Deep learning approach for denoising Monte Carlo dose distribution in proton therapy

Promotor: John Lee

Jorge Ricardo Asensi Madrigal
2018, June.

Document delivered on: 10 June, 2018

Acknowledgments

The current thesis is the result of a semester of daily work. Almost five months ago I came to Brussels from my lands. A semester of adventures, making new friends and learning about biomedical stuff and even about myself.

It was a Tuesday when I arrived to MIRO's laboratory and I met John and Umair for my first time. From that very moment I felt comfortable there. They told me everything about *Deep Learning* and the project Umair was developing. They offer me a plenty of different projects, but two weeks after, I chose Kevin's offer, and I started this thesis.

This acknowledgment should refer such a huge number of people that it is technically impossible to mention everyone. But I would like to mention my supervisors, John Lee, Kevin Souris and Umair Javaid.

I would like to thank the new friends I made in Brussels: Elena Escudero, Ana, Nahia, Rubén, Paula; but specially thanks to Roger and Elena. I will never forget the days we spent travelling, visiting Brussels, trying new beers and enjoying our conversations and *salseos*.

My Valencia friends deserve a mention here too. From Rubén, Paco, Richi, Sonia, Carmen, Heura, Oscar, Claudia and Cristina; to Fernando, Carlos, Toni, Pablo, Pedro Miguel, Juanfer, Alberto, Ryan, Mari, Laura, and Marta. Every one of you gave me the proper thing even in the bad and the good moments.

To *La Resistencia* for our marvelous summer trip I will ever remember, and for every incredible single moment we spent before and after that trip.

To my parents and my grandparents, both those who are and those who are not.

To my cousin Antonio and his beautiful family.

To my girlfriend Mónica.

And to Martina.

ABSTRACT

Radiation therapy planning requires to simulate the dose distribution on a CT patient image. It is used an algorithm based on Monte Carlo to generate that simulation, but this algorithm produces some noise that need to be removed.

Convolutional Neural Networks (CNN) have improved the state-of-the-art in the recent years by recognizing hierarchical features on an image. The purpose of the current work is to build a Neural Network that take a 3D Monte Carlo Dose Distribution as an input and denoise it through the different layers it includes, to use it in hospital practice. The quality of Monte Carlo generated images depends on the number of particles employed, consequently, improving the quality of the images involves an exponential increase of the computing time. Simulations generated with 1e9 particles could be considered as free-noisy because the residual noise they have does not compromise the clinical application. We filtered distributions generated with 1e7 and 1e6 particles, what result in one minute and 10 seconds of computing, respectively. Both networks architectures are U-Net, commonly used in the segmentation task, both of them exceed the state-of-the-art, achieving a signal-to-noise ratio of 73.03 and 35.69 respectively, and they spend 45 seconds around on filtering the whole 3D-image.

RESUMEN

La planificación en radioterapia requiere simular la distribución de la dosis basada en la imagen del paciente, obtenida mediante Tomografía Axial Computarizada. La forma de generar esta simulación es mediante algoritmos basados en Montecarlo, pero, estos algoritmos producen un cierto ruido que debe ser eliminado.

Las redes neuronales convolucionales (RNC) han revolucionado el *state-of-the-art* en los últimos años mediante el reconocimiento de características de forma jerarquizada. El propósito del presente trabajo es la construcción de una red neuronal que tome por entrada una distribución de dosis tridimensional generada mediante algoritmos Montecarlo y, mediante los diferentes parámetros que esta incluye, remover el ruido que tenga para obtener una imagen perfectamente funcional en la práctica hospitalaria. La calidad de las imágenes generadas por Montecarlo depende del número de partículas que se modelicen, por lo que, mejorar la calidad de las mismas implica un aumento exponencial del tiempo de cálculo. Las distribuciones de dosis consideradas "libres de ruido", requieren el uso de 1e9 partículas, lo que implica una hora de computación. Nosotros hemos filtrado imágenes generadas con 1e7 y 1e6, que conllevan, respectivamente, un minuto y 10 segundos de computación. Ambas redes neuronales se corresponden a la arquitectura U-Net, ampliamente usada en el área de segmentación de imágenes. Ambas redes superan el *state-of-the-art* actual, obteniendo un *signal-to-noise* ratio de 73.03 y 35.69 respectivamente, y empleando, aproximadamente 45 segundos en filtrar la imagen 3D.

CONTENTS

Table of figures

Figures

Tables

Graphs

INTRODUCTION

Cancer is a major public health problem in the world. Only in United States, 1,688,780 people suffered cancer last year, and 600,920 deaths were projected to occur [1]. That very year, the number of tumor deaths in Europe was 1,373,500. These statistics show us cancer is the second most common illness in Europe, behind cardiovascular problems [2].

Common treatments for cancer are surgery, chemotherapy and radiation therapy. The area of this project is radiation therapy, which consists in applying ionizing beams to the tumor with the aim to destroy every cancer cell, but preventing non tumor cells of any damage. Furthermore, there are three main types of radiation therapy, the one made with photons, other one made with electrons, and the one made with protons. The first option is the most used technique in the world, and it has been developed for years, but, recently, proton therapy has been revealed as a more precise and safe technique [3], because the beam dispersion is lower than photons one due to its distribution, called Bragg Peak. This distribution has three different parts: Firstly, there is a constant low dose area where protons do not interact much with the matter. Afterwards, there is the Bragg peak itself. It is a narrow area where all the protons interact with the environment and release their energy. Finally, behind the Bragg peak, no energy is delivered.

For keeping safe healthy cells, physicists try different beams on a patient CT-model, seeking the best angles, intensities and dimensions. They attempt to minimize the dose out of the tumor and maximize it inside the tumor. There are two class of algorithms to model the behavior of real beams: one of them is based on analytical calculations, which is fast but approximate, not exact, and other one based on Monte Carlo method [4]. This last method is more accurate, but it requires a lot of time to do a good prediction, what is not assumable for clinical practice. Modeling protons behavior is more complicated than photons, because the Bragg Peak is very sensitive to little changes, and there is a necessity of having good simulations as we are applying protons for preventing damaging healthy tissues. We can spend less computational time to generate dose distributions, but it results in a noisy image that cannot be used.

The main problem of these dose distribution is that the noise distribution is completely unknown. Best noise filters have been made by discovering the distribution of that noise and considering it. Obviously, we cannot apply these filters to our dose distributions. That is the reason why we are going to use Deep Learning filters.

The aim of this project is to filter these noisy images by using Convolutional Neural Network [5] that could learn very complex noise distributions. The methods of the current thesis should be applied in clinical practice, so we need a fast denoise system which provides a considerable good performance. Slow systems will keep the current problem of the Monte Carlo dose distributions, and bad performances of the system will give unrealistic doses, something undesired.

An Artificial Neural Network (ANN) is a virtual Machine Learning system that can learn high abstract information from one signal (images, sounds recordings, bio signals, etc.) through its structure of hierarchical parameters. Each layer of a ANN provides higher abstract information, and, depending on the layer's connections we will set different ANN architectures. There are two different architectures we are going to use: U-Net and DenseNet. First one provides more robust predictions, but the second one needs a much smaller number of parameters to filter the images, by reducing the quality of the performance.

This work is broken down into five chapters. The first one is about the two different radiation therapy techniques, radiotherapy and proton therapy; the benefits and the losses of both of them will be explored there, the concept of planning and the radiation dose and how it depends on the chosen technique. Chapter two is focused on the Neural Networks, the different parts of one network, the perceptron, the layer and the whole network; the different phases of a network set, training, validation, test and the possible issues that could appear; the state-of-the-art of the current networks, and an explanation of the chosen architectures. The third chapter explains the data we have and the preprocessing we are doing to train the networks, focusing on the general features of the image, intensity, which is equivalent to the dose, the dimensions and the anatomy of the tumor, afterwards we will explain how we standardize all our samples, and, finally, a method called *data augmentation*. The fourth one will analyze the methods used in this thesis. This chapter will focus on both architectures: U-Net and DenseNet; on the fine-tuning process, where we are going to explain each hyperparameter, on the method for generating the whole 3D output and on the metrics chosen for comparing and validating how good are our results. Finally, the last chapter presents the results of the networks and the viability of that system in the clinical practice.

Once we have presented the results, we are going to discuss them and the features that could be improved. Finally, the conclusions of the current research will be presented there, exposing some future perspectives for the denoising issue with deep learning techniques.

Chapter 1. Basic notions of Radiation Therapy

Chapter 1

Basic notions of Radiation Therapy

Radiation therapy (RTx) consist in the application of ionizing radiation, delivered by radioactive isotopes, linear accelerator or cyclotrons, to tumors. Sometimes, radiation therapy is but one treatment in an extensive planning that may incudes surgery and chemotherapy. Other times, radiation therapy is the only treatment applied.

There are two kinds of radiological treatments: *brachytherapy*, consisting on introducing the radioisotope inside the patient, and *teletherapy*, where a certain number of ionizing beams are applied to the patient. The aim of the current thesis resides in the teletherapy area; hence, we are going to develop only that treatment [6].

Linear accelerators (linacs) are devices that accelerates charged subatomic particles, but, firstly, a tungsten filament submitted to a high voltage delivers electrons. We can apply these electrons directly, or we can project them to a metallic anode, which will produce the photons. The linac accelerate both particles, photons and electrons. These devices give the required energy to the particles before interacting with the patient. That energy allows the particle to maximize the interaction with the tumor, reducing the damage to the healthy cells.

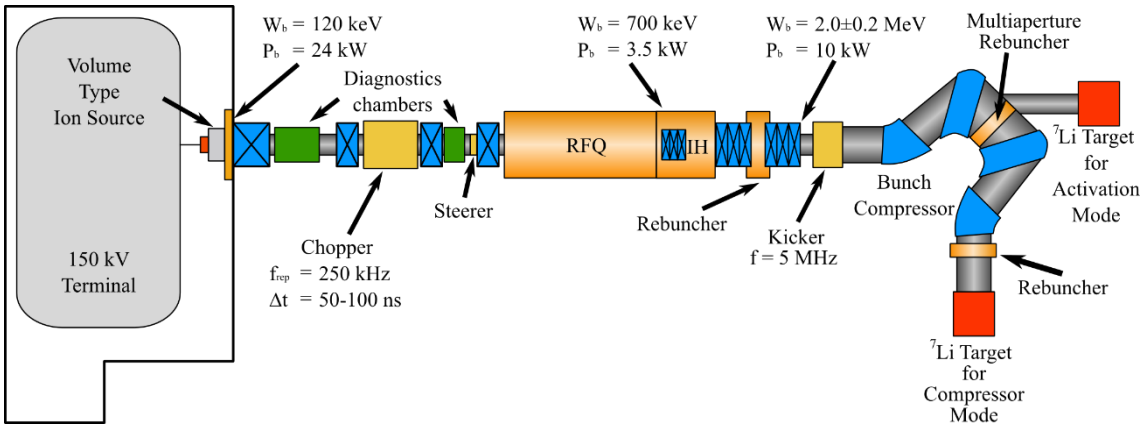


Figure 1. Lineal accelerator scheme. The ion source is connected at the beginning of the accelerator, as we could see on the left side of Figure. The different cameras provide the required energy, and they could be activated or deactivated in order to achieve the energy wanted. Finally, there are some cameras that focus the beam on the blank.

1.1. Radiotherapy and proton therapy

Photons are the particles more commonly used in radiation therapy treatments. There are two types of photons in teletherapy:

- *X-rays*:

X-rays are a kind of electromagnetic ionizing radiation produced by the electron deceleration. When an electron returns to its true orbital, it losses energy in form of light. The energy of that photon depends on the distance between the external orbital, where the electron was located, and the current orbital. The longer the distance, the higher the energy, which is proportional to its frequency. This energy could be calculated through the Plank Equation, which is referred on Figure 2.

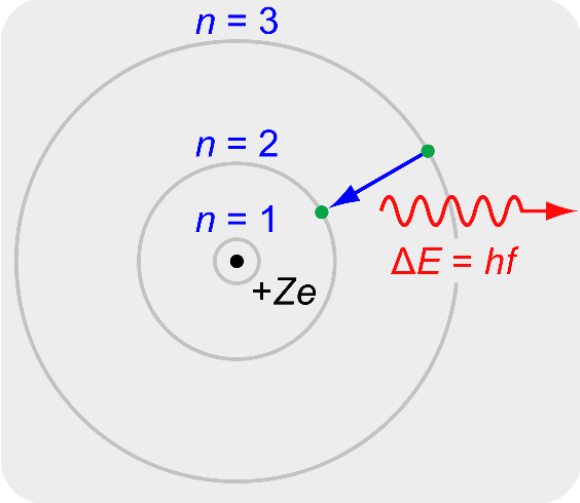


Figure 2. Photon generated by the electronic deceleration. Its energy depends on the difference between two different levels {n = 1, 2, 3}. These possible energies are proportional to its frequency by the Plank constant.

When this energy is high enough, it could ionize other atoms. When an atom is ionized, it losses an electron, as, in consequence, if this atom belongs to a higher molecule, it will change its electronic configuration and even, it would be broken. Certain molecules are extremely important for our survival, as DNA. If DNA is broken, cells could die or mutate. These mutations can happen on germ cells, which is not dangerous, or on somatic cells, which could imply a cancer.

Depending on the energy, an X-ray can be classified in: *superficial X-ray*, used to treat skin illnesses, *diagnostic X-rays*, used in radiological imaging, *orthovoltage X-rays* and *supervoltage X-rays*, used to treat some no deep tumors, and *megavoltage X-rays*, the most energetic beams and the most commonly used for radiotherapy.

- *Gamma-rays*:

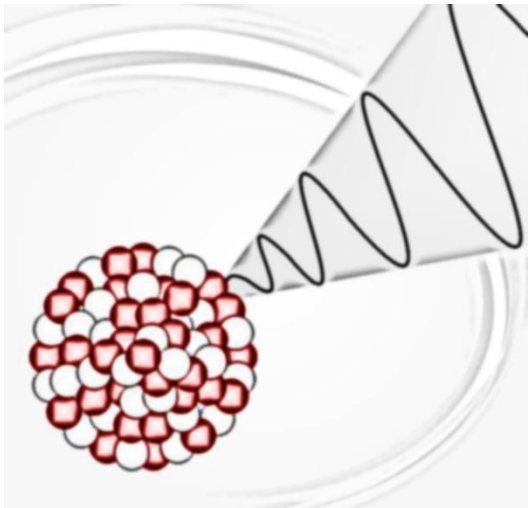


Figure 3. Representation of a gamma decay. An electromagnetic wave is produced as a consequence of a nuclear particle decay.

Gamma rays and X-rays are both electromagnetic radiation. Energies are overlapped on the electromagnetic spectrum. This process is called gamma decay, and, usually, it happens after other kind of decay, as alpha or beta decay. However, the nucleus is still excited, which decays once again producing a gamma photon. Unlike X-rays, gamma-rays are produced by radioactive isotopes within the process of a positron-electron annihilation. Gamma-rays are far more energetic than X-rays, therefore they will arrive deeper inside the human body and they are applied when X-rays cannot arrive to a tumor located on the deepest parts of the body.

Notwithstanding, there are other particles used for the radiation therapy, like carbon atoms or protons. These particles have one feature making them interesting, the *Bragg peak*. Bragg peak is the model of dose distribution for protons, and it will be explained in the next point [7].

1.2. Radiation dose

The most important parameter in radiation therapy is the delivered dose. The dose is the amount of energy released in a certain area of a tissue by the ionizing beam. Dose is proportional to the number of ionizations occurred in this area. Doctors decide what amount of radiation must be given to the tumor, and physicists have to study how to apply the beams for minimizing the dose in no cancerous areas. The unit of dose is the Gray (Gy), equivalent to energy per mass unit (J/kg)

Moreover, it is important to know how the different particles interact with the environment, and how the energy is distributed along the body.

1.2.1. Photon and proton dose distribution

Photons and protons differ in how their energy is distributed along their depth. Photons have a peak near to the surface of the skin, and, gradually, the energy deposit decrease. Consequently, the most of the dose is delivered next to the skin, even if the tumor is deeper than that. This is a problem because the most radiated area is a healthy tissue.

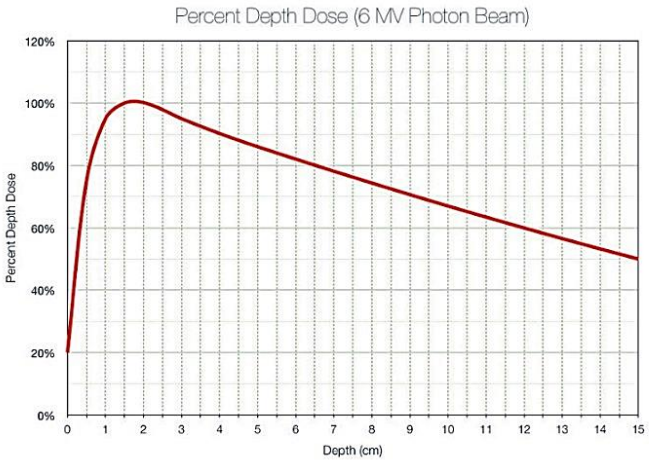


Figure 4. Percent Depth Dose for 6 MV photon beam. At the beginning of the distribution, the delivered energy is maximal, and, afterwards, that energy decreases linearly.

Nevertheless, these kind of beams are currently applied because healthy cells are not so sensitive to radiation than tumor cells. This happens because healthy cells have their reparation mechanisms totally functional, and tumor cells have not. In addition, physicists explore how to apply different beams to increase the dose on the tumor and minimize it on the healthy areas.

Photons are not a charged particle, but electrons, protons, etc. they are. This charge is which results in a completely different distribution, what is explained in the following paragraph.

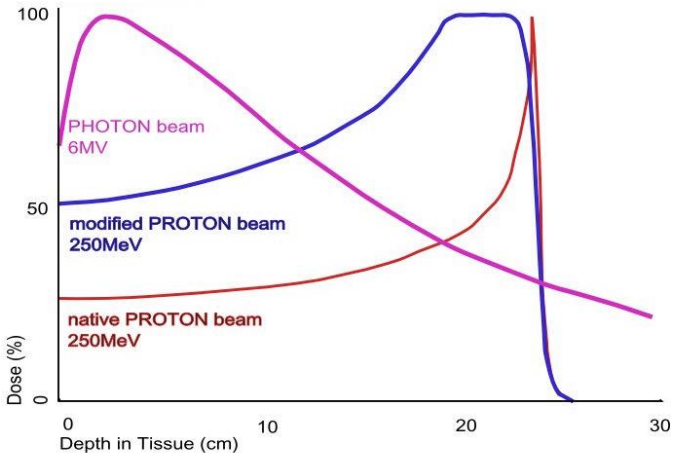


Figure 5. Comparation of dose distribution curves. Pink curve shows the same information than figure 4, and red curve shows the behavior of the proton delivered dose. This distribution is almost the same than the high energy carbon atoms.

