Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica

Convolutional Neural Networks in Tomographic Image Enhancement

Relatrice:
Chiar.ma Prof.ssa
Elena Loli Piccolomini

Correlatrice:
Chiar.ma Dott.ssa
Elena Morotti

Presentata da:
Stefano Andriolo

III sessione
A.A. 2019/2020

Introduzione

Negli ultimi due decenni si è assistito ad un grande aumento di popolarità dei metodi di **Machine Learning** per fare in modo che un computer possa risolvere problemi su cui tipicamente incontra grande difficoltà come segmentazione di immagini, analisi del sentimento, riconoscimento di pattern ricorrenti e più in generale, tutte le classi di problemi che gli esseri umani sono in grado di risolvere con relativa semplicità. Queste tecniche permettono ad un algoritmo di imparare da una base di conoscenza al fine di produrre dei risultati sempre migliori man mano che accumula esperienza, seguendo a grandi linee lo stesso procedimento che si verifica in un cervello biologico.

In particolare, il **Deep Learning**, un sottinsieme del Machine Learning, ha ottenuto risultati estremamente promettenti in diverse delle classi di problemi menzionate precedentemente. Le *Reti Neurali Profonde* sfruttano grandi insiemi di dati per essere addestrate, e grazie al loro alto grado di profondità in termini di livelli, sono capaci di imparare a riconoscere molti tipi diversi di caratteristiche dei dati in input e possono quindi produrre risultati ottimi se addestrate correttamente. La diffusione di Internet nel mondo ha permesso di rendere questi enormi insiemi di dati richiesti dalle Reti Neurali Profonde facilmente disponibili ed i progressi tecnologici degli acceleratori hardware come le GPU hanno reso queste tecniche di *Intelligenza Artificiale* molto appetibili a ricercatori, aziende ed anche singoli individui.

Un campo scientifico che ha beneficiato molto dalle tecniche di Deep Learning è l'analisi di *immagini mediche*. La **Tomografia Computerizzata** (abbreviata come *CT* dall'inglese Computerized Tomography) in particolare può essere agevolata da questi metodi per migliorare la sua efficacia. La CT è una tecnica a raggi X nata formalmente nel 1971 per catturare immagini accurate di parti interne al corpo. Usando i dati ottenuti da alcune proiezioni, generate da un fascio di raggi X che attraversano il corpo del pazione, è in grado di

ricostruire un'approsimazione del volume o dell'immagine originale (a seconda del tipo di scanner), sfruttando le basi matematiche fornite dalla *trasformata Radon*.

Il problema più grande di questa tecnica di imaging è l'uso di radiazioni ionizzanti, che possono nuocere al paziente nel caso ne assorba troppe. Per compensare, le tomografie solitamente avvengono effettuando proiezioni solamente in un intervallo limitato di angoli invece che nell'intero range disponibile. Questo può portare a ricostruzioni sbagliate, nel senso che potrebbero fallire nel rappresentare alcune caratteristiche morfologiche del soggetto o potrebbero contenere alcuni artefatti che potrebbero essere fraintesi da uno specialista che esamini l'immagine.

Quello che faremo in questa tesi, sarà realizzare diverse **Reti Neurali Convoluzionali** partendo da una base comune e vedere come si comportano nel miglioramento di immagini tomografiche ad *angoli limitati* catturate da uno scanner con una geometria cone-beam. Useremo delle tradizionali convoluzioni 2D, ma faremo dei test anche con quelle tridimensionali. Per addestrare e testare l'accuratezza degli output delle reti sono stati generati due dataset: entrambi verranno proiettati su un intervallo limitato di angoli per poi essere ricostruiti usando diverse tecniche al fine di determinare quale fornisca il risultato migliore. Inoltre, sperimenteremo anche un algoritmo di ricostruzione iterativo per capire se un numero ridotto di iterazioni possa essere compensato da un passaggio di post-processing usando una rete neurale con lo scopo di velocizzare il processo di ricostruzione.

Il primo capitolo sarà dedicato alla Tomografia Computerizzata. Parleremo della sua storia e di come gli scanner si sono evoluti nel corso degli anni. Dopodiché parleremo brevemente di alcuni dei problemi che caratterizzano questa metodologia.

Il secondo capitolo consiste in una panoramica sulle Reti Neurali. Spiegheremo brevemente i principi alla loro base e come funzionano. Dopo aver parlato dei concetti fondamentali ed aver dato un idea di come possano imparare, oltre a descrivere alcune tecniche per migliorare le loro performance, illustreremo i component principali delle *Reti Neurali Convoluzionali*, una versione particolare di rete neurale che si comporta molto bene nel trattare immagini. Una breve sezione pratica sugli strumenti che possono essere utilizzati per realizzare e sfruttare le reti neurali chiuderà quindi questo capitolo.

Nel terzo capitolo si parlerà di alcuni algoritmi usati in questa tesi. Alcuni di questi si occupano di ricostruire l'immagine/il volume originale partendo dalle proiezioni, di cui illustreremo le caratteristiche principali. Si parlerà poi di come il dataset sintetico di ellissoidi sia stato costruito e delle sue particolarità. La sezione finale del capitolo conterrà una breve descrizione della rete che verrà usata come punto di partenza per i nostri esperimenti oltre che ad alcune indicazioni pratiche riguardo la loro esecuzione.

Il quarto capitolo verrà utilizzato per mostrare i risultati ottenuti nei test sui due dataset e per discuterli. Mostreremo diverse versioni di reti al fine di capire quale abbia avuto i risultati migliori; sperimentermo inoltre un algoritmo di ricostruzione iterativo per osservare se possa essere eseguito per un numero minore di iterazioni per poi andare a migliorare la ricostruzione effettuando un passaggio di post-processing usando una rete neurale.

Per concludere, condivideremo alcuni pensieri finali per riassumere e discutere i risultati ottenuti e come possano essere interpretati.

Introduction

The last two decades have seen a great rise in popularity of **Machine Learning** methods for making computers solve problems that are known to be difficult for them like image segmentation, sentiment analysis, pattern recognition and more in general, all the classes of problems whose humans excel to solve. These techniques allow an algorithm to learn from a base of knwoledge in order to provide increasingly better results as it accumulates experience, much like what happens in a biological brain.

In particular, **Deep Learning**, a subset of Machine Learning, accomplished extremely promising results in various of the aforementioned classes of problems. *Deep Neural Networks* take advantage of big datasets that are used to train them, and thanks to their high degree of deepness in terms of levels, they are able to learn many different kinds of features from the input data and can then provide extremely good results if trained correctly. The spread of the Internet around the world made these huge datasets needed by Deep Neural Networks easily available and the technological progress of computing accelerators such as GPUs made these *Artificial Intelligence* techniques very appealing for researchers, companies and also individuals who want to experiment.

A scientific field that has greatly benefited from Deep Learning techniques is the processing of *medical images*. **Computed Tomography** (abbreviated as *CT*) in particular can take advantage of these methods to improve its effectiveness. CT is an X-ray technique formally born in 1971 for taking accurate images of internal parts of the body. It works by using projection data, obtained from a beam of X-rays which flows through the patient's body, to reconstruct an approximation of the original volume or image (based on the type of scanner) and the mathematical foundations of the process are provided by the *Radon transform*.

The major problem of this imaging method is the use of ionizing radiations, which can harm the patient if too many are absorbed by its body. To compensate for this, tomography is usually performed by taking projections on a limited interval of angles instead of the full available range, leading to reconstructions errors due to missing spatial information that cannot be inferred from the limited data that is available. This translates in error-prone reconstructions, in the sense that they could fail to represent some morphological features of the subject body or they could contain some artifacts that could be misinterpreted by a specialist looking at the image.

What we will do in this thesis, is building different versions of **Convolutional Neural Networks** starting from a base layout and see how they perform in enhancing *limited-angle cone-beam* tomographic images. We will use traditional, 2D convolutions as well as experiment with 3D convolutions. Two datasets will be used to train and test the accuracy of out networks: both will be projected only on a limited set of angles and then reconstructed using different techniques to determine which yield the best results. In addition, we will experiment with an iterative reconstruction algorithm to understand if a reduced number of iterations could be compensated by a post-processing step using a neural network to speed up the reconstruction process. In the end, we will discuss on how different networks performed on the two datasets and on the obtained results.

The first chapter will be dedicated to Computed Tomography. We will talk about its history and how scanners evolved through the years. After that, we will briefly expose some of the problems of this approach.

The second chapter consists of an overview of Neural Networks. We will provide a brief explanation of the principles at their base and how they work. After explaining the core concepts and giving an idea of how they can learn, as well as describing some techniques used to improve their performance, we will illustrate the main components of *Convolutional Neural Networks*, a particular version of neural network that performs very well when dealing with images. A quick practical section on the tools that can be used to build and exploit neural networks will then close this chapter.

In the third chapter we will talk about some of the algorithms we used in this thesis. Some of these are the ones that are responsible of reconstructing the original image/volume starting from projections, for which we will perform a

quick overview of the main features. We will then talk about how the synthetic ellipsoid dataset has been built and its characteristics. The final part of the chapter will contain a brief description of the network layout that will be used as the basis for our experiments as well as some practical indications on how they were performed.

The fourth chapter will be used to accurately show the results we obtained in the tests on the two different datasets and discuss them. We will show different versions of networks in order to understand which one performs better, and we will try to experiment with an iterative reconstrucion algorithm to check whether it could be run for a lower number of iterations and followed by a neural network post processing in order to reduce the time it takes for performing an accurate reconstruction.

To conclude, we will give some final words discussing the results we obtained and how they can be interpreted.

Contents

Chapter 1

Computed Tomography

Computed Tomography is a medical imaging technique whose roots date back to the beginning of the 20th century. Its name is derived from the Greek words *tome* (slice) and *graphein* (to write). These words are a perfect description, although basic and simplified, of what CT does: building a digital representation of a volume by examining it slice by slice (more precisely projection by projection). In practice, these projections are taken by emitting a beam of X-rays around a patient and measuring their attenuation values using a detector surface. The recorded values are then processed by the machine's computer to generate cross-sectional images of the body.

The first, fundamental contribution for the development of CT was the formal discovery of X-rays in 1895 by the German physics professor Wilhelm Röntgen (who, after that, was awarded a Nobel Prize in Physics). Then in 1917, the mathematical theory of the Radon transform was proposed, which provided a mathematical demonstration of how a function could be reconstructed from an infinite set of its projections: it is one of the core concepts that led to the development of CT. By the 1930s, the mathematical foundations of CT were established and in 1972 the first, commercially available CT scanner, invented by Sir Godfrey Hounsfield at EMI Central Research Laboratories, was publicly announced. This first version of CT scanners was limited by the technology of those years: it acquired image data in about 4 minutes, computation time was about 7 minutes per picture and the final images had a resolution of only 80x80 pixels. Also, the use of an hollow water tank that had to enclose the patient's head was required in order to reduce the dynamic

range of the radiation reaching the detector device (due to the high difference between the air, bones and body tissues attenuation values).

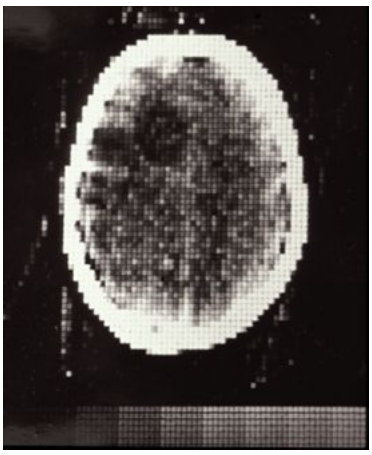


Figure 1.1: The first clinical scan taken at the Atkinson Morley's Hospital (Copse Hill, England) in October 1971 using a first generation scanner

Despite these initial limitations, CT quickly became the de-facto standard for taking medical images of a patient's body, replacing old (and sometimes very invasive) techniques like pneumoencephalography. Given their rise in popularity and number of applications, CT scanners have vastly improved throughout the last 50 years. Scanners now produce faster, more accurate and higher resolution images allowing specialists to diagnose diseases more easily and dramatically improve their precision: in 2008 Siemens introduced a new generation of scanners that was able to produce an accurate image in less than 1 second. Nowadays CT is one of the most important medical imaging procedures and is widely considered among the most important advances in medicine.

To conclude, CT continued to increase in popularity and new variations and enhancements continue to arise. One notable mention is the invention of **CBCT** (Cone Beam Computed Tomography), the variant examined in this thesis. It was first introduced in the European market in 1996 and in the American one in 2001. In 2013, during *Festival della Scienza* in Genova, Italy, the original members of the Italian research group received an award for its invention.

1.1 CT Scanners Generations

As mentioned earlier, CT works by emitting a beam of X-rays which passes through the patient's body. In the process, the rays intensity is attenuated in different ways depending on what material they intersect with; for example, a bone has a higher attenuation value with respect to a soft tissue, so it will absorb more energy from the ray, that will then reach the detector with less power. This procedure is repeated many times at different angles/positions (depending on the scan technique) to generate the complete projection. Once a full pass is performed, the computer processes the incoming data and using the mathematical tools that allows CT to work, computes the projection image; the bed moves forward and the process starts again to produce the other image slices.

