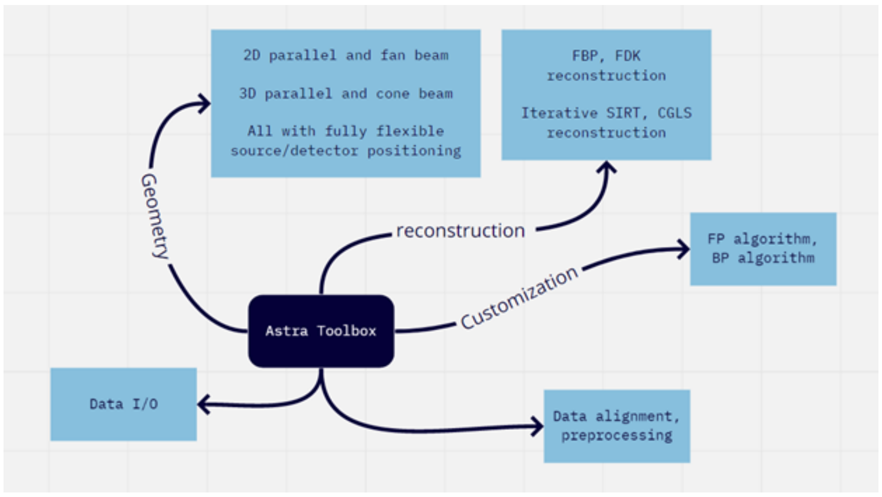
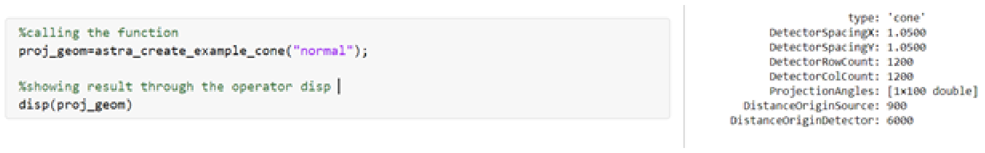

2D parallel and fan beam
3D parallel and cone beam
All with fully flexible source/detector positioning
FBP, FDK reconstruction
Iterative SIRT, CGLS reconstruction
Geometry
reconstruction
FP algorithm, BP algorithm
Customization
Astra Toolbox
Data I/O
Data alignment, preprocessing

Fig. 3.1: Main features of packages. geometry, reconstruction, customization and other functions for data processing.

- *vec*: creates a vector geometry,
- *helix*: creates an helical trajectory vector geometry,
- *deform_vec*: creates a deformed trajectory vector geometry,
- *deform_vec*: creates deformed vector geometry courtesy Van Nguyen [53].

This function returns the projection geometry as **ans** variable. For example, if execute the **astra_create_example_cone** function, will be returned the data as shown in fig. (3.2) below.



```
%calling the function
proj_geom=astra_create_example_cone("normal");

%showing result through the operator disp
disp(proj_geom)
```

```
                type: 'cone'
   DetectorSpacingX: 1.0500
   DetectorSpacingY: 1.0500
   DetectorRowCount: 1200
   DetectorColCount: 1200
   ProjectionAngles: [1x100 double]
 DistanceOriginSource: 900
DistanceOriginDetector: 6000
```

Fig. 3.2: On the right side shown the output of function.

3.1 Main concepts of toolbox

During the process, the simulation's geometry must be specified defining a volume and projection geometries. The *geometry* represents in abstract space the object or phantom is located. According to the dimensions, it can be a 2D or 3D shape and centered around the origin. The most important thing is defining the space of objects. It should fit inside the defined volume geometry. There are some limitations for volume geometry:

- All the lengths and sizes are defined relative to voxel size,

- The default voxel size is equal to 1 unit (cube voxels).

The projection geometry represents the location and transformation of the ray source and detector. In the toolbox, there are **parallel beam, fan beam and cone beam.** In fig. (3.3) shown the various projection geometry. In the first row, there is blue rectangle which is the object's space, the yellow line shows the beam directions and the green line represents the detectors. In the second row - the red box represents the volume geometry. The asterisk in the left side image represents the location of the source at each step.

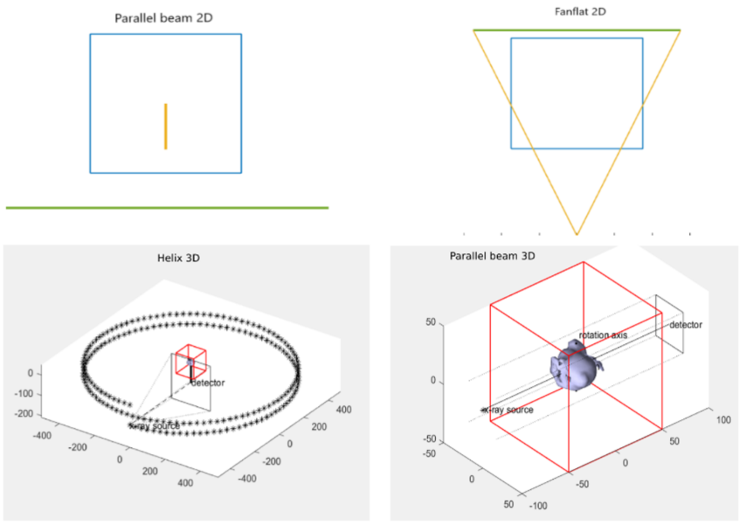


Fig. 3.3: The first row- parallel beam 2d, fanflat 2d, the second row- helix 3d, parallel beam 3d.

In fig. (3.4) is shown detailed version of 3d volume geometry.

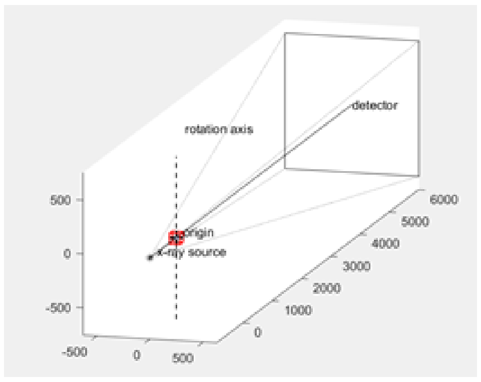


Fig. 3.4: 3D volume geometry with detailed projection geometry.

Once the volume and projection geometry is defined, the next process is the implementation of the algorithm of reconstruction. Throughout the simulations, is possible to tuning the parameters and get results at a certain point before the final reconstruction in order to evaluate the produced sinogram and choose the right method of generation.

For example in fig. (3.5) shown the image and its sinogram which was generated using the parallel 2D method.

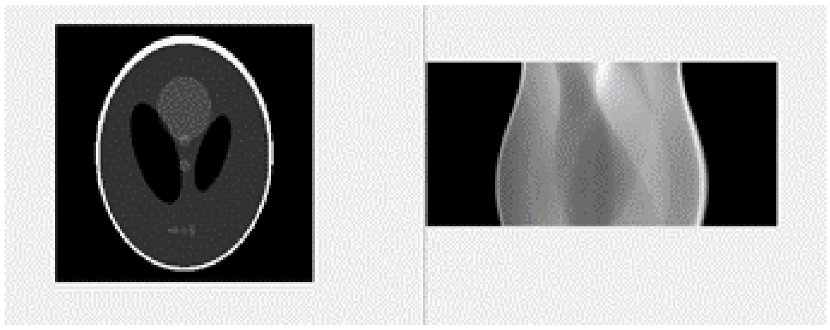


Fig. 3.5: The phantom image is on the right side and on the left side its sinogram.

The reconstruction's algorithms depend on the dimension of simulation. The general approach of algorithms for 3D and 2D cases is the same from a mathematical point of view. The difference is only the dimensions of matrices for correctly handling the data in simulation. The main reconstruction algorithms implemented in ASTRA toolbox are: **FBP, SIRT, SART, EM**. Moreover, each one could be implemented exploiting the GPU computational power and defined for 3D or 2D case.

Once the sinogram is generated, the toolbox re-generates its reconstruction. Since the ASTRA is flexible, the reconstruction stage could be set up with some parameters. For example in GPU reconstruction there are options: **SIRT_CUDA, SART_CUDA, EM_CUDA, FBP_CUDA**. In fig. (3.6) shown images such as ground truth, sinogram and reconstructed image.

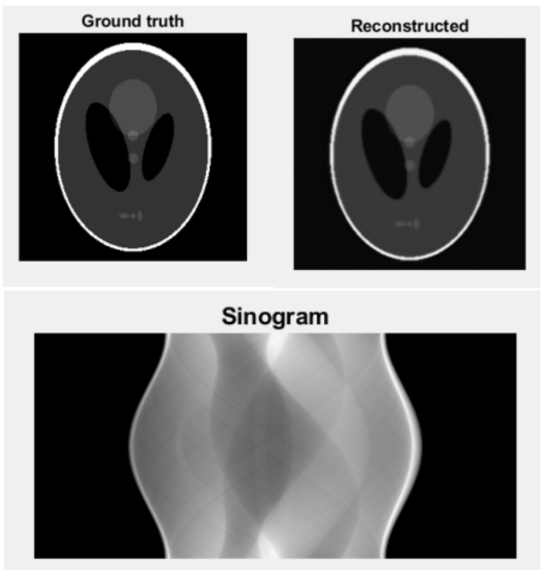


Fig. 3.6: Ground truth, reconstructed image and sinogram.

3.2 Generating of single case data by simulated computed tomography using ASTRA toolbox

Starting from the 2D case will be show how to generate the data. According to the main concepts of computed tomography using ASTRA toolbox, the first stage is setting up the volume geometry for object and setting up the projection geometry, see cod. (3.1) and cod.(3.2)

```
vol_geom = astra_create_vol_geom(256, 256);
```

Cod. 3.1: Geometry creation code with size 256x256.

```
proj_geom = astra_create_proj_geom('type', det_width,det_count,angles);
```

Cod. 3.2: Projection generator code

In fig. (3.7) is shown the volume geometry output obtained.

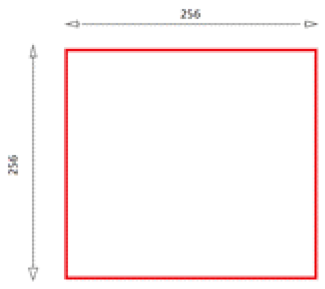


Fig. 3.7: 2-dimension volume geometry.

In order to create the projection geometry,see cod. (3.3) in ASTRA is possible to consider different options and generating the right geometry based on the usage considering parameters like:

- *type*: type of geometry considered;
- *det_width*: distance between two adjacent pixels;
- *det_count*: number of detectors in a single projection;
- *angles*: projections angles defined in radiant with a range [0, 180°]

```
proj\_geom = astra\_create\_proj\_geom('parallel', 1.0, 384, linspace2(0,pi,180))
```

Cod. 3.3: Example of projection geometry creation.

In fig. (3.8) is shown the geometry with parameters mentioned in cod.(3.3).

In fig. (3.9), there is the geometry with **fanflat** option. Therefore, the 2D fanflat beam geometry and the number of inputs is increased. Moreover, the distance from source to origin, and from origin to detectors should be set up. For example as code below:

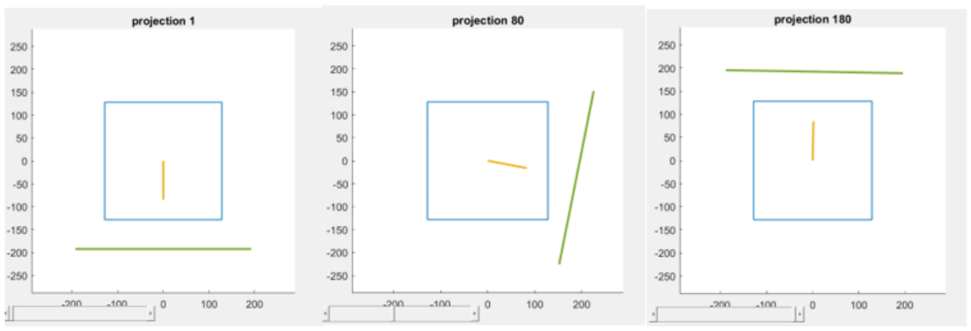


Fig. 3.8: Geometry with parallel beam with rotation from 0 up to 180 degree, width = 1 and number of detectors = 384 The blue square is volume geometry, the yellow line is the direction of rays, and green one - detectors.

```
astra_create_proj_geom('fanflat',1.38,800, linspace2(0,(120*pi)/180,207*4),900,400)
```

Cod. 3.4: Creating projection geometry.

In code represented: angles from 0 to 120-degree, distance between source and origin as 900, distance between origin and detectors as 400.

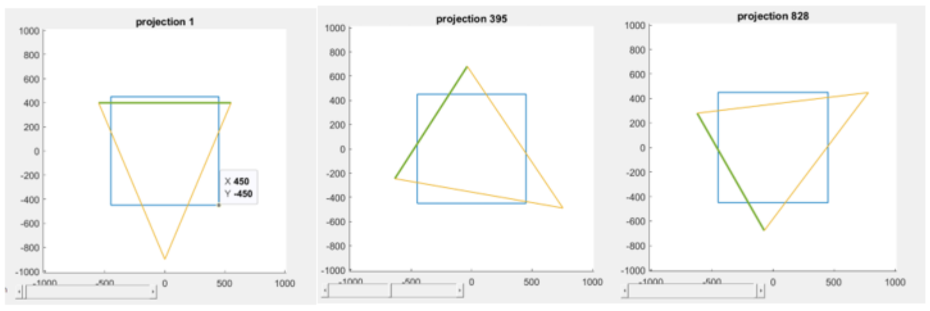


Fig. 3.9: Geometry: parallel, with rotation from 0 up to 120 degree, The blue square is volume geometry, yellow line is the direction of rays, and green one – detectors.

Considering the **fanflat** projection geometry it is flexible for customization, the code is shown in cod.(3.5). The main difference with respect the previous type is the different kind of variables considered.

```
 proj_geom = astra_create_proj_geom('fanflat_vec', det_count, vectors).
```

Cod. 3.5: Code for fanflat generation type.

Where, the variable *det_count* is number of detectors in a single projection, and the variable *vectors* is a matrix which contains the entire geometry considered, each row corresponds to a single projection and it contains, for every spatial coordinate, the following parameters:

- *src*: coordinate of the ray source,

- d : coordinate of the detector centre,
- u : the vector between the centers of detector pixels 0 and 1.

In this research, as a testing sample, the Shepp-Logan phantom image with different sizes regarding the volume geometry for each approaches were used, see fig (3.10)..

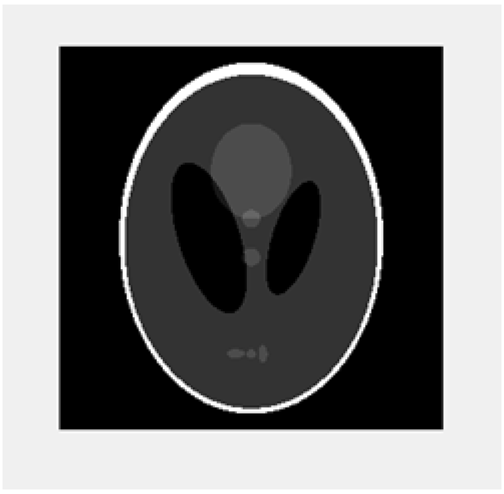


Fig. 3.10: Phantom image.

In this section, the recostruction of the Shepp-Logan phantom is performed by setting up the parameters for reconstruction algorithm using the CPU. The implementation of the code used in shown in cod. (3.6).

```
alg_id = astra_mex_algorithm('create', cfg),
astra_mex_algorithm ('iterate', alg_id , 20),
```

Cod. 3.6: Part of the code which produce the reconstructed image.

This code runs 20 iterations of algorithm, have a runtime in the order of 10 seconds. The complete representation of the code of CPU based 2D reconstruction is in cod. (3.7).

```
vol_geom = astra_create_vol_geom(256, 256);
proj_geom = astra_create_proj_geom('parallel', 1.0, 384,linspace2(0,pi,180));
proj_id = astra_create_projector('strip', proj_geom, vol_geom);
P = phantom(256);
[sinogram_id, sinogram] = astra_create_sino(P, proj_id);
astra_mex_data2d('delete', sinogram_id);
sinogram_id = astra_mex_data2d('create', '-sino', proj_geom, sinogram);
rec_id = astra_mex_data2d('create', '-vol', vol_geom);
cfg = astra_struct('SIRT');
cfg.ReconstructionDataId = rec_id;
cfg.ProjectionDataId = sinogram_id;
cfg.ProjectorId = proj_id;
alg_id = astra_mex_algorithm('create', cfg);
astra_mex_algorithm('iterate', alg_id, 20);
rec = astra_mex_data2d('get', rec_id);
astra_mex_projector('delete', proj_id);
astra_mex_algorithm('delete', alg_id);
astra_mex_data2d('delete', rec_id);
astra_mex_data2d('delete', sinogram_id);
```

Cod. 3.7: CPU based reconstruction for 2D case.

Finally, in fig. (3.11) is shown the result of the reconstruction with only the CPU.

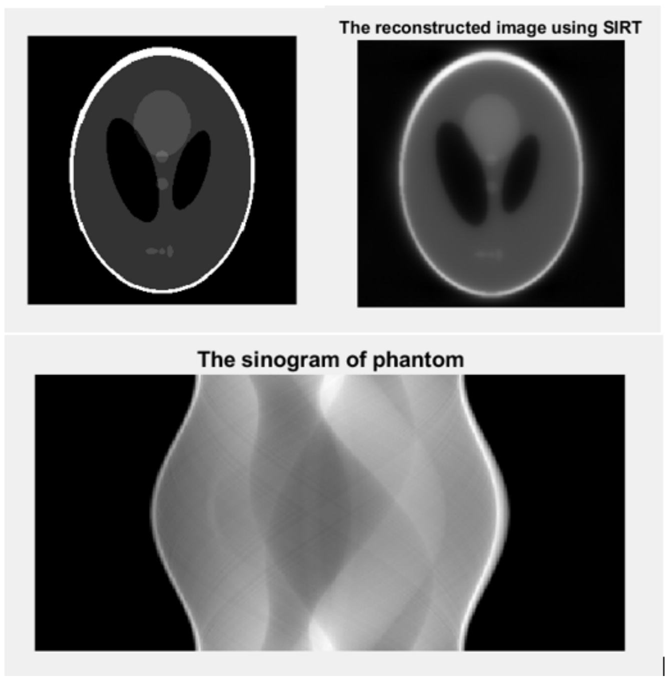


Fig. 3.11: From left to right, the original image, the sinogram and reconstructed image.

In fig. (3.12) there are reconstructions which were based on various algorithms.

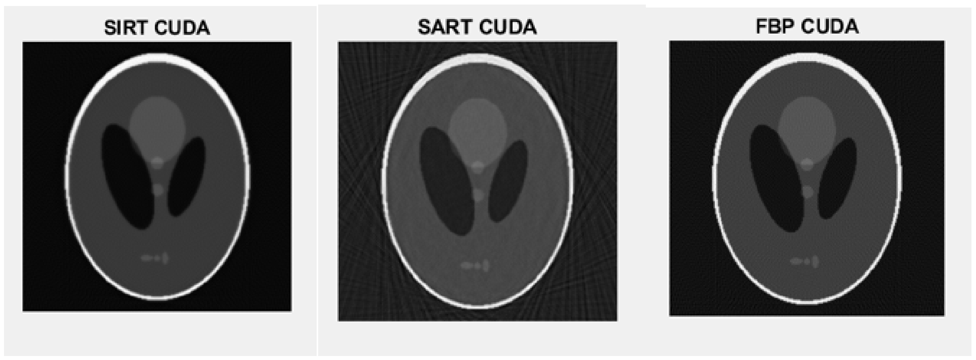


Fig. 3.12: Phantom image.

The method of reconstruction by using GPU is similar to CPU method. The choice between them does not impact the quality of the results but, since the GPU method uses a higher number of computational resources, it is important to find out a threshold between a faster reconstruction and a limitation on the resources consumption. In this method there are available the following algorithms:

- *SIRT_CUDA*,
- *SART_CUDA*,

- *EM_CUDA*,
- *FBP_CUDA*.

In ASTRA toolbox there is also possibility to create a reconstruction in a circular region, instead of the usual rectangle. This is done by placing a circular mask on the square reconstruction volume:

- c = -127.5:127.5,
- $[x \quad y]$ = meshgrid (-127.5:127.5, -127.5:127.5),
- mask = $(x^2 + y^2 < 127.5^2)$.

Further steps remain the same as before. In fig. (3.13) shown reconstruction with various algorithm using GPU as an example.

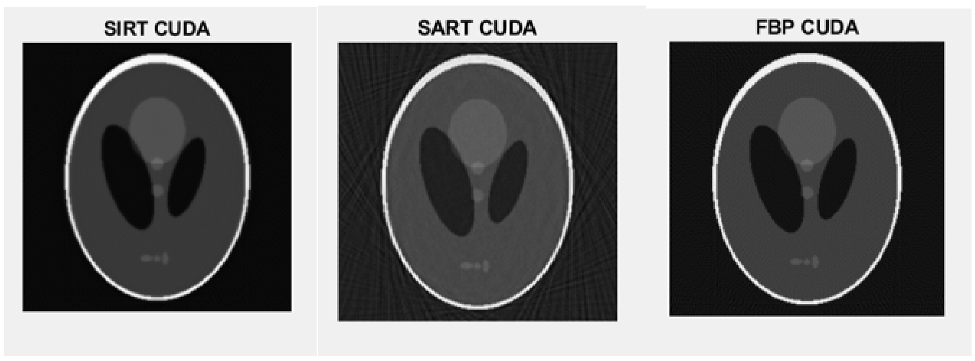


Fig. 3.13: Phantom image with SART, SIRT, FBP algorithm.

Since the creation of the sinogram is caused by the application of the Radon transform, see eq. (2.6), it can be considered as a filter. ASTRA toolbox provides options to use specific kinds of filters:

- *Ram-Lak filter*: which is sensitive to noise signal and multiplied by a suitable window to increase the performance;
- *Shepp-logan*: commonly used in FBP algorithm;
- *Cosine filter*: which is raised cosine filter. It shapes the pulsing for transmissions;
- *Hamming filter*:which uses hamming window;
- *Hann filter*: which uses a raised cosine window;
- *Tukey filter*: is tampered cosine which uses rectangular window;
- *Lanczos*: could be used as low-pass filter.