Protons spread their energy in a different way. They deliver a nearly constant low dose to the tissues, but, suddenly, the dose delivered increase, reaching a peak called *Bragg peak*, and the dose disappears. This is completely useful, because the delivered dose is minimum in healthy tissues, but, if the Bragg peak is located in the tumor region, those cancerous cells receive a high energy depose.

Considering the Bragg peak depth depends on the energy of protons, we can translate the tumor depth to a required energy for arriving to the blank, and we can give that energy to the beam by the cyclotron or the synchrotron [8].

1.3. Radiotherapy planning

The search of the best beam configuration is called *the radiotherapy planning*. It is made by physicists, and it consist in testing different numbers of beams, angles, intensities and sizes on a simulator. This simulator must be precise and make no error on the dose estimation.

Having good simulation requires two things: firstly, a patient scanning image (tomography scan) with the region of interest correctly segmented, and, secondly, a robust estimator algorithm of the dose distribution

1.3.1. Segmentation

Segmentation is the process of partitioning an image in regions of interest. In the biomedical imaging issue, segmentation is commonly used to delimit tumors for the radiotherapy service. With a partitioned image, physicists could manage the ionizing beams and estimate its distribution in the simulator system.

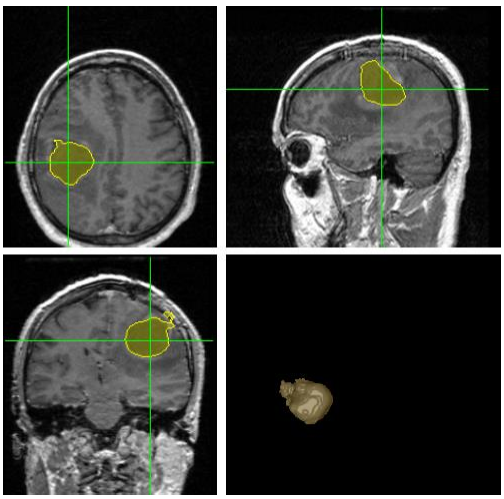


Figure 6. Segmented brain tumor. Each image corresponds to a different perspective: axial, sagittal and coronal.

Nowadays, CT-images are manually segmented. One physician seeks in the different slices of the 3D-image to find which pixels

belong to the tumor and which others do not. This takes a considerable time, and there are a lot of approaches to automatize this process. Machine Learning experts are developing systems based on Artificial Neural Networks to do it automatically [9].

1.3.2. Monte Carlo algorithm

There are two kinds of algorithms that provide an estimation of the dose distribution. There is the *analytic method*, quicker, and less accurate and more precise [10]. This kind of algorithms can be used for fast estimations of the dose distribution within the search of the better beams on the simulator. However, being sure of the real dose is impossible with those methods, we require something more powerful.

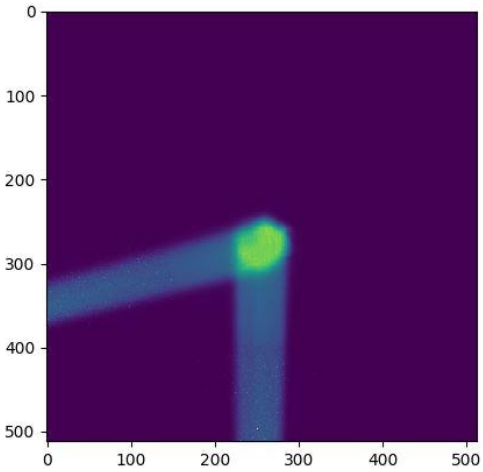


Figure 7. Example of Monte Carlo dose distribution provided with 1e9 particle interactions. High dose corresponds to the green-yellow core, and blue areas correspond to the beams.

The *Monte Carlo method* is based on random samples of the physical process. Firstly, we need to introduce all the possible interactions in the model, with an associated likelihood. Each interaction depends on several parameters: position, energy, mass, voxel value, etc.

Each proton interaction will be defined by the distribution of probabilities resulting of the combination of all the physical possible interactions. The core of Monte Carlo success is the fact that, repeating the process enough times, the distribution of interactions and energy deposit will converge to what actually happens. The higher the number of simulated particles, the more realistic the model is.

One problem of Monte Carlo algorithms is that the improvement of the performance, the noise reduction (nR), is proportional to the squared root of the number of simulated particles:

$$nR \propto \sqrt{N}$$

Consequently, doubling the quality of the estimation requires four times more time, and logically, that means time required is quadratic for improving the performances.

Furthermore, performances whose noise does not compromise the clinical practice require 1e9 particles, which means one computing hour, what is extremely slow. That is the reason why we need to denoise doses generated with a lower number of particles. Simulating 1e7 particles requires one minute of computation, and the simulation of 1e6 particles spends a few seconds [11].

We are going to filter those two simulations in order to obtain a reasonable simulation, similar to the one of 1e9 particles. The mainly problem is that the noise distribution is still unknown. There are "intelligent" denoising filters that, provided with the noise distribution, could make a really good performance. Unfortunately, as we do not know that distribution, we cannot apply that filters. This is the scenario in which deep learning appears.

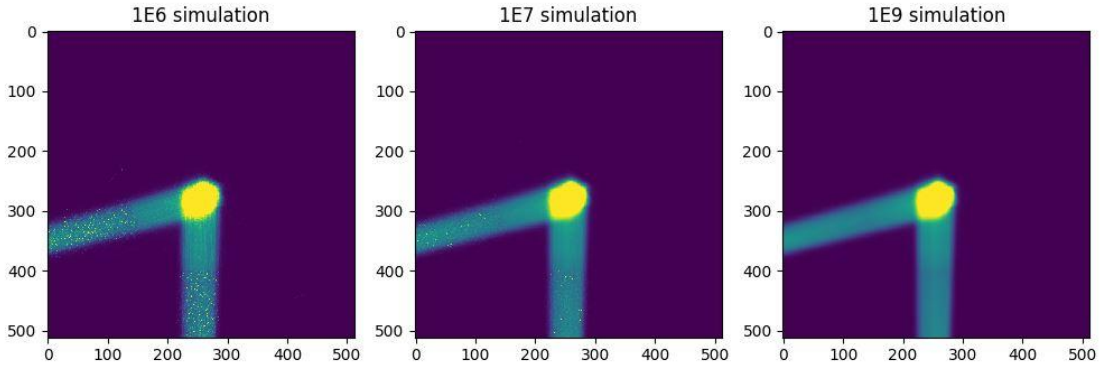


Figure 8. Different Monte Carlo simulations. The first one is produced with 1e6 particles, and, as we can see, there is a lot of Noise. The second one is produced with 1e7 particles, and, although is less noisy than the 1e6 image, it has a visible noise too. Finally, the last simulation is generated with 1e9 interactions, and we can consider it as a free noise image, so we define it as the reference.

Theoretically, deep learning systems could learn by themselves high complex patterns by a hierarchical procedure. That is the reason why we are going to use Artificial Neural Networks for denoising.

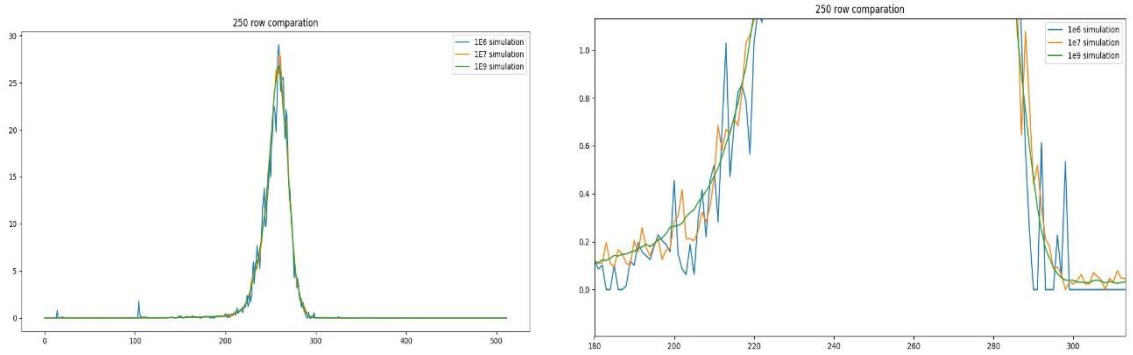


Figure 9. Noise level comparison for different simulations. Blue line corresponds to the noisiest image, generated with 1e6 interactions on Monte Carlo simulator. Orange line corresponds to 1e7 interactions image, which is less noisy than the previous one, but is still noisy. The green one is the reference, the result we would like to achieve by out neural network. On the left side we have the enlarged image, where we can see the difference between the three images. Blue noise is clearly higher, and green noise does not exist.

2. Artificial Neural Network (ANN)

Chapter 2

Artificial Neural Networks (ANNs)

Within this chapter, we are going to explain, step by step, what an Artificial Neural Network (ANN) is, the current state-of-the-art in the biomedical area, and the architecture we chose for the denoising issue.

2.1. Machine Learning system

Nowadays, there are two ways of programming. Firstly, there is the "traditional" way of programming, based on defining commands to the computer with the purpose of obtaining an exact result. In this case, we need to know the mathematical or logical procedure for solving the problem, and, afterwards we had to communicate these instructions by a programming language to the computer. The scheme of this first case would the one of Figure 10.

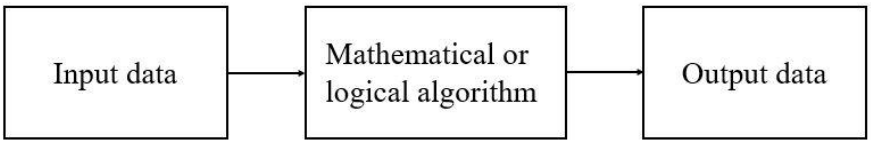


Figure 10. "Traditional" programming way scheme. The first square corresponds to the input data, which passes through the machine learning system (second square) and, this interaction produces the output)

However, there is another way of programing, consisting in developing a virtual system with many parameters that interact with the input data, and generates the output. The parameters need to be tuned by a process called training, which consist in showing the data to the system and let it learn the internal structure of these data.

We estimate the performance of the machine learning system with an unseen package of data, called *validation data*. When the model is completely fitted, we apply another independent batch of data, called *test data*. Validation data is used to check the performance of the network in each epoch, so, the network will save the best networks for that dataset. However, feeding the network with another independent dataset will show its quality with unseen samples.

2.1.1. Types of learning

One of the most important decisions we should make before starting to develop our Machine Learning system is deciding which kind of learning we will apply to the system. There are mainly three types of learning:

- *Supervised* learning:

Supervised learning consist in giving references or examples to the system in the learning phase. This would mean that, if we want to develop a system that predicts a handwritten number, we will need to show these handwritten numbers along with the corresponding label. Indeed, MNIST dataset is used to make the first approach to deep learning, and is the basic dataset where researchers apply their last discovers.

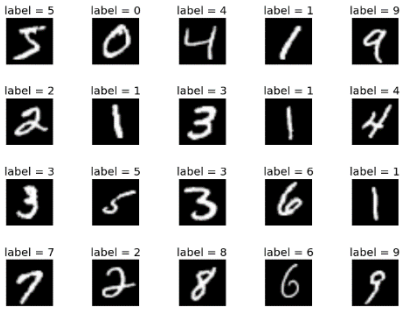


Figure 11. MNIST dataset. Each picture has its corresponding label, which is the value of the number. In this case, the output is not a number, but a vector of likelihoods.

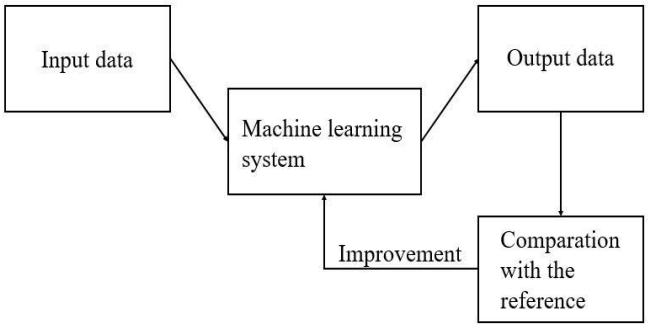


Figure 12. Supervised learning scheme. The input feeds the Machine Learning system and it produces an output. Afterwards, this output is compared with the expected output and, through this comparison, the parameters of the systems are improved.

The aim of the supervised learning is to achieve that the system could generalize from the training data to unseen data. We can see a scheme in Figure 12.

Sometimes, the model performs very well with the training data, but, when we show new data to it (in validation or test phase), the performance is worse. That problem is called *overfitting*, and it means the model does not generalize for unseen data.

- *Unsupervised* learning:

Unsupervised learning consists in training the machine learning system with no reference, keeping it learn the hidden structure of the data. Nowadays, this kind of learning only works in clustering task, thus it is useless for the denoising issue.

- *Reinforcement* learning:

Reinforcement learning is a special kind of supervised learning, where the feedback is given by the users of the system. We can find these systems in the spam folder on our email, or in websites like YouTube or Google, that use them for showing personalized recommendations to their users or to give better language translations.

2.1.2. Applications

We can make another categorization of different Machine Learning systems, depending on the desired output:

- *Classification*:

Classification task consists on having inputs from two or more classes, and the system must find the class of each of them. We can distinguish between binary classification, where the input is split into two different labels, while in multi-label classification we have more than two classes. Classification is a task performed by supervised systems.

- *Clustering*:

On the other hand, we have the clustering task, which is similar to the classification, but without classes. In this case, we do not give any label, so the system will find the natural distribution and will classify the data according to that distribution. This is, of course, an unsupervised learning.

- *Regression*:

Regression consist on predict a feature-value from a certain data. This means that the output of the system is a continuous value. Denoising issue is a regression task as we want to find a continuous free-noise pixel value based on a noisy image. Additionally, we need to provide the reference of each sample, so we work with a supervised system.

- *Dimensionality reduction*:

There are a lot of problems caused by having a high dimensional dataset. This is the reason why the dimensionality reduction is one step in the majority of the Machine Learning researches. The goal of this is to map the dataset in a lower-dimensional space, keeping almost the whole of the information. As the comparison is made with the very input, this is an unsupervised system.

2.2. Components of an ANN

An Artificial Neural Network is a Machine Learning system based on the structure and behavior of the human brain (Pitts, 1942). This system is composed by neurons (called perceptrons in the computational world) grouped in layers, which, in turn, are grouped in different ways. These neurons are connected through parameters called weighs, which assess the importance of each data for the output. A huge number of connected neurons form an Artificial Neural Network.

2.2.1. Perceptron

The perceptron is the minimal structure of an Artificial Neural Network, and it is split into three different parts [12], as we can see in Figure 13. However, the perceptron per se is a system that can work in simple classification tasks.

- *Weights*:

Weights are the parameters of the perceptron, they are multiplied by each data, in order to assess the importance of every data. These weights can have any value, but we use to initialize it with random values next to zero. Furthermore, we have a bias term.

- S*ummation*:

The next step of the perceptron is a sum of every weighted data and the bias term.

- *Activation function*:

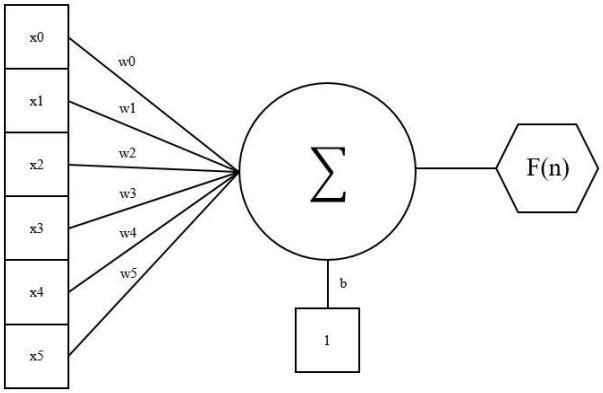


Figure 13. Perceptron scheme. The different weights are multiplied by the dataset. Additionally, there is a bias term that gives more robustness to the system. These multiplications are summed afterwards, and the result of this sum is applied on an activation function.

Once we have made the weighted sum, we have an input the network can use. The activation function works like an activation threshold in the biological neurons. There are a lot of activation functions, which we will focus on afterwards.

2.2.2. Layer

A neuron is the smallest part of an ANN. We use to group perceptrones in layers that act on the same dataset. Each neuron of the same layer is connected to each previous data, with its own weights what will be trained in a different way.

One perceptron adapts itself for detecting one feature of the dataset, and its activation function intensifies the output depending on what clearly that feature is. So, a layer that contains a lot of neurons detects a lot of features, and, consequently, is able to catch more information from the data.

2.2.3. Multilayer perceptron

The last step for building an Artificial Neural Network is to join different layers in order to increase the abstract dimension of the information. The network starts with a simple layer called *input layer*, which acts on the dataset. Afterwards, there are the *hidden*

layers, which act on the output of the previous one. So, if the first layer detects hidden features on the dataset, the second layer detects certain combinations of features, the third one detects combinations of combinations of features, etc. Finally, we have the *output layer*, that takes the abstract information and produces the desired output. The more layers we put, the more abstract information the network can learn [13].

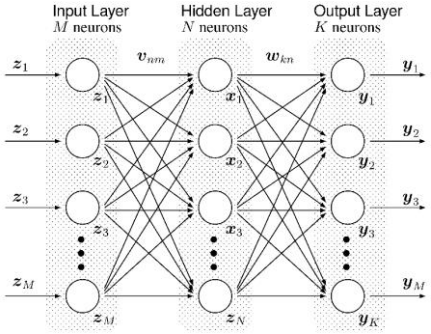


Figure 14. Multilayer perceptron scheme. We can see three different layers on this scheme. First one is called input layer, which takes the data and transforms it through the unitary structure seen before, the neuron. Each circle corresponds to a different one. The output layer gives the solution we are looking for.

But we do not need to make a sequential multilayer perceptron. State-of-the-art networks make a huge number of complex concatenations, like DenseNet, which is based on connect each layer to all the others. Additionally, we have U-Net or Capsule Networks [14], that, by the moment, provide the best performance.

2.3. Training

Within every single Machine Learning, the parameters are tune in the learning phase. In the case of Neural Networks, these parameters are adjusted by a process called *Backpropagation*.

2.3.1. Backpropagation algorithm

Firstly, it is mandatory to clarify that Neural Network are a Machine Learning system with supervised learning, so we need to show it a label or a reference with the sample.