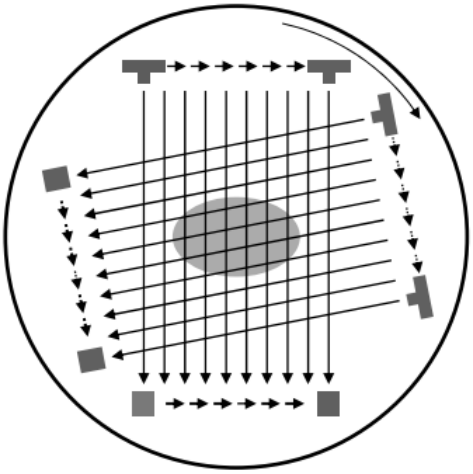


Figure 1.2: Simplified representation of a first generation scanner. The T-shaped object is the emitter that is initially translated, together with the detector, to acquire a single projection and then both rotate to complete the process.

As CT scanners improved, the layout of the projector and the detector and how they moved in relation to each other or the patient changed in order to accomodate for better types of scan. The first generation of scanners, like the one from Hounsfield and Cormack, used a type of scanning called "Pencil Beam" with a Rotate/Translate pattern. The name comes from the single beam that is emitted from the source, which uses a parallel beam geometry. This means that the source/detector devices have to move linearly in order to

acquire the data representing a slice of the body, before rotating the position of the X-ray tube to acquire data at different angles. The major benefit of such layout are the high performance in terms of scatter reduction since there is just a single emitter/detector. On the other hand, the biggest disadvantage of this first generation was the high amount of time it took to capture a full projection: to acquire a full image of the head, the X-ray tube had to be placed at a specific angle, then it started translating linearly to acquire multiple projections. After approximately 160 were taken at that angle, the X-ray tube and detectors were rotated by one degree and the process would start again, until all the two-dimensional projection images were acquired at 180 different angles.

With the goal of decreasing the amount of time it took to acquire a full CT, the first improvement to CT devices was using a narrow fan beam with an angle of approximately 10 degrees instead of a single ray and consequently an array-shaped detector to measure rays intensities. A translation to cover the full width of the body was still required, but even with just a 10 degrees-wide beam, the acquisition time was dramatically decreased, with a reduction of two to three minutes per slice. In the end, this second generation was measured to be about fifteen times faster than the previous one. This new layout introduced the problem of scatter radiation due to the narrow fan beam, unlike in the first generation where there was a single ray. A (potentially) unexpected side-effect of this new layout was the slowdown in the image-acquiring step that would slightly impact on the benefits it provided due to the added complexity in the imaging protocol given by the rotation and translation of the X-ray tube and the detector. Another disadvantage of this second generation is the high susceptibility to moving objects, that would cause te development of artifacts in the reconstructed image. For this reason, this generation of CT scanners too wasn't still a good fit for scanning body parts other than the head.

The biggest problems of the first two CT scanners generations was the translational motion which was very time consuming. Given that, the natural following step was to widen the angle of the fan beam and increase the length of the detector array with it, allowing the device to capture an entire slice of the patient at one time. Specifically, the fan beam angle now ranged from 40 to 60 degrees, the new detector array now consisted of 400 to 1000 elements and the two parts were joined together to ensure the synchronization of the

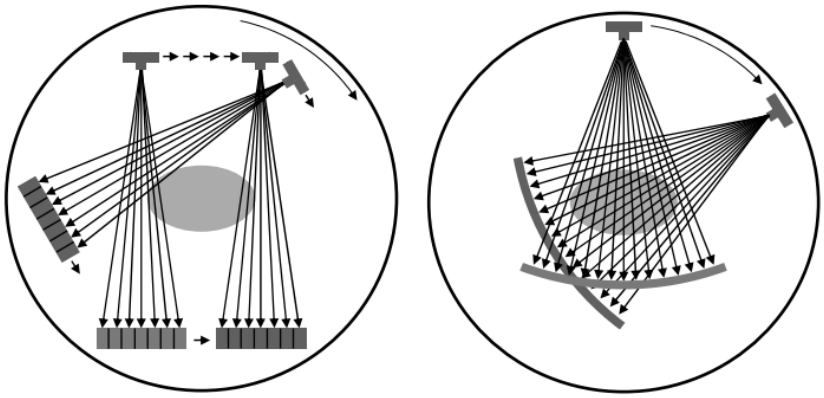


Figure 1.3: Representations of narrow fan-beam and wide fan-beam scanner layouts. Here we can see that both emit a fan-shaped beam of ray, the main difference being the absence of linear translation in the latter, which could take an entire projection with a single shot.

rotational movement between them. This generation of scanners, that could still be found operating somewhere, could deliver scan times as low as 5 seconds per projection angle. The two biggest disadvantages were both due to the high number of detector elements: firstly they were very expensive (although the benefits they provided were argued to overcome this); secondly, this generation produced a characteristic image artifact (ring artifact) that is traceable to the frequent lack of calibration between the detector elements.

To solve this last problem, the fourth generation of scanners replaced the rotating detector array with a static detector ring placed all around the patient, allowing the detector elements to keep the calibration and stay in sync. This new layout allowed the X-ray emitter to be placed both outside or inside the detector ring, with the constraint that in the last case it has to be tilted so that the X-rays only interact with the detector ring after they passed through the patient, not beforehand.

The fifth generation was specifically designed for use in cardiac tomographic imaging. This means that extremely short acquisition times ($\tilde{5}$0ms) were needed to acquire images without motion artifacts due to the heart's movement. These types of scanners used a completely different layout that included, among other things, of an electron beam emitter whose output was deflected towards a target ring that enclosed the patient, generating the X-rays that then passed through the patient and were captured by the detector on

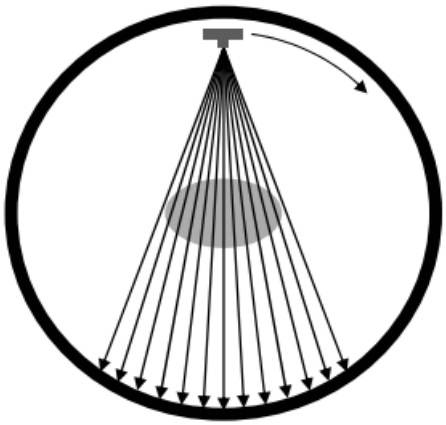


Figure 1.4: Layout of the fourth generation of scanners. The bold black circle represents the still detector ring that captures the attenuated ray emitted by the source, which in this case is located inside the ring.

the opposite side. However, since this type of scanners was essentially single-purpose, it was very expensive and not much versatile, so it wasn't considered a popular addition in the field of medical imaging.

The sixth generation removed the need to stop the gantry (the emitter/detector group) after every slice to advance to the next position thanks the development of the slip ring technology for CT devices in 1990. This technology allowed the rotating components to always have a source of power as well as a data connection to send image data, task once reserved to cables which were the primary limitation to freely move the gantry. Specifically, this new generation introduced for the first time in CT devices history a system that could rotate continuously around the patient while the bed moved forward, forming a sort of helical shape, and giving the name to this generation of "helical CT"s. The major problem of this is that no full slices are taken since the bed is in perpetual motion during the scan and the scanner is not producing planar section, but this can be solved in the reconstruction process.

The most recent type of scanners consists of a cone-beam emitter and a detector matrix layout. This layout is obtaining after a similar reasoning that led the transition from the first to the second generation of CT scanners. In that case, the pencil beam geometry was extended to create a fan beam geometry, and this time the fan beam geometry has been extended itself, creating a cone-shaped ray. To capture a ray with this shape, the linear detector had to

be changed too to make a flat panel detector composed by a matrix of detector elements. With this new layout, multiple slices could be taken in a very short timespan, leading to an enormous reduction in acquisition times. The main drawback of this new implementation was a higher level of sophistication in the reconstruction process.

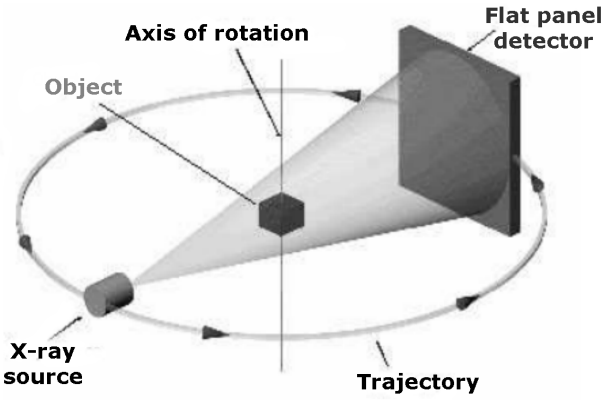


Figure 1.5: Cone Beam scanner layout: the source emits a cone-shaped beam that expands as it approaches the detector matrix, whose attenuation data can be used to produce the projection images.

Since the detector now composed by a matrix of elements, each projection is actually an image. Putting together all the projected images with a reconstruction algorithm, we can obtain a 3D object that represents the original body. The volume is reconstructed as a grid of *voxels* and it can be processed slice-by-slice as a sequence of single images, for example for helping to visualize its shape at a specific index on the sliced axis, or as a whole volume. In this thesis both representations will be used in order to understand which of the two leads to the best results.

1.2 Problems of CT

One of the common goals that every generation of scanner tried to achieve was minimizing the time it took to acquire image data. This is not only for reducing artifacts due to the patient's (voluntary or involuntary) movement but also to minimize the time she is exposed to X-rays. The radiation they emit can damage body cells, including DNA molecules, which can lead to radiation-induced cancer. One study estimated that 0.4% of cancers in the US resulted from CT scans, and that this value could have increased to to as much as 1.5% to 2% based on the rates of use of CT in 2007, although this estimate is largely disputed, considering the (relatively) low dose used in CT scans is not demonstrated to cause considerable damage. To minimize the exposition to X-rays, CT devices tend to adopt one of two techniques so the patient's body is exposed to less radiation, which we will talk about later.

Reconstructing the original image starting from the projections is performed using algorithms that have the Radon transform (in particular, its inverse) at their base. Essentially, given an input function f defined over a plane, it outputs a function defined in the two-dimensional space of lines in the plane, whose value at a particular line is equal to the line integral of the subject function over that line. The inverse Radon transform allows to "back-project" this set of lines to calculate the initial function. This mathematical concept is used for the reconstruction step in all the generations of CT scanners, with increasing refinements and adaptations as the geometry evolved.

The mathematical model works well for an infinite set of projections and continuous projection functions, giving an exact representation of the subject. In practice, things are different. First of all, only a finite number of projections can be taken with any type of CT. This finite set of projections can be graphically represented into an object called *sinogram*. Having just a finite set of data means that the information between one projection and the following one is missing and in order to reconstruct the original object where this data is missing, the reconstruction algorithm has to interpolate the existing data. This often leads to unstable solutions and interpolation errors, which cause the reconstructed image to contain artifacts and, in general, loose quality and details. In addition to this, to reduce the amount of used radiation that are potentially harmful to the patient, two techniques have been consoli-

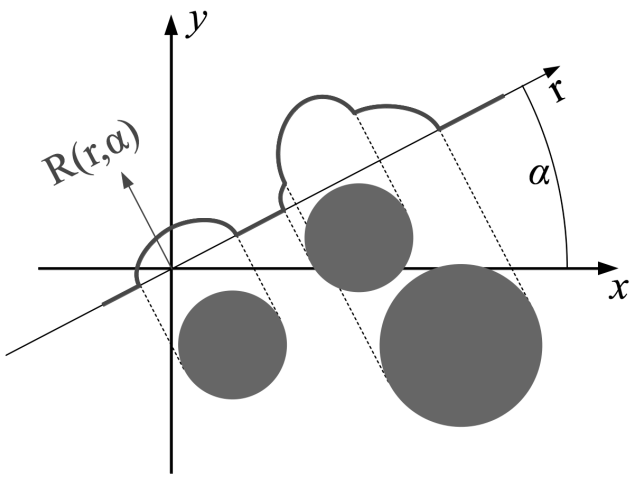


Figure 1.6: Conceptual representation of the radon transform: the thick line can be seen as the line integral of the function (that in this case is represented by the three circles) over the r straight line.

dated during the evolution of CT: **limited-angle** and **sparse-view** computed tomography. In the former case projections are taken in a subinterval [-ϕ, ϕ] of the full [-$\pi/2$, $\pi/2$) range. This approach is mostly used in tomographical applications where the shape of the examined body region doesn't allow for a full revolution around it, or it wouldn't be useful due to the human anatomy like dental tomography or breast tomosynthesis. Since this thesis mainly focuses on enhancing breast CT reconstruction images, this is the type of scan that will be examined. Sparse-view CT on the other hand reduces the number of projections by still taking them in the full [-$\pi/2$, $\pi/2$) range, but increasing the interval between projections. In both cases however the smaller set of data impacts on the reconstruction quality and increases the chances of generating artifacts. This is further worsened by non-deterministic measurement errors or random noise in the projection.

Many algorithms have been developed to solve the ill-posed problem of building an approximation of a function starting from limited CT data. One of the first (and still most used) is *Filtered Back Projection*, a stabilized and discretized version of the Radon transform which filters the projections before calculating their inverse. This algorithm produces non-optimal results and is very sensitive to projection quality, but its simplicity and and computational speed are the main reasons for its popularity. It is also relevant because it provides the foundation for many other reconstruction algorithms for many of

the scanners layouts previously described. For the purposes of this thesis, relevant algorithms are *iterative reconstruction* ones, which are computationally intensive but produce overall better results [5], and **FDK**, an algorithm for reconstructing cone beam CT scans based on a modified version of the Radon transform for two dimensions that results in a fan-beam-like reconstruction formula [2].