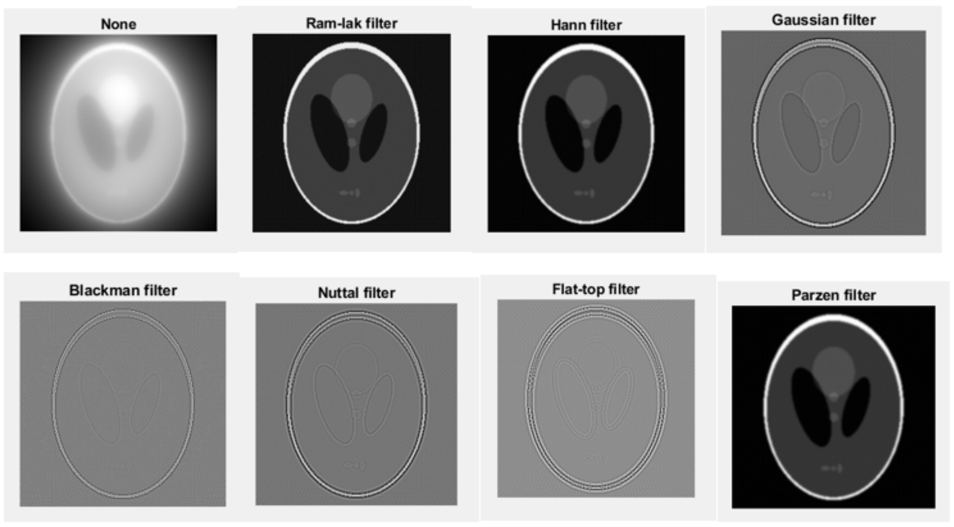


Fig. 3.14: Phantom image reconstruction with various filters.

Also other filters that can be mentioned as: *triangular, gaussian, barlett-hann, blackman, nuttall, blackman-harris, blackman-nuttall, flat-top, kaiser, parzen*, see fig. (3.14).

In ASTRA, the reconstruction can be done performing the FBP, the implementation is exposed in (3.2) where there are three steps for applying the filters, these methods works with the FBP for both the technologies, CPU or GPU:

1. Use a standard Ram-Lak filter;
2. Defining the filter in Fourier space. This is assumed to be symmetric, and ASTRA therefore expects only half band. The full filter size should be the smallest power of two that is at least twice the number of detector pixels;
3. Define a spatial convolution kernel directly. For a kernel of odd size 2*k+1, the central component is at kernel(k+1). For a kernel of even size 2*k, the central component is at kernel(k+1);

In fig. (3.17) shown the results based on 3 ways of applying the filter. In ASTRA toolbox there is also another option of simulation. The projection geometry with shifted center of rotation requires additional option as *cor_shift*. So, additional script is defined as *cor_shift=30.6*. Volume geometry and projection geometry remains the same. The projection geometry with shifted center of rotation recreated:

```
Lproj_geom_cor=astra_geom_postalignment(proj_geom, cor_shift).
```

Cod. 3.8: Creating projection geometry with shifted center of rotation.

The steps of creation are the same, firstly created a sinogram from a phantom, using the shifted center of rotation. Now recreated the sinogram data object as loading an external sinogram, using standard geometry and try to do a reconstruction,

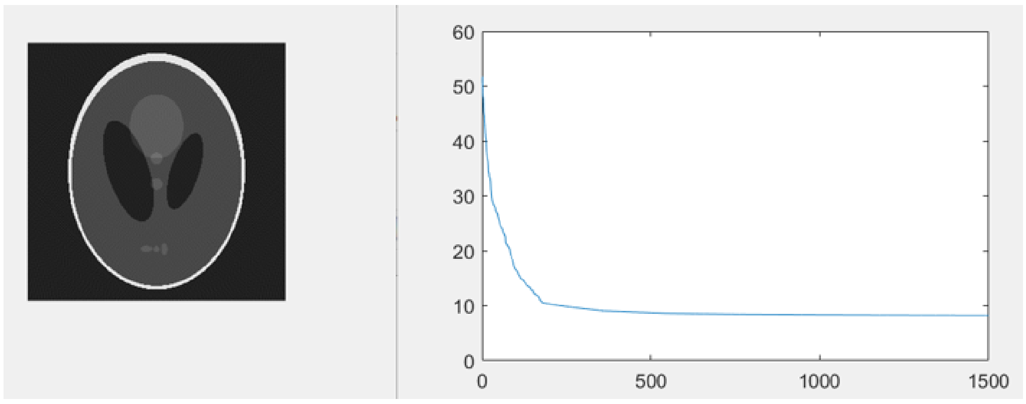


Fig. 3.15: Phantom image 1500 iterations.

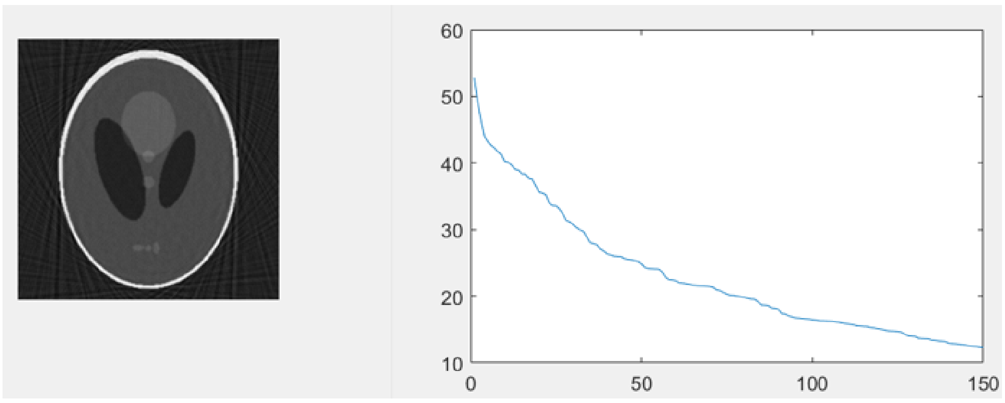


Fig. 3.16: Phantom image 150 iterations.

to show the misalignment artifacts caused by the shifted center of rotation. In fig. (3.18) shown the reconstruction with shifted center of rotation.

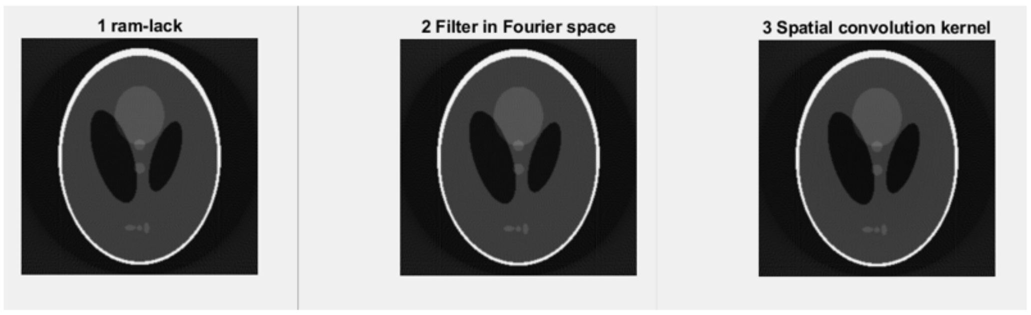


Fig. 3.17: Reconstruction based on Ram-lack, Filter in Fourier space, Spatial convolution kernel.

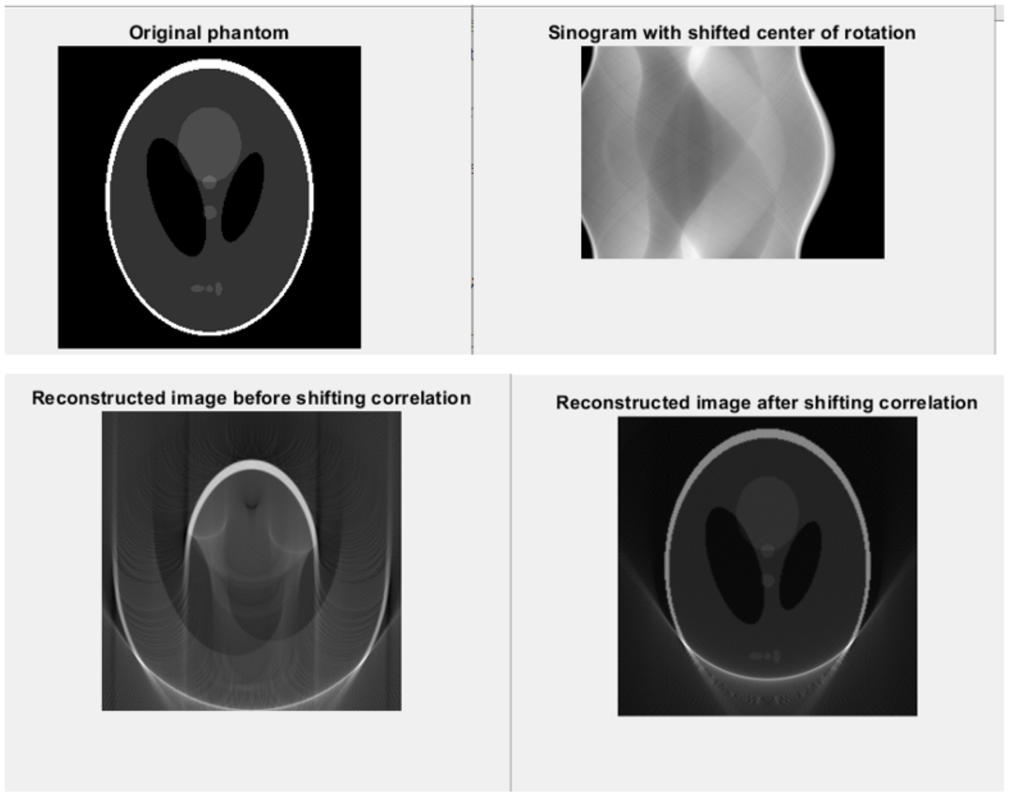

Original phantom
Sinogram with shifted center of rotation
Reconstructed image before shifting correlation
Reconstructed image after shifting correlation

Fig. 3.18: Phantom image with shifted center.

3.3 Generating multislice data by simulated computed tomography using ASTRA toolbox

In 3D case the volume geometry is 3-dimensional region in space defined by function:

```
vol_geom = astra_create_vol_geom(x, y, z)
```

Cod. 3.9: Volume geometry creation

Where the variables x, y and z represent the spatial coordinates sizes. There are considered two 3D projection geometry types which are:

- *Cone beam*;
- *Parallel beam*.

In fig. (3.19) and fig. (3.20) are shown the details of the beams geometries.

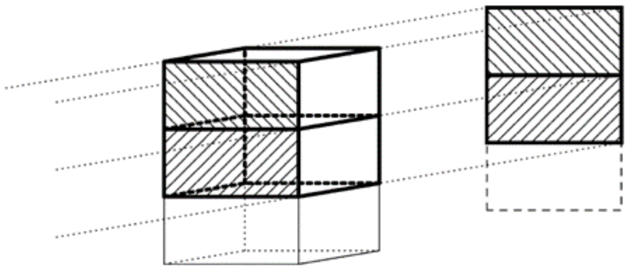


Fig. 3.19: Two volume slabs Parallel-beam projection of the cubic volume in the detector plane centre on the right. The slabs are outlined in black, and indicated by North-West (NW) and North-East (NE) diagonal patterns. The projections of these two slabs are correspondingly patterned with NW respectively NE diagonals, and do not overlap.

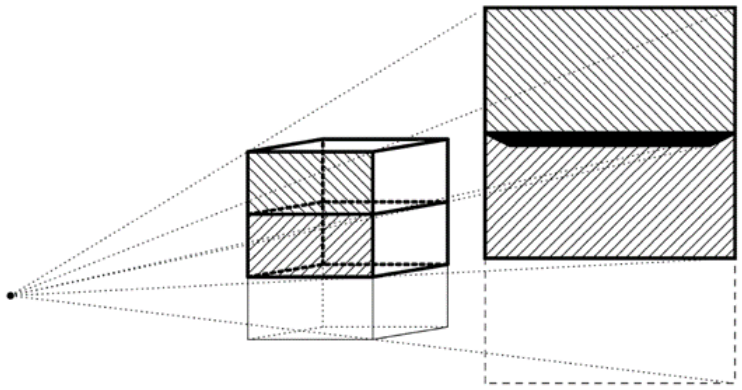


Fig. 3.20: two volume slabs Cone-beam projection of the cubic volume in the detector plane center on the right. Two slabs are outlined in black and indicated by North-West (NW) and North-East (NE) diagonal patterns. The projections of these two slabs are correspondingly patterned with NW, respectively, NE diagonals. The solidly filled area shows where the projections of the two slabs overlap.

Each function has a *regular* variant, that allows circular trajectories of the source and of the detectors around the z-axis, and a *vec* variant that allows a completely freedom of choice in the placement of source or detector [51]. These geometric functions can be divided into two different methods:

- *Circular*: the projection geometry is defined by width of detector column, height of detector row, number of rows, number of columns.
- *Free*: the projection geometry is defined by the rays directions, detector centre, height detector vector with pixels from (0,0) to (0,1), length detector vector with pixel from (0,0) to (1,0).

In fig. (3.21) are shown the results of the generator function that produces the 3D volumes and the cone beam projection geometry.

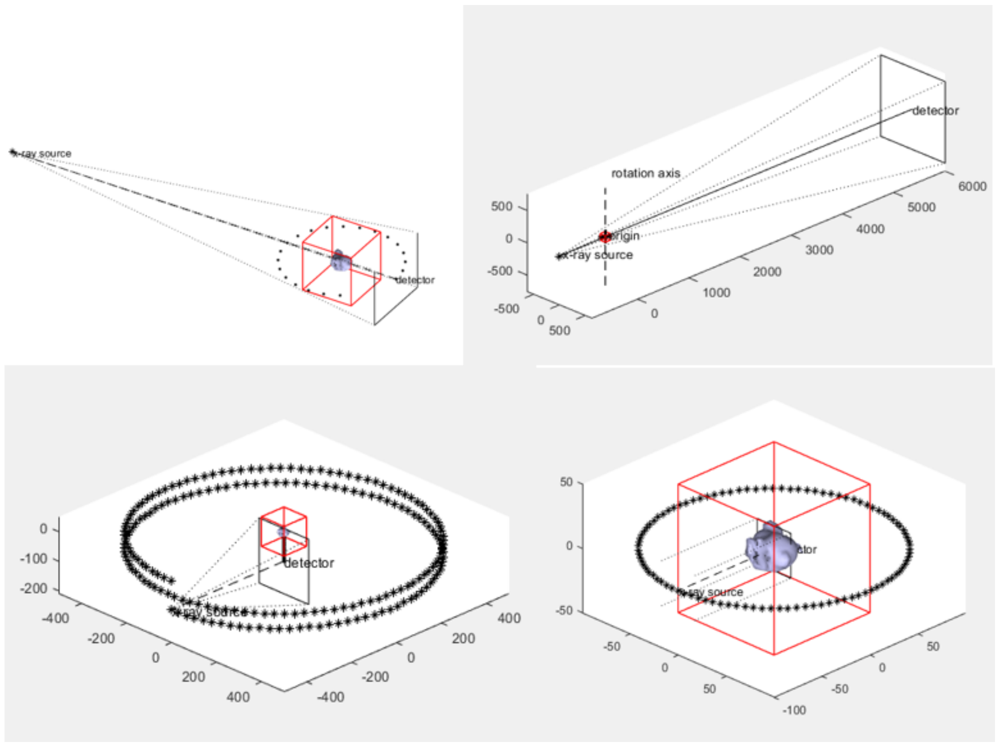


Fig. 3.21: Parallel3d, cone, cone with vector parameters.

Considering the projection geometry in 3D case,see cod.(3.10)

```
proj_geom = astra_create_proj_geom()
```

Cod. 3.10: Project geometry creation.

It create the project geometry and it has different types and parameters of customizations.

Parallel3D

The first type is *'parallel3d'* this function create a 3D parallel beam with the following parameters that can be modified:

- **det_spacing_x:** distance between two horizontally adjacent detectors,
- **det_spacing_y:** distance between two vertically adjacent detectors,
- **det_row_count:** number of detector rows in a single projection,
- **det_col_count:** number of detector columns in a single projection,
- **angles:** projection angles in radians, should be between -pi/4 and 7pi/4.

Cone

The second type is *'cone'* that create a 3D cone beam rays with the following parameters:

- **det_spacing_x:** distance between two horizontally adjacent detectors,
- **det_spacing_y:** distance between two vertically adjacent detectors,
- **det_row_count:** number of detector rows in a single projection,
- **det_col_count:** number of detector columns in a single projection,
- **angles**: projection angles in radians, should be between -$\pi/4$ and $7\pi/4$,
- **source_origin:** distance between the source and the center of rotation,
- **origin_det:** distance between the center of rotation and the detector array.

Cone_Vec

Third type is *'cone_vec'* that create the same 3D cone beam the same as before with options to custom the geometry and handles the following parameters:

- **det_row_count:**: number of detector rows in a single projection,
- **det_col_count**: number of detector columns in a single projection,
- **vectors:** a matrix containing the actual geometry. Each row corresponds to a single projection, and store all the coordinate of every spatial coordinate for:
  1. **src**: coordinate of ray source,
  2. **d** : coordinate detector center,
  3. **u** : vector between the centers of detector pixels (0,0) and (0,1),
  4. **v** : vector between the centers of detector pixels (0,0) to (1,0).

Parallel_3D

The last type is **'parallel_3D'** which is the same as *'parallel'* with options to custom the geometry with the following parameters:

- **det_row_count:** number of detector rows in a single projection,
- **det_col_count:** number of detector columns in a single projection,
- **vectors:** a matrix containing the actual geometry. Each row corresponds to a single projection, and store all the coordinate of every spatial coordinate for:
  1. **src**: coordinate of ray source,
  2. **d** : coordinate detector center,
  3. **u** : vector between the centers of detector pixels (0,0) and (0,1),
  4. **v** : vector between the centers of detector pixels (0,0) to (1,0).

In fig. (3.22) are shown some of the 3D geometries described.

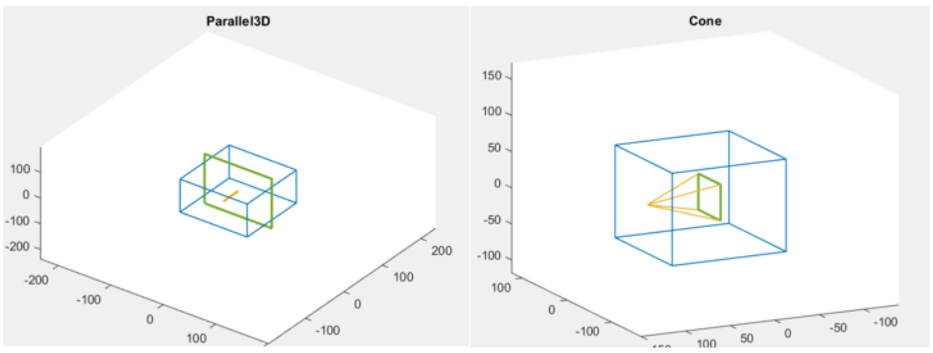


Fig. 3.22: Example of 3D beam geometries, on the left the parallel geometry, on the right the cone beam geometry.

After modelling the geometry, based on the requested specifics, and defining the sinogram the reconstruction image is generated, see fig. (3.23).

```
cube = zeros(128,128,128),
cube(17:112,17:112,17:112) = 1,
cube(33:96,33:96,33:96) = 0.
```

Cod. 3.11: Generating a cube.

For 3d case, there are some algorithms of reconstructions using GPU:

- **FP3D_CUDA**, forward projection;
- **BP3D_CUDA**, backprojection;
- **FDK_CUDA**, supports only cone,except cone_vec.Takes projection data as input and generates reconstruction;

- **SIRT3D_CUDA**, the standard reconstruction, defined as 2D case with additional options related to 3D case;
- **CGLS3D_CUDA**. the state is reset at each step astra_mex_algorithm('iterate') called. So runs CGLS for n-iterations and runs for next m-iterations leads to different outcomes from running (n+m)iterations at once. Supports all 3D geometry.

Forward and back projection are the most computationally intense operations in iterative image reconstruction due to the large size of system matrix. It is redundant to store matrix, thus the back projection and forward projection are computed on the fly. The forward projection is mathematically based on the radon transform. Back projection is the operation that smears the projection in detector space back into the object space to reconstruct the 3D volumes.

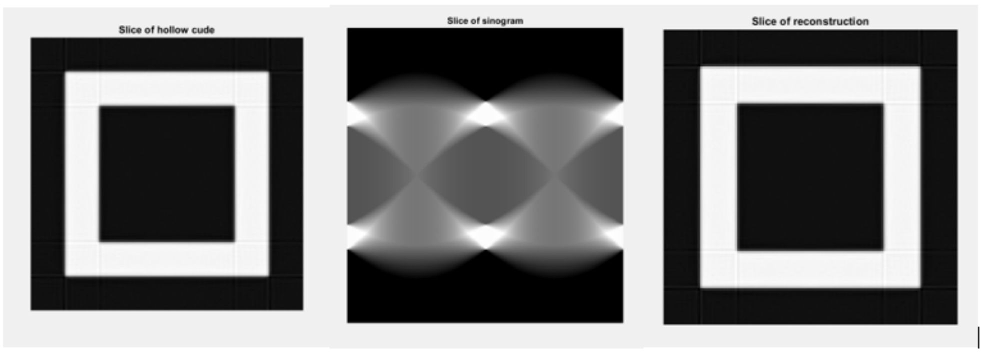


Fig. 3.23: Parallel3d projection,from left to right: slice of cube,sinogram of cube's slice, and reconstructed slice. Using a geometry 128x128x128, with parameters: parallel3d type, from 0 up to π radiant, detector size 128x192. As an object, chosen the hollow cube

Considering CUDA accelerated SIRT implementation for 3D data problems algorithm internally configures the projection building blocks that it will use. For this specific case, a non-negativity constraint is also applied. This is a simple form of prior knowledge that can lead to improved reconstruction quality. The result of this configuration is an identifier referencing the algorithm object.

The result is the reconstruction of images obtained from MATLAB memory and ready for subsequent analysis:

```
Recon=astra_mex_data3d('get',vol_id).
```

Cod. 3.12: Getting the reconstruction of image.