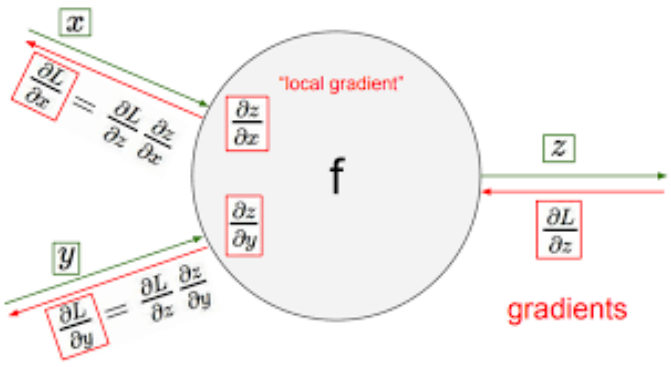


Figure 15. Gradient of backpropagation. Green lines represent the forward propagation, the process of generating one output from the input. At the beginning, the weights are randomly set, son the output will be random too. Red lines represent the backpropagation phase, which consist in calculating the derivatives of the error in each weight and correct the value of those weights by applying a step in the direction of the gradient.

The process of training for ANN has two parts: first, the forward propagation phase, in which we take one sample and we pass it through the network. As the weight values have been randomly initialized, we obtain a random output. Afterwards, we compute the error between the output and the reference.

The second phase is the backpropagation, we take that error and we calculate the derivative of that error. Then, we

apply the chain rule from the output layer to every single weight of the network. With the derivative of the error in each weight we compute the gradient descent and we tune them by applying a certain step [15].

There are a lot of hyperparameters here (a hyperparameter is an external parameter not tuned by the training): the cost function we apply, the size of the step, the size of the batch, and even, we can improve the step size by applying ADAM algorithm, which decrease the size of the step and even improves its orientation [16].

2.3.2. Training issues

We exposed the core of the Neural Networks, but it is common to find some problems while the building of the network. There are two main issues we can find in every Machine Learning system, and one extra that appears only in ANN environment.

- *Underfitting:*

Underfitting appears when the model is limited by its complexity, and, consequently, there are some samples it cannot classify or regress. In this case, we need to increase the number of neurons and even to change the layer's configuration.

- *Overfitting:*

Overfitting is just the opposite thing than underfitting. It is produced when the system focuses on the training subsample and cannot generalize to the validation and test samples. In this case, we can do two different things. Firstly, we can increase the number of different samples. If there are more samples, the system cannot learn a particular structure, so it will generalize by force. Secondly, we can simplify the model in order not to catch every detail of the training samples. Additionally, we can add layer as called *Dropout* [17], which "kills" every neuron with an output under a certain threshold, or *Batch Normalization* [18], which assumes that outputs follow a Gaussian distribution and normalizes the average and the standard deviation. These two layers remove less important details of the network, however, they add some noise to the final result.

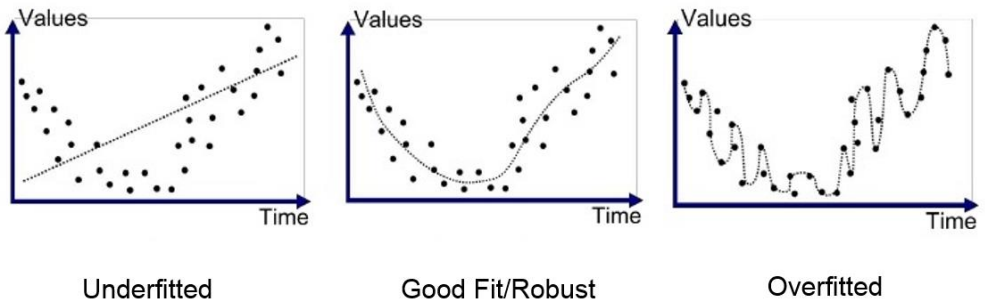


Figure 16. Example. Decision frontiers of Machine Learning systems. A) The decision frontier is not adjusted to the shape of the data, so the classification or regression will be bad. B) The decision frontier is perfectly adjusted to the data shape, so the model is robust. C) The decision frontier is over adjusted to the data, thus, the prediction will be imperfect.

- *Gradient vanishing:*

Gradient vanishing was a main problem in Neural Network area until the year 2012. It occurred that, when experts in Artificial Intelligence were trying to train deep networks (networks with more than two hidden layers), only the last layers tuned its weights. The gradient of the error tended to zero for the first layers, and, as the improvement of the weight depends on that gradient, these layers remained randomized.

2.3.3. ReLU function

In the year 2012, AlexNet applied a new activation function called *Rectified Lineal Units* (ReLU), which was defined as:

$$ReLU(x) = \max(0, x)$$

This simple function works like a thresholder function, because if the weighted sum result in a number lower than zero, the output is zero too; but if the number is positive, the output is a linear function. With this activation function, AlexNet overcome the gradient vanishing issue [5].

Nowadays, there are a lot of ReLU based functions, like Flexible ReLU (FReLU), which have an origin under zero [19], Parametric ReLU (PReLU), which, instead of outputting a zero with inputs under zero, outputs a fraction of it [20], etc.

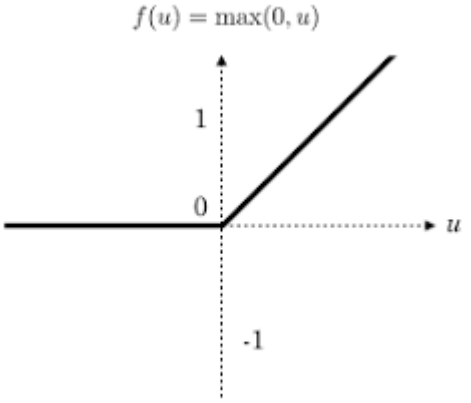


Figure 17. ReLU function graph. Input values over zero are equally outputted, but values under zero are outputted as a zero.

2.4. Convolutional Neural Networks (ConvNets)

Convolutional Neural Networks are based on the convolutional operation between an image and a kernel. They are used in the image context, where they have shown their power in detecting hierarchies of features [21]. The main idea of ConvNets is to replace the scalar weights by matrix or kernels. Each neuron has a number of weights equivalent to the indexes of the kernel, and each kernel is applied, not only to a located part of the image, but to the whole of it. The weights, instead of being connected to only one data, are shared by everyone. The idea remains in that every kernel detects a concrete feature, but, this feature is interesting in the whole picture, not only in a reduced region, so, this kernel is convolved with the entire image.

Every single kernel produces an activation map, a kind of image resulting from the convolution of the image by the kernel.

In the following layers, the input of every neuron is a bunch of activation maps, so the kernels of these layers will be 3-dimensional, convolving all the batch and producing a new one activation map.

We use to define 3x3 kernels, so there is repeated information in the successive layers. Furthermore, all these convolution layers have a lot of parameters, so the memory capacity be at stake. These are the reasons why we apply *maxpooling layers*.

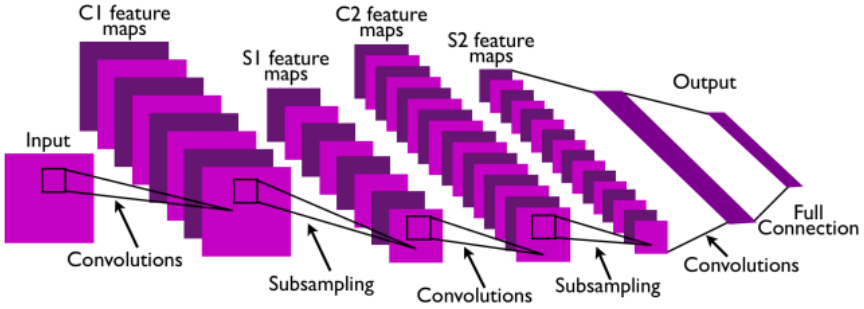


Figure 18. ConvNet scheme. First layer applies a convolution with 8 different kernels, what produces 8 different activation maps. Afterwards, we applied a MaxPooling operation, reducing the original dimension. We repeat this process once again and, therefore, we apply a fully connected layer, which produces a vector instead of a matrix.

- *MaxPooling layer:*

A maxpooling operation consists in reducing the dimension of the activation maps by selecting the maximum number in a located region of that map and generating a new map only with these values. The most common maxpooling operations is 2x2 dimensional, so for each square of four pixels, we only select the higher, reducing the total dimension to the fourth part.

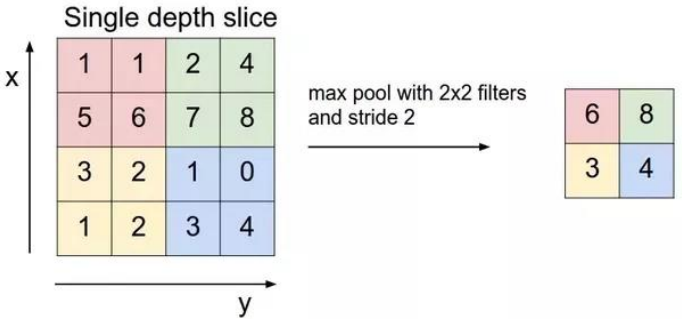


Figure 19. MaxPooling operation. It consists in, for each four squared values, taking only the highest one and reducing the output dimension.

2.4.1. Common uses of the ConvNets

Convolutional Neural Networks were created for image analysis, so all the applications of these networks are focused on that. We could divide the different approaches into three parts:

- *Segmentation task:*

In image analysis, the segmentation is the division of the image in different regions of interest. One common segmentation task is to identify the background and the foreground, but there are a lot of complex segmentations issues. In the biomedical imaging area, one of the most studied problems is the segmentation of organs and tissues for radiation therapy, or the identification of tumors for diagnosis [22].

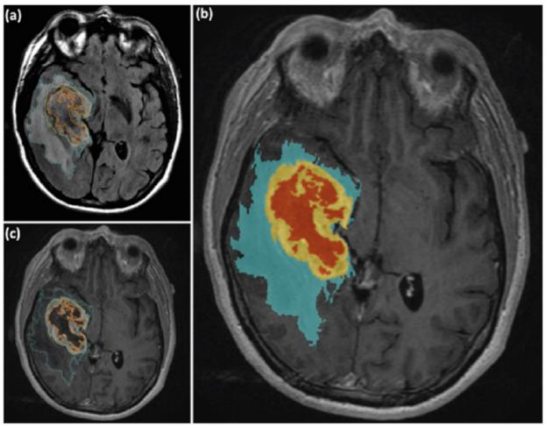


Figure 20. Segmented brain tumor. In this case, the image has 4 different masks. First one, the background, afterwards the blue region, probably the tumor vascular area, the yellow region, the tumor, and the red one, the necrosis region

Within this case, it is common to use a binary mask as a reference, in which we distinguish the region of interest and the other pixels.

- *Classification task:*

Another important task is the classification of the image. Sometimes, networks are designed to identify the most important object in the picture, but other times, the network recognizes all the objects and explain the link between them. For that task it is compulsory to have two phases in the network. Firstly, a sequence of convolutional layers for seeking the features of the image, and, afterwards, some fully connected layers.

- *Dimensionality reduction task:*

It is known that images take a lot of space in our memory, so we need to compress them. *Autoencoders* are a kind of networks with a bottleneck in the middle of its architecture. In this bottleneck, there is less information that in the input [23]. We train these networks by giving the very image as an input and as a reference [24]. The main idea is to keep most of the information with fewer variables.

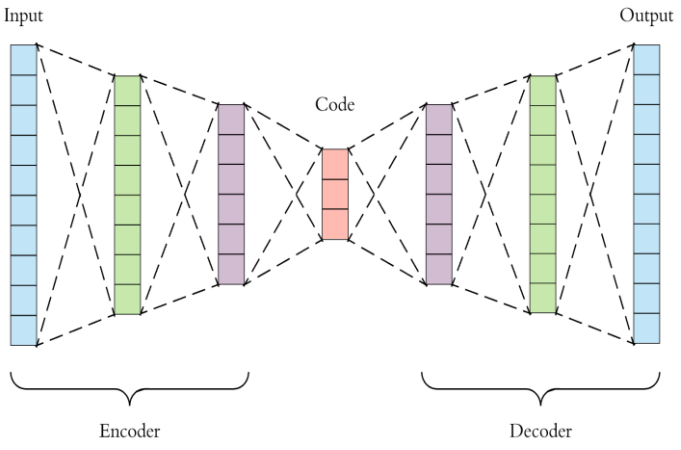


Figure 21. Autoencoder scheme. There are two phases, the downsampling phase, sometimes called the encoder phase; and the upsampling phase, called the decoder phase too.

This architecture allows to reduce the amount of data from one signal (one image in our case), but keeping almost all the information.

- *Denoising task:*

The denoising task consists in filtering the image to reduce the amount of noise, improving the vision quality on artistic photography and improving the performance of studies and clinical forecasts in biomedical imaging.

A huge number of denoising networks are considered to be autoencoders, but the core of this research is a network that is not an autoencoder, but improves the state-of-the-art.

2.4.2. U-Net architecture

Researchers discovered that, when using ConvNets for predicting the segmented area, the more abstract the information was, the less located it was. That was an actual probleme, because the key of segmentation is to identify which exactly pixels belong to the region of interest. That is the reason why they developed U-Net, an architecture that, through its concatenate layers, preserve the located information with the abstract one [25].

Standard U-Net architecture is structured in batches of two convolutional layers and a 2x2 maxpooling layer. It is divided in two phases, the *downsampling phase*, where the network learns the noise model by its hierarchical behavior, and the *upsampling phase*, where the network recomposes the abstract information with the help of the concatenated layers (what appears in Figure 22 as a copy line [9].

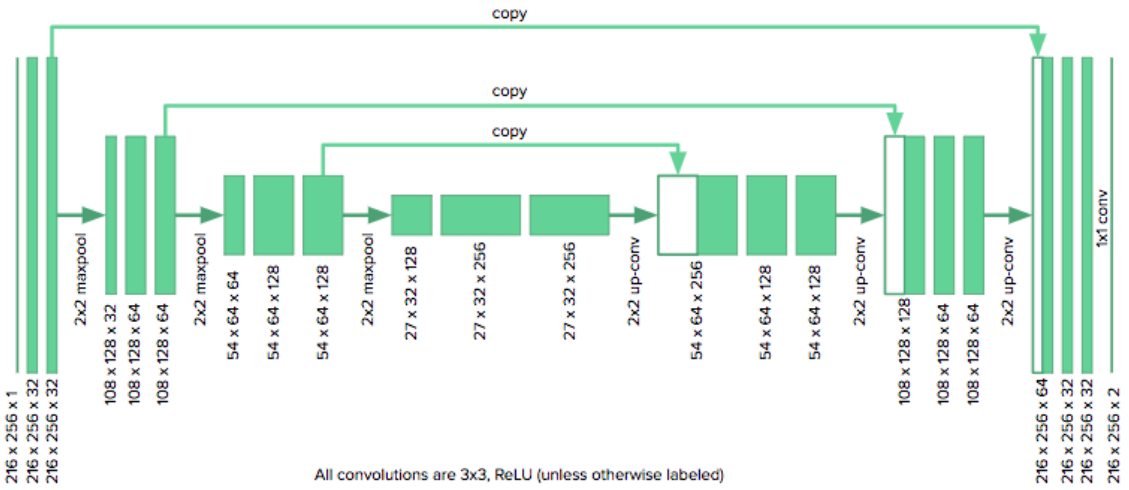


Figure 22. U-Net architecture scheme. Firstly, we have the downsampling phase, with three parts which includes two Convolutional layers and a MaxPooling layer. Afterwards, we have the upsampling phase with three parts including two Convolutional layers and an UpSampling layer. Additionally, we copy the last feature map of each downsampling part to the upsampling one.

This architecture cannot be considered as autoencoder due to the concatenated layers, which provide information from the encoder phase to the decoder one. Therefore, autoencoder references are the very inputs, and, in this case, the reference is a free-noise version of the input.

2.5. State-of-the-art

Nowadays, there are papers that explains how to denoise medical images by using wavelet transforms, intelligent filters and neural networks. We are describing the following results with metrics we are explaining in Chapter 4.

In 2017, Kaur et al. achieved a *mean squared error* under 0.01 by using a wavelet approach. Nevertheless, this approach was applied on MR images with artificial noise. Noise was generated by NUMPY algorithms with a Gaussian model [26].

In 2017, Ali achieved a PSNR over 66 with an adaptive median filter, but he used normal MR images with Salt and Pepper noise. When he used Gaussian noise, the performance decreased to 39. For this kind of noise, he achieved a PSNR of 51.98 with a normal median filter. Nonetheless, all these noises had a low intensity. Higher noises worsen the performances [27].

In 2017, Bai et al. applied Machine Learning techniques for denoising MR images with artificial Gaussian noise. They get a PSNR of 36.77 in male brain images with a 10% of noise amplitude. However, their results worsen with the augmentation of those intensities [28].

In 2017, Jifara et al. applied a Convolutional Network for denoising CT images. They used artificial noise with $\sigma = 15$ and 25; and they achieved a PSNR of 41 and 38.6 respectively [29].

In 2018, Yang et al. achieved a PSNR of 24.25 using Adversarial Networks for denoising Low Dose CT images. Adversial Networks are two different networks where, one of them produces the output and the other one tries to identify if the output is a denoised image or a reference. The main idea is to make extremely good outputs that the second network cannot difference. Nevertheless, a CNN used before achieved a better result in 2016 [30].

In 2016, Dong et al. denoised CT images by using standard Convolutional Neural Network, learning an end-to-end mapping between low resolution images and high resolution ones. They achieved a PSNR of 24.48 [31]

Chapter 3. Image preprocessing

3.1. General features of Monte Carlo doses

3.1.1. Intensity/Dose

3.1.2. Image dimension

3.1.3. Anatomy

3.2. Standardization

3.2.1. Batch selection

3.3. Data augmentation

3.3.1. Issues

Chapter 3.

Image preprocessing

In this chapter, we are going to explain the necessity of preprocess the image we are using to feed the Neural Networks and we are going to detail how we are preprocessing those images with the tools given by the DICOM format and the NUMPY python library.

Artificial Neural Networks are very sensitive to little changes in the input dataset because weights are tuned for a determined input. However, if these inputs change (in terms of dimensions, sizes, etc.) the different weights cannot match with them. Consequently, we need a robust normalization all these features.