The last years have seen the rise of neural networks applied to traditional reconstruction techniques to improve the overall reconstruction quality. In particular, **Deep Neural Networks** achieved impressive results in the field of medical imaging enhancement and continue to improve as time passes, as demonstrated by the introduction of U-Nets [12], which are especially good in these kind of tasks. In this thesis the traditional reconstruction methods will be used to provide a starting point and see how they behave in limited-data scenarios. After that, we will train a Deep Neural Network to improve these reconstructions.

Chapter 2

Neural Networks

2.1 An introduction to Neural Networks

Neural networks (properly **Artificial** Neural Networks) are a biologically-inspired programming paradigm which enable a computer to learn from observational data. They imitate biological neural networks in order to perform tasks that are known to be very easy to solve for humans but at the same time very difficult for computers: image analysis, speech recognition, adaptive control and so on. The structure at the base of Neural Networks is a **Neuron**, a processing unit that is linked to other neurons through directed weighted connections; there are different kinds of neurons, some of which will be illustrated in the following paragraphs.

By combining many neurons together we can create a neural network that can be *trained* to perform some task. In the context of NNs, a **layer** is a set of neurons which can detect the same type of feature of its input data: it could be the input layer, for example, the set of neurons where initial data is fed into, or an intermediate layer dedicated to edge detection in images. Different types of networks can be created by connecting neurons in different ways. The first and most simple network types are *feedforward neural networks* [10], which are directed acyclic graphs, where data is always fed from one layer to the following one, never backwards. *Recurrent neural networks* instead contain some links between neurons called **loopback connections**. Through these connection, a neuron can process not only information that comes at time t, but also the output of some neuron at time $t-1$. These networks are particularly powerful

in tasks where having the notion of time or in general of a sequence of data is extremely important to perform a prediction. They have indeed shown very good results in problems of speech recognition, hand-writing recognition and sequence prediction.

One of the first significant result for the development of ANNs is the creation of the **perceptron** in 1958 by the American psychologist Frank Rosenblatt [1]. It is a binary classifier that could tell if a vector of inputs belongs to some specific class. It takes a series of binary inputs $x_1, x_2, ..., x_n$, applies an activation function and then produces a single binary output. The perceptron includes the concept of *weights* $w_1, w_2, ..., w_n$, a set of real values paired with the input connections which express the "importance" of the associated input in relation to the output. It then emits 0 or 1 as the output based on whether the *weighted sum* of the input is greater than a threshold value, called *bias*, or not; this parameter is used to make it easier for the neuron to output a certain value than the other (from this the name bias, since the neuron is biased towards a value). Formally, a perceptron's output is defined as follows:

$$output = \begin{cases} 0 & \text{if} \quad w \cdot x + b \leq 0 \\ 1 & \text{if} \quad w \cdot x + b > 0 \end{cases} \tag{2.1}$$

where $w \cdot x$ is the dot product of the vectors of inputs and weights, and b is the bias. This is called the *activation function*, since it tells whether the neuron should activate or not; in the case of perceptrons it is the *Heaviside step function* [7]. Here, it is easy to see that for big bias values, the neuron is very likely to *fire* (output 1), while it is very unlikely to do so when bias is very negative.

This structure is quite powerful since it can compute many functions by choosing the right weight values, but it is not able perform operations as simple as detecting when two inputs are different, i.e. the XOR function [14]. More precisely, the perceptron only works with textitlinearly separable input datasets.

For the learning to be possible, it is necessary that a small change in the input data only cause a small change in the output. This allows the training algorithm to continuously change the weights values by confronting the updated values with the old ones, since they are relatively easy to keep track of. For example, a network could refine some weights to more precisely identify

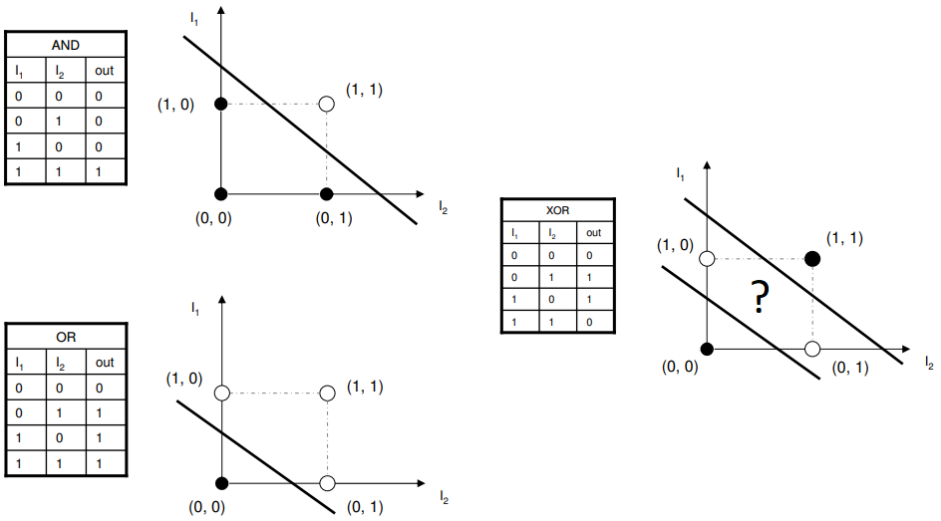


Figure 2.1: Geometrical representation of two linearly separable problems on the left (the OR function and the AND function) and a non-linearly separable one (the XOR function).

a feature of the input data, changing them in a way that could progressively reduce the difference between the expected result and the actual one, making learning happen.

It is quite clear that perceptrons networks do not expose such behavior: a small change in the input of a single neuron could completely flip its output, if the previous value was close to the threshold boundary. This could then start a chain reaction that will propagate through the entire network, even coming to completely alter the network output.

For this reason a new type of neuron has been proposed with the name of **sigmoid neuron**, which is called like that after the activation function it uses to compute the output value. The *sigmoid function* looks like a smoothed version of the Heaviside step function and has the previously illustrated property of not amplifying the input error unlike the perceptron's activation function. It is defined as follows:

$$\sigma(z) = \frac{1}{1 + e^{-Z}} \tag{2.2}$$

where $z = w \cdot x + b$ as before. In the case of sigmoid neurons however, the inputs are real numbers and not just bits, i.e. $x \in \mathbb{R}^n$.

The biggest advantage of sigmoid neurons over perceptrons is their low

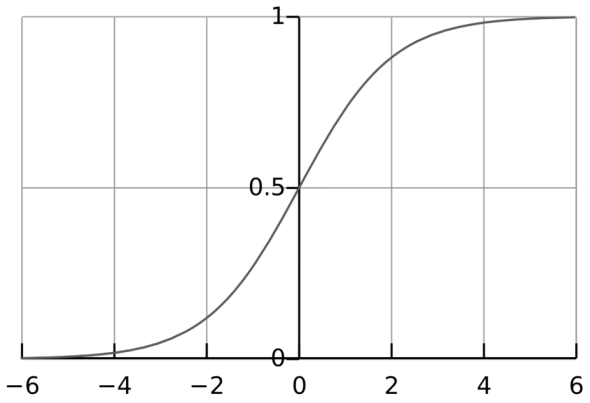


Figure 2.2: Graph of the sigmoid function

susceptibility to small changes Δw_j in the weights and Δb in the bias, which only lead to a small change $\Delta output$ in the output. To put in other terms, it is desirable that the activation function defines a linear relationship between $\Delta output$ and the changes Δw_j and Δb. Many other functions can substitute the sigmoid function: $ReLU$[8], *tanh*, or the *Softmax activation function*, just to name a few. These types of neurons have entirely substituted perceptrons and are the elementary processing unit of modern neural networks.

The basic idea behind the training of a neural network is to adjust the weights of every single neuron based on the result of a *loss function* C , which could be the *Mean Squared Error* function, the *Cross Entropy* function, or something else. The most famous algorithm for training a network is the *backpropagation* algorithm, which updates each weight by using some gradient method as the *gradient descent* to compute the gradients of the loss function and then "backpropagating" the error. In other words, what we call learning is actually minimizing the *loss function* by adjusting the weights of the neurons in the network. The backpropagation algorithm indeed tries to find the minimum of the loss function using gradient descent, which in turns uses partial derivatives with respect to the weights to find the steepest path to the function's minimum. Note that the resulting value could be just a *local* minimum and not the global one, but this has been argued not to be a problem in practical applications [11].

To speed up the learning process, at each step the gradient is multiplied by a value η defined a-priori called *learning rate*. The concept is that the gradient

contains the information about the *direction* that should be taken to reach the minimum and, multiplying it by the learning rate, we can get to such value faster. Obviously the value of this parameter must be chosen balancing the potential speedup of the learning and the need to avoid overshooting: this happens if the learning rate is too high, causing the gradient value to not stop at the minimum but pass it and reach the opposite slope.

Calculating the gradient of the cost function for every input in all neurons, however, would be very time consuming for large networks, slowing down the learning. To get around this problem, a technique called *stochastic gradient descent* is often used. It works by selecting a random subset of training inputs and estimating the gradient ΔC by computing the gradient ΔC_x only for such subset, which is called *mini-batch*. The averaging of the results provides a good estimate of the actual gradient value, overcoming the problem of slow gradient update. This process is usually repeated until the entire training set has been examined, which is the event that signals the completion of an *epoch* of training. At this point, another epoch is started, and the process is repeated for a number of epochs that could either be chosen in advance by the network designer or variable, depending on the accuracy of the network: if it doesn't improve after many epochs, the training algorithm halts; this technique is called *early stopping*.

This strategy is also useful to contrast *overfitting*. Overfitting occurs when a network learns intrinsic features of the training set, losing the ability to generalize well on new data: an overfitted model is a statistical model that contains more parameters than can be justified by the data [6]. This could be due for example to a low number of training samples compared to the high number of network parameters: the network is actually capable of remembering the exact characteristics of every single input image so it would memorize them, thus reducing its capabilities to perform well on previously unseen data. The image below shows this phenomenon: although the polynomial function exactly interpolates each point (input), it has a very specific shape that works extremely well for these points, but it is very likely that it wouldn't behave as good with new, not yet examined data; the linear function however, despite the average error that is obviously higher than the one of the polynomial, is expected to provide better predictions.

We can have a clue that a network is overfitting if the cost of the network

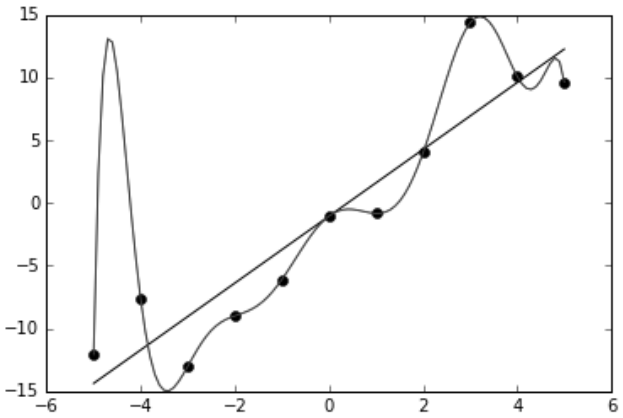


Figure 2.3: In the image we can see a noisy set of (roughly linear) points fitted into a linear and a polynomial function. Although the polynomial function exactly approximates this dataset, the linear function is expected to generalize better, which in turns would lead to better predictions.

keeps decreasing while accuracy on test data stops getting better: the network learned specific features of the input images which led the cost to become very low so there will be a certain point where, from there onward, accuracy will not improve anymore. Another indicator of an overfitting model is the accuracy when examining inputs from the different datasets: extremely high on training data (that could be as high as 100% in some cases), while quite low on test/previously unseen data.

To prevent the network from overfitting, applications can use several methods. The easiest one is to increase the number of training samples: doing so eliminates the main cause of the problem, forcing the network to learn general features of input data instead of exactly remember each sample. Unfortunately, this is not always possible since data could be expensive or difficult to retrieve; think about medical images or other restricted, domain-specific data. In such cases however, the training dataset could be *artificially* expanded using some pre-processing algorithm, for example applying some kinds of geometrical transformations if the data is a set of images.

Other ways for reducing overfitting are *regularization* techniques [11]. One of the most used methods among these techniques is the *weight decay* or *L2 regularization*. The intuition behind it is to add a term to the cost function that expresses the preference for the weights to have smaller L2 norm; For-

mally, the cost function becomes $C = C_0 + \frac{\lambda}{2n} \sum_w w^2$ where C_0 is the original, unregularized cost function and λ is the *regularization parameter*. Using this cost function, the network is forced to prefer lower weights values, with larger values being considered only if they cause C_0 to heavily decrease. This helps reducing overfitting because the network is more inclined to learn small weights, and as such they can be adjusted more easily than bigger ones.

Other regularization techniques include, among the others, *L1 regularization*, which works like L2 regularization but the added term measure weights by their L1 norm, and *dropout*. This last methods consists of randomly "ignoring" a subset of neurons in the hidden layers. We then train the network using a mini-batch, update the weights of the neurons that are still in the network and finally restore the ignored subset of neurons. After that, a different subset is ignored and the process starts again. The intuition behind this approach is that neurons in the same network have actually been trained like parts of different networks. When we use the network to get a prediction, the result can be considered as a sort of averaging over the results of such different networks. One of the heuristics which explain this method tells that distinct networks are likely to overfit in different ways, so averaging over their results will provide a quite reliable result. Another heuristic explains that a neuron is forced to learn stronger features of the input data, since it cannot rely on other neurons to help it [9]: “This technique reduces complex co-adaptations of neurons, since a neuron cannot rely on the presence of particular other neurons. It is, therefore, forced to learn more robust features that are useful in conjunction with many different random subsets of the other neurons”.