The creation data object for the reconstruction is the next step after setting up the geometries:

```
rec_id = astra_mex_data3d('create', '-vol', vol_geom);
```

Once the data object is done the reconstruction algorithm is set up with variuos algorithms:

```
cfg = astra_struct('CGLS3D_CUDA');
cfg.ReconstructionDataId = rec_id;
cfg.ProjectionDataId = proj_id;
```

Cod. 3.13: Reconstruction using CGLS3D$_CUDA$.

For profitable reconstruction the algorithm is executed with iterations. For example 150 iterations running consume approximately 750MB of GPU and has a run-time in the order of 10 seconds:

```
astra_mex_algorithm('iterate', alg_id, 240);
```

Cod. 3.14: Setting up the iteration of algorithm.

Finally, the result is obtained by using the following script:

```

rec = astra_mex_data3d('get', rec_id);
```

Cod. 3.15: Reconstruction the image.

Below described an example code of 3d reconstruction with parallel 3D projection geometry, angles from 0 up to π radians. Used **CGLS3D_CUDA** algorithm by GPU and 240 iterations see code(3.16)

```
vol_geom = astra_create_vol_geom(128, 128, 128);
 angles = linspace2(0, pi, 380);
proj_geom = astra_create_proj_geom('parallel3d', 1.0, 1.0, 128, 192, angles);
 cube = zeros (128,128,128).
cube (17:112,17:112,17:112) = 1;
cube (33:96,33:96,33:96) = 0;
 [proj_id, proj_data] = astra_create_sino3d_cuda(cube,proj_geom, vol_geom);
rec_id = astra_mex_data3d('create', '-vol', vol_geom);
cfg = astra_struct('CGLS3D_CUDA');
cfg.ReconstructionDataId = rec_id;
cfg.ProjectionDataId = proj_id;
alg_id = astra_mex_algorithm('create', cfg);
astra_mex_algorithm('iterate', alg_id, 240);
rec = astra_mex_data3d('get', rec_id);
astra_mex_algorithm('delete', alg_id);
astra_mex_data3d('delete', rec_id);
astra_mex_data3d('delete', proj_id).
```

Cod. 3.16: Reconstruction script using CGLS3D_CUDA

In fig. (3.24) shown the results of reconstruction.

Let us consider **parallel3d** projection geometry with parameters:

- proj_geom = astra_create_proj_geom('parallel3d', 1.0, 1.0, 128, 192, angles),
- angles() angles = linspace2(0, pi, 180).

In this case, the phantom3d model is used as an object in ASTRA toolbox see fig. 3.25.

Finally, following the same steps the reconstruction is obtained for 3d object see fig. 3.26.

For STL models the toolbox provides several functions for customization. The *'stlTools'* is a collection of functions, samples, and demos to illustrate how to deal with STL files. Some of them are contributions published in MATLAB. This toolbox contains the following files:

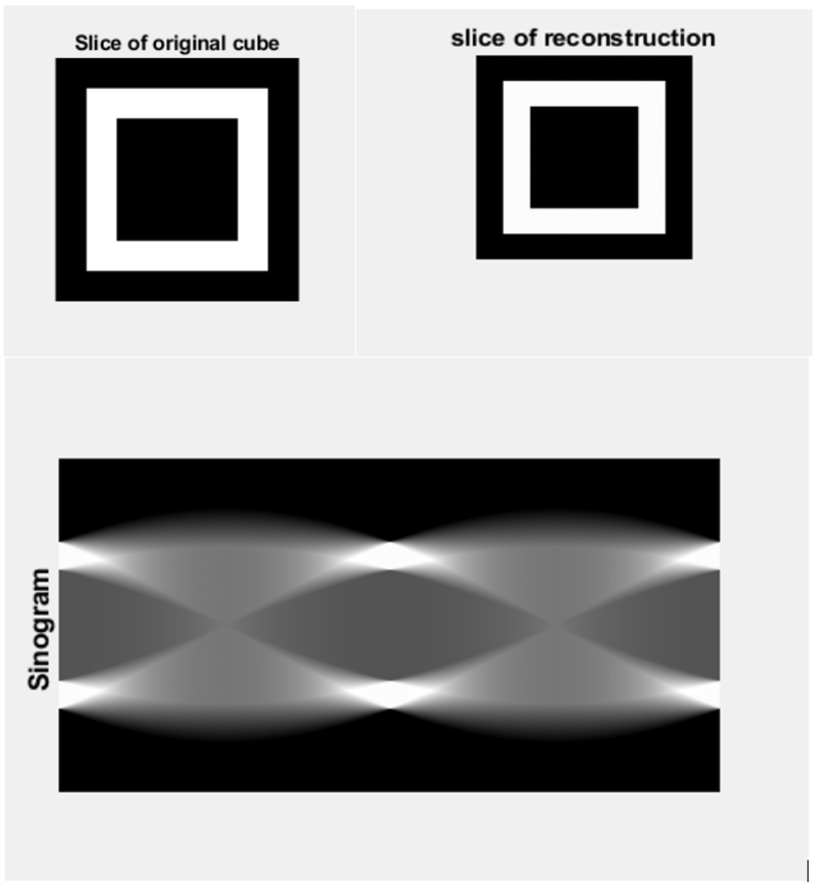


Fig. 3.24: CGLS3D_CUDA algorithm applied.

- *stlGetFormat*: identifies the format of the STL file and returns 'binary' or 'ascii',
- *stlReadAscii*: reads an STL file written in ASCII format,
- *stlReadBinary:*reads an STL file written in binary format,
- *stlRead*: uses 'stlGetFormat', 'stlReadAscii' and 'stlReadBinary' to make STL reading independent of the format of the file,
- *stlWrite*: writes an STL file in 'ascii' or 'binary' formats,
- *stlSlimVerts*: finds and removes duplicated vertices,
- *stlGetVerts*: returns a list of vertices that are 'opened' or 'closed' depending on the 'mode' input parameter. An 'open' vertices is the one that defines an open side. An open side is the one that only takes part of one triangle,
- *stlDelVerts*: removes a list of vertices from STL files,
- *stlAddVerts*: adds the new vertices from a list (and consequently, new faces) to a STL object,

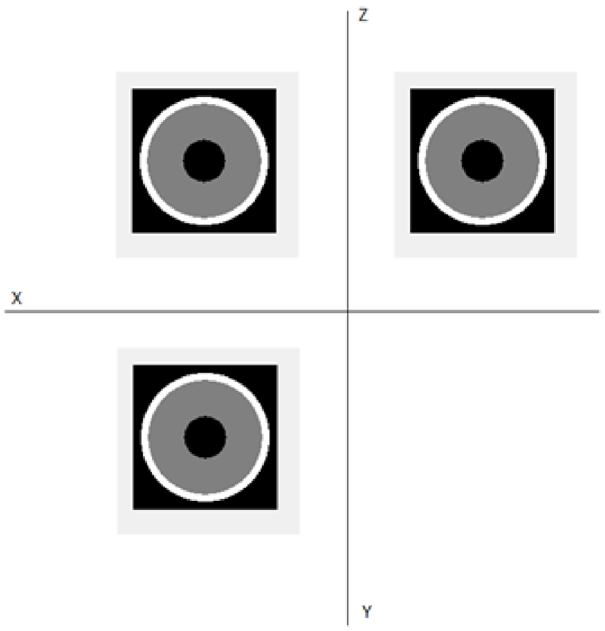


Fig. 3.25: Here is volume sphere, on figure is shown his projections.

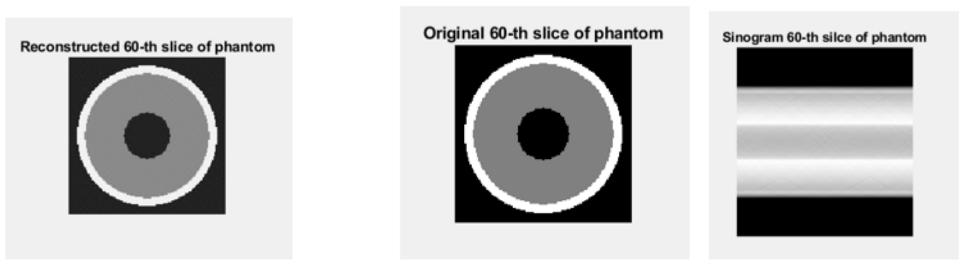


Fig. 3.26: Here is volume sphere, where applied the CGLS algorithm with parallel3D projections.

- *stlPlot*: is an easy way to plot an STL object,
- *stlDemo*: is a collection of examples about how to use stlTools,
- *femur_binary*: is an ASCII STL sample used in 'stlDemo',
- *sphere_ascii*: is a binary STL sample.

In this article, the ASTRA toolbox has been introduced and several use-cases were presented showing off its main features. These include its free, open-source nature [52], its ability to describe virtually any 2D and 3D projection setup using vector geometries and its flexibility regarding the easiness with which it can be included in other frameworks. Combined, they make the ASTRA Toolbox very suitable for fast prototyping of new applications, new geometry setups and new reconstruction methods. Furthermore, due to its efficiently implemented building blocks, these prototypes can straightforwardly be up scaled to realistic data sizes and usages. However, it should be noted that the ASTRA Toolbox is by no means the first and only tool to consider when dealing with tomographic reconstruction. Many users might prefer commercial software packages as they provide an easy-to-use graphical user interface, which the ASTRA Toolbox does not. Also, because of its flexibility regarding application fields, scanning devices, and protocols, the ASTRA Toolbox cannot provide out-of-the-box support for various file formats. In order to use the toolbox, the user thus requires knowledge of these file formats, and the skill to parse and process them in the MATLAB or Python layer. Moreover, the algorithms bundled in the ASTRA Toolbox are limited to reconstruction methods, and do not include typical preprocessing (e.g., flat field correction, phase retrieval) and post-processing (e.g., segmentation, morphological operations, mesh generation) algorithms. For practical use, the toolbox must thus be linked with other tools such as TomoPy [34]. These disadvantages limit the target audience of the ASTRA Toolbox mainly to researchers and users with expertise in computer science. In summary, the ASTRA toolbox is an excellent platform to bridge the large gap between application scientists and researchers in the field of numerical mathematics and linear solvers [50]. That way, new and advanced numerical solvers can be tested on realistic data, benefiting both communities.

Chapter 4

Limited range angle CT reconstruction

In this chapter the generation of reconstructed images exploiting the ASTRA simulation toolbox is explained. Subsequently the denoising deep learning based method is introduced, describing both the network architecture and training procedure. The main goal is to train the CNN and perform a denoising operation over the reconstructed images. The dataset is composed of CT scan DICOM files containing images of a human's chest and abdomen from The Cancer Imaging Archive (TCIA) [6]. These images can be considered as a *ground-truth dataset*. They are obtained by measurements without angle limitations, so the *sinograms* contain every projection for every angle. These images were then processed using the ASTRA toolbox, simulating a reconstruction with limited angles, using the geometric parameters in tab.(4.1).

Parameter	**Value**
Distance between the center source and detector(mm)	1300
Distance between the center and the rotation axis (mm)	900
Sampling interval between two adjacent projection views (deg)	0.145
The angle between the first source and the last source (deg)	30
Number of source	207
Number of detector	800
Diameter of the field of view (mm)	352.90
Size of each detector element (mm)	1
Pixel size (mm^2)	1.38 x 1.38

Table 4.1: Geometric parameters

Those geometrical features can recreate a virtual representation of a CT scanner and produce virtual reconstructions using the *groundtruth dataset* providing limited sinograms which contain projection's portions. Exploiting the SART algorithm, eq. (2.15), was possible to obtain *reconstruction images* that includes noise and measurement's artifacts for three different ranges using the specific simultaneous movement of the sources and detectors:

- (0 - 90°), which consider three scanning phases;
- (0 - 120°), which consider four scanning phases;
- (0 - 150°), which consider five scanning phases.

This setup allows to deal with different qualities reconstructions, and consequently, have different levels of noise and artifacts. In fig.(4.1) is exposed the movement of the components that rotate around the object in order to make the measurement, in red are highlighted the sources, and in front of these, there is a flat green detector panel that rotates every acquisition time T.

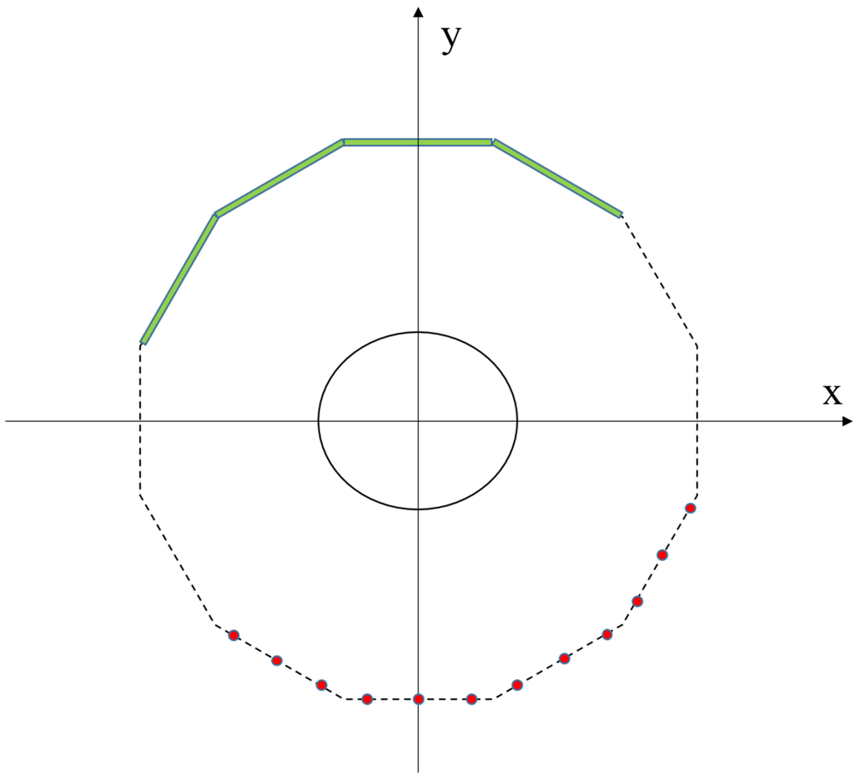


Fig. 4.1: Simple view of CT scanner setup. In this case, can be seen a 4T acquisition obtain the scanning range of (0 - 120°).

In our assumptions were considered two different approach for the image's denoising with the aims of finding a methods which can enhance the CNN's performances. There will be exposed a method which consider a single slice in input of the CNN and another one considering a multi slices input. The difference is in the number of images taken into account for the extraction features process of the network. Both of them will be exposed in the following sections showing the results and the improvement obtained.

4.1 Single Slice Denoising Case

In order to improve the performance of the CT scanning, different methods could be considered and done by:

- Increasing the power of source emission,
- Increasing the number of simultaneous sources,
- Multiply the reconstruction's time.

The main drawback of these applications is the appearance of errors and noise in the measurement. To attenuate the imposed noise, a trade-off during the design of these options should be considered. One of the many solutions provided was founded in research based on measurements with limited angle ranges, considering three different intervals: (0,90°), (0,120°) and (0,150°) with respectively three, four and five phases measurement are conducted. In this work is performed a simulation of measurement with ASTRA toolbox, and considering the steps for the creation of a virtual measurement, the definition of the volume geometry is the first quantity to define.

4.1.1 Simulated CT data generation for single slice denoising case

The considered TCIA dataset is composed by DICOM files. Digital Imaging and Communications in Medicine (DICOM) is the standard for the communication and management of medical imaging information and related data [62]. It is most used format for storing and transmitting medical images enabling the integration of medical imaging devices such as scanners, servers, workstations, printers, network hardware, picture archiving and communication systems (PACS) from multiple manufacturers. It has been widely adopted by hospitals and is making inroads into smaller applications like dentist's and doctor's offices.

The data generation part was implemented using MATLAB through function **dicominfo** the information about file is returned and with **dicomread** the file is imported into MATLAB:

```
filename = dicominfo('dicomfilename');
img = dicomread(filename);
```

In fig. (4.2) is shown one sample of real TCIA image, For the data generation the image should be doubled by applying function **double()**. The **Double** function is the default numeric data type (class) in MATLAB, providing sufficient precision for most computational tasks. Numeric variables are automatically stored as 64-bit (8-byte) double-precision floating-point values.

The following step is to define the size of volume geometry. Since the acquired data size is 512x512, therefore the volume geometry will have the same size. Below is shown the code.

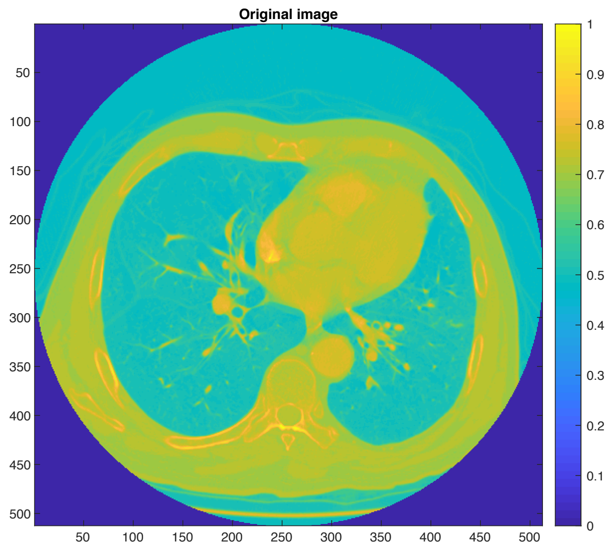


Fig. 4.2: Image of real data.

```
vol_geom = astra_create_vol_geom(512, 512)
```

Cod. 4.1: Volume geometry creation

The initial parameters of computed tomography simulation meet the requirement of the aforementioned table.

- *Ranges*: (0,90°); (0,120°) ; (0,150°),
- *Sub-angles*: 30 deg,
- *Distance source to detector*: 1300 mm,
- *Distance source to rotation axis*: 900 mm,
- *Sampling interval*: 0.145 deg,
- *Number of source*: 207,
- *Number of detectors*: 800.

The next step is setting up the projection geometry:

```
proj_geom=astra_create_proj_geom('fanflat',1.38,800,linspace2(0,(90*pi)/180,207*3)
    ,900,400)}
```

Cod. 4.2: Projection geometry code in ASTRA toolbox

Here used **fan-flat** type for range angle from 0 to 90° and the function astra_create_sino_gpu is implemented in order to generate the sinogram of real data:

```
[sinogram_id , sinogram] = astra_create_sino_gpu(img,proj_geom , vol_geom);
sinogram_id = astra_mex_data2d('create', '-sino', proj_geom,sinogram);
rec_id = astra_mex_data2d('create', '-vol', vol_geom);
```

Cod. 4.3: Code implemented for the sinogram's generation derived from the real data

Since ASTRA working also with CUDA, here is shown the set up of the parameters for a reconstruction algorithm using the GPU in order to obtain a faster reconstruction:

```
cfg = astra_struct('FBP_CUDA');
cfg.ReconstructionDataId = rec_id;
cfg.ProjectionDataId = sinogram_id;
```

Cod. 4.4: CUDA code for the reconstraction phase

The algorithm object is created from the configuration structure here:

```
alg_id = astra_mex_algorithm('create', cfg);
```

In order to get high quality reconstruction, here is implemented 300 iterations of the algorithm:

```
astra_mex_algorithm('iterate', alg_id , 300);
```

Finally the code returns the result:

```
rec = astra_mex_data2d('get', rec_id);
```

Cod. 4.5: Reconstruction the data.

At the end of the simulation the memory is cleared by using the commands:

```
astra_mex_algorithm('delete', alg_id);
astra_mex_data2d('delete', rec_id);
astra_mex_data2d('delete', sinogram_id);
```

Cod. 4.6: Cleaning up the memory.

In cod. (4.7) is shown the full implemetation code:

```
filename = dicominfo('1-005.dcm');
img = dicomread(filename);
img=double(img);
vol_geom = astra_create_vol_geom(512, 512);
proj_geom=astra_create_proj_geom('fanflat',1.38,800,linspace2(0,(90*pi)/180,207*3)
    ,900,400)};
[sinogram_id , sinogram] = astra_create_sino_gpu(img, proj_geom , vol_geom);
sinogram_id = astra_mex_data2d('create', '-sino', proj_geom , sinogram);
rec_id = astra_mex_data2d('create', '-vol', vol_geom);
cfg = astra_struct('FBP_CUDA');
cfg.ReconstructionDataId = rec_id;
cfg.ProjectionDataId = sinogram_id;
alg_id = astra_mex_algorithm('create', cfg);
astra_mex_algorithm('iterate', alg_id , 300);
rec = astra_mex_data2d('get', rec_id);
astra_mex_algorithm('delete', alg_id);
astra_mex_data2d('delete', rec_id);
astra_mex_data2d('delete', sinogram_id);
```

Cod. 4.7: Reconstruction with FBP_CUDA

In fig. (4.3) are shown the results of the SIRT and SART reconstructions starting from same geometry and considering the same image for both of them. In order to perform the correct simulation process is possible to provide a drawing of the designed geometry using the function: *astra_geom_visualize(proj_geom,vol_geom)*.

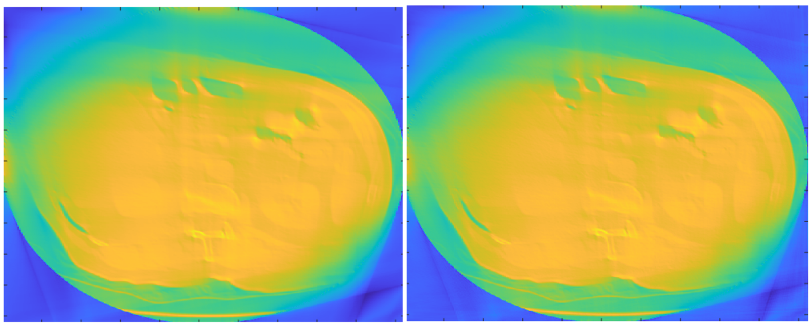


Fig. 4.3: On the left- SIRT_CUDA algorithm, on the right-SART_CUDA.