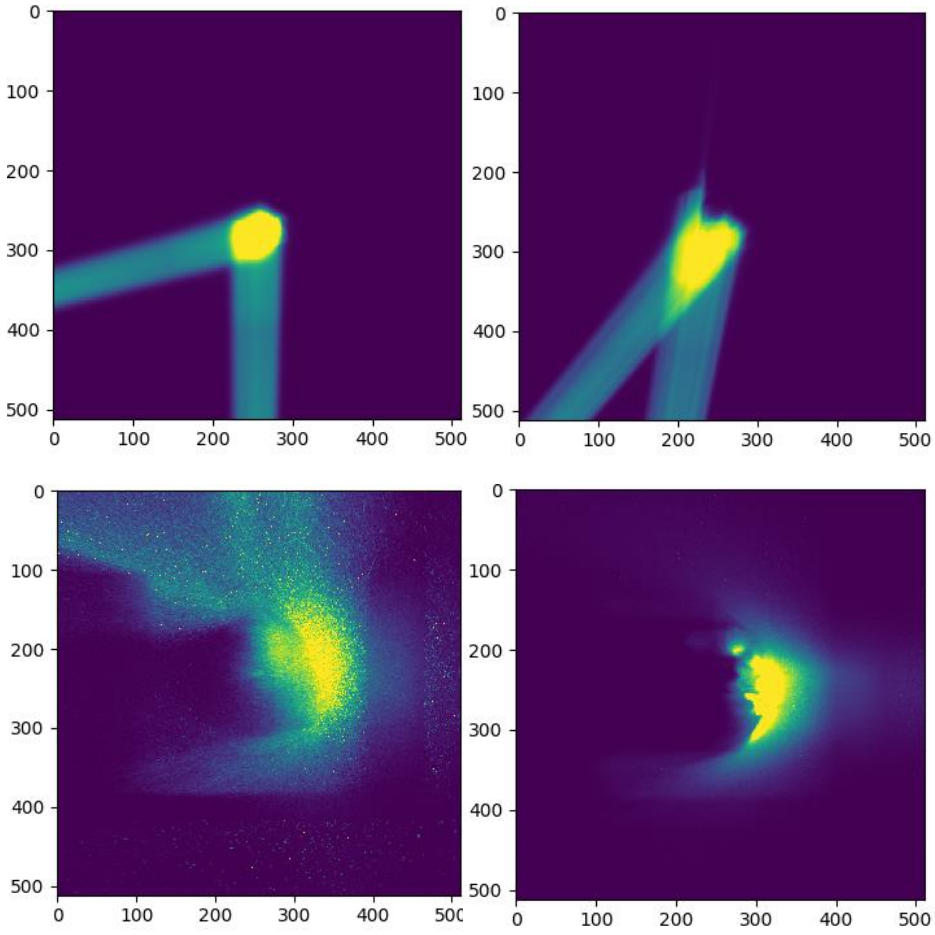


Figure 23. Shape and intensity variability. Blue color represents low doses areas and yellow one represents high dose areas (a) and (b) images are liver dose distributions. We could see thin beams with no noise. Intensities varies around 50-80 Grays. (c) image corresponds to a lung dose distribution, which implies larger beams with big areas affected by them and a very intensive noise. (d) image is a brain tumor dose distribution. It is thinner than the previous one but with a complex shape.

Firstly, we will describe the images we are working with and their main features (dimensions, size, intensities and shape). We are going to clarify how these features influence the final performance of the network, and how the changes on them could seriously affect the quality of the final result.

Furthermore, memory is a limiting factor for the training phase, so we cannot filter the whole image at once. We need to reduce the size of the 3D image or training different batches that will be rejoined afterwards.

Our dataset growths slowly, so the number of samples is another limit we could fight with data augmentation. Data augmentation consists in making the very transformations to the input and to the reference, so, that increases the geometrical robustness of the network. However, 3D data augmentation involves some issues commented on this chapter.

3.1. General features of Monte Carlo doses

Dose distributions images are featured by their size and their variability of intensities, which depend on the patient size and the treatment requirements. Images intensities are split in two different parts: low dose areas and high dose areas. These two different areas appear in every single image, even in reference images. Reference images are those simulated with 1e9 particles because they are considered free-noise.

3.1.1. Intensity/Dose

- *High dose areas:*

There are other parts of the image where the proton beams appear themselves. Those areas have pixel values over 1 Gray, and the noise level is lower than in low dose areas. However, these noise levels are enough important to be considered, as they distort the estimation of the dose that will be delivered in the patient.

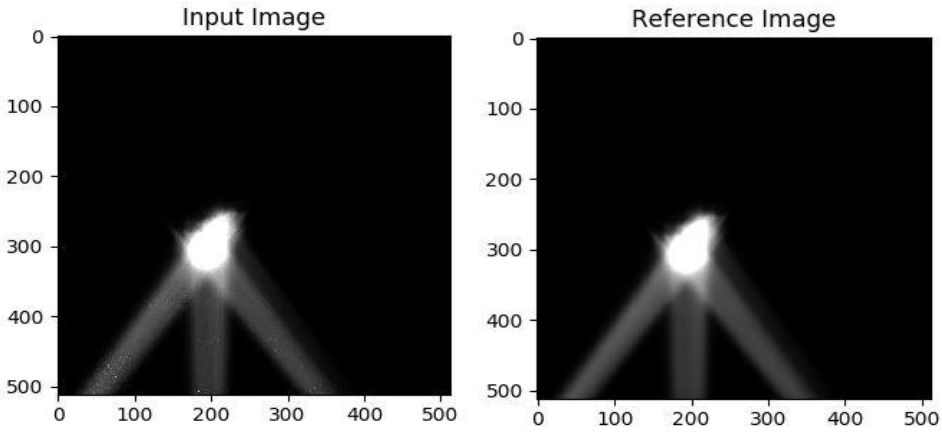


Figure 24. High dose slice. (a) 1e7 simulation, we could appreciate the noise in the beams. (b) 1e9 simulation, completely free-noise. We could see that the beams are homogeneous, thus, this is the reference image, the model we want to replicate with the input image (1e7)

The noise in these areas is neglectible on the reference image, so, theoretically, the network could discover the noise distribution and remove almost all of them from the input image.

- *Low dose areas:*

These are the areas out of the beam focus. They are characterized, even in 1e9 particles images, by a dose lower than the 1% of the maximum dose for that patient.

Notwithstanding, as the dose here is lower than the 1% of the maximum dose, the noise effects are almost unimportant and any improvement here will be fine, but not essential.

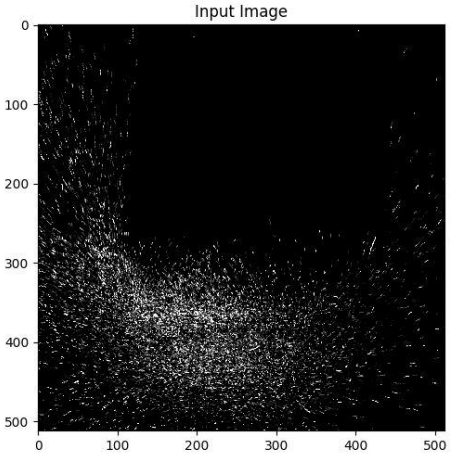


Figure 25. Low dose slice (1e7 simulation). We should see a homogenous image, but we see set of high intensity points with no rapport between them.

It can be deduced from the previous point that, considering low and high dose areas, there is a lot of variability within the model in terms of intensities and shapes. That is not a problem, considering the robustness of neural networks, because a good performance on low dose areas is not essential. All of this will be explained in Chapter 4. The real problem is the variability between different patients, and, furthermore, the no normalization of the dimensions of the images.

Fortunately, intensities of those dose distributions are normalized through a parameter called *DoseGridScaling*, which belongs to the DICOM file. *DoseGridScaling* adjust the input intensities of each realization because, depending on the Monte Carlo performance, it could be different. That parameter depends not only on the patient, but on the realization.

The dose given to the patient depends on the dose prescription made by the doctor, so there is a variability from 30 Gy to 90 Gy for different patients.

3.1.2. Image dimension

Dose distributions image sizes are partially standardized. On the one hand, height and width are totally normalized, every slice is 512x512. On the other hand, depth dimension is variable in order to standardize the pixel size.

Neural Networks must be provided with constant dimensional signals, so we have to standardize the dimensions. There are two options:

- Normalization of the third dimension:

There is a plenty of techniques to normalize a certain dimension, for instance, the *reshape* function from NUMPY library. These techniques imply a problem, the pixel size is denormalize, which involves a distortion of the noise model. Distorting the noise models could mean that the network would not be able to learn it, so the performance will be worsened.

Additionally, GPU's memories cannot allow a whole 3D image, so we cannot deal with the entire images. We will apply different patches from the volume in order to make possible the training process.

- Patch training:

The other thing we can do is to feed the network with subsamples of the original image. These subsamples are different parts of the original image, for instance, the first three slices. In this case, the patch will have 512x512x3 pixels. The dimensions of that applied patches will be discussed afterwards, in the next chapter.

Subsampling the original image preserves pixel size and noise distribution, but the network is fed with fewer pixels, and, consequently, with less information. A whole volume provides all the geometrical and shape information, with the organs perfectly define, or, in this case, the beam shapes. With all this information it is easier to keep the original shape with no deformities. However, applying subsamples means not feeding the network with a global picture of the whole body, so not all the beam shape is watches by the system at the same time, which results in a worse performance.

3.1.3. Anatomy

Another important difference between patients is the location of their tumors and their size. Some of them are located on liver or on lung, but there are others from neck, brain and prostate. These changes imply different shapes and different beams.

The beam model is another thing the network has to adapt to. There are different angles the beams could incise in the tumor, which adds more variability to the model. Unusual beam shapes will reduce the quality of the network performance, but this could be addressed by increasing the variability within the training dataset. That means to include the highest number of patients we can.

3.2. Standardization

As we have said in the previous point, it is necessary to standardize the data because the neural networks need to be fed with normalized samples: in terms of dimensional size and intensities. Here we are going to define how that intensity standardization will be made.

Firstly, we are going to normalize pixel intensities by multiplying each one by *DoseGridScaling* parameter. This parameter belongs to each realization, so we will read the sample from the dataset and, afterwards, we will normalize pixel value.

3.2.1. Patch selection

Image dimensions must be the same for every sample, so, we are going to select subsamples of the same dimension from the data. The dimension of those subsamples is a very important hyperparameter, because it will define the architecture of the network, the memory limit and the computational time.

We can take simple 2D slices, which will provide less information to the network, but will be faster. 2D images imply 2D convolutions with 2D kernels. These 2D kernels have l^2 parameters, but 3D kernels have l^3. As we used to apply 3x3 kernels (or 3x3x3). The number of parameters increases from working in 2D to work in 3D, but it remains constant for all 3D images, even the ones of the full volume. Nevertheless, we must notice that going from 2D to 3D implies an increase of the memory requirement

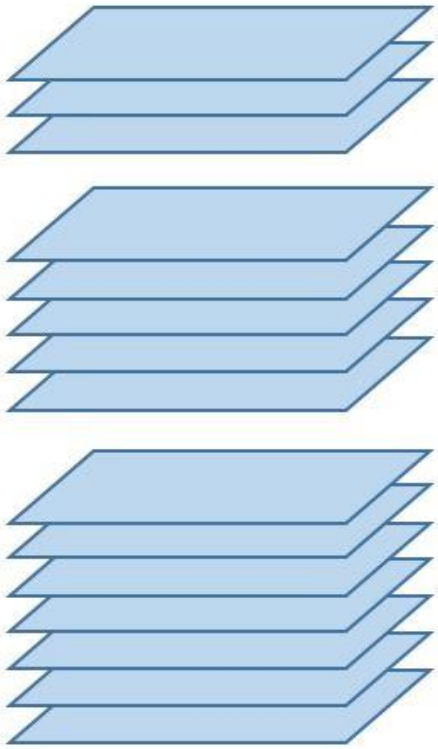


Figure 26. Patches of 3, 5 and 7 axial slices. Apart from the 2D patches, we are applying patches with 3, 5 and 7 axial slices.

Therefore, we will define different networks for different patches. First, we are going to create a network provided with 512x512 images; but we are creating others networks provided with 512x512x3, 512x512x5 and 512x512x7 batches. These networks will improve the performance as they consider more information, but training and operational time growths considerably.

3.3. Data augmentation

One of the most important problems for our research is the lack of data. Overfitting problem appear when the amount of data is not enough. In our case, Monte Carlo algorithm spends more than one hour on generating the simulation and upload to the store.

We are applying data augmentation, which means to make the very transformations to the sample and to the reference, with the aim of increasing the samples. We need to apply the same transformations to the reference because we are looking for generating it, so, if the reference is not rotated, we are going to generate a rotated output, and, when making the comparison between the output and the reference, it will appear an error that will influence on the weight tuning.

We are going to do rotations of 10 degrees and zooms of the 1% of the whole image. There are a lot of methods that can be used, for example, non-linear deformations, but, as the geometry of the dose images is very defined, it is not necessary to apply them, and, furthermore, training with those methods could introduce some noise on the result.

It is important to mention that these transformations only improve the performance in case of very similar images with different orientations. Nevertheless, these augmentations are important to give more robustness against the network and to avoid overfitting, because we are keeping noise model but we are changing the geometric of the image, so it cannot learn the particular situation as every time it is different.

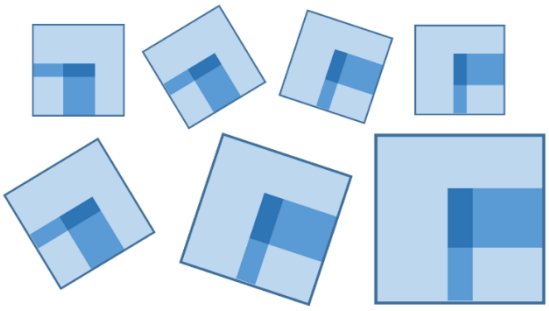


Figure 27. Different transformations of one image. All these images are the original one with rotations and zoom transformations.

3.3.1. Issues

Data augmentation is applied in 2D images. Our mainly problem is that we want to fed some networks with 3D patches, and there is no method for that problem.

We could do our own method to augment the 3D data, applying 3D transformations or 2D transformations to each slice. However, that requires time we have not, so we are not going to augment 3D data.

Chapter 4. Methods

Chapter 4.

Methods

In this chapter, we are going to develop the methods we applied in the research, focusing on the exact architecture, the fine-tuning (improvement of the hyperparameters) and the metrics we are applying in order to evaluate the results.

4.1. Architectures

Artificial Neural Networks are a certain number of neural layers connected in a proper way. There are a lot of architectures (different ways of connecting these layers, activation functions, layer sizes, sharing or no sharing weights, etc.) for very different purposes.

Usually, the first step on deep learning research is connecting some consecutive layers to have a fast picture on our minds. Those networks offer a primary idea of what you can achieve with more complex ones.

Indeed, it is impossible to mention anything about a certain architecture without focusing on a task. For imaging, there are a lot of different architectures: from Consecutive Convolutional Networks to Capsule Nets. In our research we will use U-Net and DenseNet. These two networks are commonly used in the segmentation issue, but it is uncommon to use it for the denoising issue.

We are going to explore these two architectures in the current chapter, focusing on the hyperparameters they have and how to tune them.

4.1.1. U-Net

U-Net was introduced in *Chapter 2*, but here, we are going to define exactly how the network we are using is.

This network is based on three downsampling phases and three upsampling phases. With the downsampling phases we are obtaining the noise distribution, and with the upsampling phases we are using that abstract information to remove the noise from the original image. These upsampling layers are provided with the symmetric activation map. This information can be perfectly understood on the next figure.

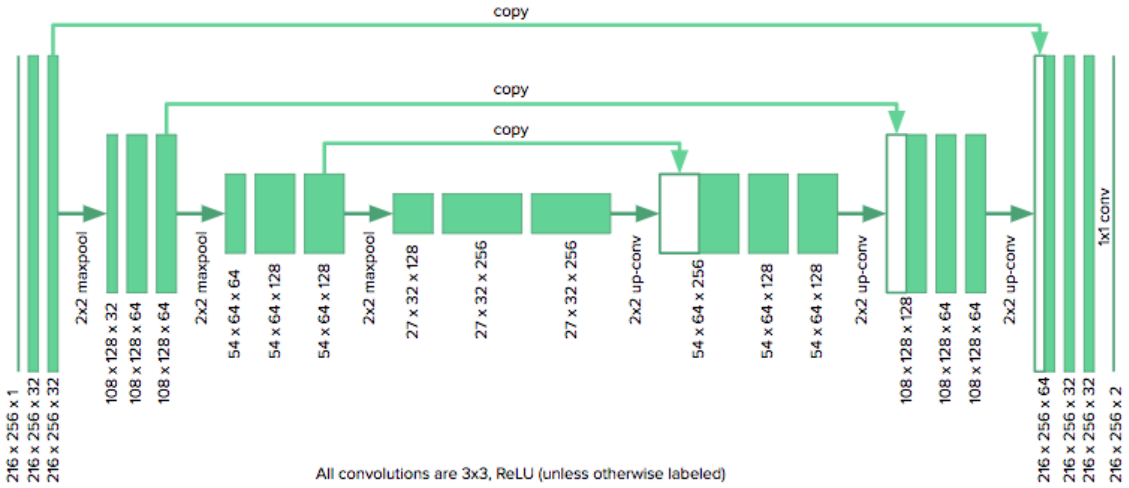


Figure 28. U-Net architecture scheme. Firstly, we have the downsampling phase, with three parts which includes two Convolutional layers and a MaxPooling layer. Afterwards, we have the upsampling phase with three parts including two Convolutional layers and an UpSampling layer. Additionally, we copy the last feature map of each downsampling part to the upsampling one.

However, this figure does not represent our actual networks. We are working with 4 different U-Nets detailed on the next chart. All of those networks have some hyperparameters in common.

- Kernel size:

Every single network has different kernel sizes, because everyone works with a different sample size. For the 2D dimensional images, we are working with 3x3 kernels in all the networks, but the last layer, which has a single kernel of 1x1. Nevertheless, all of the 3D images require 3D convolutions with 3D kernels (3x3x3), except the last layer, that has a kernel of 1x1x1.

- Number of kernels:

U-Net architecture use an increasing number of kernels, from the first layer to the center of the network. Each block of convolutions has the double number of kernels than the previous one. Afterwards, in the upsampling phase, there is a decreasing number of kernels. Nonetheless, instead of starting with 32 kernels per layer we are using 8 kernels because it provides better performance than the one with 32 due to that one has too much parameters and it overfits. The details of that issue will be commented on the discussion.

- Dilation ratio:

We are going to apply a dilation ratio in some networks. This dilated ratio is a way to improve the feature map by making convolutions with dilated versions of the original kernels. Dilated kernel is a larger kernel with zeros between the different weights. This can be seen on Figure 30. We are applying 3x3(x3) kernels that will be dilated to 5x5(x5),

afterwards to 7x7(x7), etc. These dilated kernels could use information located further than the one catch by no dilated kernels.