Apart from the parameters that are learned during the training (weights and biases), there are number of other ones which are called **hyperparameters**. These are all the parameters that are fixed and don’t change during the learning phase. Many of the values described earlier are hyperparameters: the learning rate, the batch size, the number of epochs, the regularization parameter and the number of epochs before early stopping occurs are all hyperparameters. Of course, these cannot be chosen randomly, but some heuristic exist to set good initial values. To adjust these values, usually the network is trained in a way such that the learning phase takes little time: this cause the network not to be very accurate, but doing so the designer can have fast feedbacks about network performance and tune hyperparameters without having to wait

a large amount of time for the network to be trained.

In this chapter the basics of neural networks have been explained. In the next one some particular structures of neural networks will be illustrated. These structures are useful in some specific tasks such as image segmentation, sequence prediction and others.

2.2 Convolutional Neural Networks

As modern neural networks rose in popularity, many researchers studied them to perform different tasks in various fields. As a result, new network layouts and techniques were born and are now widely used in both research and industrial applications. Even *deep neural networks* themselves are an evolution of single-layer neural networks, where the network contains many intermediate hidden layers.

Convolutional Neural Networks [4] use a special architecture that performs very well in image classification, and most of the existing image recognition networks use this kind of architecture, or a variation of it.

At the core of CNNs there is the concept of *local receptive fields*. They are regions of the input data each one connected to one neuron in the following layer allowing it to retrieve spatial information about that single region. In other words, a local receptive field allows to recognize a *feature* of the input data inside the region it describes; this is why they are used for image recognition networks. Imagine for example to have a network that accepts an image as its input and should be able to identify some objects inside that image. The network's input layer can be seen as a matrix with the same shape as the input images. Local receptive fields in this case are regions of adjacent pixels of the same size for all the neurons in the same layer. By limiting the input a neuron receives to only a small window allows it to learn a feature that is bound to that small region of input data, contrary to what would happen in the case of fully connected layers. A feature of the input data could be an edge, a shape, a line or something like that. The concept of feature anyway changes as the data flows through the network: layers closer to the start of the network will learn low-level features like lines and edges, while farther layers will learn more complex and abstract features due to the fact that the local

receptive fields they interact with have all the aggregate information from previous layers. Supposing that a network should detect a number in an image, the first convolutional layer will just see raw pixel data and could identify lines or edges. The results are then processed by another convolutional layer that could put together this information to recognize shapes like the upper circle of the number 9 or the crossing lines of the number 4. Finally, one last convolutional layer could aggregate all the input data to reconstruct the final number, and then output the result.

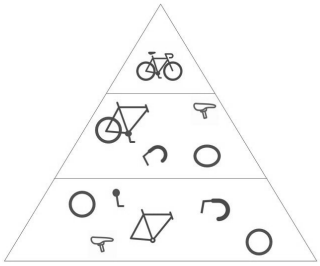


Figure 2.4: Example of different types of features recognized by convolutional layers at various levels. Each level of the pyramid detects the same type of feature, whose level of abstraction continuously increases as data advances into the network.

Local receptive fields are the core concept behind *convolutional layers*. In practice, in the context of convolutional layers local receptive fields have small dimensions compared to the input image, but span across the entire depth of the input, like the number of channels in an RGB image. These define a set of learnable *filters*, or *kernels*. At each learning step, this filter is used to perform a **convolution** between its values and the input data, where the output bidimensional *activation map* captures the spatial information contained in a particular receptive field. Since this spatial information is local to a restricted area of the input, it can only capture information about that area, but this process is performed for the entire input space, so the same kind of feature can be detected across all the input. For this, all the receptive fields in the same filter share the same weights, and the map from the input layer to the destination layer is called a *feature map*. Note that a single convolution layer can contain multiple feature maps, each one with its own set of weights. In

each layer, the kernel has to go from one edge to the other of the input image, and the amount of pixels it moves at each step is called *stride*, which is another hyperparameter. To control the size of the output feature maps, an additional parameter can be specified: the *padding*. With this, additional data is added to the input image to keep the output the same size as the input, or to expand it.

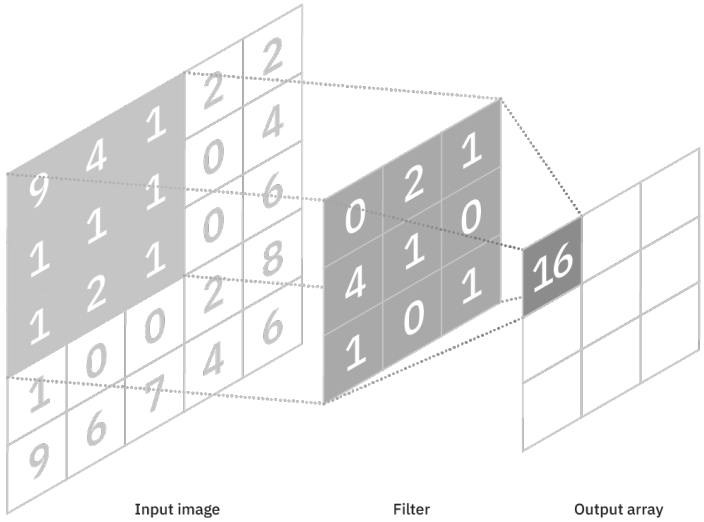


Figure 2.5: Example of convolution in a convolutional layer. Each value in the current window is multiplied by the respective weight in the filter and then all the results are summed to obtain the final result to insert in the output array.

An important type of layer that is usually paired with convolutional ones is the *pooling layer*. It works in a similar way to the convolutional one, but it does not contain any weights. It crosses all the input data in a similar way as the convolutional layer, but it instead applies an aggregator function to the values inside the receptive field. The pooling type can either be *max-pooling*, where the maximum value in the receptive field is outputted, and *average-pooling*, where the output value is the average of the receptive fields neuron values. While the latter could look like the best choice since it produces a value that, in some way, contains information about all the neurons in the receptive field, the former is the most used. The reason is that the neuron with the highest value is the one with the most impact across its receptive field, so by choosing it we can simplify the network layout but without losing much important information.

2.2.1 3D Convolutions

In this thesis we will train a convolutional network which containd *3D convolutions*. This type extends the 2-dimensional since it can move along all the three axes of the space. It is particularly useful when dealing with 3-dimensional data like in this case: as we explained before, a single projection performed using a *cone-beam geometry* is actually an image and consequently a full scan describes a volume. Similar as 2D convolutions which encode spatial relationships of objects in a 2D domain, 3D convolutions can describe the spatial relationships of objects in the 3D space.

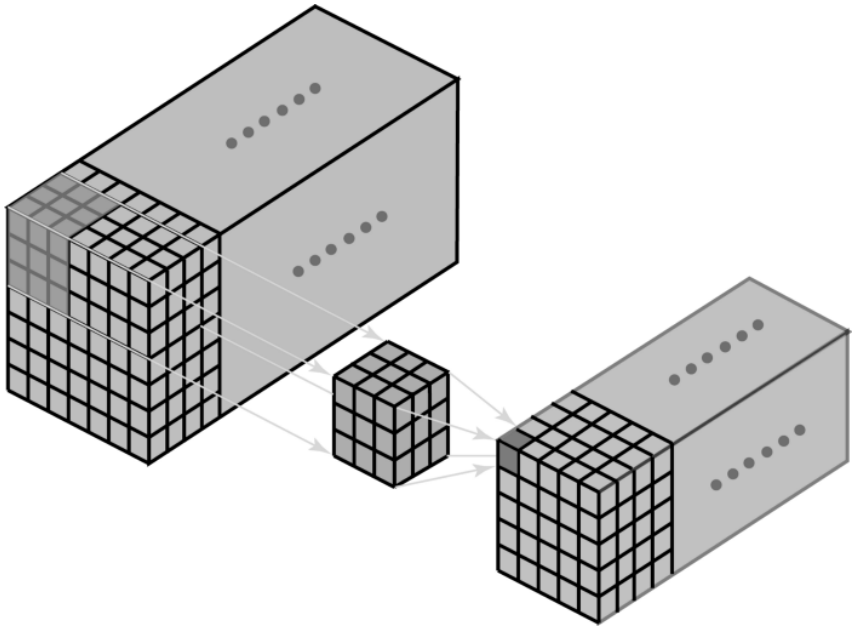


Figure 2.6: Graphical representation of a 3D convolution.

2.3 Neural Networks in practice

Over the last decade many frameworks for building and training neural networks have been created. Almost all provide GPU-accelerated functionalities to reduce training time by exploiting the parallelizability of the training algorithm. Three of the most famous are Tensorflow, Keras (interface to the Tensorflow library) and PyThorch. They provide many structures ready to be used like convolutional layers and LSTMs, different types of loss functions,

various activation functions, optimizers and so on. Keras is the framework used in this thesis because it provides a clear and nice interface to Tensorflow structures and there is a vast literature about it. In order to speed up the training of the network, it is useful to run the training algorithm on a GPU. Unfortunately, it is a component that may not be available in all computers and, even if it is, it could not be suitable for this job. All the ML libraries indeed use a set of APIs that is exclusive to NVIDIA cards called CUDA which allow algorithms to run on the GPU using their computational power.

Due to this constraint, Google Colaboratory [15], better known as *Colab*, has been used as the development platform for our neural network. It provides an isolated environment where users can run Python code in an interactive playground. It also offers the possibility of selecting a runtime with a dedicated high-end GPU: the most powerful one users are offered is an NVIDIA T4 with 16GB of RAM, a model specifically designed for accelerating machine learning programs. Although it is a very powerful card with a quite high amount of memory, this has often been a limit on the network size: it was very demanding in terms of memory due to the high number of feature channels in convolutional layers, the timestep count in LSTM layers and the resolution of input images. This caused the need to reduce the network size, both in terms of layers and feature channels, and we cannot know for sure if having more memory available could have improved the network performance.

Chapter 3

Technical Notes

In this chapter a brief description of algorithms and structures used in this thesis will be given. We will talk about back projection and iterative algorithms for tomographic reconstruction to provide an overview of how they work. After that, the building of a 3D phantoms dataset will be described and finally we will show the layout of the neural networks that will be used to enhance the tomographic images in the following chapter.

3.1 FDK

FDK [2] is an analytical reconstruction technique to reconstruct the original object starting from the projections proposed in 1984 by *Feldkamp, Davis ans Kress*. It derives from the Filtered Back Projection algorithm for reconstructing a function based on projection data but suitably changed for the 3D cone-beam geometry. In this case, as the name suggest, a *filter* is used to reduce the blur in low frequency areas, where many projections are combined, but at the same time it accentuates high frequencies ones (contrasting features). Due to this, the reconstructed object often contains some artifacts which look like trails or shadows. Such artifacts are even more evident when to measure a large number of projections, or the projections are not uniformly distributed over 180 or 360 degrees.

FDK provides approximate, direct reconstruction of a three-dimensional density function from a set of two-dimensional projections, reducing to the standard fan-beam formula in the plane that is perpendicular to the axis of

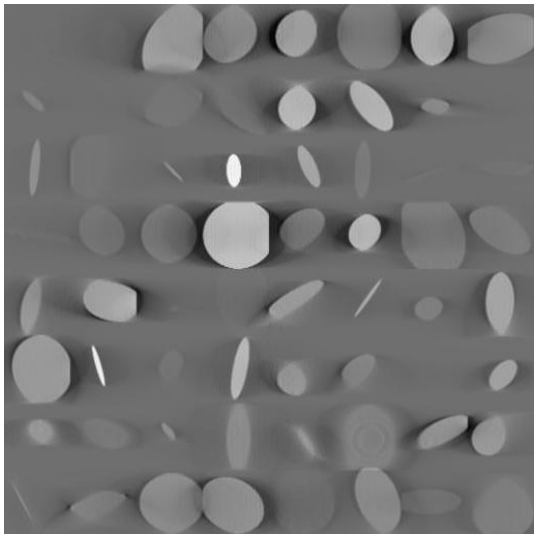


Figure 3.1: Example of a slice of a volume reconstructed using the FDK algorithm. Some artifacts that resemble trails or shadows can be seen expanding from actual objects.

rotation and contains the point source.

FDK has the advantage of being quite fast since it perform the backprojection in a single pass. Astra Toolbox provides a CUDA implementation of the algorithm [16] that can benefit from GPU acceleration, and it will be used to reconstruct the original volumes in the synthetic dataset that will be shown later.

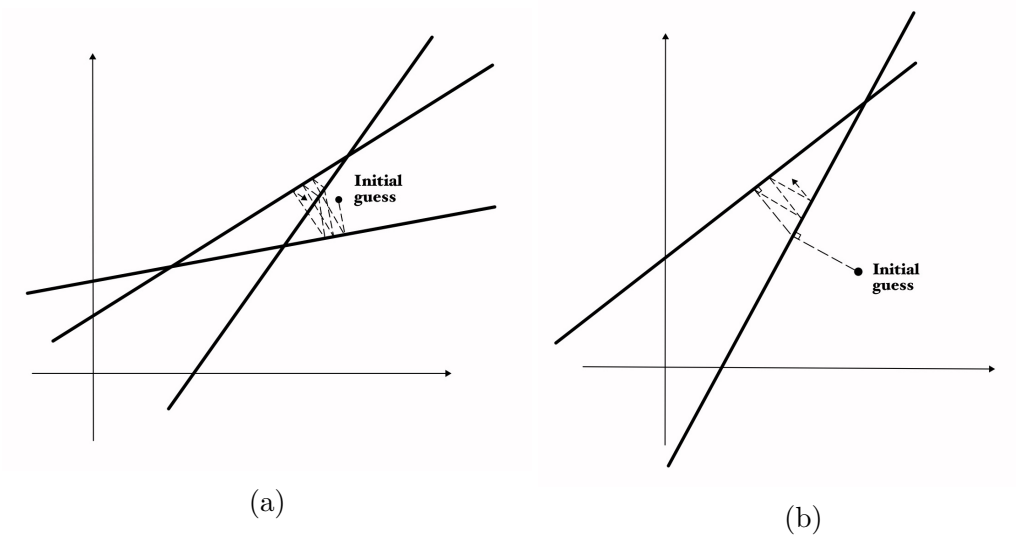
3.2 SIRT

Simultaneous Iterative Reconstructive Technique (SIRT), as the name suggests, is an **iterative** reconstruction technique. Iterative methods work by repeatedly improving their output by performing multiple passes. Specifically, SIRT calculates the reconstruction by resolving a system of equations which map the pixels (or voxels) of the reconstructed object to the rays values.