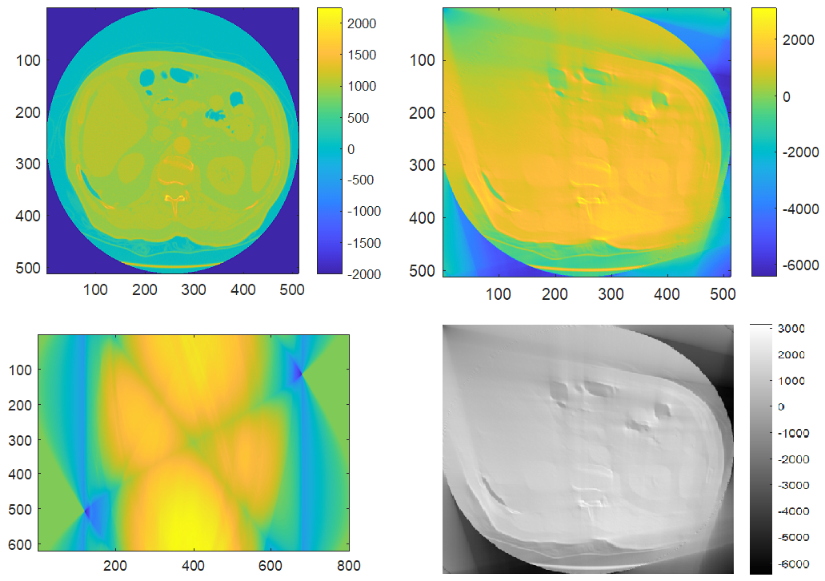


Fig. 4.4: In the first row - from left to right: The original and reconstructed images, in the second row- from left to right: The image's sinogram and the reconstructed image in grey scale.

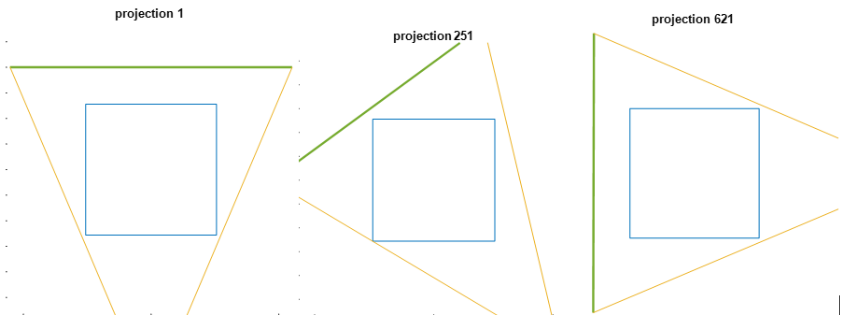


Fig. 4.5: Three steps of rotation projection visualization,considering the range (0,90°). The green line represent the detectors, the yellow line is range sub-angle, and the blue rectangle is the object's volume geometry.

In fig.(4.4) is shown the reconstruction by applying the FBP_CUDA algorithm for angle range (0,90°). In fig.(4.5) is shown the result of this function with the evolution of the projection's process.

One of the checkpoint of our research work is to compared each the results obtained in order to evaluate the enhancement of the process, for tis reason are considered two evaluation metrics useful for a deep investigation. The first considered is the PSNR that computes the peak signal-to-noise ratio between two images. This ratio is used as a quality measurement between the original and compressed image. The higher PSNR leads to a better quality of the reconstructed image.

PSNR

Peak Signal-to-Noise Ratio, it is used to estimate the difference between two images.

$$\begin{aligned} PSNR(x,y) &= 10 \cdot \log_{10} \left[\frac{(max(max(y)))^2}{MSE}] \right] \\ MSE(x,y) &= \frac{\sum_{i,j}(x_{i,j} - y_{i,j})^2}{N} \end{aligned} \tag{4.1}$$

Where:

- y = original image;
- x = reconstructed image;
- i,j = denote the pixel position of the images
- N = image's pixels

In tab. (4.2) are proposed the results of the PSNR computation, there are shown the performances of various algorithms over different rang of measurements.

PSNR (dB)		
Range (degree)	SART	SIRT
0-90	-50,4918	-59,8961
0-120	-46,7528	-57,2586
0-150	-42,2501	-54,2711

Table 4.2: Comparison between SART and SIRT algorithms over different ranges.

In this research work, another evaluation metric is considered, the Structural Similarity Index Measure (SSIM) is also used to check the improvement of the reconstruction image. SSIM is a method for predicting the perceived quality of digital television and cinematic pictures, as well as other kinds of digital images and videos. SSIM is used for measuring the similarity between two images [63]. The higher SSIM leads to a better quality of the reconstructed image.

SSIM

Structural Similarity Index Measurement, which is used to measure the component similarity between the reconstructed image and the original one.

$$SSIM(x, y) = \frac{(2\bar{x}\bar{y} + C_1)(2\sigma_{xy} + C_2)}{(\bar{x}^2 + \bar{y}^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{4.2}$$

Where:

- $\bar{y}$, $\bar{x}$ = respectively, are the mean values of the original and reconstruced images;
- σ_x,σ_y = respectively, are the standard deviations of the original and reconstructed images;
- σ_{xy} = covariance of the images
- C_i = constant value used as a normalization factor.

In tab. (4.3) are proposed the results of the SSIM computation, there are shown the performances of various algorithms over different rang of measurements.

SSIM		
Range (degree)	SART	SIRT
0-90	0,296	0,2628
0-120	0,3651	0,3337
0-150	0,4143	0,4362

Table 4.3: Comparison between SART and SIRT algorithms over different ranges.

The computing of the equations was done on MATLAB considering the calculations over the correspondence between the reconstructed image and his respective original image, see cod. (4.8).

```
SSIM = zeros(1,10); %Variables useful for the storage
PSNR = zeros(1,10);
N = 512 * 512
for d = 1:10
    x = squeeze(pr(d,:,:));
    y = squeeze(og(d,:,:));
    tmp = 0;
    % Drawing up for the computation
    for i = 1:512
        for k = 1:512
            diff = (x(i,k) - y(i,k))^2;  % Pixels difference
            tmp = tmp + diff;            % Variable for the storage
        end
    end
    MSE = (tmp/N);
     % PSNR computation
    PSNR(1,d) = 10*log10(((max(max(y)))^2)/MSE);
    x_mean = sum(x(:))/N;
    y_mean = sum(y(:))/N;
    covxy = cov([x(:),y(:)])
    %SSIM computation
    SSIM(1,d) =
    [(2*x_mean*y_mean)*(2*covxy(1,2))]/[(x_mean^2 + y_mean^2)*(covxy(1,1) + covxy(2,2))];
end
```

Cod. 4.8: MATLAB code for the evaluation parameters' computing.

The reported results show that the SART method could achieve a better reconstruction in limited range angle setup so this reconstruction method should be used in order to generate the input to be donoised by the CNN. This is also in accordance to results that could be found in literature, for example in [54].

4.1.2 Convolutional Neural Network implementation - Single Slice Case

In this section will be explored the convolutional neural network used in our research, analysing the implementation of the network and the management of the data for the single slice case and the multi slice case. In our assumptions were used a neural network, composed of a U-net,originally used for image segmentation applied to human's membrane. Within the network was added a *skip connection* that links the input and the output of the CNN. The latest layer provides the outcome based on the decision made during the entire process. The overall network, called *SARTConvNet* can be divided into two different parts, the *downhill path* and the *uphill path* one mirrored to the other, see fig.(4.6). The network is composed by:

- *Zero Padded Convolutional Layers*: with 3x3 dimension, it computes the feature's extraction, see eq. (2.50). In this steps are carried out operations of *batch normalization* and *rectified linear unit*, in this case is considered a *ReLu Unit*, see eq. (2.4.1).

- *Pooling Layers*: with 2x2 dimension, it computes the image's down-sampling in the downhill path while, in the uphill path, it computes the image's up-sampling. After this operation, the number of feature channels is increased, see eq. (2.51). For that reason, the capability of features extraction is enhanced, providing a better CNN [27].

- *Skip connections*: implementing these operations is helpful because they can avoid information loss due to the operations done in every layer [26].

The CNN is implemented considering the Python environment taking into account two *deep learning framework* which are *Tensorflow* and *Keras*. The model in [64] was considered; it was modified, changing the input size considering a dimension of 512x512 pixels. For the creation of the skip connection, a *merge layer* was added. Also, the loss parameter was modified, considering a loss metric provided by MSE, eq.(2.36). Lastly, the *Stochastic Gradient Descent* was considered as an optimizer parameter, eq.(2.41), with a learning rate in the range of [0.01 - 0.0001] and a *momentum* with value 0.99.

```python
def unet(pretrained_weights = None,input_size = (512,512,1)):
inputs = Input(input_size)
    conv1 = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(inputs)
    conv1 = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv1)
    pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)
    conv2 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(pool1)
```

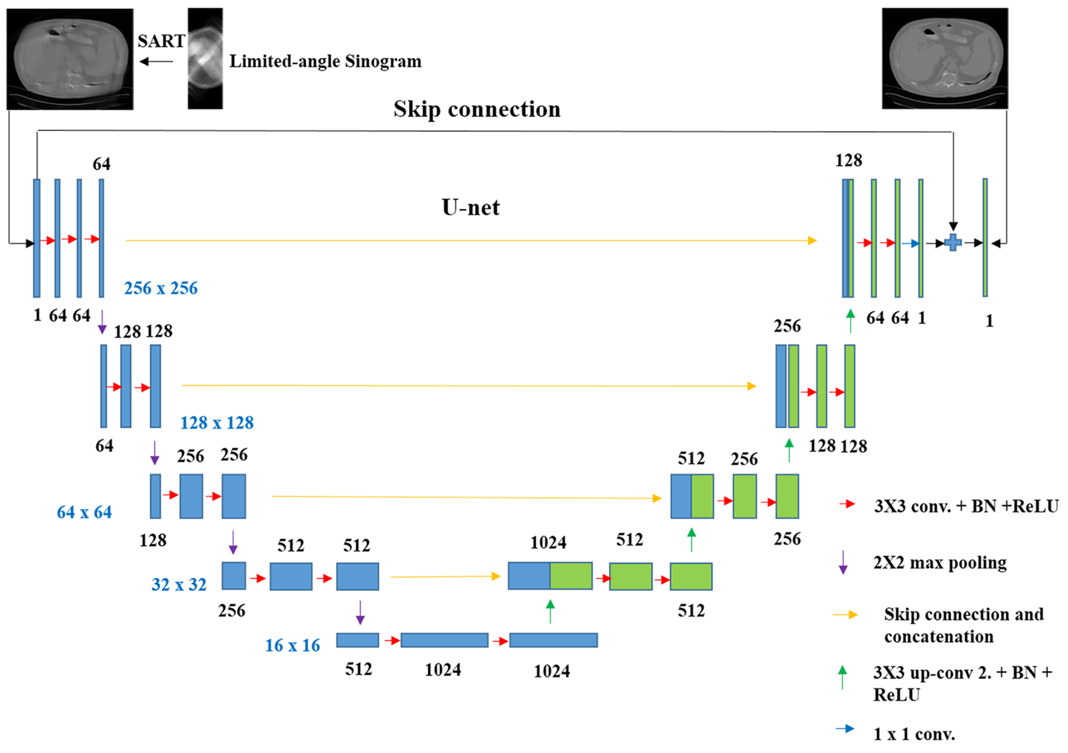


Fig. 4.6: Network architecture, here can be seen the distinction between the downhill and uphill paths with each layer and proper dimension.

```
    conv2 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv2)
    pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)
    conv3 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(pool2)
    conv3 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv3)
    pool3 = MaxPooling2D(pool_size=(2, 2))(conv3)
    conv4 = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(pool3)
    conv4 = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv4)
    drop4 = Dropout(0.5)(conv4)
    pool4 = MaxPooling2D(pool_size=(2, 2))(drop4)
    conv5 = Conv2D(1024, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(pool4)
    conv5 = Conv2D(1024, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv5)
    drop5 = Dropout(0.5)(conv5)
    up6 = Conv2D(512, 2, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(UpSampling2D(size = (2,2))(drop5))
    merge6 = concatenate([drop4,up6], axis = 3)
    conv6 = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(merge6)
    conv6 = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv6)
    up7 = Conv2D(256, 2, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(UpSampling2D(size = (2,2))(conv6))
    merge7 = concatenate([conv3,up7], axis = 3)
    conv7 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(merge7)
    conv7 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv7)
    up8 = Conv2D(128, 2, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(UpSampling2D(size = (2,2))(conv7))
    merge8 = concatenate([conv2,up8], axis = 3)
    conv8 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(merge8)
    conv8 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv8)
    up9 = Conv2D(64, 2, activation = 'relu', padding = 'same', kernel_initializer = '
     he_normal')(UpSampling2D(size = (2,2))(conv8))
    merge9 = concatenate([conv1,up9], axis = 3)
    conv9 = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer = '
     he_normal')(merge9)
    conv9 = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer = '
     he_normal')(conv9)
    conv9 = Conv2D(2, 3, activation = 'relu', padding = 'same', kernel_initializer = '
     he_normal')(conv9)
    conv10 = Conv2D(1, 1, activation = 'relu',padding = 'same', kernel_initializer = '
     he_normal')(conv9)
    merge10 = concatenate([conv10,inputs], axis = 3)
    conv11 = Conv2D(1, 3, activation = 'sigmoid',padding = 'same', kernel_initializer = '
     he_normal')(merge10)
    model = Model(inputs = inputs, outputs = conv11)
    model.compile(optimizer = SGD(learning_rate= 0.0001, momentum=0.99),loss = '
     mean_squared_error', metrics =['accuracy']);
    return model
```

Cod. 4.9: Implementation of the CNN model in Python

Using the ASTRA toolbox, starting from the sinograms created by measurements without limitations, it was possible to obtain three different ranges of scanning considering a limited angle measurement. For a better interpretation of the results, have been created 350 reconstruction images from the limited sinograms subdivided into three different categories:

- *Train set*: 300 images used for train the CNN;
- *Test set*: 25 images used for testing the solidity of the CNN;
- *Validation set*: 25 images used for the validation, useful for the loss' compute.

The division process was done using a simple script, see lis. (4.10). Initially, a source CSV file was created to collect the entire dataset. Then, exploiting the *scikit-learn* module, a random split was performed using the function *train_test_split*. In this way was possible to divide the images into *train*, *test* and *validation* sets based on the considerations made before. After all, different CSV documents which contain the files of every set, were created. They will be used in the file's recalling during the training phase of the network.

```
import pandas as pd
from sklearn.model_selection import train_test_split
import os

dataset = '/home/ubuntu/TESI GERACE/unet-master/dataset/namefile.csv'
data = pd.read_csv(dataset)

x_train, x_test = train_test_split(data, test_size= 50, train_size=300)

#creazione file validation
x_test, x_val = train_test_split(x_test, test_size = 0.5)
x_train.to_csv('/home/ubuntu/TESI GERACE/unet-master/dataset/x_train_90.csv', index =
    False)
x_val.to_csv('/home/ubuntu/TESI GERACE/unet-master/dataset/x_val_90.csv', index = False)
x_test.to_csv('/home/ubuntu/TESI GERACE/unet-master/dataset/x_test_90.csv', index = False)
```

Cod. 4.10: Dataset division script.

Successively, the main script was compiled considering the file subdivision and the correspondence with the a priori information provided by the groundtruth files obtained from the unlimited sinograms. In this way, the training phase of the network can begin. Starting with the creation of empty python's arrays, reading the CSV files created in lst.(4.10) and fill up the right one considering the same source file for the management of the reconstruction files and the originals maintaining the correspondence between them.

```
#Creation of arrays for a better management of the files
filenames_data = [];  #array for the reconstruction data
filenames_og = [];    #array for the original data
filenames_val = [];   #array for the validation data
filenames_pred = [];  #array for the test data

from csv import reader
#For cycle useful to append file names in the exact order

inputfile = open('/home/ubuntu/TESI GERACE/unet-master/dataset/x_train_90.csv','r')
csv_reader = reader(inputfile)
header = next(csv_reader)
for row in inputfile:
    place = row
    place = place.strip()
    filenames_data.append(place)
    filenames_og.append(place)
```

Cod. 4.11: Array's loading example. Here we have the fill up of the original and reconstruction arrays. For the others arrays we just need to change the inputfile and the name of the final variable.

Proceeding with the design and in accordance with the division of the dataset, see in (2.4.1), the entire *test set* was stored into a specific variable. So, during the training phase, the quality of our network can be verified viewing the intermediate results and monitoring the situation in case of error or unsatisfactory results. Considering the *filename_pred* array containing the file's name of the test set's component, the loading of the image into a multidimensional array, exploiting the *scipy.io* package with *loadmat* module, can be done.

```
d=512                                  #image dimension
z=len(filenames_pred)                  #number of file in testset
filenames_pr = np.zeros((z,d,d,1))     #array of zeros

for i in range(z):
    mat = loadmat('/home/ubuntu/TESI GERACE/unet-master/dataset/SART90_NO/test/' +
     filenames_pred[i])
    img = mat['rec']
    filenames_pr[i,:,:,0] = img
```

Cod. 4.12: Image loading script.

The processing of *train and validation sets* is different with respect to the first one. Due to a limited amount of resources, the physical machine can not handle the volume of data provided to the CNN input. To resolve this issue a particular script that helps us to divide the datasets into portions with the dimension of *batch size* variable and provide these in the input of the CNN avoiding the *out of memory problem* was considered. To perform this operation a *Custom Keras Generator* was implemented, see cod. (4.13), which is composed of 3 steps:

- *Element initialization,*
- *Batch's length computing,*
- *Test and ground-truth images' loading.*

The fundamental step is to provide the exact correspondence between the input images and the ground-truth images avoiding computational problems.

```
class My_Custom_Generator(utils.Sequence) :

   def __init__(self, image_data,image_og, batch) :
    self.image_filenames_data = image_data
    self.image_filenames_og = image_og
    self.batch_size = batch;

   def __len__(self) :  #compute the number of samples in each batch
        return (np.ceil(len(self.image_filenames_data) / float(self.batch_size))).astype(np.int)

   def __getitem__(self, idx) :  #operation for every batch

        batch_x = self.image_filenames_data[idx * self.batch_size:(idx + 1) * self.batch_size]  #filenames noisy and gt data
        batch_y = self.image_filenames_og[idx * self.batch_size:(idx + 1) * self.batch_size]

        x = np.zeros((self.batch_size, image_size, image_size, 1),np.float)
        y = np.zeros((self.batch_size, image_size, image_size, 1),np.float)
        for i in (range(0,self.batch_size)):
                mat = loadmat('/home/ubuntu/TESI GERACE/unet-master/dataset/SART90_NO/'+batch_x[i])
                if(mat is not None):
                    img = mat['rec']
                    x[i,:,:,0] = img
                    mat_2 = loadmat('/home/ubuntu/TESI GERACE/unet-master/dataset/og_data/'+ batch_y[i])
                    img_2 = mat_2['img']
                    y[i,:,:,0] = img_2
```

Cod. 4.13: Keras generator for the input management.

After this preparatory work, all the data were provided to the CNN recalling the Keras generator function and imported the CNN model. Inside a *for cycle* were implemented the *model.fit_generator* function providing in input:

- *Generator*: recalling the Keras generator containing the file names' array of training data and the original one.
- *Validation_data*: recalling the Keras generator containing the file name's array of validation data.
- *Number of epochs*: in this case is equal to 1 due to the for cycle, in this way it possible to works with one epoch at time limiting crash problem or applying the early stop avoiding *overfitting*.
- *Verbose*: graphical information useful for a better live interpretation of the progress.

The parameters' results of the *model_generator* function were stored and used for better visualization of the data. Moreover, for every epoch, the model and the predictions are saved providing all the parameters, weights and bias which compose the CNN, and the expected outcome of the network in a particular epoch, see the cod. (4.14).

```
image_size = 512    #input size definition
batch_size = 1      #input batch size

#recall of keras generator
my_training_batch_generator = My_Custom_Generator(filenames_data,filenames_og, batch_size)
my_validation_batch_generator = My_Custom_Generator(filenames_val,filenames_val_og,
     batch_size)

#importing of model and fundamentals packages
import math
from model import *
from tensorflow.keras.models import save_model, load_model
from scipy.io import loadmat
import tensorflow as tf
from scipy.io import savemat

model=unet()
epoch = 150     #number of epoch for the training phase

#reloading of an eventual presaved model
model = load_model('/home/ubuntu/TESI GERACE/unet-master/model_save.h5')

#for cycle for the training phase
for k in range(1,epoch+1):
    print('Epoch {0} of {1}'.format(k,epoch))
    hist = model.fit_generator(generator=my_training_batch_generator,
                               validation_data =my_validation_batch_generator,
                               epochs=1 ,verbose = 1)
    predictions = model.predict(filenames_pr)
    model.save('model_save3d.h5')
    savemat('prediction.mat', {'pr':predictions})
```

Cod. 4.14: Training code.

4.2 Multi Slice Denoising Case

4.2.1 Simulated CT data generation for multi slice denoising case

In order to implement a multislice denoising, spatial information of each slice must be known. Unfortunately this information is not present in the TCIA dataset, so a new dataset had to be considered. This dataset was provided by hospital A.O. San Carlo Borromeo in Milano to simulate the limited-angle TCT projection data to demonstrate the performance of our research work. In fig. (4.7) are shown the images of the considered dataset.

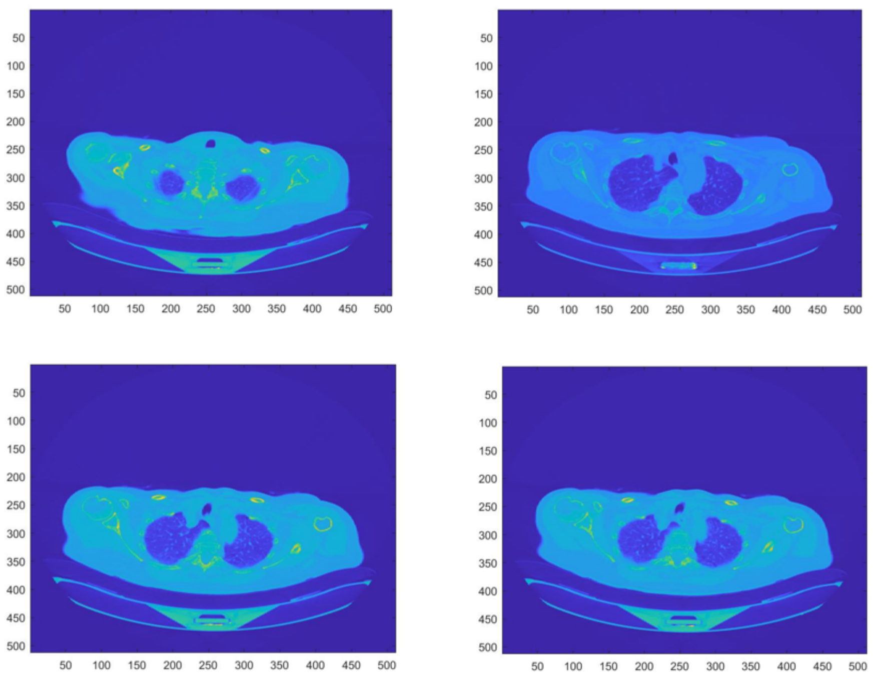


Fig. 4.7: Example of images extracted from the dataset.