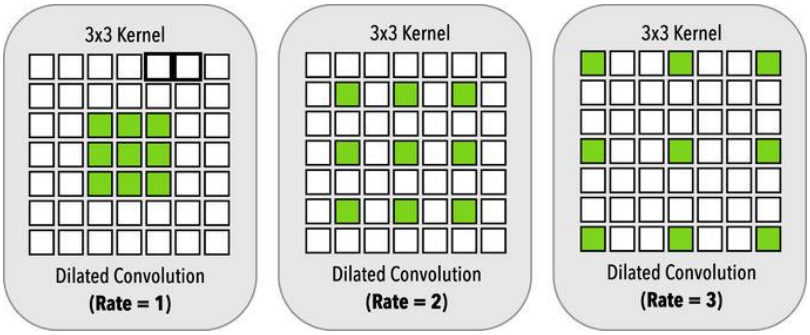


Figure 29. Dilated kernel scheme (a) image is the defect kernel. (b) image is a kernel with a dilation ratio of 2, which results in the same kernel than the previous one with zeros between its weights. (c) image is a dilated kernel by a factor of 3.

Finally, the last layer consists of one kernel of 1x1 that compress the last feature map in only one slice. In addition, for 3D patches, the third dimension of this kernel has the size of the depth patch.

U-Net	Kernel size	Number of kernels	Dilated ratio	Number of parameters
512x512x1	3x3	8,16,32,64	No	121,969
512x512x3	3x3x3	8,16,32,64	No	365,201
512x512x5	3x3x3	8,16,32,64	No	326,307
512x512x7	3x3x3	16,32,64,128	Yes	1,456,681

Table 1. U-Net structures. The first column corresponds to the network we are working with, the second one shows the kernel size, the third one shows the number of kernel per layer, from the first patch of convolutions to the central one. The upsampling phase is just symmetric. The fourth column shows if we are applying a dilation kernel and the last one corresponds to the number of parameters or weights we are using.

4.1.2. DenseNet

DenseNet was introduced in *Chapter 2* too. Nevertheless, we are going to focus on its structure at this point.

DenseNet is different than U-Net. Not only in terms of concatenations, but in terms of internal variability. DenseNet is not a predetermined architecture, but a general concept that focus on concatenating a huge number of layers.

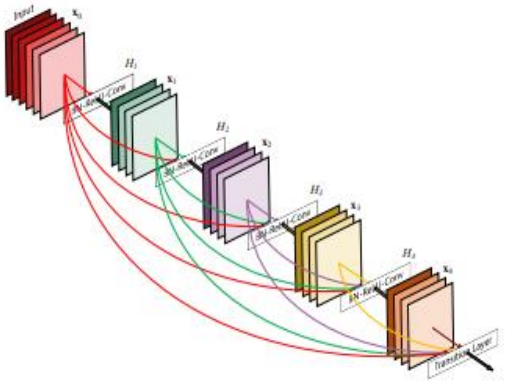


Figure 30. DenseNet scheme. Every layer is connected to all the followings. Nevertheless, in our case, this will be a patch of the whole network, where we are applying three patches like this one but with three convolutional layers instead of five.

As we can see in Figure 31, each layer is connected to all the following layers, providing its information to all the network. Nonetheless, DenseNets are divided into blocks where all the layers are densely connected, but there is no concatenation between layers of different patches, as we can see in the next figure.

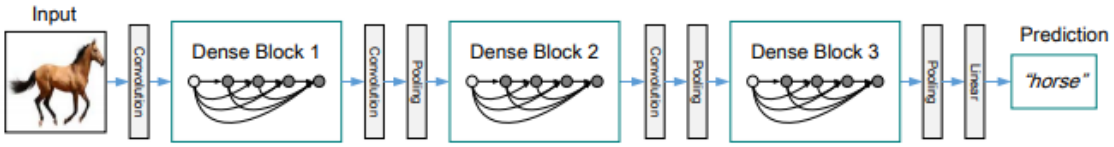


Figure 31. Common DenseNet scheme. We apply three Dense blocks where we the layers are connected like in Figure 31. Dense blocks are connected by a MaxPooling operation, which downsamples the activation maps. Finally, there is a fully connected layer in order to produce a likelihood vector.

The scheme we can observe in Figure 32 is the most common DenseNet applied to image classification [32]. However, improving the result of a complete output image requires taking the information from the abstract layers and use it for denoise the input. Those concatenations are based on the U-Net structure, so we are going to mix both of them.

In our case, we are considering a three dense blocks network, starting with three convolutional layers, doing a MaxPooling operation after, other three convolutional layers, an UpSampling operation, and finally, other three convolutions. We can see this architecture on the next figure.

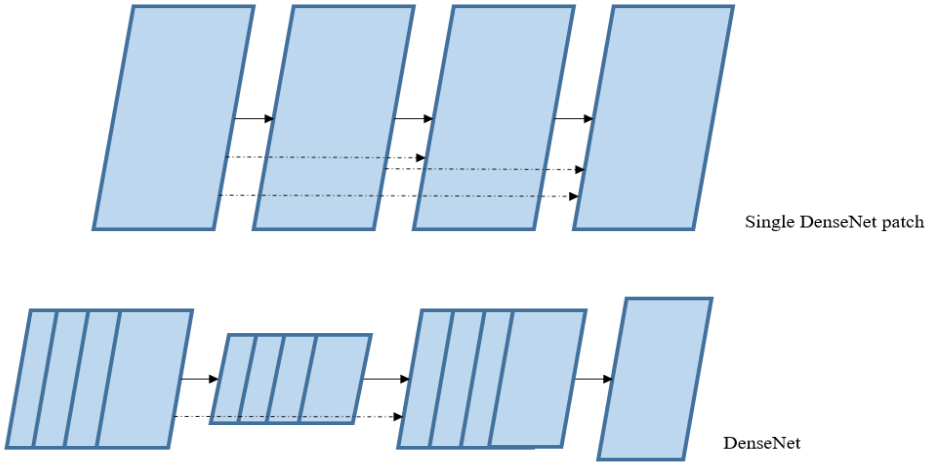


Figure 32. DenseNet and Dense block scheme. Top image explains the Dense Blocks we are applying. They have three convolutional layers. Each activation map is connected to the other layers. Afterwards, we can see in the bottom part of the image the whole network. Three Dense blocks are connected, between the first and the second one there is a downsampling layer and, between the second and the third one there is an upsampling layer.

We are going to apply different DenseNet for providing better images or for being faster in the denoising task. The most important feature of this architecture is that it requires a much smaller number of parameters than U-Net. This could reduce memory usage and the computational time, but all the concatenations demand more memory, so, at the end, there is not a huge memory usage improvement.

We are applying three different networks. One fed with 2D patches and two others provided with three dimensional patches of 512x512x3 and 512x512x5 pixels, each one requires different specifications, which will be defined on the following lines.

- 512x512x1 network:

This network follows the scheme proposed on Figure 32. First patch has 3 layers of only 4 kernels each one, the central patch has 4 kernels per layer, and the last one 4 too. That result in only 2,741 parameters, what allows to deal with a huge number of samples without breaking the memory.

- 512x512x3 network:

This other network based on the scheme of Figure 5 has the same structure than the previous one. Nevertheless, the number of parameters is higher because these 3D kernels are larger than 2D kernels, which implies 14,649 parameters. However, this number is very small too, comparing with the 3.000.000 parameters of U-Net.

Furthermore, the last layer of the network only contains one kernel of 1x1x3, which takes all the channels and compress it in the resulting image.

- 512x512x5 network:

This last network is based on the previous scheme. Nonetheless, the structure changes as there are the double of kernels per layer. The first patch has 8 kernels per layer, the second one has 16 kernels per layer, and the last one 8 per layer. This kernel augmentation implies a great increase of parameters: 57,809. The last layer has one kernel too, but in this case the dimensions are 1x1x5 because there are five channels instead of three. All this information is summarizing in the following table.

DenseNet	Kernel size	Number of kernels	Dilated ratio	Number of parameters
512x512x1	3x3	4, 4, 4	No	2,741
512x512x3	3x3x3	4, 8, 4	No	14,649
512x512x5	3x3x3	8, 16, 8	No	57,809

Table 2. Dense Networks summary. Each network is fed with different patches. The first column corresponds to those sizes, the second one to the kernels we are applying, the third one to the number of kernels. The forth column shows the networks that will have a dilated kernel and the last one corresponds to the number of parameters the network has.

4.2. Fine-tuning of the hyperparameters

Fine-tuning is the process whereby the different hyperparameters are optimized by training the network, looking at the results, and changing their values. Commonly, these hyperparameters are tuned with low resolution images, for instance, with 64x64 pixels instead of 512x512, what implies a training phase much faster.

Unfortunately, changing the dimensions of the images changes noise model because the pixel dimension changes. The images third dimension is variable to keep the

same pixel size, so, if we change this size, the noise model will be variable depending on the image. Then, we are working with the whole images for tuning the network, but not with the 3D patch versions.

There are a lot of hyperparameters that will be explained in the following points.

4.2.1. Loss function

Training the different parameters implies computing the derivatives of the error in each neuron of each layer. However, this error could be calculated with different functions. All these functions are used in different contexts depending on the output we are looking for.

We are applying a function called *Mean Squared Error*, commonly used in the denoising task because it is applied assuming the noise model is Gaussian. However, this function performs well with our noise model too. There are other tasks when the output is a binary mask and we should apply different functions, belonging to the *categorical* group: as *categorical crossentropy*, *sparse categorical crossentropy* and *binary crossentropy*.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(f_i - y_i)^2$$

4.2.2. Optimizer

When we compute the derivatives of the loss function in each weight, we apply a step. Derivatives indicate the directions in which the loss is minimized, and the steps are the distance we advance in that direction. This method is called *Gradient Descent*, because we are using the direction of the gradient for descent to the lowest part of the loss function, which is the solution we are looking for.

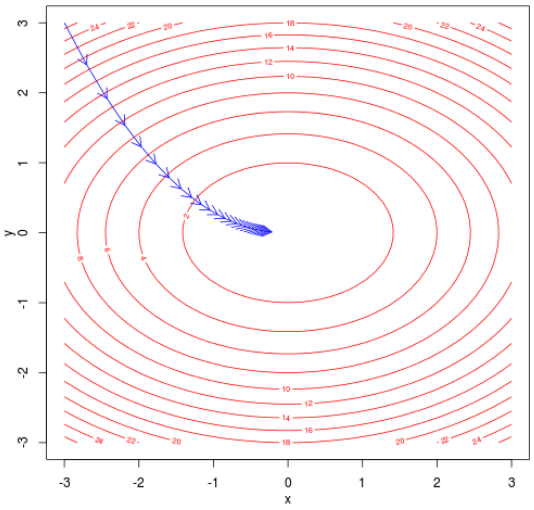


Figure 33. Gradient Descent graph. Blue arrows are the different iterations the model is doing. Each one is directed to the global minimum, which, using a loss function, is the sought

Nevertheless, this method has several problems: gradient descent could be stucked in local minima or in flat zones. Considering the irregular surface of the loss function, there are a lot of

minimizing directions that are not the propest one. Finally, if the step, also called learning rate, is very small, the method will take a lot of time to find the solution; notwithstanding, a large step will be faster, but it will not find the best solution, because finding the best point of a function requires a small step which does not spin around it.

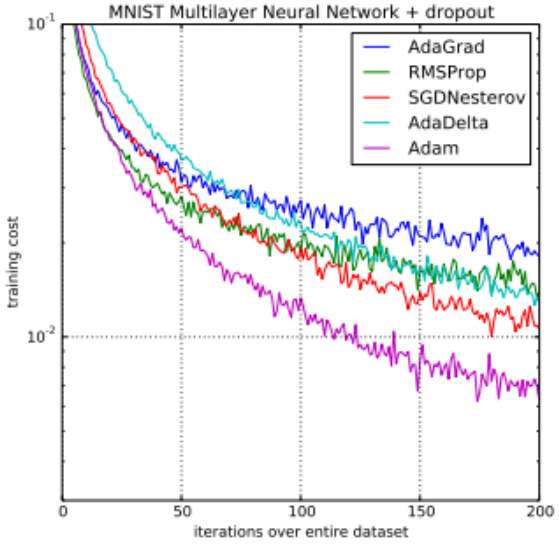


Figure 34. Different optimizers effect on the number of iterations. Adam is the best one, but AdaDelta, SGDNesterov, RMSprop and AdaGrad improve the basic performance.

We could apply a momentum to the learning rate, which is a small contribution of the previous derivative. This momentum helps the gradient to avoid local minimums and saddle points because, even if the current derivative is zero, it could leave from there. In addition, applying a momentum decrease the time it takes to find the absolute minima. There are other mathematical improvements that results in different optimizers, as AdaGrad, which gives more importance to the current gradient by squaring it, RMSprop, which applies a mix of AdaGrad and a learning rate decay, explained in Callbaks point, and Adam, which applies the momentum and the RMSProp idea, and we are using it in the current research.

4.2.3. Callbacks

Callbacks are certain tools Keras, the Python library for deep learning, give us for improving the training phase. These tools are all applied in the training phase, and we are going to use three of them.

Adam optimizer allows us to avoid local minima through the momentum, which keep the gradient out the local directions. Nevertheless, irregular function surface slows down the computation of the gradient because those local directions influence in the final step.

The original way of computing the gradient consist in calculating the learning rate of each weight with each sample, and, afterwards, computing the average of all of them. This average is robust against noise, but requires a lot of time, usually, high number of samples implies a memory failure, so that is not practical.

Reducing the computing time requires to estimate the gradient with less data, and the simplest way of calculating this is the *stochastic gradient descent*, consisting in using only one sample to make the estimation. This is faster, but it could not avoid irregular surfaces, so, the number of epochs will increase.

However, we could calculate a *minibatch gradient descent*. This gradient is computed by the average of a sample batch, instead of compute the average of all the dataset. Minibatches allows a good approximation of that average, is faster, and is less memory expensive.

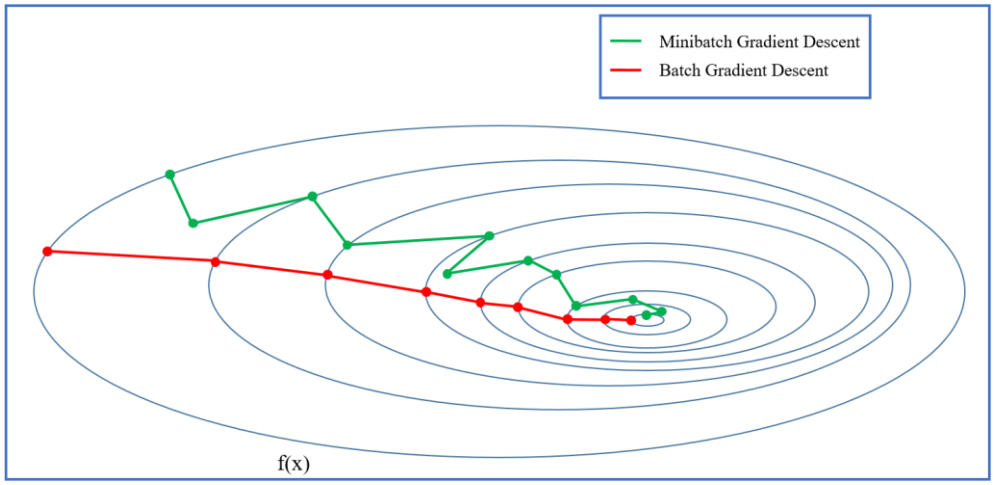


Figure 35. Complete dataset (red) and Minibatch (green) gradient descent comparison.

The hyperparameter is the size of that batch. A bigger batch will estimate better the gradient, but is slower. On the other hand, a smaller batch will perform worse, but faster. This size will be different for each network, defined in the following table.

U-Net	Kernel size	Number of kernels	Dilated ratio	Number of parameters	Batch size
512x512x1	3x3	8,16,32,64	No	121,969	16
512x512x3	3x3x3	8,16,32,64	No	365,201	4
512x512x5	3x3x3	8,16,32,64	No	326,307	4
512x512x7	3x3x3	16,32,64,128	Yes	1,456,681	4

Table 3. U-Net summary. The different columns explain the size of the kernel, the number of kernels we are applying, the dilated ratio, the number of parameters that the network has to improve and the size of the minibatch.

DenseNet	Kernel size	Number of kernels	Dilated ratio	Number of parameters	Batch size
512x512x1	3x3	4, 4, 4	No	2,741	16
512x512x3	3x3x3	4, 8, 4	No	14,649	16
512x512x5	3x3x3	8, 16, 8	No	57,809	8

Table 4. DenseNet summary. The different columns explain the size of the kernel, the number of kernels we are applying, the dilated ratio, the number of parameters that the network has to improve and the size of the minibatch.

There is also the problem of choosing the learning rate. It would be acceptable to choose an intermediate learning rate, large enough to be fast, but small enough to find a proper result. Notwithstanding, the best thing we could do is to use a variable learning rate. We start with a large learning rate (2E-4) and, when its size is too large to improve the performance, use a smaller one. We can adapt this learning rate with a callback called *ReduceLROnPlateau* which needs as parameters the number of epochs without any improvement necessary for reduce it, in our case 5 epochs, and the factor between the

previous learning rate and the current one, in our case 0.2, which means a decrease of 80%.

Furthermore, we are applying other callback methods for saving the best model, *ModelCheckpoint*. This function saves the weight values in an archive '.h5'. The parameter this callback needs to be provided with is the metric it must monitor to deduce which epoch gives the best performance. We are choosing the one with the lowest validation loss, in this case, the validation mean squared error. Additionally, we need to write the name of the archive, but it is not an hyperparameter. This callback only saves the weight value, so, each time we use the network we need to build it up and load the correspondent file.

Finally, the last callback we are using in this research is *EarlyStopping*. This callback is used for ending the training up. EarlyStopping needs to be provided with the number of consecutive epochs without any model improvement and the metric used for monitoring it. Obviously, the number of epochs in this callback must be lower than the one from ReduceLROnPlateau, because we need to give time to the model to improve with the newest learning rate. In our case, these two parameters are: 11 epochs and validation loss.

Callbacks	Number of epochs		
ReduceLROnPlateau	5	Learning Rate decay	0,2
ModelCheckpoint	-	Metric	MSE
EarlyStopping	11	Metric	MSE

Table 3. Callbacks summary. The first column explains the number of epochs for activate the function, and, afterwards, we explain the details of each function: the learning rate decay and the metrics we are applying for considering the best performance.

4.2.4. Other parameters

We commented all the hyperparameters in the last points, but there are other parameters that need to be set for developing a network. Firstly, we must define the number of epochs in the training phase. We set an automatic saver and an early stopping, but we need to specify a default number of epochs. *EarlyStopping* will apply under that number of epochs. If this callback is not activated before arriving to that number, the training will stop by itself.