To briefly describe this process, we could initially think of an image with N pixels as a single point in an N-dimensional space. Each of the equations in the previously mentioned system then describes an hyperplane in such space. So, when the solution to the system exists, it is the intersection of all these hyperplanes, which in turns corresponds to the original image thought as a

point in the N-dimensional space. Geometrically, SIRT initially performs a guess on the position of the intersection point then, at each iteration, the current guessed point is alternately projected to each line. If a unique solution exists, the iterations will converge to it, so it is clear that the more iterations the algorithm is run for, the closer the calculated point will be to the actual solution of the equation system.

When dealing with real-world data however, the projection data could have some noise, which could come from a number of different sources. In the geometrical representation just given, this could lead to a change in the position of some hyperplanes, causing the system not to have anymore a unique solution. In this case, the guessed intersection point would not converge to a single point, but it would oscillate near the intersections of the hyperplanes [3].

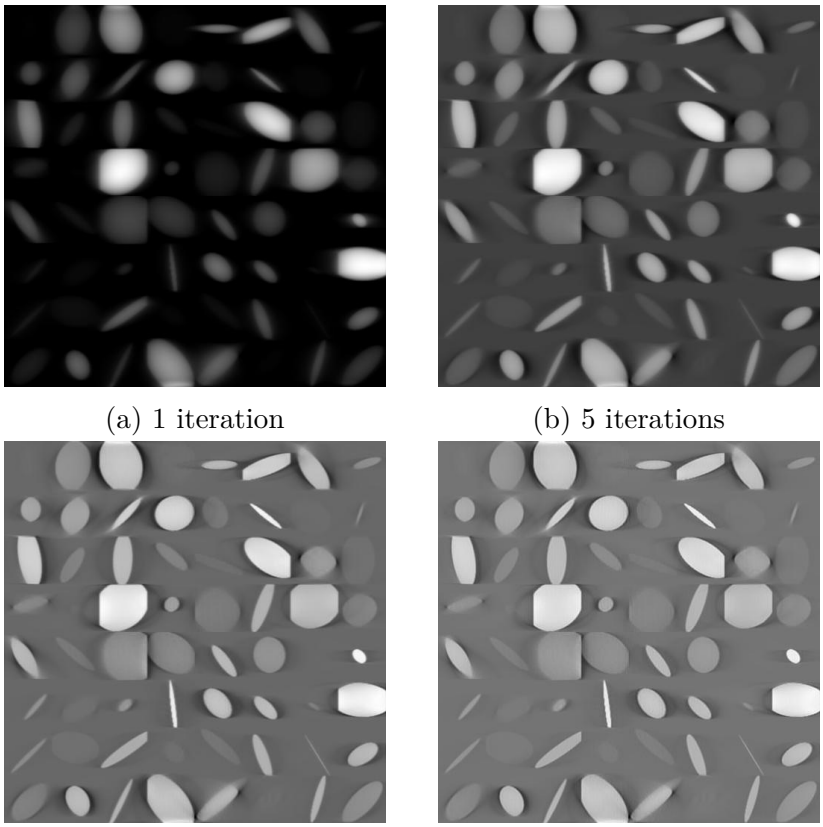


(a) (b)

Figure 3.2: In the images above a geometrical representation of how SIRT works can be seen. Image 3.2a provides a geometrical representation of how SIRT works. The lines are the equations of the system to solve assuming it has only two degrees of freedom, the intersection of the two being the solution to such system. The dashed line represents the "path" SIRT takes for reaching the convergence point. In 3.2b we can visualize what happens in presence of noisy data: a single solution to the system does not exist anymore, so the computed solution will keep oscillating near the triangle defined by the three lines.

At each step, the algorithm computes the change $\Delta f_j^{(i)}$ in the jth pixel caused by the ith equation in the system. The value of the jth cell is only

updated after all the changes relative to each line which intersects the cell's center have been calculated, and is assigned the average value of such changes. This process is performed for an arbitrary number of times, the higher this number the better the quality of the reconstructed object. An example of the improvements in the reconstruction as the iterations number increases is shown in figure 3.3.



(a) 1 iteration (b) 5 iterations

(c) 20 iterations (d) 50 iterations

Figure 3.3: Same slice of a volume reconstructed with the SIRT algorithm run for different numbers of iterations.

Although this algorithm can lead to good results, it is a computationally intensive calculation and performing many iterations can be very time-consuming. On the other hand, a low number of iterations cause the reconstruction to contain heavy artifacts. Astra Toolbox provides a CUDA implementation of the SIRT3D algorithm too [21].

source-origin distance	1000
origin-detector distance	10
detector pixel size	1
horizontal detector spacing	1
vertical detector spacing	1

Table 3.1: Parameters of the (virtual) scanner used for the scan

3.3 Dataset Generation

Since actual CT scans are difficult to acquire due to their strictly private nature, not many collections are available. To compensate for this, a synthetic dataset has been created. It consists of 200 phantoms of size $64 \times 512 \times 512$, where each one contains a 8×8 grid of 64 ellipsoids which vary in size, position and rotation inside their grid blocks. For the generation of a phantom, we built one ellipsoid at a time using a **Python** library called *pyellipsoid* [20] that returns a 3-dimensional array with ellipsoid data starting from center, radii and rotations data. In each phantom there is one ellipsoid of very small size since an element with such features is inaccurately represented by projections and thus the reconstruction algorithm could not be able to correctly backproject it.

Once the original phantoms dataset was built, the Astra Toolbox framework was used to simulate CT scans on it. Since we are mainly focusing on *limited-angle* CT, the scan interval was set to $[-50°, 50°]$ with a projection for each degree, resulting in a total of 101 scans per phantom. Scan parameters are illustrated in table 3.1.

The second dataset contains 20 samples of artificial breasts that closely resembles real ones, which will be used to generate some projections using a cone-beam geometry and then reconstructed using two different reconstruction algorithms, just like with the synthetic dataset.

After that, reconstructions have been calculated using both *FDK_CUDA* and *SIRT3D_CUDA* reconstruction algorithms of Astra Toolbox. The SIRT algorithm performed 5 iterations for each phantom but one, for which the algorithm has been run multiple times in order to showcase different levels of accuracy by changing the number of iterations it took each time.

3.4 Network Layout

For the scope of this thesis, various networks have been tested in order to show how *convolutional* ones could perform on tomographic image enhancement. All of these however share a common structure which is known to perform very well on image segmentation tasks, called **Unet** [12] after its shape. Image 3.4 show the architecture of the original Unet.

The input image is fed into the network, then the data then is propagated through the network along all possible paths (arrows) and finally it outputs the segmentation map. In the image, each dark box represents a multi-channel feature map, whose number of feature channels is specified on top of it and its x, y size in the lower left corner. Note that here the network was used to solve *segmentation* problems, hence the two feature map classes in the output: one for the relevant data and the other for the background data. The darker horizontal arrows denote a 3×3 convolution operation followed by a non-linear activation function, $ReLU$ in this case. Down-facing arrows represent 2×2 *max pooling* operations which halve the x, y dimensions of the data at each step. After each one, the number of feature channels is doubled. This sequence of convolutional operations followed by max pooling ones leads to a spatial contraction with a gradually increase of the **What** and decrease of the **Where**. For that, this part of the network is called the *contraction path*.

The Unet then contains also an *expansion path* (on the right side of the image) to create a high-resolution segmentation map. This expansion path consists of a sequence of *up convolutions* and *concatenations* (here represented by the lighter horizontal lines) with the corresponding high resolution features from the contraction path. The final convolution is performed using a 1×1 kernel to keep the image size the same but reducing the number of feature channels.

In our experiments, the GPU VRAM was the biggest constraint to the size of the network. At the time of writing, the highest-memory GPU available in Colab is either an NVIDIA Tesla P100 or an NVIDIA Tesla T4, both equipped with 16GB of VRAM. What this means in practice is that the network had to be downsized, particularly in the number of feature channels, or the batch size kept as low as 8 samples per batch. This caused the *convolutional LSTM networks* to be left out since they required a very high amount of memory

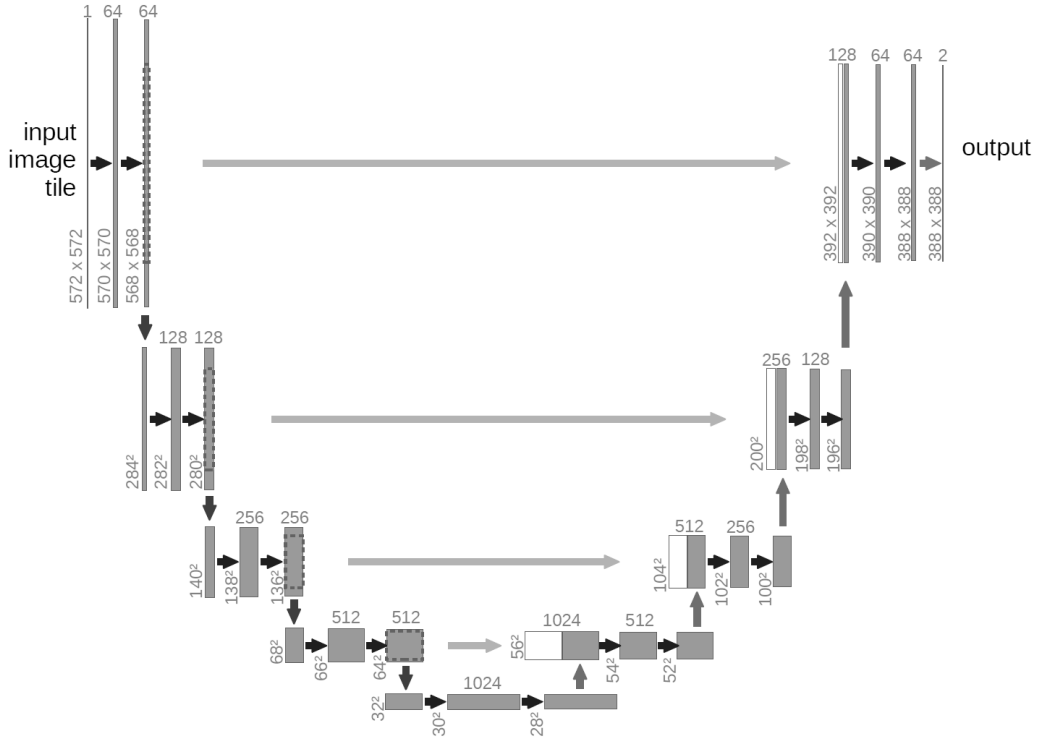


Figure 3.4: Architecture of the original Unet

even for the smaller ones, due to their need to store the weights of previous updates. The results from a convolutional LSTM network downsized to fit in GPU memory were not acceptable, so they will not be discussed. Due to this, we cannot say if such network layout could have been a viable solution given more resources.

For the synthetic dataset, we managed to build a Unet with 4 initial convolutions and 4 levels of max pooling operators (implying as many up convolutions). It was trained for 100 epochs using a batch size of 64 images (all the slices of a single phantom) on a dataset containing 50 samples. We also experimented with 3D convolutions to see if they could improve the results of the 2D versions but the increase in the dimensionality of convolutions layers implied a reduce in the feature maps number of these.

Beside the synthetic dataset, a real one called **VICTRE** was also used. It was taken from the *NBIA Cancer Image Archive* [18] and contains tomographic images obtained by performing cone-beam projections on a dummy object built to resemble a real breast.

3.5 Training of the Network

All the networks used in this thesis use the *ADAM* optimizer which has been demonstrated to be computationally efficient, have little memory requirements and be well suited for problems that are large in terms of data and/or parameters [13]. Its parameters were set accordingly to the batch size, dataset size and network layout.

For computing the loss during the training of the network we used the *Mean Squared Error* function, which is defined as follows:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{Y_i}\right)^2, \tag{3.1}$$

where n is the length of the vector of predictions generated from a sample of n data points, Y is the observed vector of variables being predicted and $\hat{Y}$ is the vector of such predictions [17]. In other words, it is the mean of the squares of the errors.

As often happens in medical imaging contexts, real-world tomographic data is very hard to obtain due to privacy concerns and legal implications. To make the VICTRE dataset more effectively represent one of these, this last limitation was taken into account, and only the data from 20 different "patients" was used, of which 18 were used as the training set and the other 2 were used to test the network. The same has been done for the synthetic dataset, resulting in a training set of 20 samples.

We did not use a validation set mainly for two reasons: first, this is not a classification problem, so a validation step wouldn't have been much useful; secondly, the limited number of available data made this unfeasible.