The considered data set have a dimension of 512x512 pixels. The measurement were done applying the same volume and projection geometries considered in the Single Slice case. The noisy result, that can be seen in fig. (4.8), was obtained by apply the SART reconstruction algorithm, taken into account an angle range of (0,120°) for the projection process.

Following the same approach as before is also possible to analyse the performance of the entire process by computing the PSNR and SSIM for the various algorithms for each range of measurement. In the tab. (4.4) and tab. (4.5) are shown the PSNR and SSIM values for each range highlight, in agreement with [54], a higher value for the SART algorithms with respect the SIRT and also a higher value for the range with a higher range value .

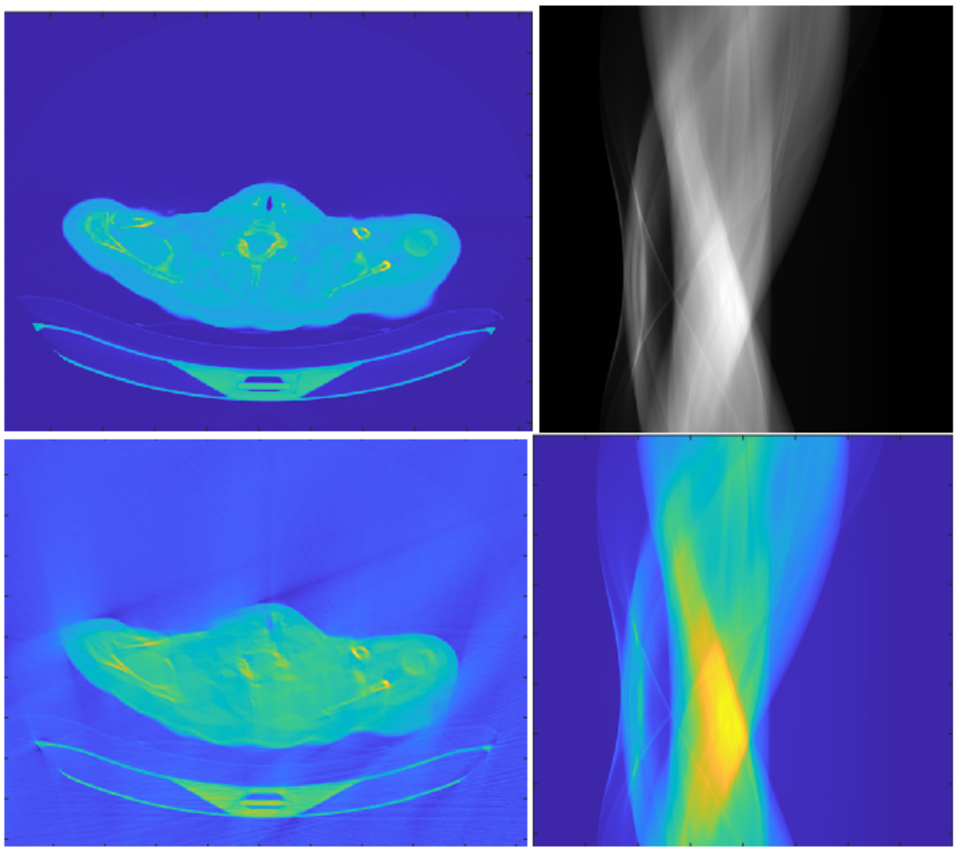


Fig. 4.8: In the first row-original image and grey scaled sinogram, in the second row- reconstructed image and sinogram.

PSNR (dB)		
Range (degree)	SART	SIRT
0-90°	-42,6137	-53,2635
0-120°	-37,914	-49,8115
0-150°	-34,5684	-45,1639

Table 4.4: Comparison between SART and SIRT algorithms over different ranges.

SSIM		
Range (degree)	SART	SIRT
0-90°	0,1777	0,0933
0-120°	0,2153	0,1433
0-150°	0,2295	0,2099

Table 4.5: Comparison between SART and SIRT algorithms over different ranges.

4.2.2 Convolutional Neural Network implementation - Multi slice Case.

In this section is provided the implementation of the Convolutional Neural Network in the multislice case. A complete analysis of the modifications made over the U-net model are reported, providing all the information useful for the management and the creation of the datasets which are composed by slices of different dimensions. In this way it is possibile to realize the denoising operation of an image exploiting a higher amount of data by considering the neighbor images. Starting from the model in [64], it was modified considering a variations on the first layer where it considers a different input dimension based on the number of slices taken into account, the output provided is always in one dimension, see cod. (4.15). In fig. (4.9) there is the graphical representation of the CNN, notice the change in the dimension's input. This change is applied in order to perform the denoising process over the central image of the multislice array, increasing the information considered in the computation with the aims of enhancing the denoising process.

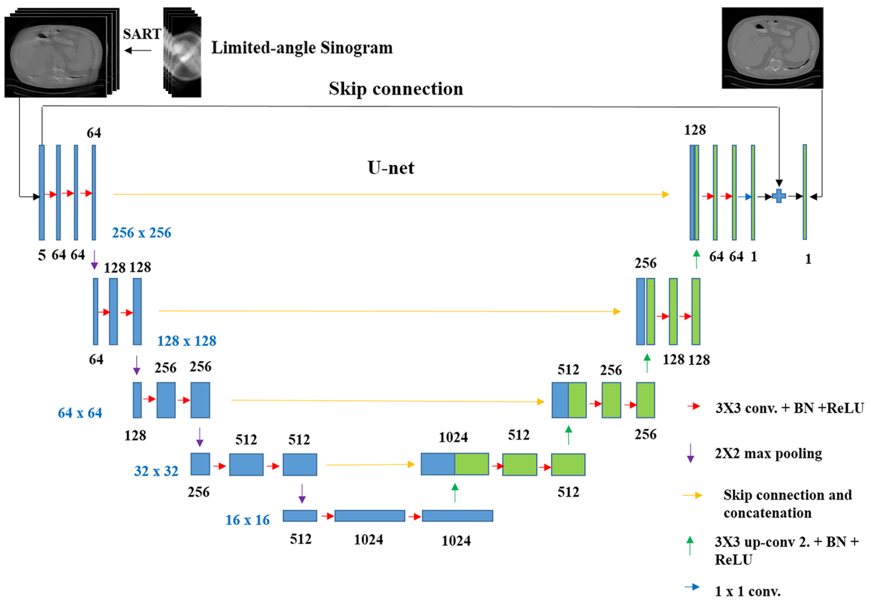


Fig. 4.9: Modified U-net, in this example, the input of the network uses 5 slices. The denoising is performed on the central image of the array, for the comparison is used the corresponding ground-truth image.

```
def unet(pretrained_weights = None,input_size = (512,512,k)):
inputs = Input(input_size)
    conv1 = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(inputs)
    conv1 = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv1)
    pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)
    conv2 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(pool1)
    conv2 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv2)
    pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)
```

```python
    conv3 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(pool2)
    conv3 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv3)
    pool3 = MaxPooling2D(pool_size=(2, 2))(conv3)
    conv4 = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(pool3)
    conv4 = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv4)
    drop4 = Dropout(0.5)(conv4)
    pool4 = MaxPooling2D(pool_size=(2, 2))(drop4)
    conv5 = Conv2D(1024, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(pool4)
    conv5 = Conv2D(1024, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv5)
    drop5 = Dropout(0.5)(conv5)
    up6 = Conv2D(512, 2, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(UpSampling2D(size = (2,2))(drop5))
    merge6 = concatenate([drop4,up6], axis = 3)
    conv6 = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(merge6)
    conv6 = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv6)
    up7 = Conv2D(256, 2, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(UpSampling2D(size = (2,2))(conv6))
    merge7 = concatenate([conv3,up7], axis = 3)
    conv7 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(merge7)
    conv7 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv7)
    up8 = Conv2D(128, 2, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(UpSampling2D(size = (2,2))(conv7))
    merge8 = concatenate([conv2,up8], axis = 3)
    conv8 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(merge8)
    conv8 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv8)
    up9 = Conv2D(64, 2, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(UpSampling2D(size = (2,2))(conv8))
    merge9 = concatenate([conv1,up9], axis = 3)
    conv9 = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(merge9)
    conv9 = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv9)
    conv9 = Conv2D(2, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv9)
    conv10 = Conv2D(1, 1, activation = 'relu',padding = 'same', kernel_initializer = 'he_normal')(conv9)
    merge10 = concatenate([conv10,inputs], axis = 3)
    conv11 = Conv2D(1, 3, activation = 'sigmoid',padding = 'same', kernel_initializer = 'he_normal')(merge10)
    model = Model(inputs = inputs, outputs = conv11)
    model.compile(optimizer = SGD(learning_rate= 0.0001, momentum=0.99),loss = 'mean_squared_error', metrics =['accuracy']);
    return model
```

Cod. 4.15: Implementation of the CNN model in Python for the multislice case.; can be seen the difference in the input size

For this method, a different kind of management and a different kind of data must be considered. The dataset employed is composed by a series of images that have a spatial reference in order to know the exact location of every image and select the right neighbor. The creation of the slices is performed take into consideration an array of images, with a defined dimensions that depends on the number of slices considered where the image in the centre position is the reference data where is performed the denoising. For a better handling of the data, the input considered by the CNN is an array of arrays that contains the noisy and the ground-truth images, in this way are avoided errors in the recalling phase. Starting form the creation of array of zeros with the dimension of the slice, the data loading are performed sequentially, obtaining two variable called *noisy* and *gt* which contains the images. Then a *if loop control* happens in order to save all the data in the final variable called *dato_x.mat* which is used as a input in the CNN. The cod. (4.16) shown how the generation of these data are performed.

```
%creation of arrays
noisy = zeros(512,512,k);
gt = zeros(512,512,k);
a = 1;
c = 1;
for i = 700:900
    load(strcat("/Users/ange/Desktop/TESI/og700/",int2str(i),".mat"))  %loading of ASTRA
     reconstructed images
    load(strcat("/Users/ange/Desktop/TESI 2/sart_700_900/",int2str(i),".mat")) %loading of
      the original images
    noisy(:,:,a) = rec;
    gt(:,:,a) = img;
    x = [int2str(i)];
    %supervision loop useful for saving the variable
    if(a==k)
        save(strcat("/Users/ange/Desktop/TESI/3D CASE/5 slice/dato_",int2str(k),".mat"),'
     noisy','gt','label')
        c = c+1;
        a = 0;
    end
    a=a+1;
end
```

Cod. 4.16: Slices genertion code

In these assuptions, where firstly replicated the task made in the section 4.1, in other to have a starting point and highlight the improvement derived from the consideration of a multislice. A congruence in the confrontation of the datasets results must be performed for this reason, the same data used as a *test set*, must be considered in every dataset. Also the creation of these slices reduce drastically the dimension of the dataset and consequently reduce the dimension of the *train set*, in order to limit this problem an operation of *Data augmentation* is realized flipping vertically all the images and store them into new variable, see cod. (4.17).

```
for i = 1:40
   load(strcat('/Users/ange/Desktop/TESI/3D CASE/5 slice/dato_',int2str(i),'.mat'));
   for k = 1:5
       x = fliplr(gt(:,:,k));
       y = fliplr(noisy(:,:,k));
       gt(:,:,k) = x;
       noisy(:,:,k) = y;
   end
   c = 40+i;
   save(strcat('/Users/ange/Desktop/TESI/3D CASE/5 slice/dato_',int2str(c),'.mat'),'noisy'
     ,'gt');
end
```

Cod. 4.17: Data augmentation code

For a better interpretation of the results, have been created 200 reconstruction images from the limited sinograms subdivided into three different categories, for the single slice sub-case were considered:

- *Train set*: 180 images used for train the CNN;
- *Test set*: 10 images used for testing the solidity of the CNN;
- *Validation set*: 10 images used for the validation, useful for the loss' compute.

For the multislice case, despite the implementation of the data augmentation, have been considered a smaller number of data respect to the single slice sub-case but the congruence between the test sets was preserved. In thee assumptions were considered two differt type of dimensions:

Three slices

In this case was considered this data division:

- *Train set*: 110 images used for train the CNN;
- *Test set*: 10 images used for testing the solidity of the CNN;
- *Validation set*: 10 images used for the validation, useful for the loss' compute.

Five slices

In this case was considered this data division:

- *Train set*: 60 images used for train the CNN;
- *Test set*: 10 images used for testing the solidity of the CNN;
- *Validation set*: 10 images used for the validation, useful for the loss' compute.

The successive steps are the same proposed in the single case approach, the only difference is the implementation of a for cycle useful for a better allocations of the data in the right position, see cod. (4.18).

```
d=512
z=len(filenames_pred)
filenames_pr = np.zeros((z,d,d,k))
filenames_metric = np.zeros((z,d,d,1))
for i in range(z):
        for k in range(0,k-1):
            mat = loadmat('/home/ubuntu/TESI GERACE/unet-master/dataset/SART90_NO/'+
    filenames_pred[i])
            img = mat['noisy']
            filenames_pr[i,:,:,k] = img[:,:,k]
```

Cod. 4.18: Image loading script for multislice case

The processing of *train and validation sets* is different with respect to the first one. Due to a limited amount of resources, the physical machine can not handle the volume of data provided to the CNN input. To resolve this issue a particular script that helps us to divide the datasets into portions with the dimension of *batch size* variable and provide these in the input of the CNN avoiding the *out of memory problem* was considered. To perform this operation a *Custom Keras Generator* was implemented, see cod. (4.19), which is composed of three steps:

- *Element initialization*,
- *Batch's length computing*,
- *Test and ground-truth images' loading*.

The fundamental step is to provide the exact correspondence between the input images and the ground-truth images avoiding computational problems.

```
class My_Custom_Generator(utils.Sequence) :

  def __init__(self, image_data,image_og, batch) :
    self.image_filenames_data = image_data
    self.image_filenames_og = image_og
    self.batch_size = batch;

  def __len__(self) :  #obtain number of samples in each batch
      return (np.ceil(len(self.image_filenames_data) / float(self.batch_size))).astype(
    np.int)

  def __getitem__(self, idx) :  #operation for every batch

      batch_x = self.image_filenames_data[idx * self.batch_size:(idx + 1) * self.
    batch_size]  #filenames noisy and gt data
      batch_y = self.image_filenames_og[idx * self.batch_size:(idx + 1) * self.
    batch_size]

      x = np.zeros((self.batch_size, image_size, image_size, k),np.float)
      y = np.zeros((self.batch_size, image_size, image_size, 1),np.float)
      for i in (range(0,self.batch_size)):
              mat = loadmat('/home/ubuntu/TESI GERACE/unet-master/dataset/SART90_NO/'+
    batch_x[i])
              img = mat['noisy']
              for a in range(0,k-1):
                  x[i,:,:,a] = img[:,:,a]
              img_2 = mat['gt']
              y[i,:,:,0] = img_2[:,:,k/2]

      return x,y
```

Cod. 4.19: Keras generator for the input slices management

In the cod. (4.19) can be seen the differences applied in the dimensions of the images. The *variable* x, with a dimension k contains the slices of noisy images, the *variable* y with a 1-dimension, contains the original image correspondent to the noisy images in the centre of the array. In this way er have the right sequence of images provided in the CNN input. The parameters' results of the *model_generator* function were stored and used for better visualization of the data. Moreover, for every epoch, the model and the predictions are saved providing all the parameters, weights and bias which compose the CNN, and the expected outcome of the network in a particular epoch, see the cod. (4.20).

```
image_size = 512    #input size definition
batch_size = 1      #input batch size
#recall of keras generator
my_training_batch_generator = My_Custom_Generator(filenames_data,filenames_og, batch_size)
my_validation_batch_generator = My_Custom_Generator(filenames_val,filenames_val_og,
    batch_size)
#importing of model and fundamentals packages
import math
from model import *
from tensorflow.keras.models import save_model, load_model
from scipy.io import loadmat
import tensorflow as tf
from scipy.io import savemat

model=unet()
epoch = 150     #number of epoch for the training phase
#reloading of an eventual presaved model
model = load_model('/home/ubuntu/TESI GERACE/unet-master/model_save.h5')
#for cycle for the training phase
for k in range(1,epoch+1):
    print('Epoch {0} of {1}'.format(k,epoch))
    hist = model.fit_generator(generator=my_training_batch_generator,
                              validation_data =my_validation_batch_generator,
                              epochs=1 ,verbose = 1)
    predictions = model.predict(filenames_pr)
    model.save('model_save3d.h5')
    savemat('prediction.mat', {'pr':predictions})
```

Cod. 4.20: Training code.

Chapter 5

Results

This thesis work proposes and evaluates two different methods for enhancing the tomographic reconstructions of acquisitions obtained in a limited angle range setup. In the first section, the result of single slice case reconstruction process are reported proving the quality enhancement that could be obtained using this denoising technique. In the second section, the results obtained from a different method, using a different dataset, are provided. In particular, the new method takes into consideration more than one slice as the input with the aim of improving the enhancing performance by taking into consideration a larger amount of information.

5.1 Single slice case

This subsection contains the results of CNN's training exploiting a dataset provided by the TCIA. Starting from the original image, obtained by a sinogram containing all the possible projections, it was possible to reconstruct, with limited angles, noisy images used as the input of the CNN. The denoising capability of the CNN are evaluated in comparing its results with the original images in the database assumed as groundtruth. All the different developed model, each one related to limited angle ranges, were tested on the same dataset in order to compare the results obtained. Denoising results present in [54] will also be reported for comparison.

SART 150

This section exposes the most straightforward and the best possible scenario in our dissertation. The high range of measurement produces a sinogram with a high number of projections and, consequently, an image with a low level of noise and artifacts, providing an excellent output from the CNN in terms of image quality, loss value, and PSNR/SSIM values.
Despite the high value of *learning rate*, the network provides a good train and validation loss pattern, see fig.(5.1), with lines following each other without significant fluctuations. In terms of denoising output, the CNN output shows little difference between the original and reconstructed images. In fig.(5.2) there is an example of

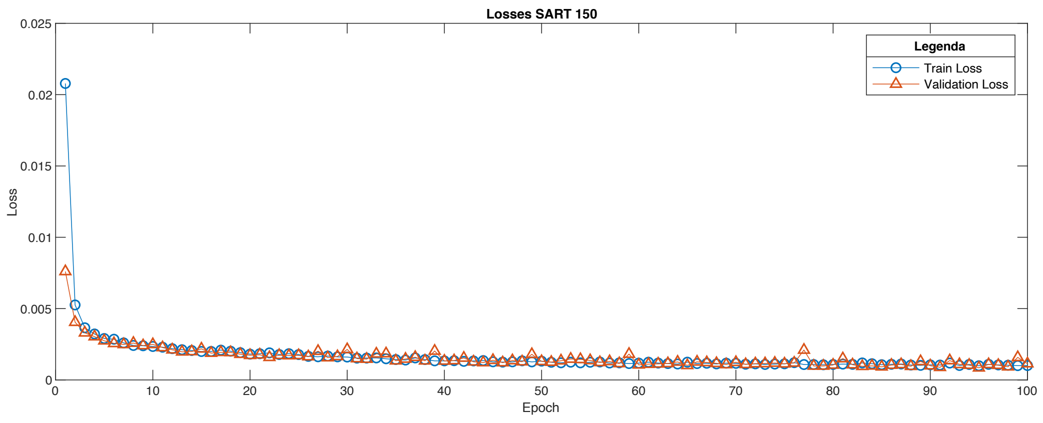


Fig. 5.1: Losses patterns for SART in range 0-150°. In blue, is represented the training loss and in red, the validation loss.

the CNN outcome. On the left is present the *original image*. On the center, there is the *reconstructed image* obtained with ASTRA and can be seen some artifacts and blurred zone due to the reconstruction phase, on the right, there is the *CNN output* without noise and artifacts having a good correspondence of values.

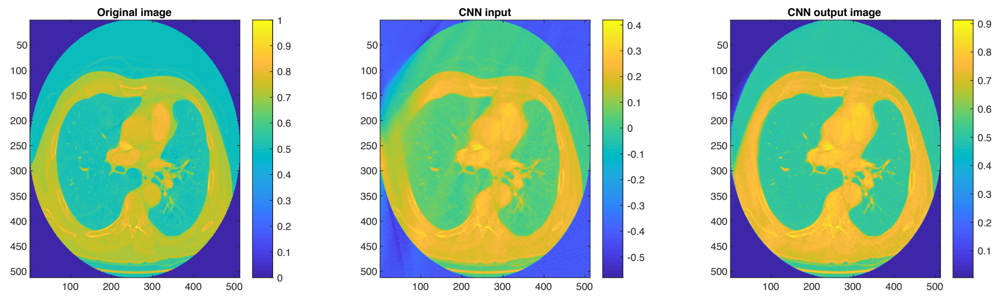


Fig. 5.2: Example of CNN result, from left to right: Original image, reconstruction SART with range 0-150°, CNN output.

For a better view, the graphics in fig. (5.3) help us quickly analyze and better compare the outcome obtained by the proposed method.

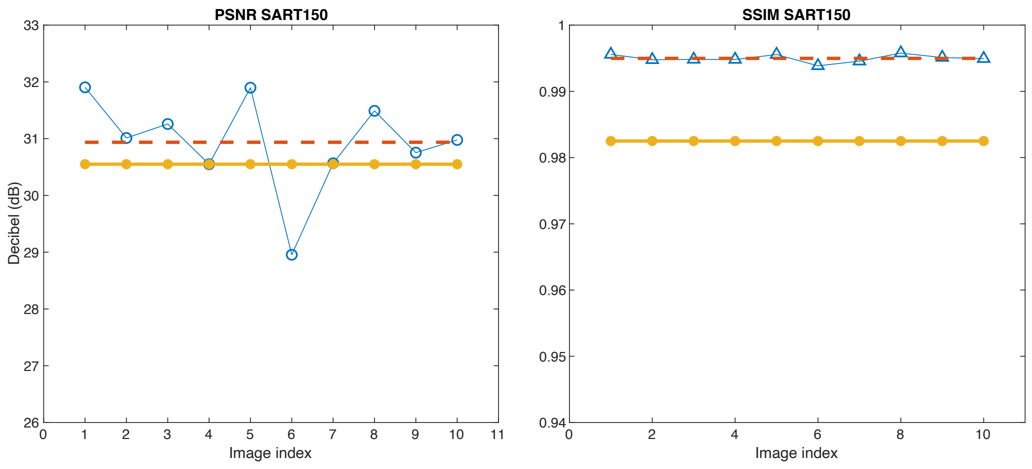


Fig. 5.3: PSNR and SSIM patterns for range 0-150°.The blue line represents the PSNR/SSIM value for the i-th test image; the yellow line represents the average value reported in [54], the red line represents the parameters' average value

SART 120

This section reports results obtained on data acquire with 0-120 angle range. Due to the reduced angle of measurement, the sinograms have fewer views in respect to the previous case, which produces an ASTRA reconstruction image with much more artifacts and noise outside and inside of the subject. Although this new setting, the CNN outcome is robust and stable, providing a sound reconstruction and a high-grade train loss pattern, while, for the validation loss, there are some fluctuations, see fig.(5.4).