In our case, we are setting a maximum of 400 epochs.

Furthermore, there is one parameter directly related to the batch size, the number of steps per epoch. *Steps per Epoch* is the number of minibatches taken by the network in each epoch. If we train the network without data augmentation, we could not define that number, but, if we use that augmentation, we must set it. For those cases, we are applying the number of training samples divided by the batch size. If that number is not an integer, we ceil it.

4.3. Volume generation

Once we have obtained a denoised batch, we must generate a whole 3D image that could be compared with the reference. This volume is generated by inputting the different batches which are modified by the network in order to produce the very central slice correspondent to the reference. All these slices are stocked in a 3D empty matrix with the same dimensions as the reference. Nevertheless, if a networks produces the central denoised slice provided with a batch of N slices (where N is an odd number), first $\frac{N-1}{2}$ original slices will not be denoised, and the last $\frac{N-1}{2}$ either.

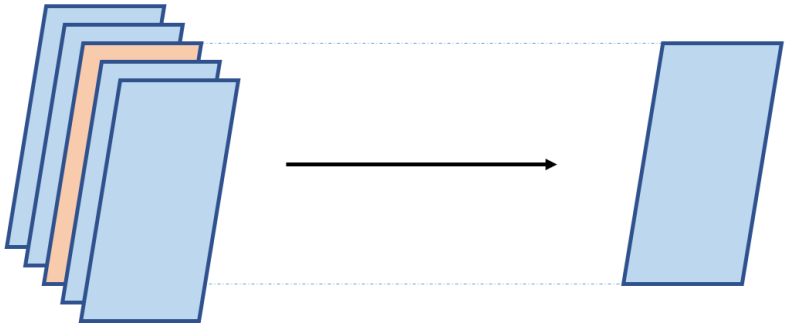


Figure 36. Scheme of denoising process. The left side is the patch of 5 slices, and we want to obtain the central one denoised, which we can see on the right side.

Considering the first slices and last slices of the image have not critical information, we are going to let it empty on the final volume, but we could apply a 2D neural network to denoise those slices.

4.4. Metrics

All these previous points focus on the best parameter setting for achieving our goal. However, we must define some metrics to estimate how good are the different performances and compare them. Metrics must clarify certain important features for the sought goal, as the computational time and the intensity levels.

- *Computational time:*

Computational time will measure the time spent on filtering every single batch and generate the volume with them. Time units used are seconds, so, lower values of time mean faster networks.

- *Mean Squared Error:*

Mean Squared Error measures the pixel intensity difference between the denoised image and the reference by summing all the squared differences. If this value is close to

zero, there is no difference between the reference and the output. This is the metric we are using to evaluate which weight configuration is the best in the training process.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(f_i - y_i)^2$$

- *Signal-to-Noise ratio:*

This metric compares the amplitude of the noise and the signal by dividing the second by the first one. We could compute the noise by subtracting denoised images and reference. After, we must sum each squared noise value to estimate its amplitude, followed by a squared root for normalizing it. Finally, we sum each value of the reference and we divide it by the noise.

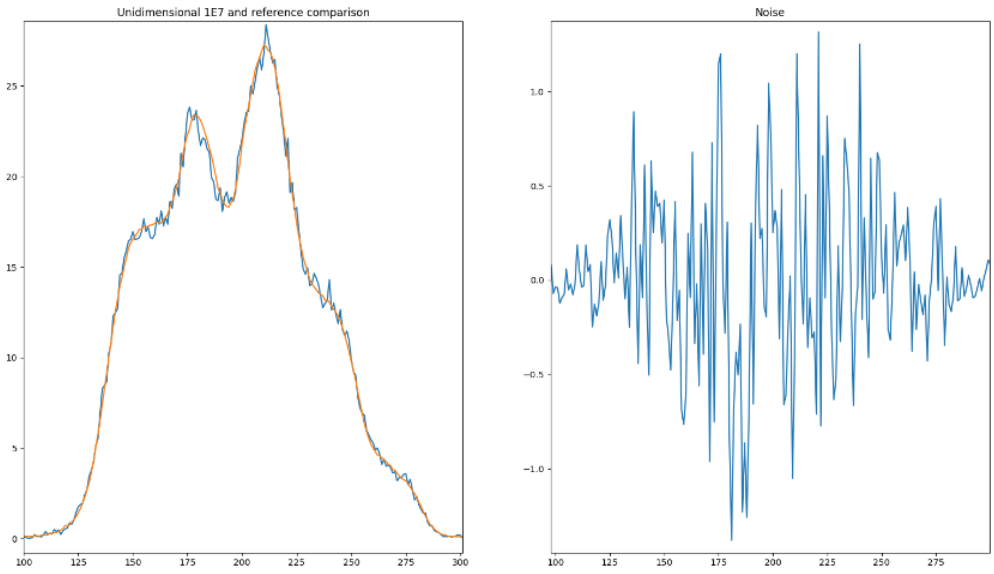


Figure 37. Comparison between reference and input. The first image is the comparison between the reference and the 1e7 interactions image. The second one is the noise of the 1e7 image, obtained by subtracting the reference to the input.

$$\sigma = Im_{ref} - Im$$

$$|\sigma| = \sqrt{\sum(\sigma[i])^2}$$

$$|\delta| = \sqrt{\sum Im_{ref}[i]^2}$$

$$SNR = \frac{|\delta|}{|\sigma|}$$

The symbols used in the previous equations are: σ, noise; |σ|, absolute value of the noise; |δ|, absolute value of the reference, where Im_{ref} = δ.

We are looking for a high signal-to-noise ratio, what implies a practically irrelevant noise. However, there is a huge variance between different patient dose distributions, which implies a directly rapport with the noise distribution and the noise amplitude. Furthermore, there are some patients whose reference images are still noisy, due to the huge size of their tumors. So, there is another metric which provides a more robust results.

- *Improvement ratio (Signal-to-Noise):*

Considering signal-to-noise ratio depends on the quality of the original input, that metric could not demonstrate the real performance of the network. Signal-to-noise ratio shows the quality of the output, but that quality depends on the network and on the input quality. But we can minimize the dependence on the input quality by calculating the improvement ratio by dividing the signal-to-noise ratio of the output and the input, or, as presented on the following equation.

$$ISNR = \frac{SNR_{out}}{SNR_{in}} = \frac{\frac{|\delta|}{|\sigma|_{out}}}{\frac{|\delta|}{|\sigma|_{in}}} = \frac{|\sigma|_{in}}{|\sigma|_{out}}$$

All these metrics are applied to four patients who compose the test dataset. We will compute the average of the four patients and the standard deviation, results that will provide robustness and reliable data.

- *Peak Signal-to-Noise ratio (PSNR):*

This is the most common parameter used for estimate the quality of a denoised image. MPEG committee has established a threshold of 0.5 dB to determine if there is improvement between two different samples. PSNR is calculated according to the following equation:

$$PSNR = 10 \cdot \log_{10}\left(\frac{MAX^2}{MSE}\right) = 20 \cdot \log_{10}\left(\frac{MAX}{RMSE}\right)$$

Where MAX is the maximum pixel intensity, that use to be between 60 and 80 Grays in our case, MSE is the *Mean Squared Error*, which is defined on the previous page, and RMSE, which is *Root Mean Squared Error*, the squared root of the MSE.

- *Dose-volume histogram (DVH):*

This last metric is not scalar. It is an decumulative histogram relating the dose to tissue volume. It is used in radiotherapy planning, and the idea is to compare the DVH graph of the reference with the DVH of the output. One single parameter we can extract from this histogram is the D_{95}, which is the minimum dose given to the 95% of the tumor volume, and its unities are Grays (Gy).

The most important feature of this histogram is the slope, which must be as steeper as possible for the target, what means that the dose delivered to the tumor is

homogeneous. In our case, we are trying to make a DVH similar to the reference, which, theoretically, is the real behavior of the radiation beam.

In our case, we could only perform the DVH on one lung tumor image because we do not have the tumor contours of the other images. For this image, the reference has a DVH_{95} of 65.45 Gy.

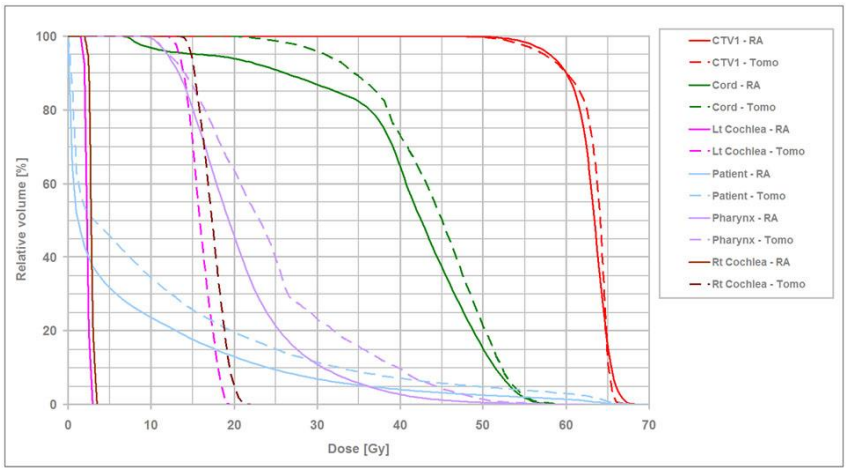


Figure 38. DVH graph. We can see a lot of lines of different colors, but the most important is the red one, that shows the dose given to the tumor.

Chapter 5. Results

5.1. U-Net

5.2. DenseNet

Chapter 5.

Results

In this chapter we are exposing the results of our research. Firstly, we are describing the performance of U-Net, applied to doses generated by 1e7 particles and by 1e6 particles. Afterwards, the results of DenseNet for 1e7 and 1e6 particles images.

5.1. U-Net

Firstly, we are going to expose the results of the 1e7 doses. We are applying 4 different networks, one fed with 2D images and the other ones fed wit 3D batches (depth: 3, 5, 7).

- 1e7 particles doses:

Firstly, the training of the networks was developed according to the chapter 4. The first network was fitted during (105) epochs, what implied a validation MSE of (0.01502).

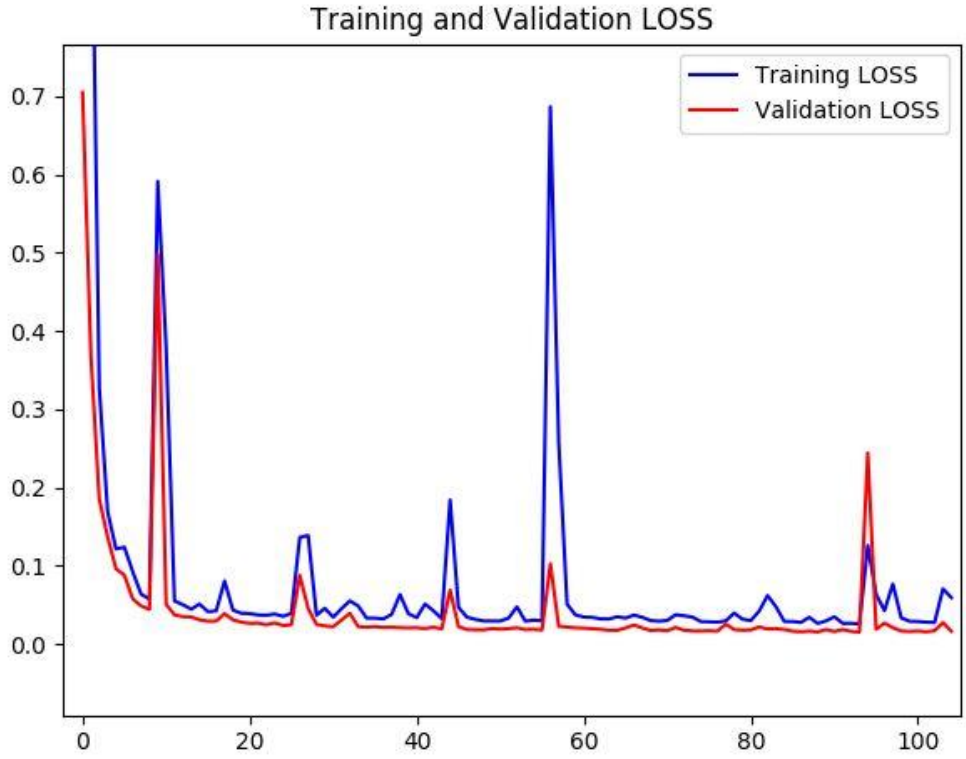


Figure 39. Training and validation loss curves for U-Net (1 slice and 1e7 particles input)

The second network was trained for (117) epochs, and the validation MSE achieved was (0.00957).

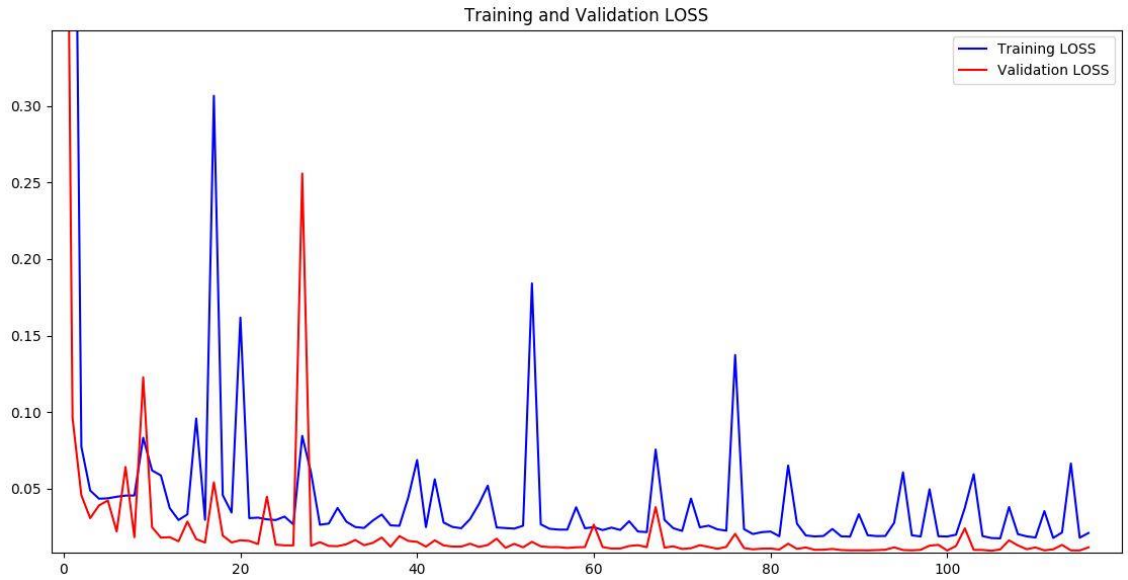


Figure 40. Training and validation loss curves for U-Net (3 slices and 1e7 particles input)

Thirdly, the network fed with a patch of 5 slices achieved a MSE of (90) in (0.01326) epochs.

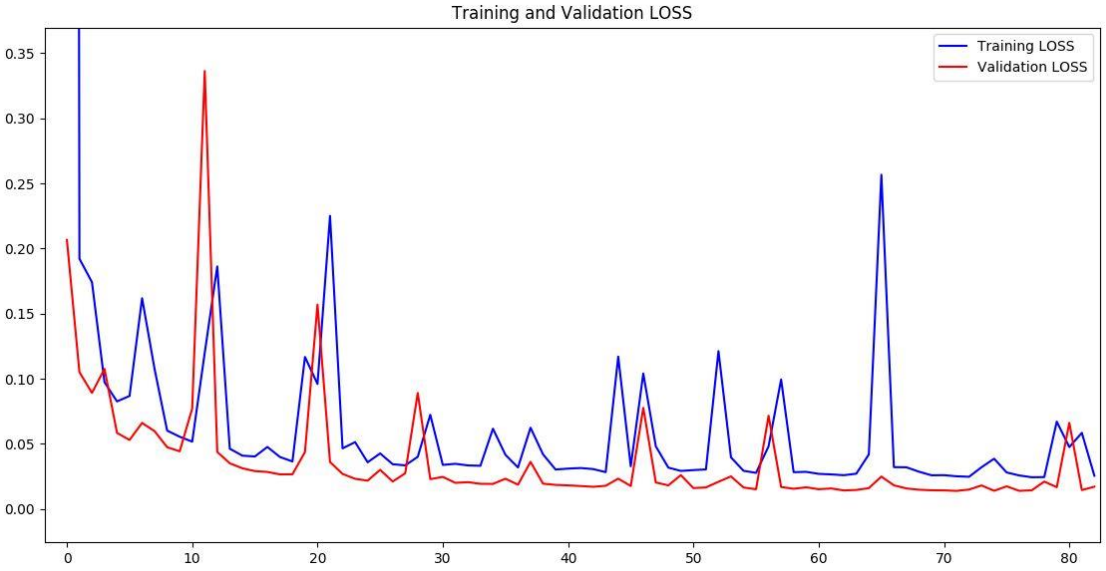


Figure 41. Training and validation loss curves for U-Net (5 slices and 1e7 particles input)

Finally, the last network achieved a MSE of (0.01486) within (25) epochs, as we can see in the following graph.