Chapter 4

Numerical Results

This chapter will be dedicated to the results of our experiments. We will show how different layouts of networks performed on enchancing FDK and SIRT reconstructions for both the synthetic and VICTRE datasets.

In order to evaluate the performance of the network and the accuracy of both reconstructions and predictions, we used the **PSNR** metric. It is defined as the ratio between the maximum possible power of a signal and the power of corrupting noise that affects the fidelity of its representation [19], so a higher value is associated with a less noisy signal. Formally, given a monochrome noise-free image I and its noisy representation K, PSNR is defined as

$$PSNR = 10 \cdot \log_{10} \left(\frac{MAX_I^2}{MSE} \right) \tag{4.1}$$

$$= 20 \cdot \log_{10} \left(\frac{MAX_I}{\sqrt{MSE}} \right) \tag{4.2}$$

$$= 20 \cdot \log_{10}(MAX_I) - 10 \cdot \log_{10}(MSE), \tag{4.3}$$

where MAX_I is the maximum value the signal can reach and MSE is calculated between I and K. In the case of images, if pixels are represented using 8 bits per sample, the value of MAX_I is 255. The ideal situation would be the absence of noise, or in other words $I = K$. This would mean that MSE is 0, making the PSNR value be *infinite* (or *undefined*, since there would be a division by 0).

4.1 Neural Networks on FDK reconstructions

In this section we will show the results we obtained on FDK images exclusively. They will be divided into two subsections, the first being dedicated to 2D convolutional networks and the latter to 3D ones.

4.1.1 2D Convolutional Network

The first experiment that has been made involved Neural Networks with 2-dimensional convolutions layers. These types of network take one image at a time and process it: the image at step t is completely independent from the other. We used the classical Unet architecture that can be seen in figure 3.4 but with small modifications. The complete network structure can be seen in table 4.1.

The numbers between the brackets in the Conv2D layers specify respectively the number of feature channels and the size of the kernel. For all of these layers, the *stride* value has been set to its default value of 1 for each dimension. The padding has been set to *same* in order to preserve the input dimensions by padding evenly to the left/right or up/down. Note also that for each layer except for the last one, where a **sigmoid** activation function has been used, the **ReLU** activation function has been chosen since it performs very well in CNNs and helps to mitigate the problem of vanishing gradient [8].

The gray lines in the table represent the *skip connections* from the contraction path to the expansion one, and the bold names indicate which layers are involved in the concatenation operation. A *dropout* layer has been inserted between the two central convolutions to prevent the network from overfitting since both training sets were small (18 entries for the VICTRE dataset and 20 for the synthetic one). The number between the brackets is the ratio of input units that are set to 0 at each learning step.

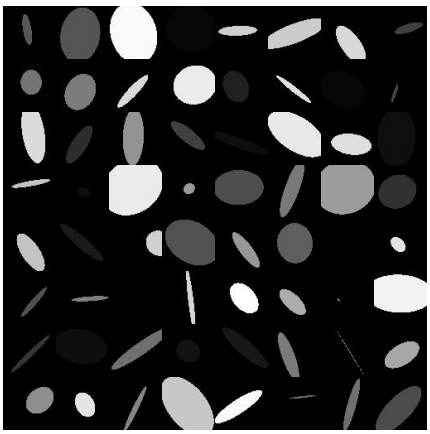
The same layout has been used for both the synthetic and VICTRE dataset with the only difference being the *batch size* that in the former case was equal to 16 since we sliced each volume in 4 groups of images, while in the latter case it was equal to 11, the entire set of slices of VICTRE elements.

In figure 4.1 we can see that the PSNR value of the prediction is much higher than the one of the reconstruction, while the MSE is lower. The pre-

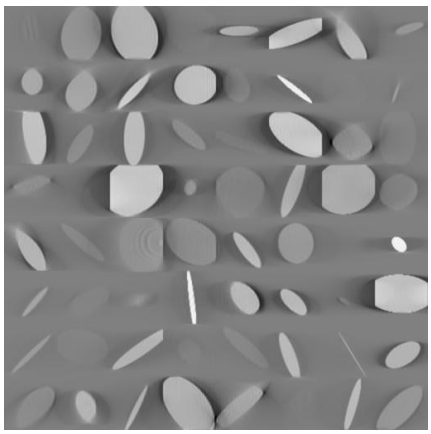
Layer type	Output shape
Conv2D(16, 3)	(512, 512, 32)
Conv2D(16, 3) [**skip1**]	(512, 512, 32)
MaxPooling2D	(256, 256, 32)
Conv2D(32, 3)	(256, 256, 64)
Conv2D(32, 3) [**skip2**]	(256, 256, 64)
MaxPooling2D	(128, 128, 64)
Conv2D(64)	(128, 128, 128)
Conv2D(64) [**skip3**]	(128, 128, 128)
MaxPooling2D	(64, 64, 128)
Conv2D(128, 3)	(64, 64, 256)
Conv2D(128, 3) [**skip4**]	(64, 64, 256)
MaxPooling2D	(32, 32, 256)
Conv2D(256, 3)	(32, 32, 512)
Dropout(0.2)	(32, 32, 512)
Conv2D(256, 3)	(32, 32, 512)
UpSampling2D	(64, 64, 512)
Conv2D(128, 2)	(64, 64, 256)
concat(**skip4**)	(64, 64, 512)
Conv(128, 3)	(64, 64, 256)
Conv(128, 3)	(64, 64, 256)
UpSampling2D	(128, 128, 256)
Conv2D(64, 2)	(128, 128, 128)
concat(**skip3**)	(128, 128, 256)
Conv(64, 3)	(128, 128, 128)
Conv(64, 3)	(128, 128, 128)
UpSampling2D	(256, 256, 128)
Conv2D(32, 2)	(256, 256, 64)
concat(**skip2**)	(256, 256, 128)
Conv(32, 3)	(256, 256, 64)
Conv(32, 3)	(256, 256, 64)
UpSampling2D	(512, 512, 64)
Conv2D(16, 2)	(512, 512, 32)
concat(**skip1**)	(512, 512, 64)
Conv(16, 3)	(512, 512, 32)
Conv(16, 3)	(512, 512, 32)
Conv2D(1, 1)	(512, 512, 1)

Table 4.1: Structure of the 2D Convolutional Neural Network.

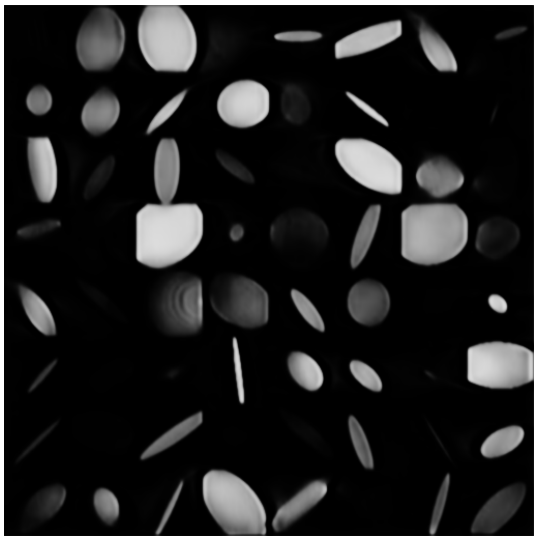
dicted image has indeed a better contrast and many imperfections are removed. However, it is not very close to the original one, since we can see that many particulars are missing or badly represented; FDK is known to produce inaccurate output when dealing with limited-angle tomographic data and our network only cannot entirely reconstructed all the missing information.



(a) Original image



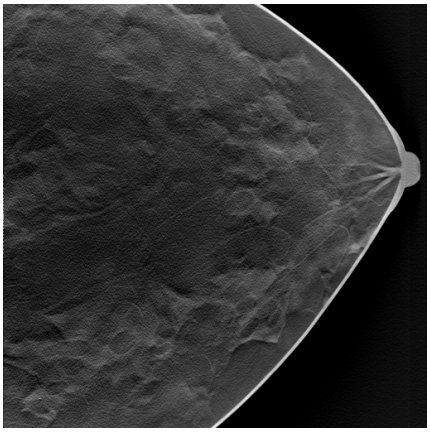
(b) FDK reconstruction:
MSE: 0.190,
PSNR: 7.19



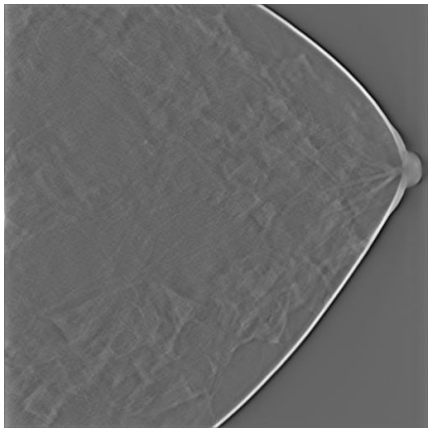
(c) NN output:
MSE: 0.013,
PSNR: 18.7

Figure 4.1: Example slice of a volume reconstructed using FDK along the original and NN-post-processed ones. In image 4.1c we can see that many details of the original image are missing in the NN reconstruction and it wrongly treats artifacts of the reconstruction as valid objects.

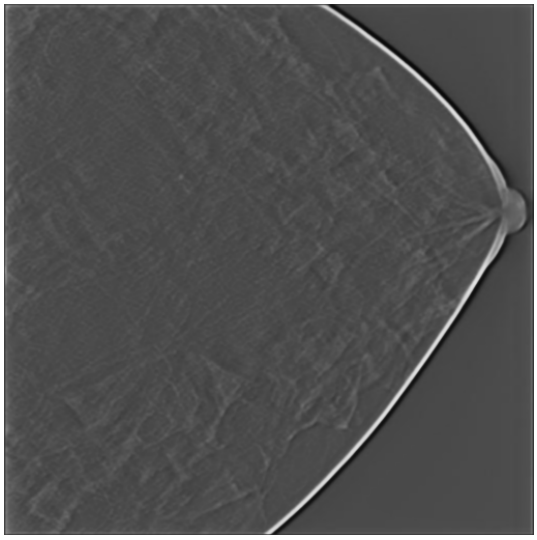
We can observe similar results with the VICTRE dataset. Figure 4.2 summarizes them. We can note that the reconstructed image has a low dynamic range and many details are blended into the background. Applying the Neural Network post-processing slightly improves this but the result is still quite off from the original image.



(a) Original image



(b) FDK reconstruction:
MSE: 0.041,
PSNR: 13.8

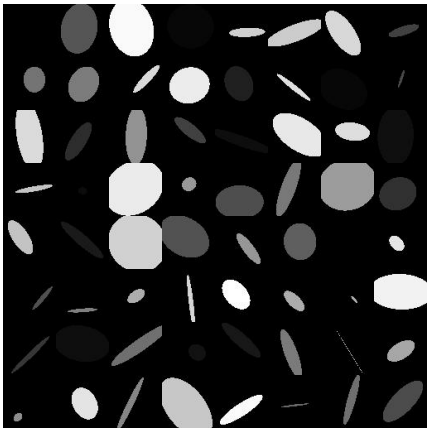


(c) NN output:
MSE: 0.019,
PSNR: 17.1

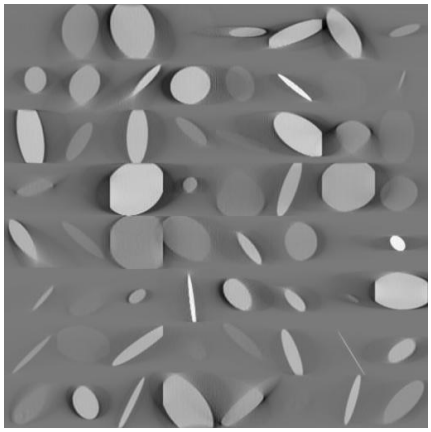
Figure 4.2: Example slice of a volume from the VICTRE dataset reconstructed using FDK along the original and NN-post-processed ones.

4.1.2 3D Convolutional Network

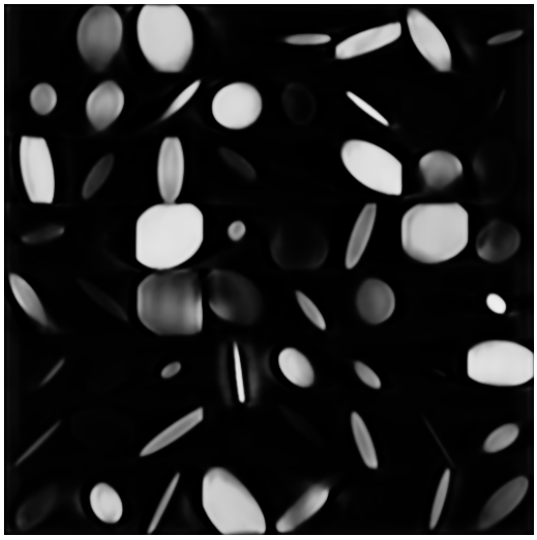
Since we are dealing with volumes, it is possible to build a Convolutional Neural Network which takes advantage of the third dimension to improve its results when enhancing reconstructed volumes. The network has the same layout as before, the only difference being the type of the convolutions and max-pooling/upsampling operations which now work with 3D entities.