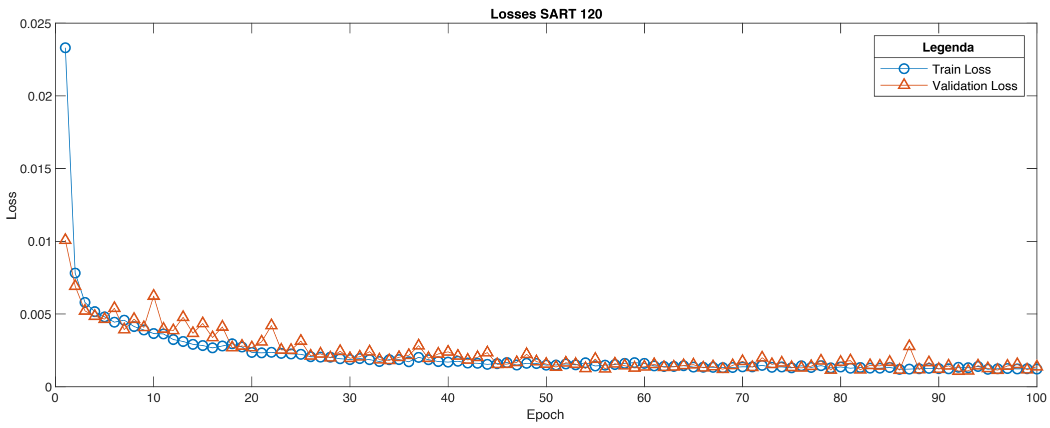


Fig. 5.4: Losses patterns for SART in range 0-120°. In blue, is represented the training loss and in red, the validation loss.

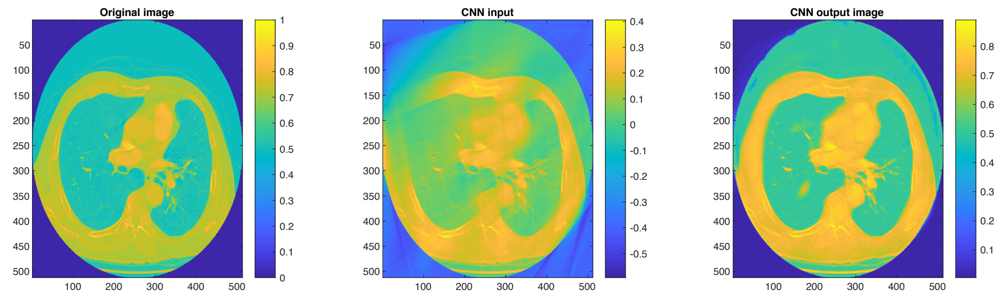


Fig. 5.5: Example of CNN result, from left to right: Original image, reconstruction SART with range 0-120°, CNN output

Visually comparing with the original images, the CNN output reconstruction provides, in a few ares of the reconstructed image, points of 'misconception' value and the validation loss fluctuations can be related to this particular result. See fig.(5.5). In terms of PSNR and SSIM, there is some difference concerning the previous case. In terms of SSIM, there is the best possible result with value, for every image, more significant than the reference values considered in [54].

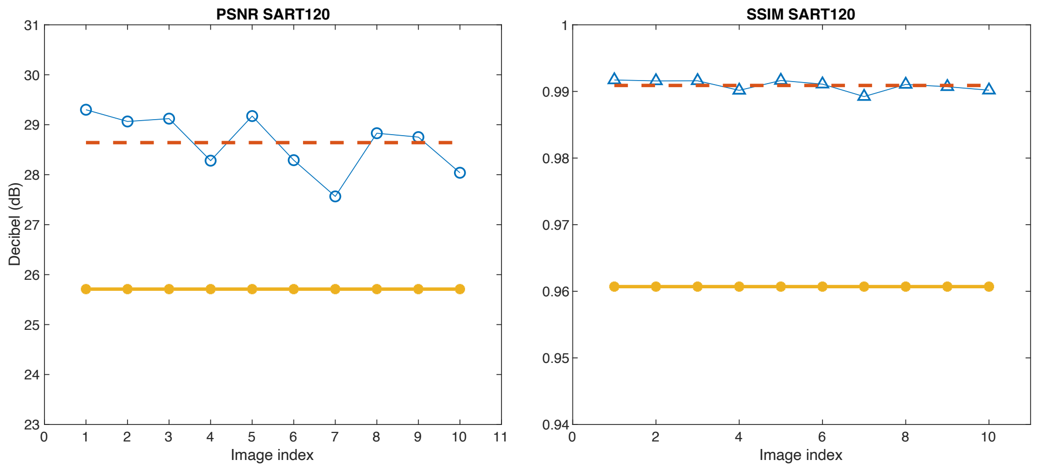


Fig. 5.6: PSNR and SSIM patterns for range 0-120°.The blue line represents the PSNR/SSIM value for the i-th test image; the yellow line represents the average value reported in [54], the red line represents the parameters' average value

indicating an excellent reconstruction and a substantial similarity between the images despite the reconstruction's errors, see fig.(5.6)

SART 90

This section reports results obtained considering the most challenging case in our dissertation. The limited range produces a sinogram with a low number of views, and it causes blurring and undefined zone on the relative image. It is tough to understand the details and the entire structure of interest. Nevertheless, CNN's response was optimal, providing an optimal trend of the train and validation losses, see fig. (5.7). Evaluating the results in the matter of PSNR and SSIM parameters is excellent; this is an excellent enhancement because an image obtained with a very limited range of angles can be reconstructed without significant issues. Nonetheless, the average value obtained is far better than the reference, see fig. (5.9).

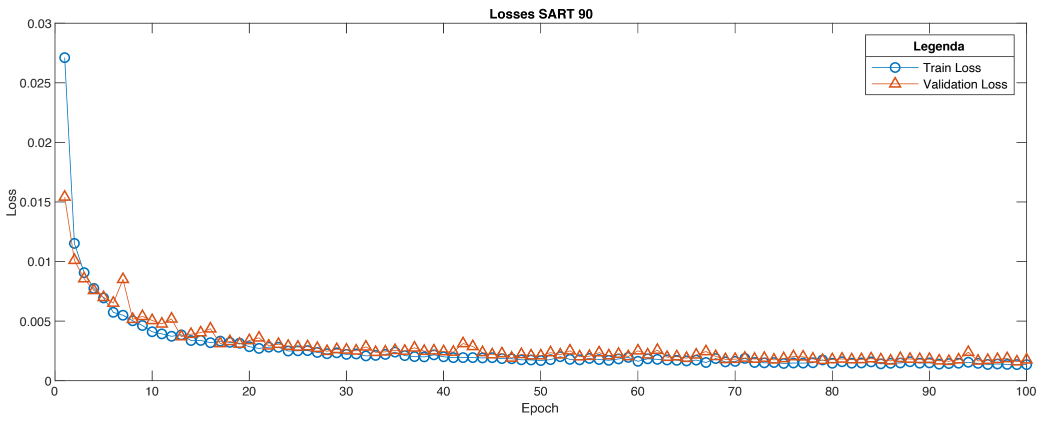


Fig. 5.7: Losses patterns for SART in range 0-90°. In blue, is represented the training loss and in red, the validation loss.

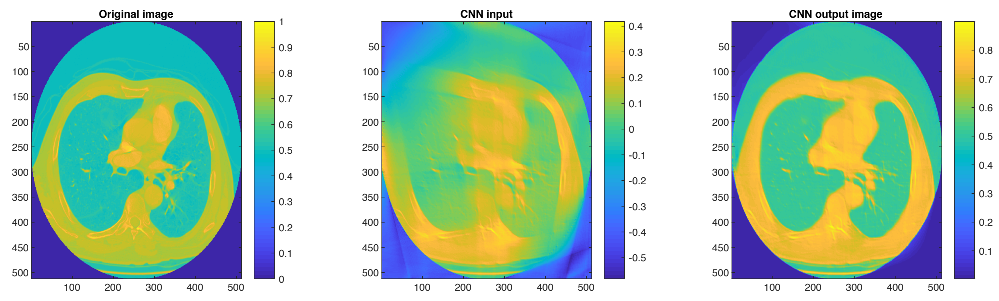


Fig. 5.8: Example of CNN result, from left to right: Original image, reconstruction SART with range 0-90°, CNN output

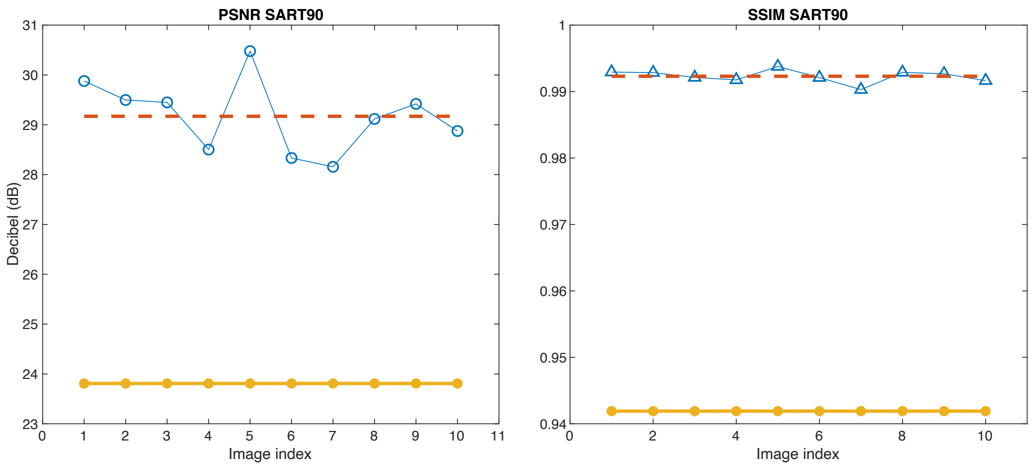


Fig. 5.9: PSNR and SSIM patterns for range 0-120°. The blue line represents the PSNR/SSIM value for the i-th test image; the yellow line represents the average value reported in [54], the red line represents the parameters' average value.

	Range 150		**Range 120**		**Range 90**	
Index	**PSNR (dB)**	**SSIM**	**PSNR (dB)**	**SSIM**	**PSNR (dB)**	**SSIM**
1	31,90	0,9956	29,30	0,9917	29,87	0,9929
2	31,01	0,9948	29,06	0,9916	29,49	0,9929
3	31,25	0,9948	29,12	0,9916	29,45	0,9921
4	30,55	0,9948	28,28	0,9902	28,50	0,9918
5	31,89	0,9956	29,17	0,9916	30,74	0,9938
6	28,95	0,9939	28,29	0,9911	28,33	0,9921
7	30,56	0,9946	27,56	0,9892	28,15	0,9903
8	31,49	0,9957	28,83	0,9910	29,11	0,9929
9	30,75	0,9951	28,75	0,9907	29,42	0,9927
10	30,97	0,9949	28,04	0,9902	28,87	0,9916
Average	30,94	0,9950	28,64	0,9917	29,17	0,9923
Reference	30,55	0.9825	25,71	0,9607	23,80	0,9419
Δ	0,39	0,0125	2,93	0,0310	5,37	0,0504

Table 5.1: Evaluation parameters for SART reconstruction. The terms with a higher value are highlighted.

Table 5.1, contains all the results obtained on a small test set (10 images) extracted from TCIA dataset providing PSNR and SSIM values for each range, the average value for each evaluation metric, the metric results reported in [54], and Positive values of delta corresponds to improvement in respect to the method reported in [54].

Looking at the column "Range 150" in tab. 5.1, the CNN produces a high value of PSNR and SSIM, proving an optimal result in terms of structures and similarities. On average, it makes higher PSNR and SSIM values concerning the reference. Those few values under the average value are due to the image's zones where there is a higher presence of details whose the network can not identify in the limited number of epochs considered. Despite this particular circumstance, the overall reconstruction is done perfectly. Highlight the column 'Range 120' in tab 5.1, in this specific dataset the presence of the 'misconception' zone, caused by the blurring effect derived by the limited angle reconstruction, creates some discrepancies in the PSNR computation. This can be seen in fig. (5.5) where the reconstruction images shows som difference respect to the original one. PSNR's value was far better for the other images, thus providing an average value of PSNR greater than the value propose in [54].

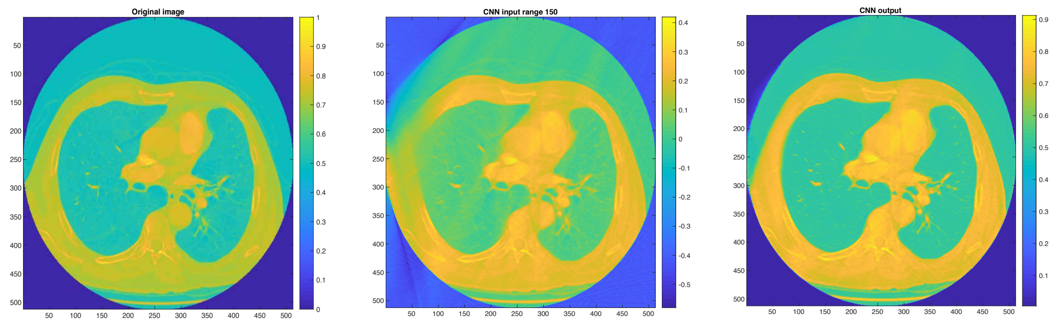


Fig. 5.10: Range 150° - Best case, first table's row.

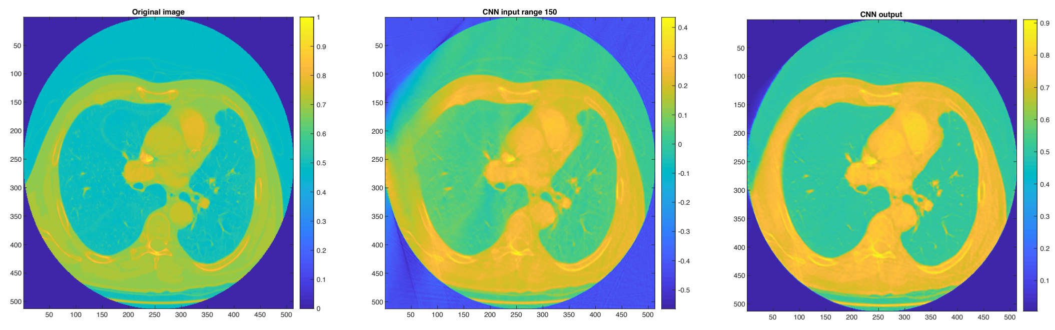


Fig. 5.11: Range 150° - Worst case, sixth table's row.

The section 'Range 90', in tab. 5.1, show significant results with better than the value obtained in [54] for the entire image's set. The few terms under average are due to blurring imposed by the limited angle scanning it leads to a non-optimal reconstruction providing a drop of the parameters.

For a better view and a deep understanding of the results, the images are provided for the best and the worst case for every range. Starting from the range 0-150°, can be seen the difference between the images, see fig.(5.10) and fig. (5.11). The first row presents the best case; in the second row, there is the worst case. The image on the left is the original one; on the center, there is the ASTRA reconstruction and, on the right, the CNN reconstruction. The main difference between these cases is in the details computation; there is a better result for the image with a better definition and a much higher number of details in the original image, leading to a better reconstruction of them.

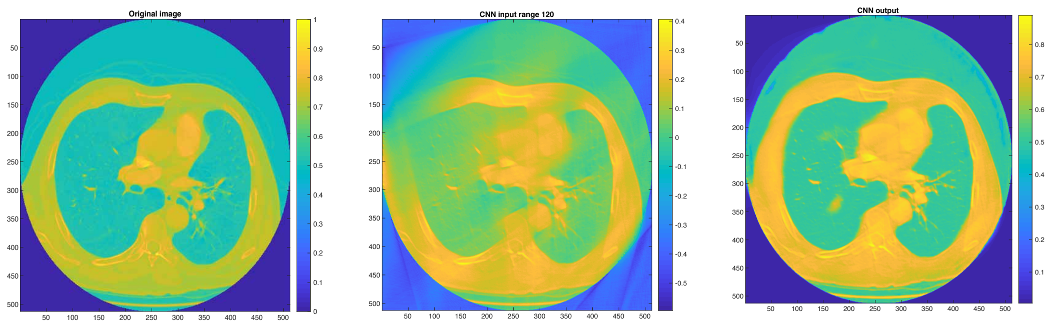


Fig. 5.12: Range 120° - Best case, first table's row.

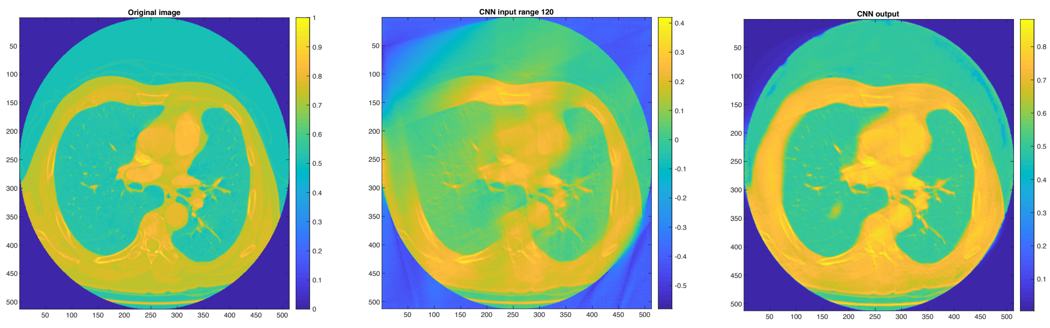


Fig. 5.13: Range 120° - Worst case, tenth table's row.

In the range 0-120°, the difference is amplified due to the limited angle measurement and this lead to some curious results, see fig.(5.12). In the ASTRA reconstruction there are zones of blurring which modify the amplitude value of the pixels mainly in the upper part of the image, this leads to an error in the evaluation as can be seen in the image on the right that present zone of error, mostly on the border. Despite this, the overall image provides a good result and a good definition of details. The worst case provides a good value of PSNR and SSIM, but respect the previous case, the reconstruction presents some holes in the border of the subject due to the noise present in the image that modifies the expected value. This justifies the difference of 1dB in the PNSR and the SSIM term.

Lastly, considering the range 0-90°, the CNN provides impressive results. The noise caused by the very limited range of angles causes a different number of blurred areas with respect to the case 0-120°. Despite this, CNN's response is excellent. In fig. (5.14), the CNN output produces some errors in the central part of the image where the blurring effect causes a non-defined reconstruction of the alveoli. The worst case, see fig.(5.15), shows the main effect of the noise over the reconstruction. The significant difference between the PSNR's values is due to the high blur obtained in the reconstruction. The overall CNN output shows remarkable similarities with the original one, but in the details can be seen a non-defined reconstruction in the central part and a mismatch of reconstruction values can be seen in the upper zone of the CNN output, a yellow stripe is visible, and that evidences a non-optimal reconstruction in this area. Despite this, the higher value available are well-defined, highlights the quality of the entire reconstruction process.

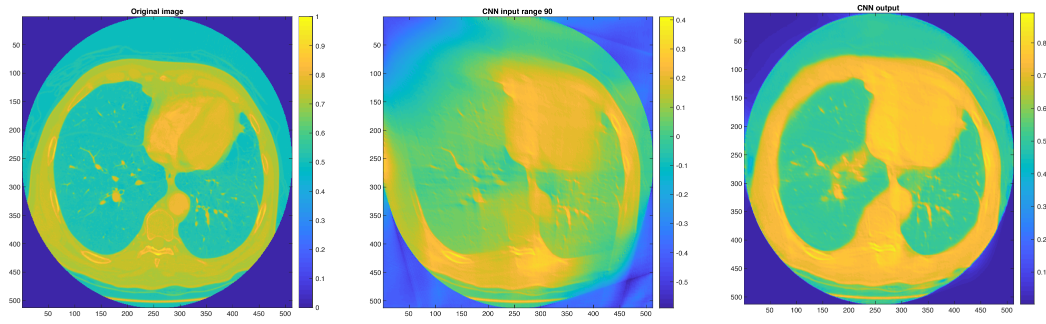


Fig. 5.14: Range 90° - Best case, fifth table's row

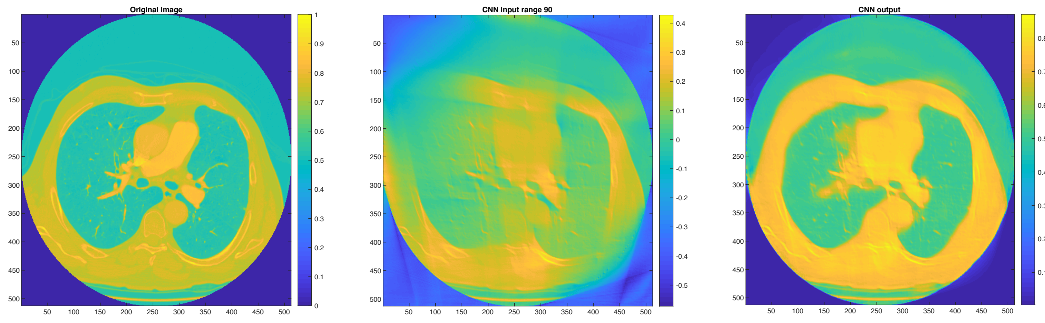


Fig. 5.15: Range 90° - Worst case, seventh table's row

5.2 Multi slice case

This section provide the results obtained with the use of a different dataset. It was considered for the necessity of a spatial reference that can denote the position of every slice to prove that, assuming a multislice in input, it could enhance the CNN's denoising performance. The results for the single slice case, five slices case and three slices case will be shown. This process was implemented using the 0-120° measurement range for each case. Moreover, for a congruent comparision of the data, the same test set containing the same images was considered and the results for each case are reported.