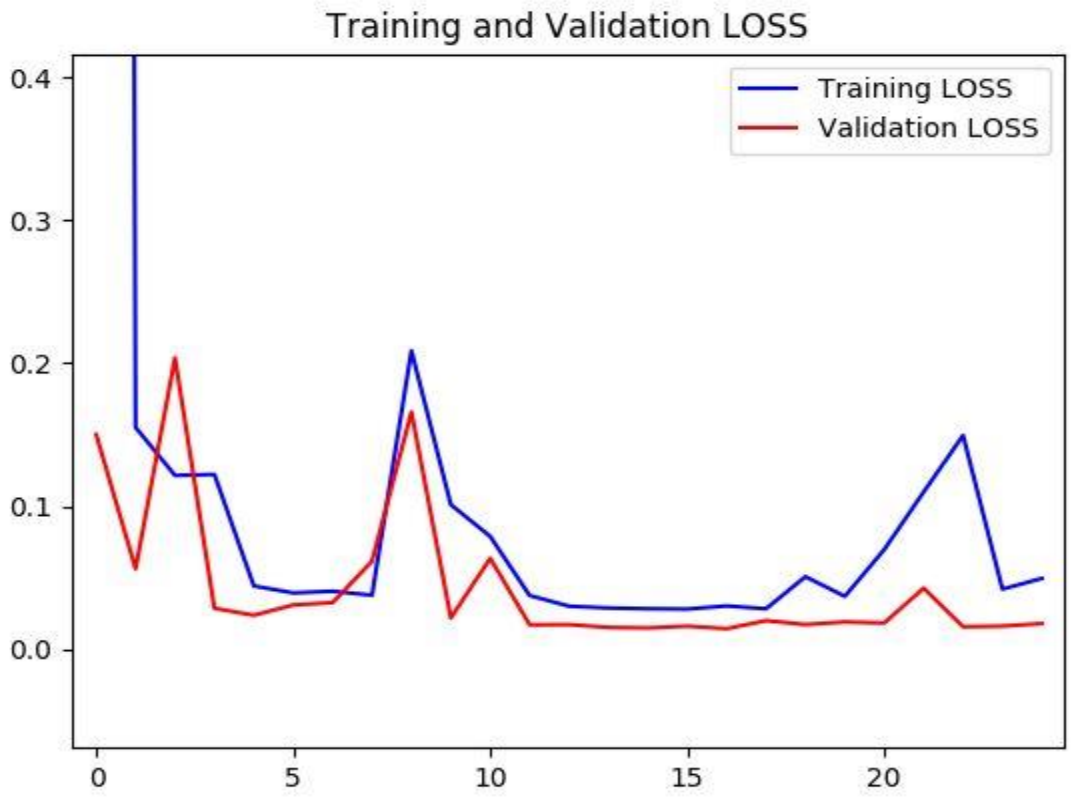


Figure 42. Training and validation loss curves for U-Net (7 slices and 1e7 particles input)

U-Net (1e7)	Computational Time	MSE	SNR	Improvement ratio	PSNR	DVH(95)
512x512x1	1.485 (0.655)	0.024 (0.020)	50.813 (7.307)	3.503 (0.958)	57.985 (7.54)	62.70
512x512x3	5.571 (1.142)	0.015 (0.012)	64.018 (9.67)	4.484 (1.381)	59.993 (7.090)	62.73
512x512x5	11.677 (1.685)	0.017 (0.016)	58.117 (5.129)	4.106 (1.308)	58.886 (6.65)	62.77
512x512x7	17.687 (2.709)	0.019 (0.016)	56.305 (9.321)	3.946 (1.224)	58.852 (7.000)	62.86

Table 4. U-Net (1e7) metrics summary. This table contains seven different columns: first one corresponds to the patch size we are using to feed the network, second one is the computational time (in seconds) required for denoising the whole image, third column corresponds to the Mean Squared Error, fourth column corresponds to the Signal-to-Noise ratio, fifth one is the Improvement ratio (ISNR), sixth one is the Peak-Signal-to-Noise ratio and last one corresponds to the DVH_{95} (Gy), remembering than the reference DVH_{95} is 62.45 Gy. The numbers out of brackets corresponds to the average of four test patients, and the numbers inside the brackets corresponds to the standard deviation. DVH_{95} is not the average of several patients because it was only one contour patient available in our test dataset.

These networks were described on chapter 4, so, according to the metrics described there, the performance is:

- 1e6 particles doses:

All these networks were trained as described on chapter 4, and the training process is summarized in the following table:

U-Net (1e6)	Epochs	Val MSE
512x512x1	101	0.06487
512x512x3	67	0.03440
512x512x5	73	0.04796
512x512x7	41	0.04098

Table 5. U-Net (1e6) training summary. This table has three columns, the first one corresponds to the patch size, the second one corresponds to the number of epochs until EarlyStopping was activated, and the third column corresponds to the validation Mean Squared Error achieved.

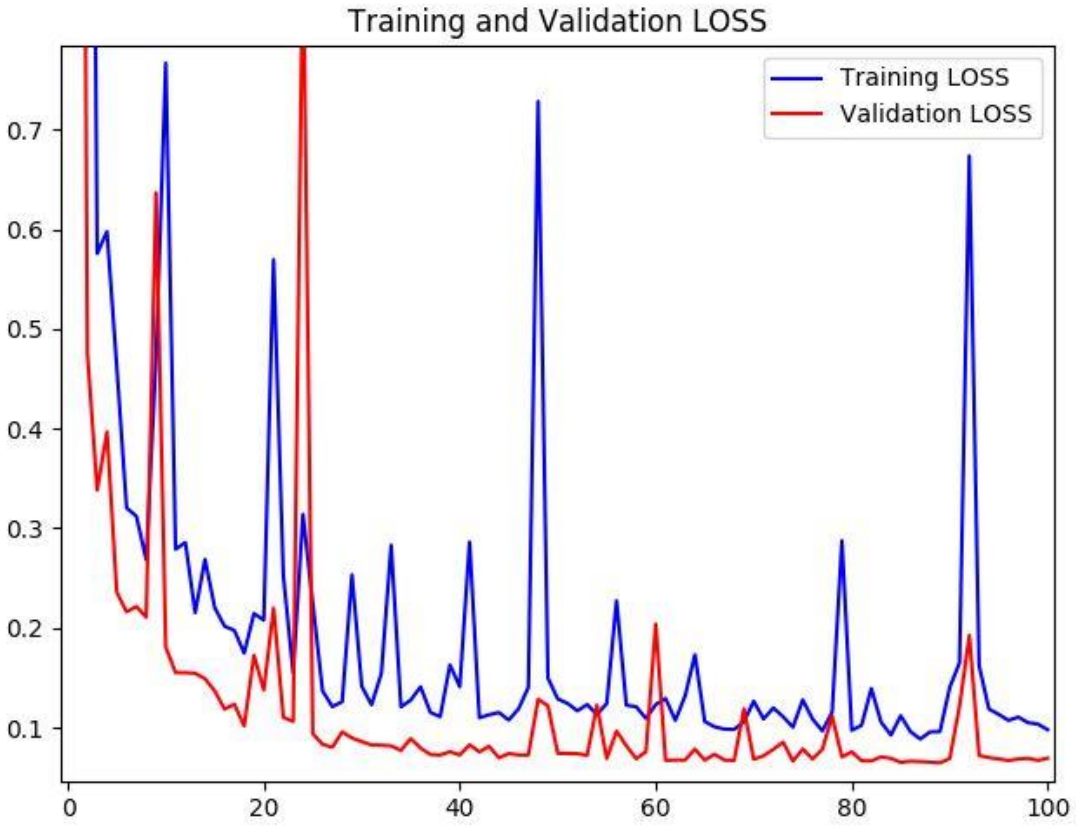


Figure 43. Training and validation loss curves for U-Net (1 slice and 1e6 particles input)

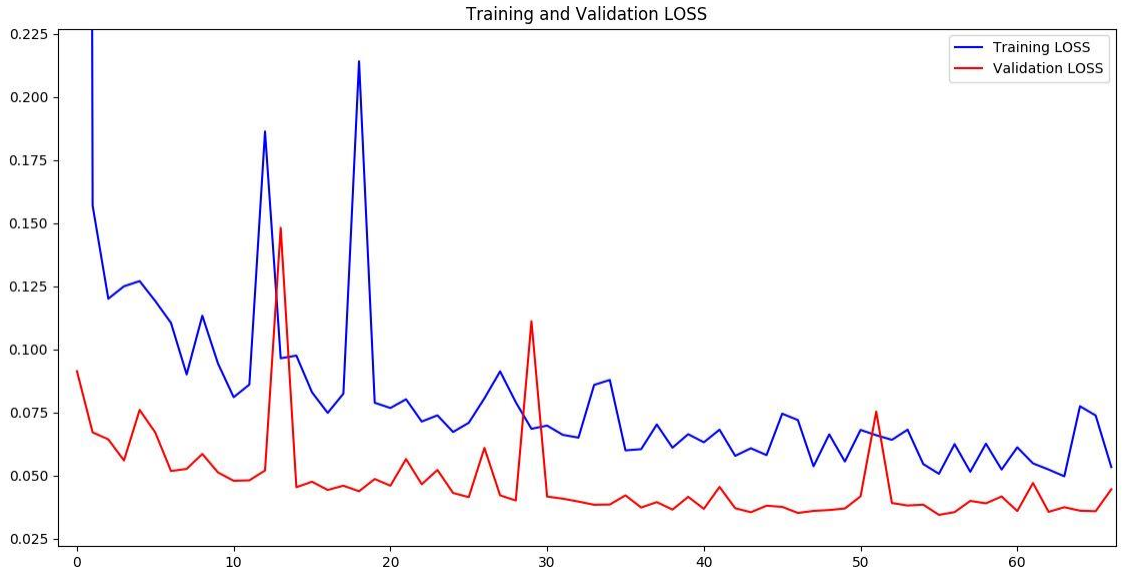


Figure 44. Training and validation loss curves for U-Net (3 slices and 1e6 particles input)

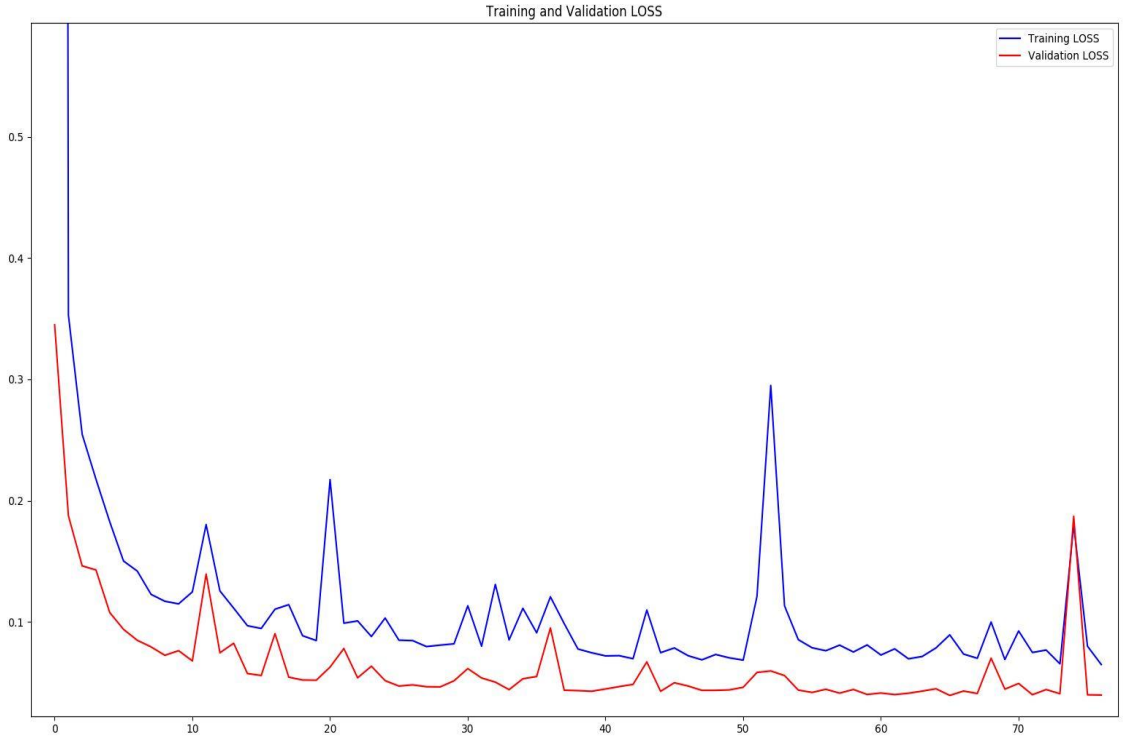


Figure 45. Training and validation loss curves for U-Net (5 slices and 1e6 particles input)

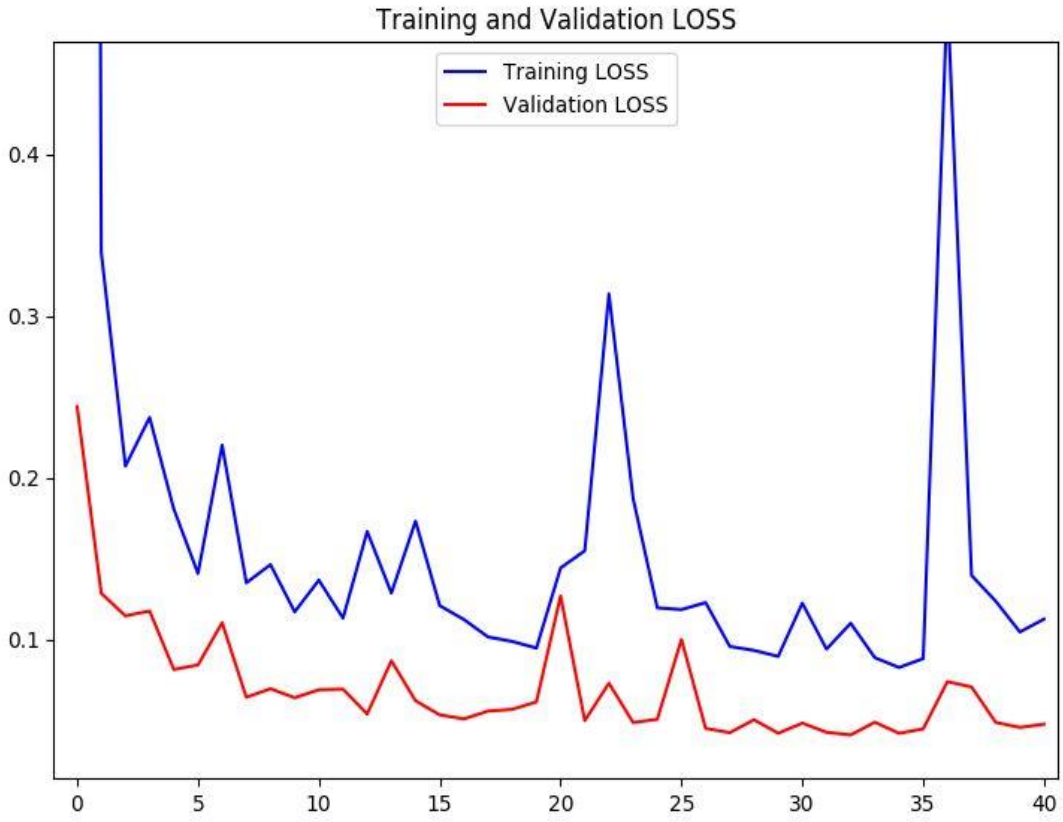


Figure 46. Training and validation loss curves for U-Net (7 slices and 1e6 particles input)

According to the metrics described on chapter 4, the performance of all these networks are:

U-Net (1e6)	Computational Time	MSE	SNR	Improvement ratio	PSNR	DVH(95)
512x512x1	1.330 (0.589)	0.134 (0.112)	26.386 (9.670)	5.051 (0.336)	51.888 (9.57)	62.39
512x512x3	5.064 (0.964)	0.050 (0.041)	35.185 (4.986)	7.347 (1.868)	54.832 (7.568)	62.49
512x512x5	11.216 (1.980)	0.058 (0.046)	30.196 (5.129)	6.572 (2.258)	53.5 (6.442)	62.50
512x512x7	10.586 (1.548)	0.098 (0.083)	28.331 (7.241)	5.632 (0.809)	52.763 (8.678)	62.42

Table 6. U-Net (1e6) metrics summary. This table contains seven different columns: first one corresponds to the patch size we are using to feed the network, second one is the computational time (in seconds) required for denoising the whole image, third column corresponds to the Mean Squared Error, fourth column corresponds to the Signal-to-Noise ratio, fifth one is the Improvement ratio (ISNR), sixth one is the Peak-Signal-to-Noise ratio and last one corresponds to the DVH_{95} (Gy), remembering than the reference DVH_{95} is 62.45 Gy. The numbers out of brackets corresponds to the average of four test patients, and the numbers inside the brackets corresponds to the standard deviation. DVH_{95} is not the average of several patients because it was only one contour patient available in our test dataset.

12.2.DenseNet

This point will be developed as the previous one. It will be break down into 1e7 particles dose and 1e6 particles dose. In this case we are applying three networks, no 4.

- 1e7 particles doses:

DenseNet (1e7)	Epochs	Val MSE
512x512x1	330	0.02190
512x512x3	174	0.0184
512x512x5	85	0.01208

Table 7. DenseNet (1e7) training summary. This table has three columns, the first one corresponds to the patch size, the second one corresponds to the number of epochs until EarlyStopping was activated, and the third column corresponds to the validation Mean Squared Error achieved.

The following table summarize the results of the training, in terms of number of epochs and validation Mean Squared Error.

Additionally, the curves of validation and training loss are attached below this paragraph.