(a) Original image



(b) FDK reconstruction:
MSE: 0.156,
PSNR: 8.05



(c) NN output:
MSE: 0.008,
PSNR: 20.7

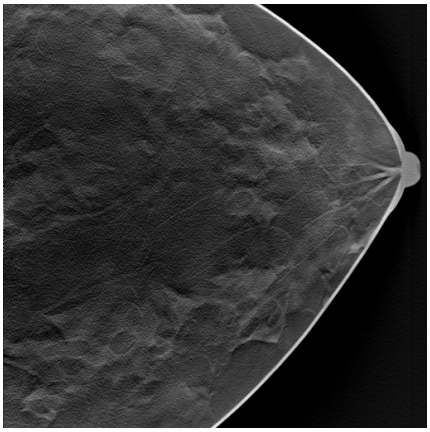
Figure 4.3: Example slice of a FDK volume reconstruction from the synthetic dataset along the original and 3D CNN-post-processed ones.

The first result that will be showed involves the ellipsoids dataset. In this case, an important difference from the 2D network is the batch size. This time the entire 64 slices of the volume must be used to preserve the spatial information since separating them would impact on the 3D convolutions effectiveness. Due to this necessity, the GPU memory constraint is even more strict, since a size increase in the third dimension has to be compensated somewhere else (assuming the amount of GPU memory is fixed), for example in the number of feature channels. In this specific instance, the number of feature channels of the convolutional layers needed to be halved.

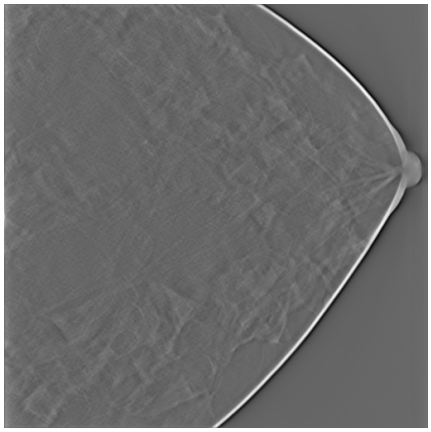
We reduced the amount of feature channels accordingly to half their number in the previously mentioned networks. This translates into less network parameters, which in turn leads to a weaker learning process and a lower improvement of the prediction. For this, the dropout layer has been removed. In figure 4.3 we can see the results of this approach. As before, the value of MSE and PSNR have improved, and we can see that some artifacts are correctly removed and most of the objects are reconstructed correctly. Unfortunately, many others are not, and there are even some artifacts that resemble actual object which our network failed to remove. This is highly likely due to the low number of parameters in our neural network, but we can see that where the reconstruction is not severely "damaged" it provides quite good results.

With the VICTRE dataset we can see an improvement from the 2D version of the CNN. Here, since there are only 11 slices per volume, some minor changes had to be done in the network layout. Due to the layers now working in the 3-dimensional space, max-pooling and upsampling now change the size of this dimension as well. Since the low and odd size in this case would not make possible to halve it for the needed number of times, max-pooling and upsampling layers were edited to keep the third dimension size the same.

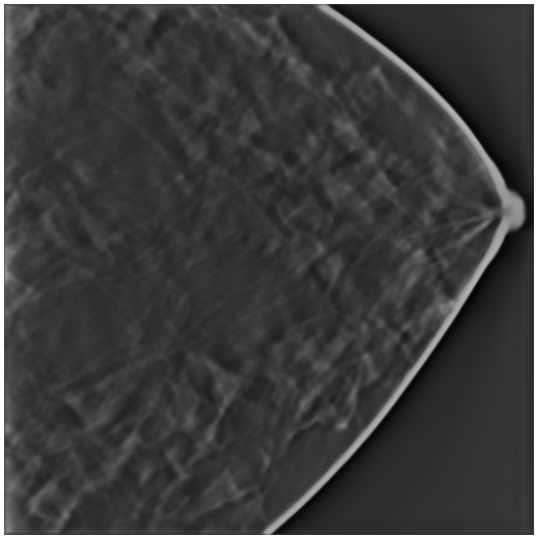
In figure 4.4 we can see that, although it seems slightly out of focus, the Neural Network output improves the quality of the FDK reconstructions and more details can be identified there than in the initial FDK image.



(a) Original image



(b) FDK reconstruction:
MSE: 0.042,
PSNR: 13.7



(c) NN output:
MSE: 0.011,
PSNR: 19.2

Figure 4.4: Example slice of a volume from the VICTRE dataset reconstructed using FDK along the original and ones post-processed by the 3D convolutional network.

Overall, in the case of FDK reconstructions the results seem underwhelming since various details of the original volume are missing from the network output: this could be linked to two factors. The first is that the images the Neural Network has to enhance are primarily not very good reconstructions, so the network can only extract little useful information from them; the other reason is the network size: as we talked earlier, the memory constraint we faced

when training the network on Google Colab forced us to limit the network size, resulting in a potential decrease of its accuracy. Having more performant hardware could probably have benefited the outcome, but we cannot know for sure.

4.2 Neural Networks on SIRT reconstructions

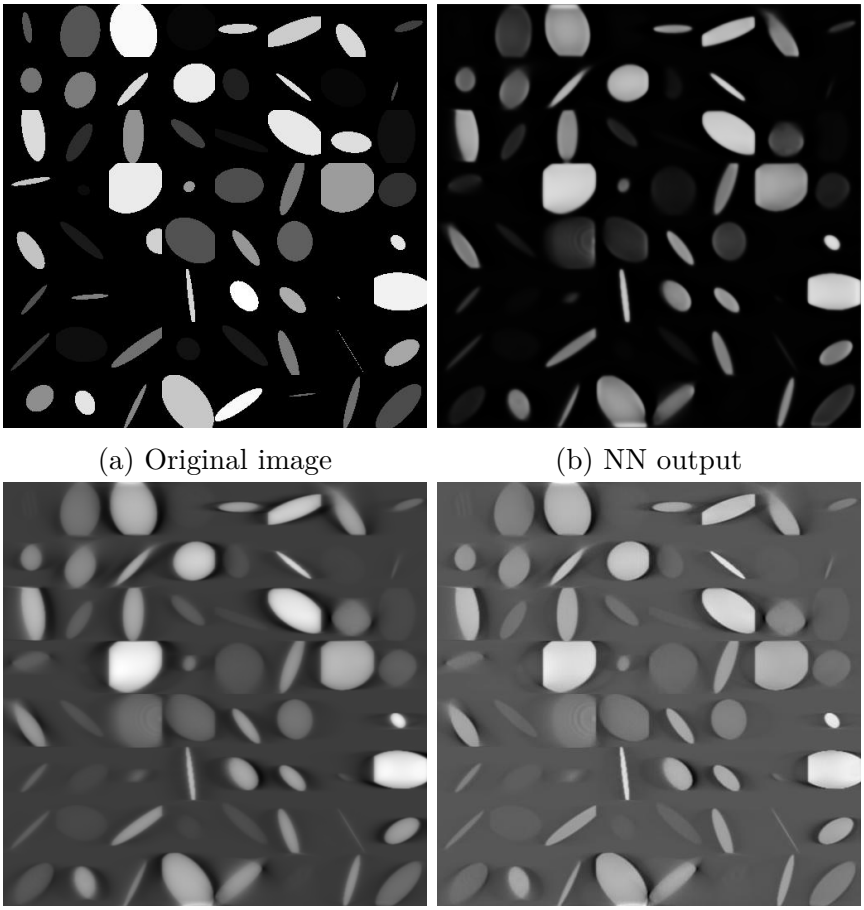
In this chapter we will show the improvements we can obtain by keeping the number of iterations of the SIRT algorithm low and redirect its output to a neural network in order to improve the accuracy of the reconstruction. For the tests, we used the same network layout we used for enhancing FDK reconstruction but using SIRT reconstructions as its input data.

To measure the effectiveness of this approach, we tried to reconstruct many times an object from the same set of projections but changing each time the number of iterations of the SIRT algorithm. For each reconstruction, we calculated the MSE between the reconstruction and the original image, as well as the PSNR metric. Then, we used the neural network to enhance the projected image obtained from 5 iterations of the algorithm and measured the MSE and PSNR metrics of the output.

4.2.1 2D Convolutional Network

For the SIRT reconstruction we used the exact same network as the 2D FDK case study. To get an idea of how it performs in enhancing the reconstruction, we used the network to get a prediction starting from the reconstructed volume and then calculated the MSE and PSNR values on it as before.

From figure 4.5 we can see that the predicted value helps enhancing the raw 5-iterations SIRT reconstruction and in some ways it provides better results than 20 iterations of the SIRT algorithm. It is however quite similar to the result we obtained starting from the FDK version in figure 4.1c.



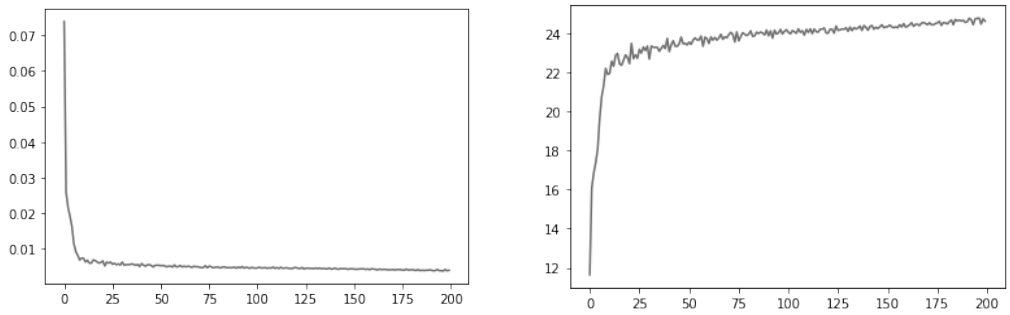
(a) Original image (b) NN output

(c) SIRT reconstruction (5 iterations) (d) SIRT reconstruction (20 iterations)

Figure 4.5: Example slice of a volume from the synthetic dataset reconstructed using SIRT after 5 iterations along the original, 20-iterations version and post-processed ones.

The same network has then been trained on the VICTRE dataset but changing the number of the feature channels in the initial convolution to 32 (and subsequent ones accordingly), thanks to the reduced size of the third dimension. This means that the network is able to recognize a greater number of feature types in the input images and, at least in theory, it should perform better.

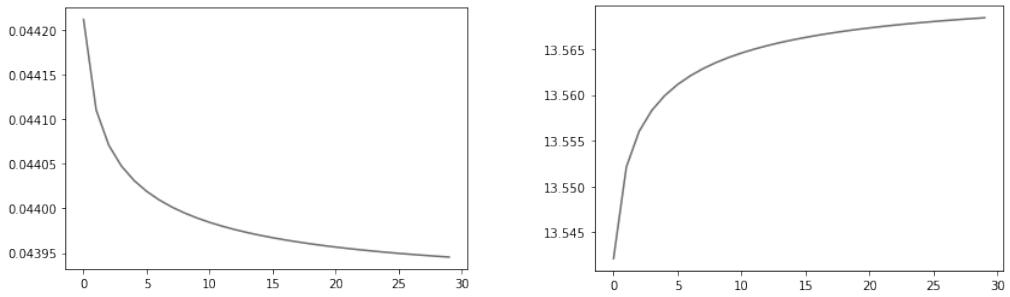
In figure 4.6 we can see the graphs of both the MSE and PSNR values during the training of the network. While MSE does not improve much after iteration 100, PSNR continues to increase until the end of the training epochs, reaching a value of about 25.



(a) MSE (b) PSNR

Figure 4.6: History of MSE and PSNR values during the training of the network.

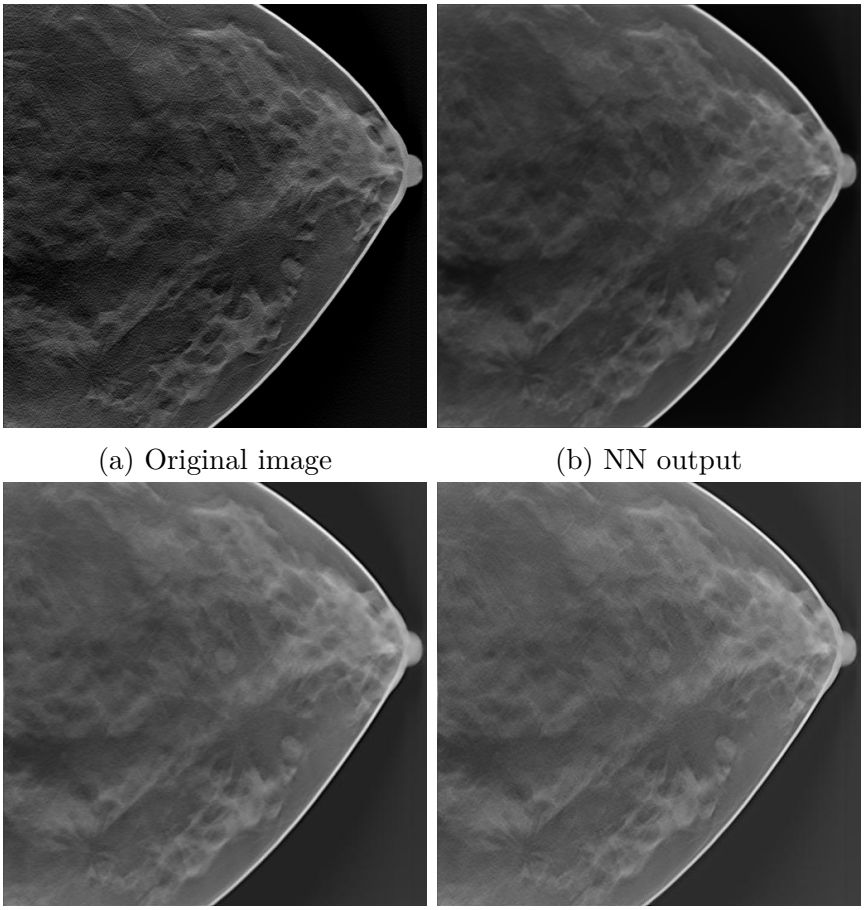
To have an idea of the MSE and PSNR values of SIRT reconstructions based on the iterations number change, we calculated some reconstructions from the same set of projections using SIRT with different iterations counts. From figure 4.7 we can see how these values change.