Single slice case

Standard case of reconstruction seen before. This case are considered in order to provide the denoising evaluation as a benchmark of the CNN and consider it as a basis of comparison with the other cases' results. This particular dataset presents some difficulties in the reconstruction. Visually some noise and artifacts do not allow us to understand the details clearly, with a great emphasis on high-value points. In

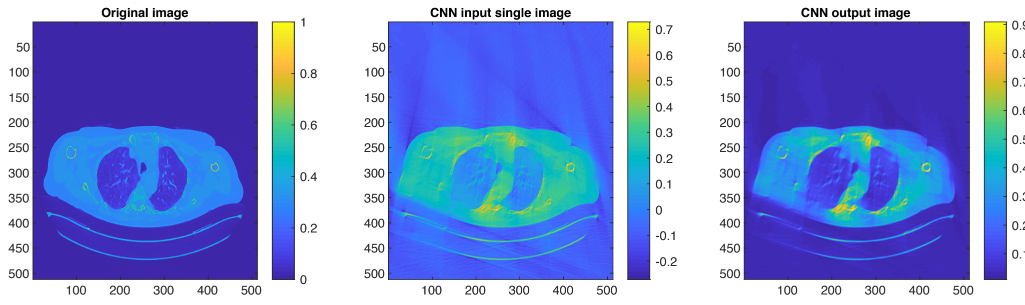


Fig. 5.16: Example of CNN result, from left to right: Original image, reconstruction SART range 0-120° with a single image, CNN output.

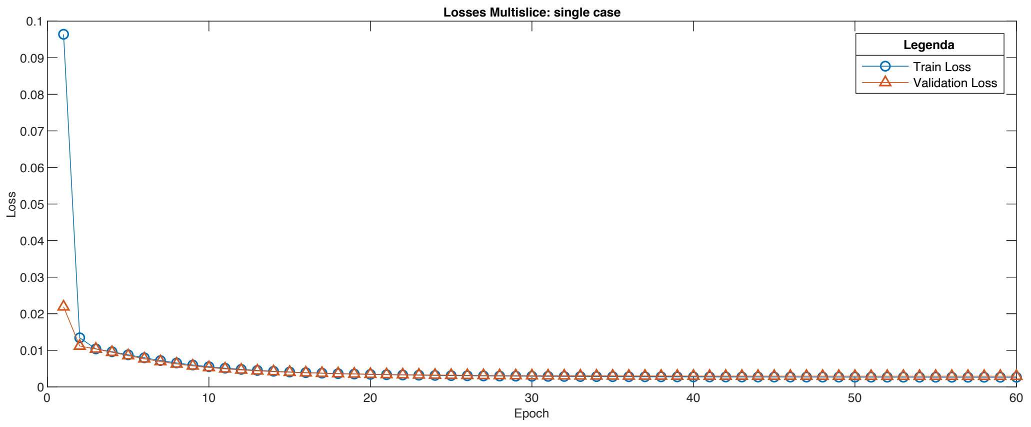


Fig. 5.17: Loss patterns. In blue, is represented the training loss and in red, the validation loss.

the overall consideration, it produces a good outcome, see fig. (5.16). Also, the loss's parameters follow an excellent pattern congruent with the reconstruction's quality, see fig.(5.17).

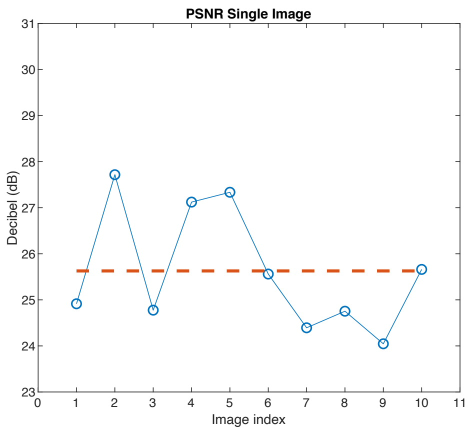


Fig. 5.18: PSNR and SSIM graphics. In blue, PSNR value of i-th test image, in red, the average value.

Three slices case

In this case, in input of the CNN, an array of three ASTRA reconstructed images is considered. This method aims to enhance the amount of data for the denoising of an image including the previous and the following images that, later, are compared with the original one. As a reference for the CNN input, the central image is the one to be compared. Visually, the denoising output obtained can not be clearly evaluated, the results at first sight is very similar to the 'Single Slice Case', see fig. (5.19). Also, in terms of losses, the train and the validation losses have a good trend, demonstrating the quality and the effective operation of the network, see fig.(5.20).

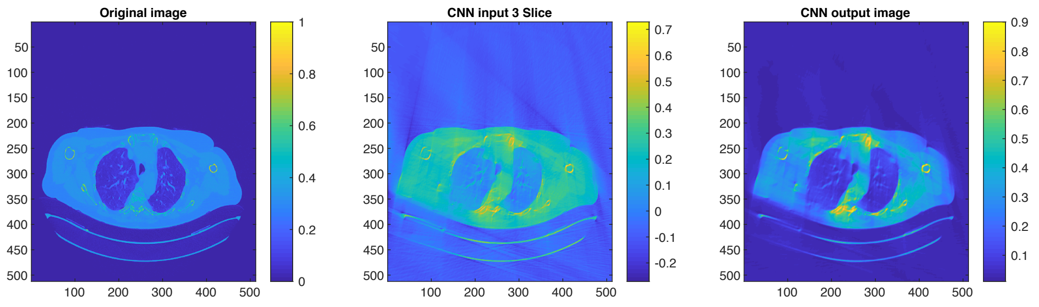
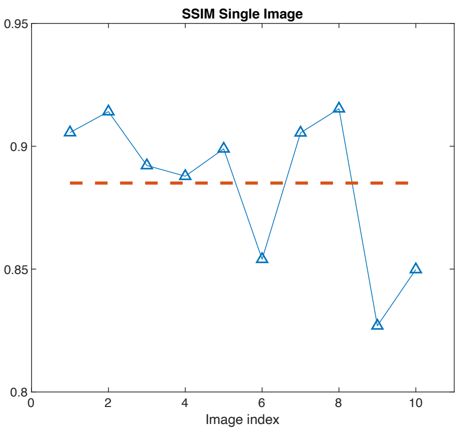


Fig. 5.19: Example of CNN result, from left to right: Original image, reconstruction SART range 0-120° with 3 slices, CNN output.



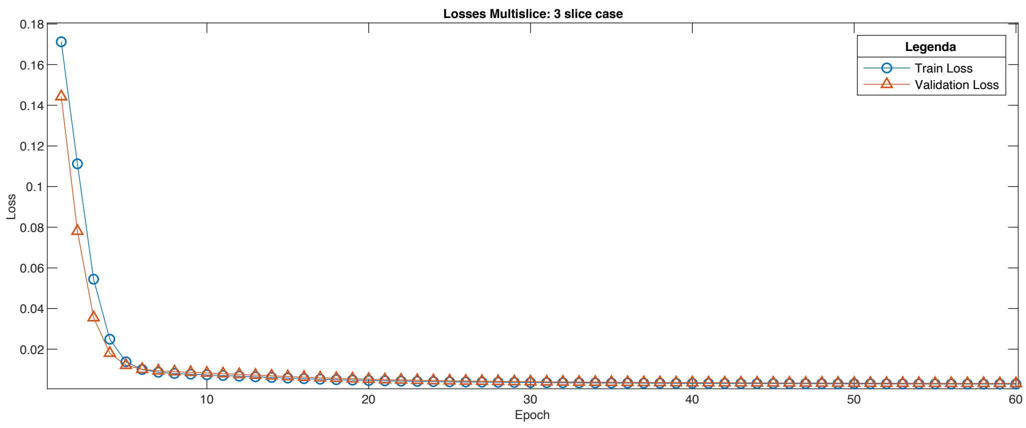


Fig. 5.20: Loss patterns. In blue, is represented the training loss and in red, the validation loss.

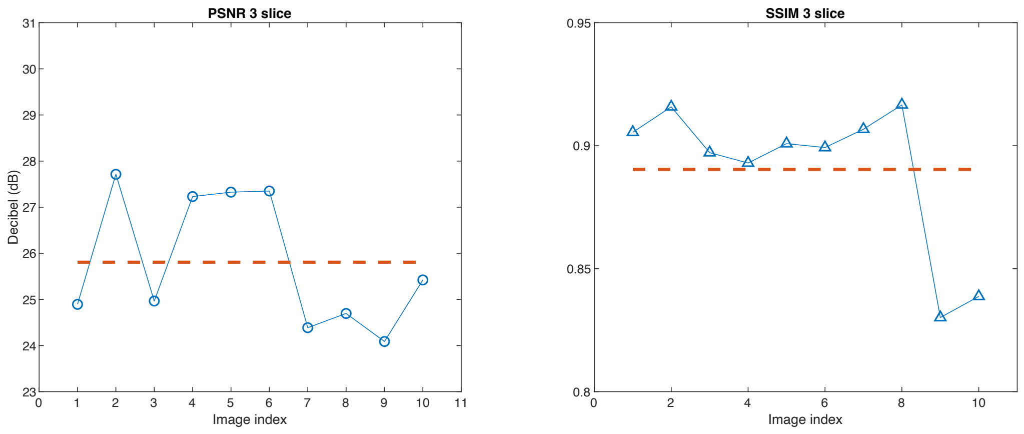


Fig. 5.21: PSNR and SSIM graphics. In blue, PSNR value of i-th test image, in red, the average value.

Five slices case

In this case, in the input of the CNN, an array of five ASTRA reconstructed images is considered. This method aims to enhance the number of data, including the two previous and the two following images, and then compared with the original one. As a reference for the CNN input, the central image is the one to be compared. The denoising output obtained is very similir with respect to the other two cases; in the overall there is a good reconstruction without a clear vision of details. (5.22). Also, graphically, at first sight, some similarity can be seen in the graphics, see (5.24). Despite the non-optimal reconstruction quality, the loss's lines follow a good pattern underlining the proper functioning of the network, see fig. (5.23)

The table 5.2, contains all the results for this particular dataset providing PSNR and SSIM values for every case, the average value for every parameters, the reference value, the difference between the *single case* parameters and the *multislice* parameters contained inside 'Δ *Single case*' section. Looking at column 'Single Case' in tab.

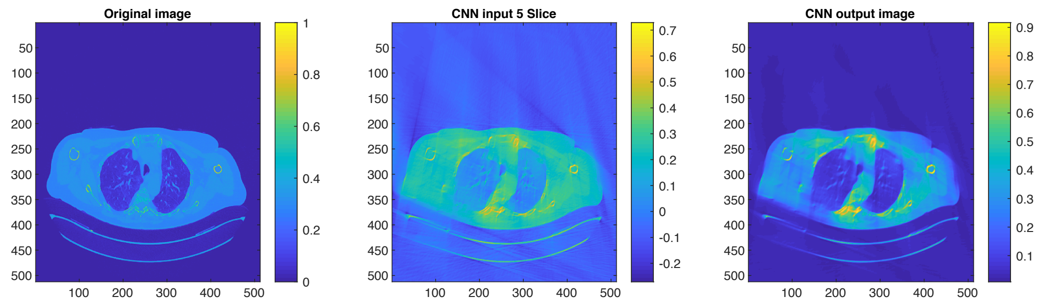


Fig. 5.22: Example of CNN result, from left to right: Original image, reconstruction SART range 0-120° with 5 slice image, CNN output

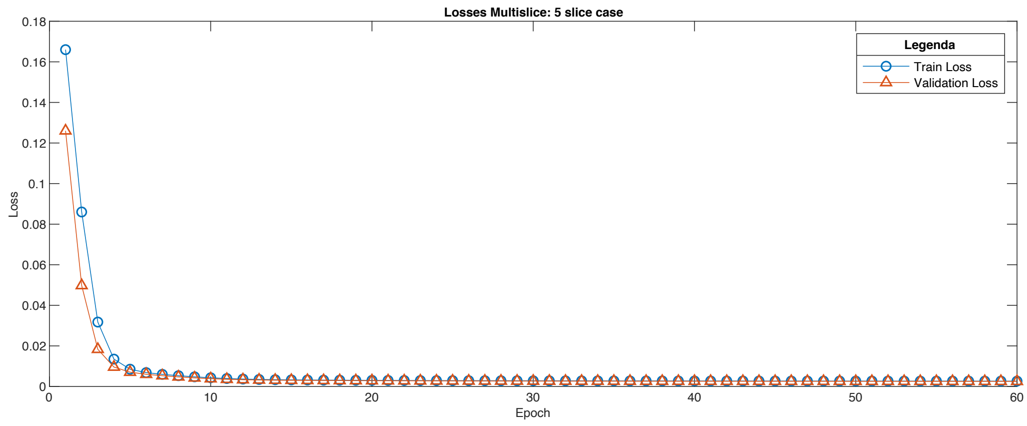


Fig. 5.23: Losses patterns. In blue, is represented the training loss and in red, the validation loss.

(5.2) the physical output's quality is corroborated by good PSNR and SSIM values. These considerations produce good average terms of PSNR and SSIM. The parameters under average are due to the presence of pixels and zones with a high value which, during the ASTRA reconstruction, produces tough uncertainty zones that creates difficulties in the CNN reconstruction, see fig. (5.18). Considering the column '3 Slice Case' comparing one by one terms of each case;can be noticed, in some example, that there is an improvement in the reconstruction evaluation's terms. On average, there is a minimum enhancement of the PSNR and SSIM values. Finally, in the '5 Slice Case' in tab. (5.2) can be seen the results. Considering PSNR and SSIM parameters, on average, is obtained a slightly difference in the average value of PSNR and SSIM respect the 'Single Case' values,and also for the values in the '3 Slice Case'. The results in tab. (5.2) could be associated to a CNN dimension's limitation, so to obtain a good result, an enhancement of the convolutional layers or a different treatment of the data needs to be done, in order to obtain a higher improvement.

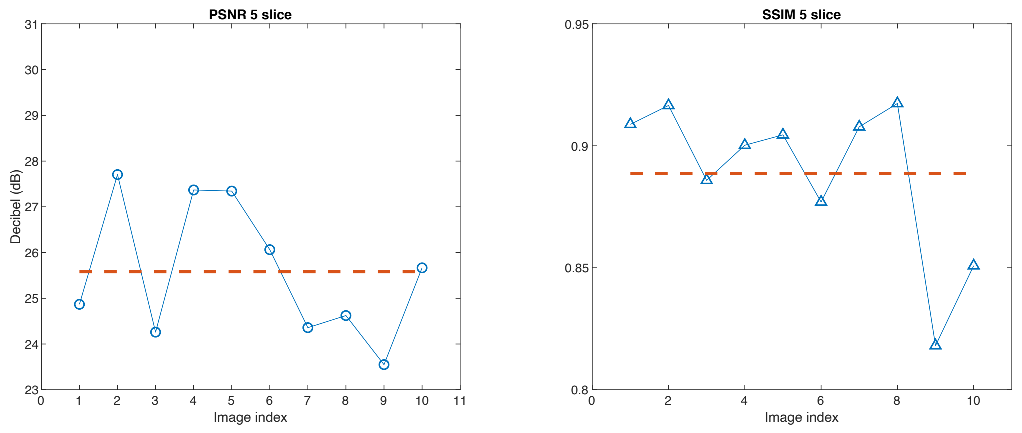


Fig. 5.24: PSNR and SSIM graphics. In blue, PSNR value of i-th test image, in red, the average value.

	Single Case		3 Slice Case		5 Slice Case	
Index	**PSNR (dB)**	**SSIM**	**PSNR (dB)**	**SSIM**	**PSNR (dB)**	**SSIM**
1	24,91	0,9056	24,89	0,9050	24,89	0,9079
2	27,71	0,9141	27,71	0,9158	27,50	0,9105
3	24,77	0,8922	24,96	0,8972	24,63	0,8935
4	27,12	0,8878	27,03	0,8930	27,38	0,8990
5	27,33	0,8990	27,32	0,9008	27,39	0,9041
6	25,56	0,8540	27,35	0,8992	25,89	0,8695
7	24,39	0,9055	24,38	0,9067	24,22	0,9024
8	24,75	0,9153	24,69	0,9166	24,71	0,9183
9	24,04	0,8269	24,10	0,8300	24,00	0,8308
10	25,66	0,8499	25,42	0,8387	25,50	0,8428
Average	25,62	0,8850	25,79	0,8903	25,61	0,8879
Δ Single case	-	-	0,16	0,0053	-0,01	0,0029

Table 5.2: Results of PSNR and SSIM in the multislice case. The terms with a higher value are highlighted

For a better view and a deep understanding of the results, the images for the best and the worst scenario are provided for every case. Starting from the single slice scenario, it produces a difference in the outcomes shown in fig.(5.25) and fig. (5.26). The first row presents the best case. In the second row, there is the worst case. The image on the left is the original one. On the center, there is the ASTRA reconstruction and, on the right, the CNN reconstruction. The overall results of this case are good, and at first sight, there is a substantial similarity between the original and the reconstructed one. In fig. (5.25), the evaluation parameters are affected by the significant presence of noise due to the limited measurement, which can be seen in the ASTRA reconstruction image and by the two minor points in yellow, which highlight the points of maximum in the image. Also, the pixels related to higher values that evidence the arm's bones' presence are clearly reconstructed. For these reasons, there is a high value of PSNR and SSIM. Considering the worst case, see fig. (5.26), the overall subject can be identified, the pixels with the higher value are detected and reconstructed. Also, a non-optimal reconstruction on the left side of the image can be seen due to the blur effect imposed by the measurement process. These produce errors in the main subject and the external object; this motivation causes a low value of PSNR and SSIM.

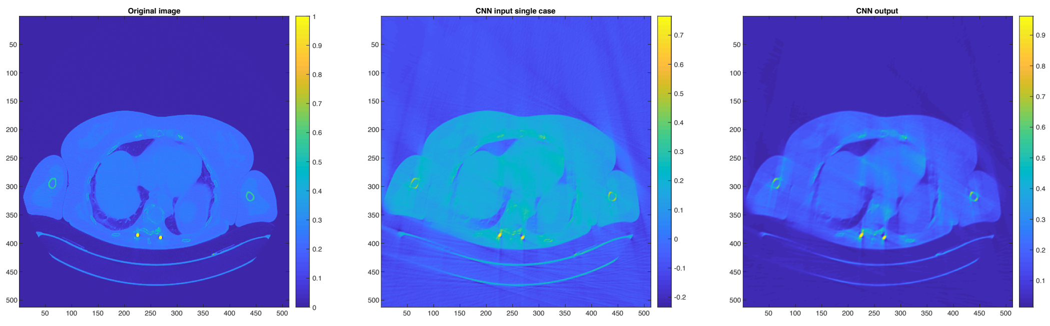


Fig. 5.25: Single slice - Best case, second table's row.

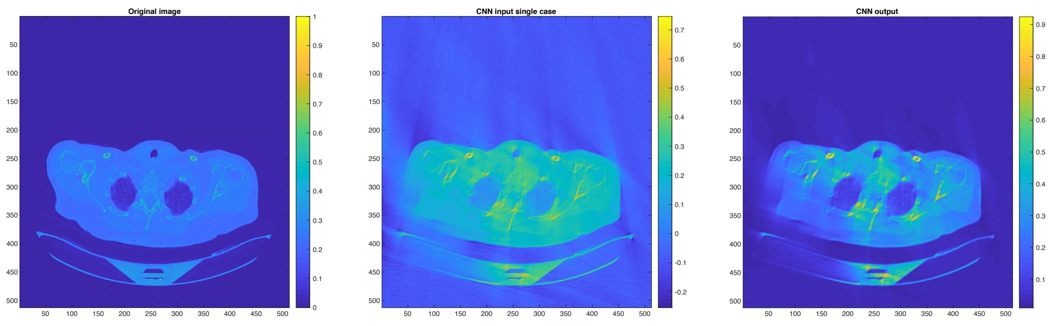


Fig. 5.26: Single slice - Worst case, ninth table's row.

Considering the results for the five slice case, see fig. (5.27) and fig. (5.28), they are very similar to the single slice case. Analyze the tab. (5.2), the only difference can be seen in the SSIM. In the best-case scenario, there is a decrease of parameters, which can be caused by the type of image considered. The slice taken into account has a low amount of pixels with high value, so the use of multiple slices for a better definition does not enhance this particular case. In the worst-case scenario, there is a small increase; the image considered is different from the previous case with a larger presence of higher pixel values, the deployment of multiple slices can enhance the reconstruction quality in this particular type of image. There is a little improvement on the background and the blurred area on the left; these motivations lead to an increment in the SSIM and take into consideration the application of multislice input.

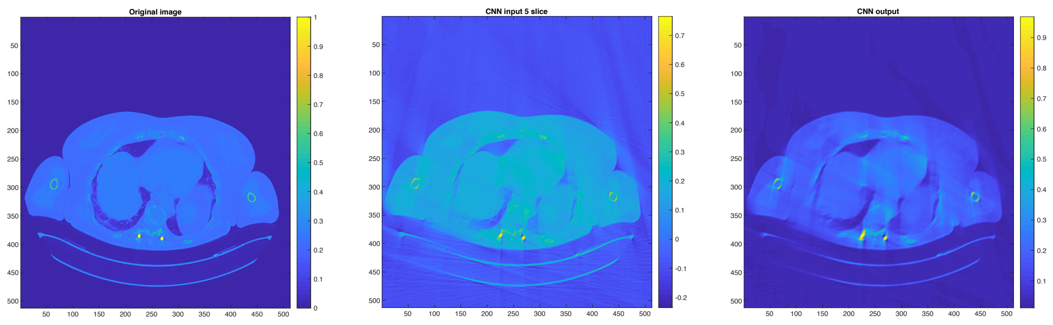


Fig. 5.27: 5 slices - Best case, second table's row.