(a) MSE (b) PSNR

Figure 4.7: History of MSE and PSNR values for different numbers of iterations of the SIRT algorithm.

In this case, the network provides good results which can be observed in figure 4.8. We can see that in the neural network prediction dark spots are accentuated while in both raw SIRT reconstructions the dynamic range is lower, but at the same time it preserves the quality of the initial reconstruction image. This is pointed out by the MSE and PSNR values, which are respectively equal to 0.004 and 23.3. Referring to figure 4.7, we can see that these values are way better even than the ones of a raw SIRT reconstruciton which runs for 30 iterations. This technique could then be a valid approach for enhancing limited-angle SIRT reconstructions.



(a) Original image (b) NN output

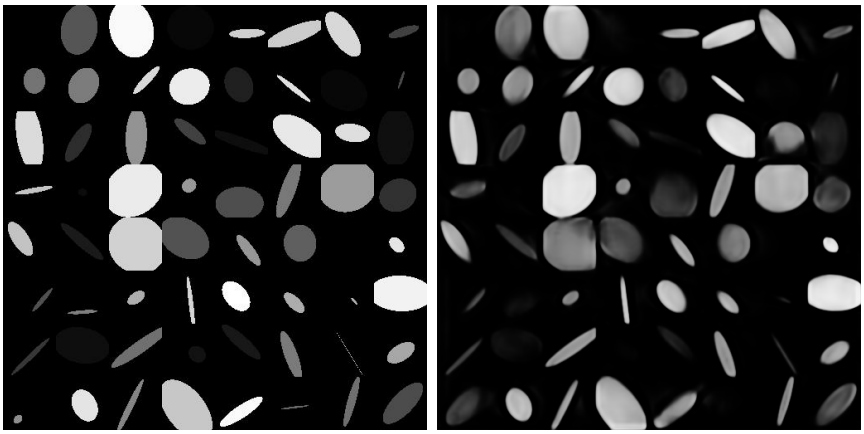
(c) SIRT reconstruction (5 iterations) (d) SIRT reconstruction (25 iterations)

Figure 4.8: Example slice of a volume from the synthetic dataset reconstructed using SIRT after 5 iterations along the original, 20-iterations version and post-processed ones.

4.2.2 3D Convolutional Network

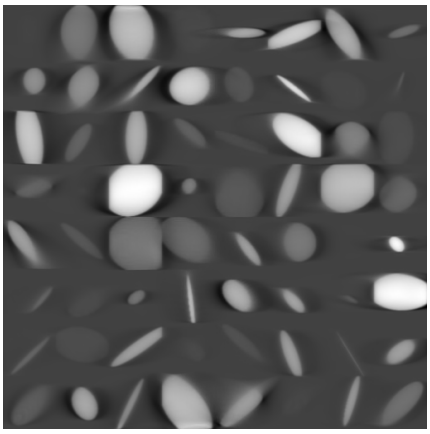
The same 3D convolutional network used for FDK images have been tested for SIRT reconstructions too, but this time it has been trained with 5-iterations SIRT reconstructions. Figure 4.9 shows the comparison between slices of the same volume in different states. We can see that projection trails are for the most removed and the overall image quality is quite good. Even some of the ellipsoids which tend to blend the most with the background seem to be reconstructed, although their shape is not very clear. Other ones still contain some artifacts, but it has to be remembered that this is the result of a Unet-

based 3D convolutional neural network with only 8 feature channels in the starting convolution. As a comparison, the original Unet had 64 of them.

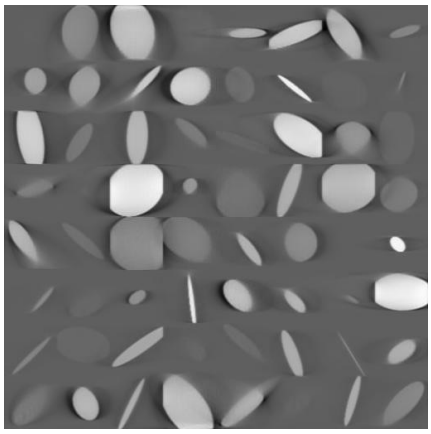


(a) Original image

(b) NN output



(c) SIRT reconstruction (5 iterations)

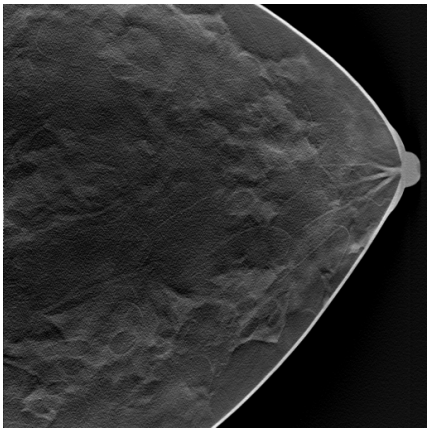


(d) SIRT reconstruction (20 iterations)

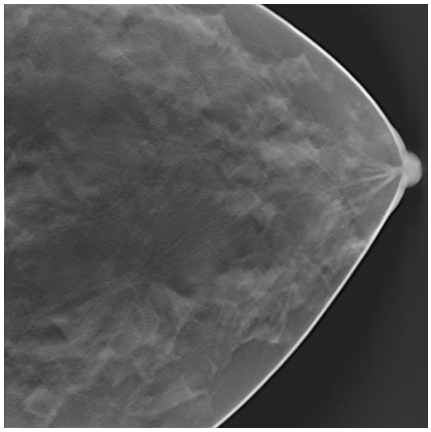
Figure 4.9: Example slice of a volume from the synthetic dataset reconstructed using SIRT after 5 iterations along the original, 20-iterations version and post-processed ones.

Finally, the 3D convolutional neural network has been tweaked and then trained on the VICTRE dataset. Its reduced number of slices allowed double the number of feature channels in convolutional layers than when dealing with the ellipsoids dataset. From figure 4.10 we can see that the neural network output contains a rather little amount of noise as indicated by the value of the PSNR metric which is 24.8. The MSE itself is very low, and we can see that the image from the prediction approximates quite well the original one. One negative aspect of this approach is the blurriness of the prediction; anyway,

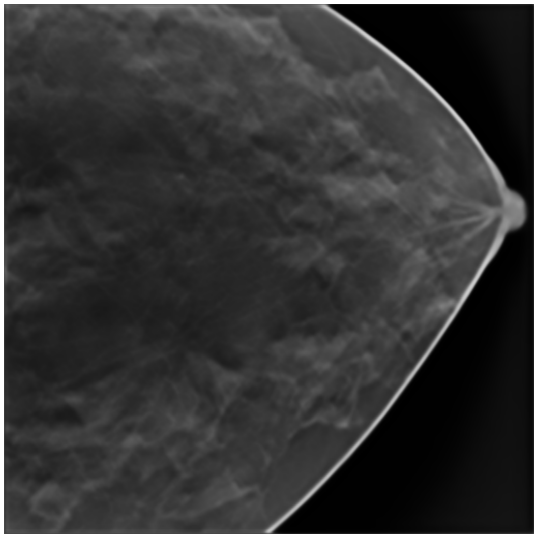
this is the result obtained starting from a SIRT reconstruction performed in only 5 steps, so this result can be considered quite good.



(a) Original image



(b) SIRT reconstruction



(c) NN output:
MSE: 0.003,
PSNR: 24.8

Figure 4.10: Example slice of a volume from the victre dataset reconstructed using SIRT after 5 iterations along the original and neural network-post-processed ones.

Chapter 5

Conclusions

In this work we showed different approaches for enhancing tomographic images using *Convolutional Neural Networks*. We generated a synthetic dataset of volumes made of ellipsoids, then calculated the projections using a cone-beam geometry and finally used these to build a reconstruction using both FDK and SIRT algorithms. The same projection/reconstruction work has been done for a dataset containing a reduced number of samples of artificial breasts built to resemble real ones as closely as possible.

After this initial generation procedure, we performed some tweaks to the Unet structure to adapt it to our use case. We first tried to enhance FDK reconstructions using both 2D and 3D convolutional networks but the inaccuracy of this algorithm when dealing with limited-angle projections combined with a reduced network size due to memory constraint didn't lead to useful results.

Then we used SIRT reconstructions to train the two types of networks. Using this dataset, we managed to get encouraging results, especially with 3D convolutional networks. The additional spatial information given by the third dimension allowed more information to be available to make an estimate, so the output volume was quite accurate, for both the synthetic and VICTRE dataset. The 3D convolutional network could also improve a 5-iterations SIRT reconstruction to a point that neither a 20-iterations one could reach.

Our results have been limited by the low number of samples available to train the networks and by the hardware we used. The first problem unfortunately is not easily solvable since real-world data gathering is strictly regulated

by privacy laws and for this is is quite difficult to build big datasets to use as training samples for a neural network. An artifically augmented dataset could however be built, but there would be the need to closely pay attention not to alter the intrinsic features of that class of data. Problems about the performance of the hardware can be solved by adding more memory or compute units to the device the network is being trained on, or using a federated environment to split the work between multiple computing entities, but this procedure can add some complexity to the process and could become economically very expensive.

In conclusion, we demonstrated that a neural network can be implemented and trained to enhance tomographic image data. We exposed the difference between FDK and SIRT reconstructions, where the latter preserve a higher level of detail than the former. We then showed that the predictions a neural network trained on SIRT reconstruction makes turn out to have a better quality than higher-iterations count ones, particularly using 3D convolutional networks. This could provide a good starting point for future improvements on the use of 3D convolutional networks for enhancing tomographic reconstrucions volumes.

Bibliography

[1] F. Rosenblatt. "The perceptron: A probabilistic model for information storage and organization in the brain". In: *Psychological Review* 65(6) (1958). URL: https://doi.apa.org/doi/10.1037/h0042519.

[2] L. A. Feldkamp, L. C. Davis, and J. W. Kress. "Practical cone-beam algorithm". In: *J. Opt. Soc. Am. A* 1.6 (June 1984), pp. 612–619. DOI: 10.1364/JOSAA.1.000612. URL: http://josaa.osa.org/abstract.cfm?URI=josaa-1-6-612.

[3] A C Kak and M Slaney. "Principles of computerized tomographic imaging". In: (Jan. 1988). URL: https://www.osti.gov/biblio/5813672.

[4] Y. Lecun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[5] Gabor T. Herman. *Fundamentals of Computerized Tomography: Image Reconstruction from Projections*. Springer Science & Business Media, 2009. ISBN: 9781846287237.

[6] B. S. Everitt and A. Skrondal. *The Cambridge Dictionary of Statistics*. Cambridge University Press, 2010. ISBN: 9780521766999.

[7] *Heaviside step function — Wikipedia, The Free Encyclopedia*. 2010. URL: https://en.wikipedia.org/wiki/Heaviside_step_function. (accessed: 21.02.2021).

[8] Vinod Nair and Geoffrey E. Hinton. "Rectified Linear Units Improve Restricted Boltzmann Machines". In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*. ICML'10. Haifa, Israel: Omnipress, 2010, pp. 807–814. ISBN: 9781605589077.

[9] Geoffrey Hinton et al. “Improving neural networks by preventing co-adaptation of feature detectors”. In: *arXiv preprint* arXiv (July 2012).

[10] Jürgen Schmidhuber. “Deep Learning in Neural Networks: An Overview”. In: *CoRR* abs/1404.7828 (2014). arXiv: 1404.7828. URL: http://arxiv.org/abs/1404.7828.

[11] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep learning”. In: *Nature* 521.7553 (May 2015), pp. 436–444. ISSN: 1476-4687. DOI: 10.1038/nature14539. URL: https://doi.org/10.1038/nature14539.

[12] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “U-Net: Convolutional Networks for Biomedical Image Segmentation”. In: *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*. Ed. by Nassir Navab et al. Cham: Springer International Publishing, 2015, pp. 234–241. ISBN: 978-3-319-24574-4.

[13] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[14] Seymour A. Papert Marvin Minsky. *Perceptrons: An Introduction to Computational Geometry*. The MIT Press, 2017. ISBN: 9780262343930.

[15] *Colab FAQ*. URL: https://research.google.com/colaboratory/faq.html. (accessed: 20.02.2021).

[16] *FDK_CUDA*. URL: http://www.astra-toolbox.com/docs/algs/FDK_CUDA.html. (accessed: 21.02.2021).

[17] *Mean Squared Error — Wikipedia, The Free Encyclopedia*. URL: https://en.wikipedia.org/wiki/Mean_squared_error#Definition_and_basic_properties. (accessed: 27.02.2021).

[18] *National Biomedical Imaging Archive Web Site*. URL: https://imaging.nci.nih.gov/nbia-search-cover/. (accessed: 27.02.2021).

[19] *Peak Signal to Noise Ratio — Wikipedia, The Free Encyclopedia*. URL: https://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio. (accessed: 27.02.2021).

[20] *pyellipsoid*. URL: https://github.com/ashkarin/pyellipsoid. (accessed: 23.02.2021).

[21] *SIRT3D_CUDA*. URL: `http://www.astra-toolbox.com/docs/algs/SIRT3D_CUDA.html`. (accessed: 21.02.2021).