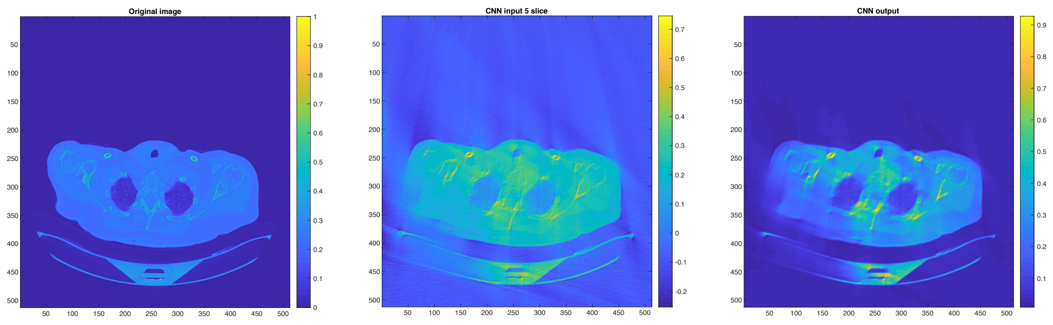


Fig. 5.28: 5 slices - Worst case, ninth table's row.

For the three slices case, the visual result is like the previous cases; the difference can be exploited make a comparison with the value in the tab.(5.2). For all the evaluation parameters, there is a slight increase which confirms the analysis made before. The images obtained in the best case have not been that different. This is due to the low number of pixels with a high value that the network can identify and reconstruct without problems. By contrast, the worst-case shows little change in the PSNR and SSIM values, the deployment of a lower number of neighbor slices can increase the quality of the reconstruction thanks to the congruence of the values with the nearest images. In these dissertation, the five slices case does not provide some enhancement because the neighbors considered are too far from the reference, so the values considered can be too different, and they cause a drop in the reconstruction quality. The use of a higher number of slices in the computation can not provide an optimal reconstruction with this particular dataset; as can be seen, the images with homogeneous values have the same PSNR and SSIM of the single slice case, see fig. (5.29). Instead, the non-homogeneous images, see fig. (5.30) produce a small enhancement in the parameters, so the CNN has a better approach with this kind of image that has a larger number of areas with a higher pixel value.

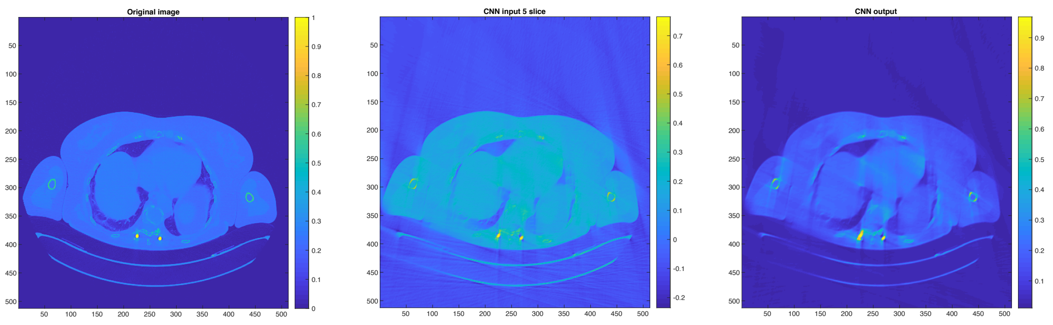


Fig. 5.29: 3 slices - Best case, second table's row

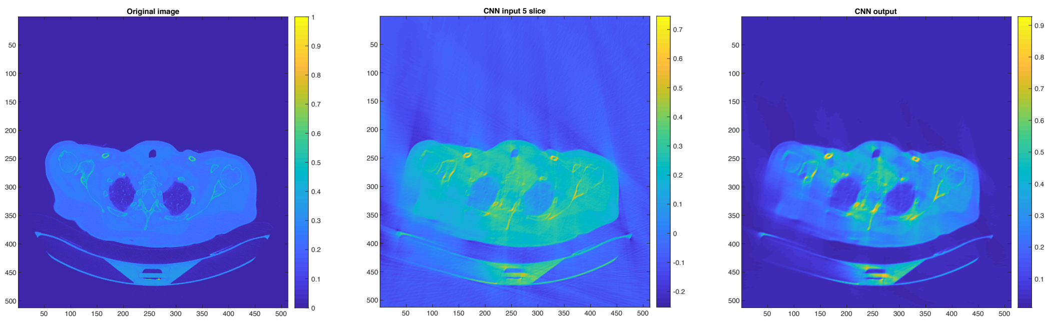


Fig. 5.30: 3 slices - Worst case, ninth table's row.

Chapter 6

Conclusions

This thesis work aimed to highlight the benefits of deep learning applied to a limited angle CT scanning for application in the medical and industrial field. The usage of a limited range angle computed tomography could be useful in a plethora of applications since it is a solution to reduce the quantity of X-ray's radiation, decrease the overall cost, and complete the entire measurement process in less time.

The application of this solution provides some contraindications; it produces noise and creates some artifacts which are counterproductive results.

Therefore, these results can be improved by applying the concepts of deep learning. In this work were developed two different methods for exploiting the problem of reconstruction with limited angles. Each method aims to produce an enhanced outcome starting from a noisy measurement.

The first method considers denoising reconstructed slices independently, and was tested using three datasets obtained from sinograms with a limited number of projections using angle range of [0-90°], [0-120°], [0-150°]. So, different level of noise and artifacts were obtained.

The response of the considered U-net was significant. We got results by evaluating the PSNR and the SSIM parameters over the reconstructed images from the sinograms with a higher degree value, and also containing a higher number of projections. In this way the CNN produce an outcome with an average PSNR = 30,94 dB and an average SSIM = 0,9950 for the SART 0-150°. These values are on the same level or slightly superior to the ones reported on state of the art works, e.g [54], tested on the same data.

In the other two datasets, the amount of noise was higher; despite this, the results were excellent. For both, there was a significant improvement in PNSR and SSIM. There was a difference, in respect to the evaluation metric reported in [54], of +2,93 dB for PSNR and + 0,0310 for SSIM in the SART 0-120°.

The best result was obtained with the noisier dataset considered for the SART 0-90°; there was an improvement of +5,37db for PSNR and +0.0504 for SSIM.

A second method was proposed in order to evaluate the possibility of jointly performing denoising on neighbor slices. In this case, it was considered a particular dataset which allowa us to have a spatial reference for each slice in order to elaborate a higher amount of information, maintaining the image's congruence for a proper CNN's mode of operation. The images considered were obtained by sinograms that contain projections within the range 0-120°.

This approach allows us to deal with an intermediate case with a discrete level of noise and with spatial references, which offer the possibility to handle voxels working with slices of images enhanced the quantity of information used for the comparison.

For this method, the considered datasets are composed of the reference image and some neighbors. In total, there are three or five images. The aim of this operation is to improve the denoising factor of the network making the convolutional operation on the reference image, which is the image in the centre position of the input array. The central image is the selected one to be restored, by considering a higher density of data that are provide by the neighbor images that were the in previous or in the following positions with respect to the reference one.

The results obtained present a slight improvement, especially for the three slices case, with an increment of +0,16 for the PSNR and +0.0053 for the SSIM. The motivations behind these modest improvements could be many. The usage of neighbor slice, in respect of denoising each slice separately, could enhance the performance within certain limits. Using a higher distance, such as in the five slices case, could be counterproductive because the introduced additional information could be not related to the central slice. Also, the network could be underpowered, and there will be the necessity to increase the deep and computational power to allow the extraction of a more significant number of features.

Lastly, the dataset employed could be the primary motivation. It was noticed that the images that show a significant number of regions with high pixel values improved the reconstruction. A better generalization of the dataset could be one solution for the problem. Despite all, this work points out the benefit and proving the quality of the deep learning for the reconstruction in a limited angle measurement case considering a single image comparison and laying the basis for the use of multislice in the standard applications.

Future directions

- Increase the simulation time by migrate the toolbox in C++ programming language and speed up the execution processes.
- Management over the CNN must be done to increase the layers' number and the computational power. Improving the network capabilities enhances the number of extracted features and, consequently, increases the details observed by carrying out a deeper analysis of the images. In this way, an output with a better quality could be produced.
- A detailed research of a proper dataset must be considered to have the correct number of images with homogeneous features that can easily be identified.

Also, the resolution, in term of the distance between two projections, is essential to have the same degree of similarity, allowing a much richer amount of data that can be processed in the multislice case.

- Operate with a different range of reconstruction algorithms to understand which one is the more suitable for this specific application.
- Conduct a different form of reconstruction measurement to evaluate the improvement obtained by generating the reconstructed image with a displacement in order to imitate a helical movement performed in a 3D computed tomography.

Bibliography

[1] Wim van Aarle. *The ASTRA Toolbox: A platform for advanced algorithm development in electron tomography.* 2019. URL: https://github.com/astratoolbox.

[2] Valentin Grigorevich Lapa Alexey G. Ivakhnenko. *Cybernetics and Forecasting Techniques.* American Elsevier Publishing Company, 1967.

[3] A.H. Andersen and A.C. Kak. "Simultaneous Algebraic Reconstruction Technique (SART): A superior implementation of the ART algorithm". In: *Ultrasonic Imaging* 6.1 (1984), pp. 81–94. ISSN: 0161-7346. DOI: https://doi.org/10.1016/0161-7346(84)90008-7. URL: http://www.sciencedirect.com/science/article/pii/0161734684900087.

[4] E C Beckmann. "CT scanning the early days". In: *The British Journal of Radiology* 79.937 (2006). PMID: 16421398, pp. 5–8. DOI: 10.1259/bjr/29444122. eprint: https://doi.org/10.1259/bjr/29444122. URL: https://doi.org/10.1259/bjr/29444122.

[5] Marcel Beister, Daniel Kolditz, and Willi A. Kalender. "Iterative reconstruction methods in X-ray CT". In: *Physica Medica* 28.2 (2012), pp. 94–108. ISSN: 1120-1797. DOI: https://doi.org/10.1016/j.ejmp.2012.01.003. URL: http://www.sciencedirect.com/science/article/pii/S112017971200004X.

[6] Kenneth Clark et al. "The Cancer Imaging Archive (TCIA): Maintaining and Operating a Public Information Repository". In: *Journal of digital imaging* 26 (July 2013). DOI: 10.1007/s10278-013-9622-7.

[7] Dudgeon and Mersereau. *Multidimensional digital signal processing.* Prentice-Hall, 1984.

[8] Vincent Dumoulin and Francesco Visin. *A guide to convolution arithmetic for deep learning.* 2018. arXiv: 1603.07285 [stat.ML].

[9] Er Flisch et al. "Industrial computed tomography in reverse engineering applications". In: *Proc. of Computerized Tomography for Industrial Applications and Image Processing in Radiology, BB 67-CD Paper 8.* 1999.

[10] K. Fukushima. "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position". In: *Biological Cybernetics* 36 (2004), pp. 193–202.

[11] Kunihiko Fukushima. “Neocognitron for handwritten digit recognition”. In: *Neurocomputing* 51 (2003), pp. 161–180. ISSN: 0925-2312. DOI: https://doi.org/10.1016/S0925-2312(02)00614-8. URL: http://www.sciencedirect.com/science/article/pii/S0925231202006148.

[12] Herman Gabor and Kuba Attila. *Discrete Tomography Foundations, Algorithms, and Applications*. Springer, Berlin, Heidelberg, 1999. URL: https://www.springer.com/gp/book/9780817641016.

[13] *Generations of CT Scanners*. URL: http://199.116.233.101/index.php/Generations_of_CT_Scanners.

[14] Lucas L. Geyer et al. “State of the Art: Iterative CT Reconstruction Techniques”. In: *Radiology* 276.2 (2015). PMID: 26203706, pp. 339–357. DOI: 10.1148/radiol.2015132766. URL: https://doi.org/10.1148/radiol.2015132766.

[15] Julien H. Girard et al. *Original use of MUSE’s laser tomography adaptive optics to directly image young accreting exoplanets*. 2020. arXiv: 2003.02145 [astro-ph.IM].

[16] Dustin Grinnel. *The Invention of Computed Tomography*. URL: https://www.criver.com/eureka/the-invention-of-computed-tomography.

[17] Said Attia al Hagrey. *Geophysical Imaging Techniques*. Springer, Berlin, Heidelberg, 2012. URL: https://doi.org/10.1007/978-3-642-22067-8_10.

[18] Rohan Hirekerur. *A Comprehensive Guide To Loss Functions — Part 1 : Regression*. URL: https://medium.com/analytics-vidhya/a-comprehensive-guide-to-loss-functions-part-1-regression-ff8b847675d6.

[19] B. K. P. Horn. “Fan-beam reconstruction methods”. In: *Proceedings of the IEEE* 67.12 (1979), pp. 1616–1623. DOI: 10.1109/PROC.1979.11542.

[20] Kurt Hornik. “Approximation capabilities of multilayer feedforward networks”. In: *Neural Networks* 4.2 (1991), pp. 251–257. ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(91)90009-T. URL: http://www.sciencedirect.com/science/article/pii/089360809190009T.

[21] G. N. Hounsfield. “Computerized transverse axial scanning (tomography): Part 1. Description of system”. In: *The British Journal of Radiology* 46.552 (1973). PMID: 4757352, pp. 1016–1022. DOI: 10.1259/0007-1285-46-552-1016. eprint: https://doi.org/10.1259/0007-1285-46-552-1016. URL: https://doi.org/10.1259/0007-1285-46-552-1016.

[22] Lu Huang et al. “Serial Quantitative Chest CT Assessment of COVID-19: Deep-Learning Approach”. In: *Radiology: Cardiothoracic Imaging* 2.2 (2020), e200075. DOI: 10.1148/ryct.2020200075. eprint: https://doi.org/10.1148/ryct.2020200075. URL: https://doi.org/10.1148/ryct.2020200075.

[23] A. G. Ivakhnenko. “The Group Method of Data of Handling ; A rival of the method of stochastic approximation”. In: *Soviet Automatic Control* 13 (1968), pp. 43–55. URL: https://ci.nii.ac.jp/naid/10004319713/en/.

[24] Sai Nikhilesh Kasturi. *Underfitting and Overfitting in machine learning and how to deal with it !!!* URL: https://towardsdatascience.com/underfitting-and-overfitting-in-machine-learning-and-how-to-deal-with-it-6fe4a8a49dbf.

[25] Henry J. Kelley. “Gradient Theory of Optimal Flight Paths”. In: *ARS Journal* 30.10 (1960), pp. 947–954. DOI: 10.2514/8.5282. eprint: https://doi.org/10.2514/8.5282. URL: https://doi.org/10.2514/8.5282.

[26] J. Kim, J. K. Lee, and K. M. Lee. “Accurate Image Super-Resolution Using Very Deep Convolutional Networks”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 1646–1654. DOI: 10.1109/CVPR.2016.182.

[27] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Advances in neural information processing systems* 25 (2012), pp. 1097–1105.

[28] Andreas Maier et al. “A gentle introduction to deep learning in medical image processing”. In: *Zeitschrift für Medizinische Physik* 29.2 (2019). Special Issue: Deep Learning in Medical Physics, pp. 86–101. ISSN: 0939-3889. DOI: https://doi.org/10.1016/j.zemedi.2018.12.003. URL: http://www.sciencedirect.com/science/article/pii/S093938891830120X.

[29] Anas Al-Masri. *What Are Training, Validation and Test Data Sets in Machine Learning?* URL: https://medium.com/datadriveninvestor/what-are-training-validation-and-test-data-sets-in-machine-learning-d1dd1ab09bae.

[30] Warren S. McCulloch and Walter Pitts. “A logical calculus of the ideas immanent in nervous activity”. In: *Bulletin of Mathematical Biology* 52.1 (1990), pp. 99–115. ISSN: 0092-8240. DOI: https://doi.org/10.1016/S0092-8240(05)80006-0. URL: http://www.sciencedirect.com/science/article/pii/S0092824005800060.

[31] Mayank Mishra. *Convolutional Neural Networks, Explained.* URL: https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939.

[32] Zeeshan Mulla. *Cost, Activation, Loss Function—— Neural Network—— Deep Learning. What are these?* URL: https://medium.com/@zeeshanmulla/cost-activation-loss-function-neural-network-deep-learning-what-are-these-91167825a4de.

[33] State University of New York at Buffalo. Department of Computer Science and A.V. Lakshminarayanan. *Reconstruction from divergent ray data*. Technical report. State University of New York, Department of Computer Science, 1975. URL: https://books.google.it/books?id=XbVCHQAACAAJ.

[34] Palenstijn WJ Pelt D, Ilya Sutskever, and Geoffrey E Hinton. "Tomopy: a framework for the analysis of synchrotron tomographic data". In: *Journal of Synchrotron Radiation* 25 (2012), pp. 1097–1105.

[35] The Nobel Prize in Physiology or Medicine 1979. *NobelPrize.org*. Nobel Media AB 2020. URL: https://www.nobelprize.org/prizes/medicine/1979/summary/.

[36] Andrea Provino. *Stochastic Gradient Descent, Batch Gradient Descent Algorithm*. URL: https://andreaprovino.it/stochastic-gradient-descent-batch-gradient-descent/.

[37] Ning Qian. "On the momentum term in gradient descent learning algorithms". In: *Neural Networks* 12.1 (1999), pp. 145–151. ISSN: 0893-6080. DOI: https://doi.org/10.1016/S0893-6080(98)00116-6. URL: http://www.sciencedirect.com/science/article/pii/S0893608098001166.

[38] J. Radon. "Über die Bestimmung von Funktionen durch ihre Integralwerte längs gewisser Mannigfaltigkeiten". In: *Akad. Wiss.* 69 (1917), pp. 262–277.

[39] Caroline Richmond. "Sir Godfrey Hounsfield". In: *BMJ* 329.7467 (2004), p. 687. ISSN: 0959-8138. DOI: 10.1136/bmj.329.7467.687. eprint: https://www.bmj.com/content/329/7467/687.1.full.pdf. URL: https://www.bmj.com/content/329/7467/687.1.

[40] F Rosenblatt. "The perceptron: A probabilistic model for information storage and organization in the brain". In: *Psychological Review* 65 (1958), pp. 386–408. DOI: https://doi.org/10.1037/h0042519.

[41] S. Roy et al. "Deep Learning for Classification and Localization of COVID-19 Markers in Point-of-Care Lung Ultrasound". In: *IEEE Transactions on Medical Imaging* 39.8 (2020), pp. 2676–2687. DOI: 10.1109/TMI.2020.2994459.

[42] Sebastian Ruder. *An overview of gradient descent optimization algorithms*. URL: https://ruder.io/optimizing-gradient-descent/index.html#gradientdescentvariants.

[43] Siemens. *A Gentleman´s Crazy Idea*. URL: https://www.medmuseum.siemens-healthineers.com/en/stories-x-ray-technology/history-of-ct.

[44] Thomas Strohmer and Roman Vershynin. "A Randomized Solver for Linear Systems with Exponential Convergence". In: Jan. 2006, pp. 499–507. ISBN: 978-3-540-38044-3. DOI: 10.1007/11830924_45.

[45] Van Tessa et al. "ITERATIVE RECONSTRUCTION ALGORITHMS The implementation of iterative reconstruction algorithms in MATLAB". In: (Jan. 2007).

[46] Van Tessa et al. "ITERATIVE RECONSTRUCTION ALGORITHMS The implementation of iterative reconstruction algorithms in MATLAB". In: (Jan. 2007).

[47] Joseph J. Titano et al. "Automated deep-neural-network surveillance of cranial images for acute neurologic events". In: *Nature Medicine* 24.9 (Sept. 2018), pp. 1337–1341. ISSN: 1546-170X. DOI: 10.1038/s41591-018-0147-y. URL: https://doi.org/10.1038/s41591-018-0147-y.

[48] J.G. Sanctorum2 V. Nguyen1 J. De Beenhouwer1. *Fast and Flexible X-ray Tomography Using the ASTRA Toolbox.* 2019. URL: https://github.com/astratoolbox.

[49] Alessandro Vallebona, ed. *Una modalità di tecnica per la dissociazione radiografica delle ombre applicata allo studio del cranio Comunicazione.* Torino.

[50] K J. Batenburg W. J. Palenstijn and J. Sijbers. *ASTRA toolbox 3D geometry reconstruction.* 2016. URL: https://www.astra-toolbox.com/docs.html.

[51] and J.Sijbers W.J.Palenstijn K.J.Batenburg. *ASTRA toolbox.* URL: https://www.astra-toolbox.com/docs/geom3d.html.

[52] and J.Sijbers W.J.Palenstijn K.J.Batenburg. *ASTRA toolbox 3D geometry reconstruction.* 2016. URL: https://www.astra-toolbox.com/docs/algs/FP3D_CUDA.html.

[53] andJ.Sijbers W.J.Palenstijn K.J.Batenburg. "Performance improvements for iterative electron tomography reconstruction using graphics processing units (GPUs)". In: *Computed Tomography.* Vol. 23. 35-47. 2011, pp. 250–253. DOI: 10.1016/j.jsb.2011.07.017.

[54] Jiaxi Wang et al. "Deep learning based image reconstruction algorithm for limited-angle translational computed tomography". In: *PLOS ONE* 15.1 (Jan. 2020), pp. 1–20. DOI: 10.1371/journal.pone.0226963. URL: https://doi.org/10.1371/journal.pone.0226963.

[55] *Ralph Alfidi su Wikipedia.* URL: https://it.wikipedia.org/wiki/Ralph_Alfidi.

[56] *Nonogram on Wikipedia.* 2020. URL: https://en.wikipedia.org/wiki/Nonogram.

[57] *Learning rate.* 2021. URL: https://en.wikipedia.org/wiki/Learning_rate.

[58] *Gradient descent.* 2021. URL: https://en.wikipedia.org/wiki/Gradient_descent.

[59] *SGD.* 2021. URL: https://en.wikipedia.org/wiki/Stochastic_gradient_descent.

[60] *Convolutional neural network.* 2021. URL: https://en.wikipedia.org/wiki/Convolutional_neural_network.

[61] *Training, validation, and test sets*. 2021. URL: https://en.wikipedia.org/wiki/Training,_validation,_and_test_sets#Cross-validation.

[62] Wikipedia. *Radiography*. 2008. URL: https://en.wikipedia.org/wiki/X-ray.

[63] Wikipedia. *Structural$_s$imilarity*. URL: https://en.wikipedia.org/wiki/Structural_similarity.

[64] Zhixuhao. *U-net*. 2019. URL: https://github.com/zhixuhao/unet.

[65] Hafidz Zulkifli. *Understanding Learning Rates and How It Improves Performance in Deep Learning*. URL: https://towardsdatascience.com/understanding-learning-rates-and-how-it-improves-performance-in-deep-learning-d0d4059c1c10.