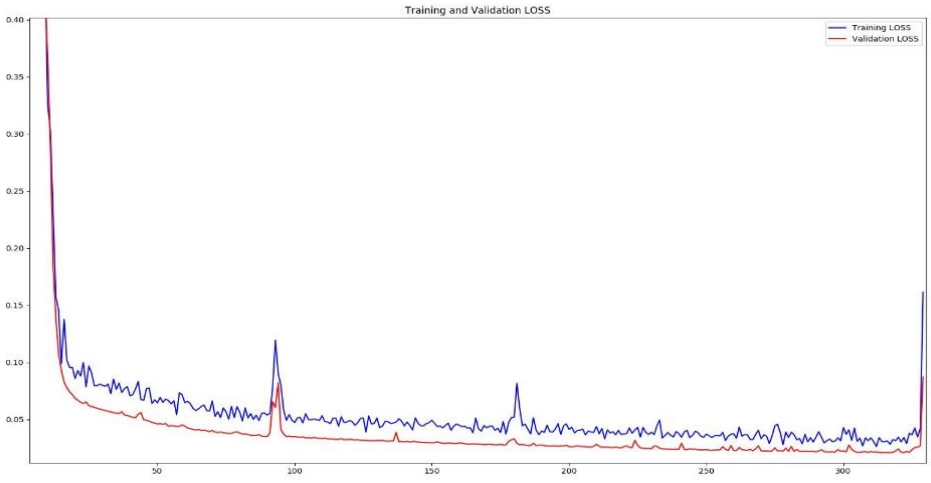


Figure 47. Training and validation loss curves for DenseNet (1 slice and 1e7 particles input)

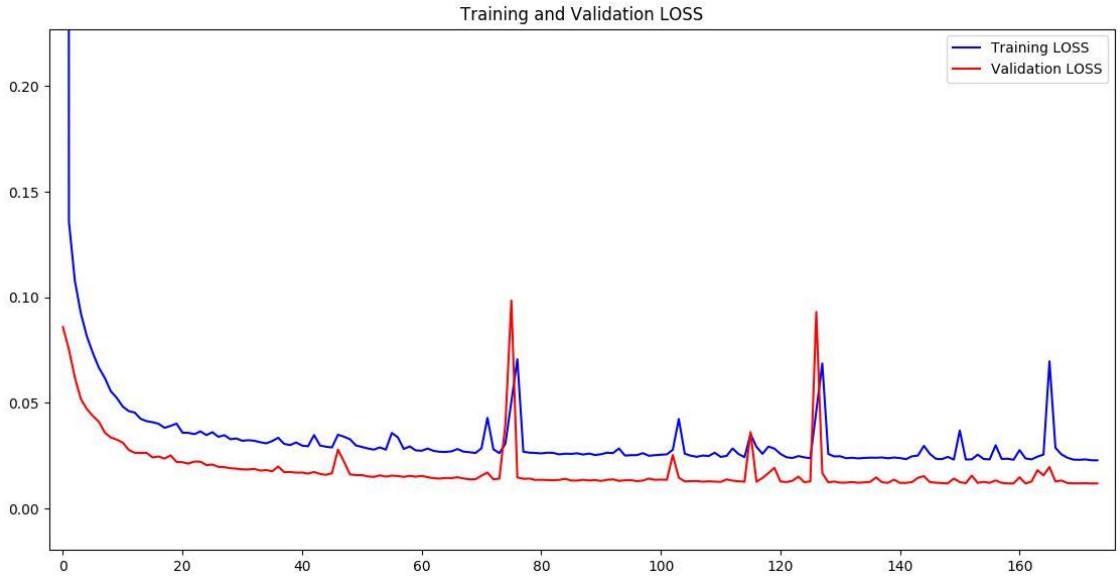


Figure 48. Training and validation loss curves for DenseNet (3 slice and 1e7 particles input)

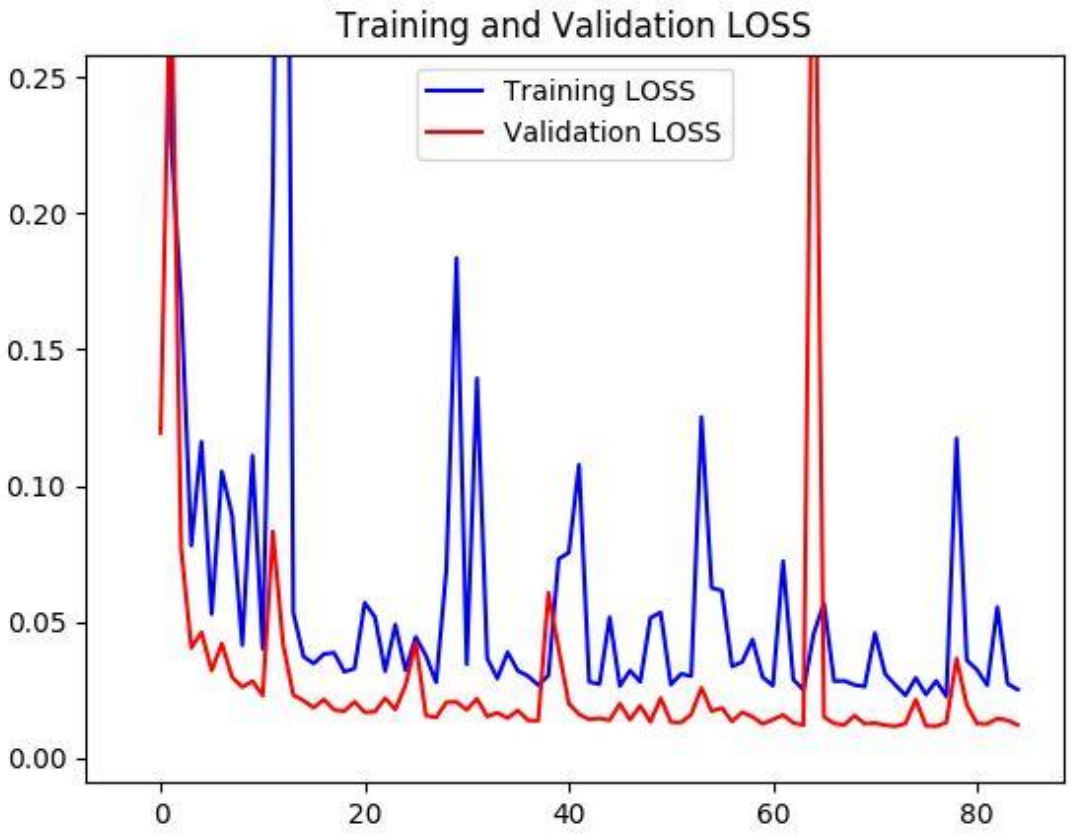


Figure 49. Training and validation loss curves for DenseNet (5 slice and 1e7 particles input)

Moreover, the performances of the different networks are described on the table below. Firstly, we can read the average and, secondly, the standard deviation.

DenseNet (1e7)	Computational Time	MSE	SNR	Improvement ratio	PSNR	DVH(95)
512x512x1	1.391 (0.638)	0.033 (0.028)	42.251 (6.432)	2.949 (0.879)	56.377 (7.139)	62.81
512x512x3	5.231 (0.916)	0.018 (0.015)	58.631 (9.228)	4.077 (1.188)	59.226 (7.244)	62.77
512x512x5	12.326 (1.902)	0.018 (0.015)	60.661 (10.309)	4.157 (1.09)	59.492 (7.600)	62.79

Table 8. DenseNet (1e7) metrics summary. This table contains seven different columns: first one corresponds to the patch size we are using to feed the network, second one is the computational time (in seconds) required for denoising the whole image, third column corresponds to the Mean Squared Error, fourth column corresponds to the Signal-to-Noise ratio, fifth one is the Improvement ratio (ISNR), sixth one is the Peak-Signal-to-Noise ratio and last one corresponds to the DVH_{95} (Gy), remembering than the reference DVH_{95} is 62.45 Gy. The numbers out of brackets corresponds to the average of four test patients, and the numbers inside the brackets corresponds to the standard deviation. DVH_{95} is not the average of several patients because it was only one contour patient available in our test dataset.

- 1e6 particles doses:

Here, there are the table with all the training information and the graphs of the training and validation loss curves.

DenseNet (1e6)	Epochs	Val MSE
512x512x1	268	0.08676
512x512x3	179	0.04474
512x512x5	90	0.40630

Table 9. DenseNet (1e6) training summary. This table has three columns, the first one corresponds to the patch size, the second one corresponds to the number of epochs until EarlyStopping was activated, and the third column corresponds to the validation Mean Squared Error achieved.

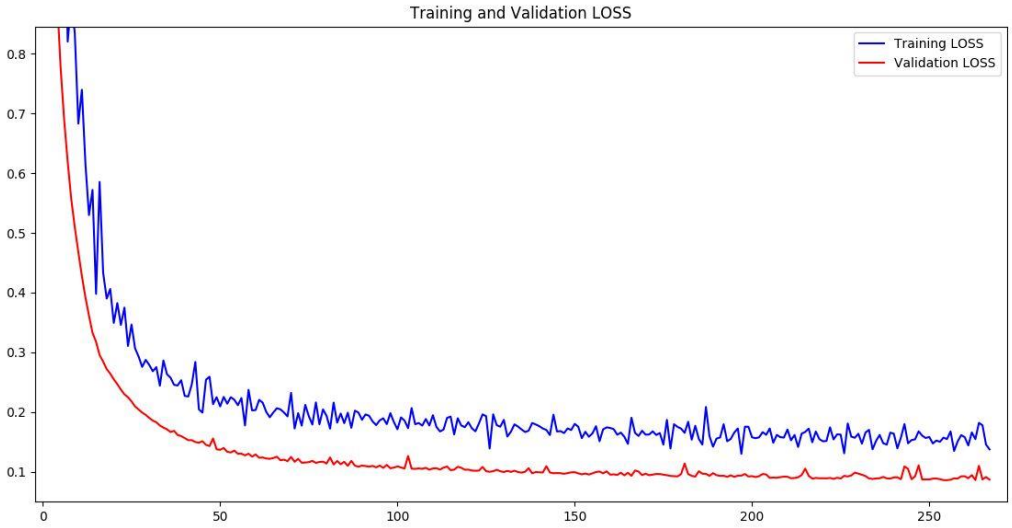


Figure 50. Training and validation loss curves for DenseNet (1 slice and 1e6 particles input)

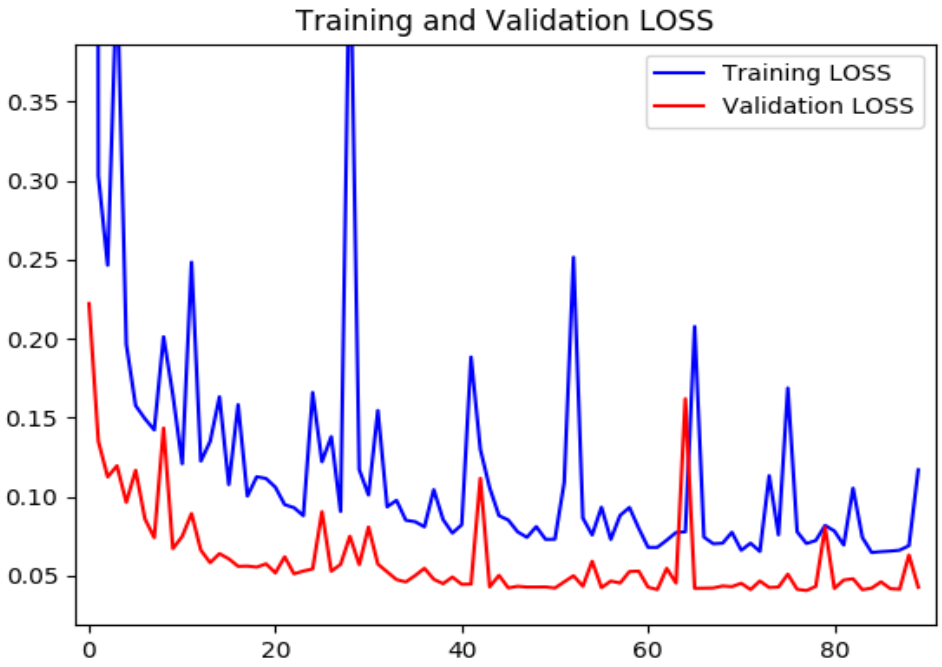


Figure 51. Training and validation loss curves for DenseNet (5 slice and 1e6 particles input)

Additionally, the results of the performances are the following ones:

DenseNet (1e6)	Computational Time	MSE	SNR	Improvement ratio	PSNR	DVH(95)
512x512x1	1.273 (0.610)	0.167 (0.139)	20.367 (3.912)	4.170 (0.881)	49.962 (7.956)	62.27
512x512x3	4.860 (0.763)	0.056 (0.045)	32.033 (4.076)	6.823 (2.032)	54.006 (7.006)	62.32
512x512x5	12.362 (1.836)	0.052 (0.042)	32.813 (4.206)	7.040 (2.200)	54.224 (6.821)	62.29

Table 10. DenseNet (1e6) metrics summary. This table contains seven different columns: first one corresponds to the patch size we are using to feed the network, second one is the computational time (in seconds) required for denoising the whole image, third column corresponds to the Mean Squared Error, fourth column corresponds to the Signal-to-Noise ratio, fifth one is the Improvement ratio (ISNR), sixth one is the Peak-Signal-to-Noise ratio and last one corresponds to the DVH_{95} (Gy), remembering than the reference DVH_{95} is 62.45 Gy. The numbers out of brackets corresponds to the average of four test patients, and the numbers inside the brackets corresponds to the standard deviation. DVH_{95} is not the average of several patients because it was only one contour patient available in our test dataset.

Discussion

Proton therapy is a relative new treatment in oncology that is being used for critical situations where the tumor is surrounded by sensitive tissues. The Bragg Peak allows physicians to irradiate the blank without damaging healthy cells. Monte Carlo algorithms make a very accurate estimation of the dose distribution of proton beams. This accuracy is an extremely important feature because the exact area of the Bragg Peak is very sensitive to little changes on the parameters (type of tissue, beam energy, etc.), and, considering the high dose is delivered there, little movements implies a great pain to healthy areas.

However, Monte Carlo algorithms produces a little noise on its simulations, and, the amount of particles that need to be simulated are too high, requiring more than one hour to compute it. The alternative is generating a dose distribution with a fewer number of particles, but the resultant image is noisier. The current thesis proposes two methods based on deep learning algorithms for denoising the distributions:

- U-Network: based on a hierarchical structure, this architecture has been used during several years for the segmentation task. Its similarity with the autoencoders is the reason why they perform well for that issue.
- Dense Network: this architecture is based on hierarchical structure where all the layers are connected to provide the most important information. These concatenations reduce the number of parameters that we need.

The details of the performance were described in Chapter 5, but the consequences of that results are going to be discussed in the following paragraphs.

Firstly, we are going to discuss the training curves, which represent the training and the validation *Mean Squared Error* along every epoch.

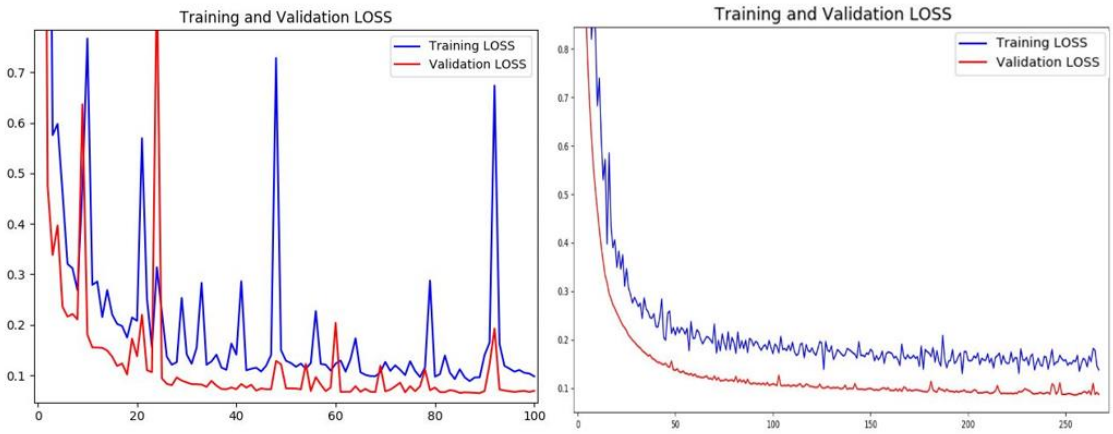
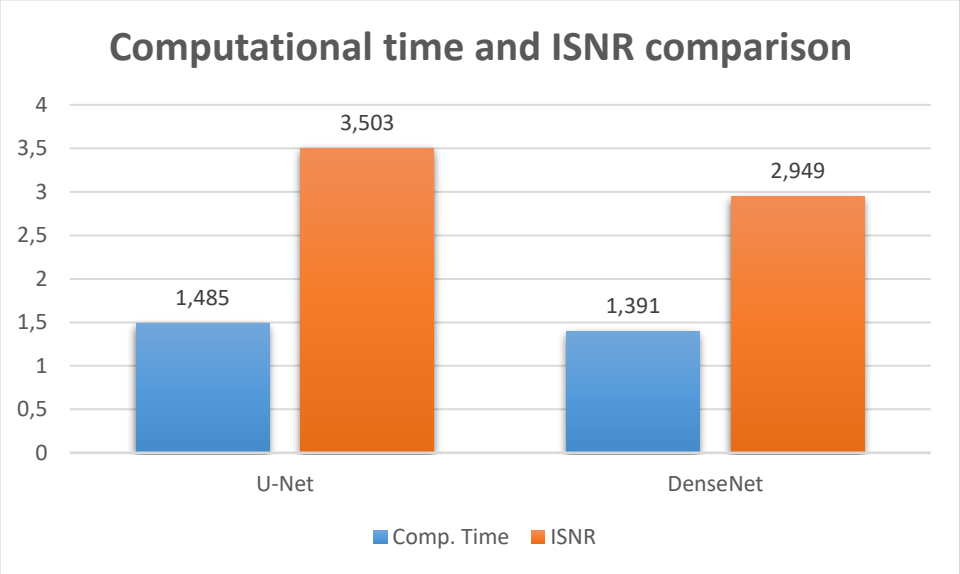


Figure 52. Training and validation loss curves for U-Net and DenseNet. Graph (a) is U-Net fed with 1e6 particles and 512x512x1 slices, and graph (b) is DenseNet fed with 1e6 particles and 512x512x1 slices. The first one took 101 epochs to achieve the best performance and the second one took 268 epochs to achieve the best result.

As we can see in *Figure 52*, there is a huge difference between the training loss curve and the validation one. Usually, validation curve is over the training loss, but, in this case, that curve is under the training one. This is caused because the validation dataset, which has four different patients, is simpler than the training dataset. This could be solved with more data. If the available data was large enough, the complexity of the datasets, the training and the validation, or even the test one, would be the same.

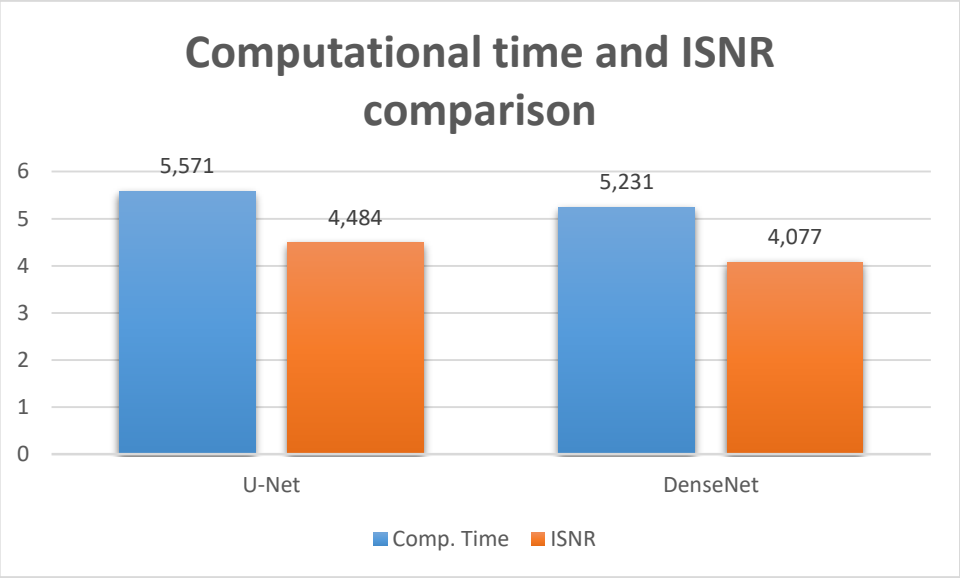
Secondly, we are going to compare the computational time required for filtering one 3D image and the ISNR, which give us information about the improvement of the image.



Graph 1.Comparison between the computational time and the ISNR of U-Net and DenseNet. This comparison is made with the networks fed with patches of one slice only. We can see a little diminution of the computational time for DenseNet, but, there is a great improvement of the ISNR for U-Net.

Considering the information provided by *Graph 1*, the little decrease of the computational time is not worth attending on the great decrease of the performance. Furthermore, taking into account the standard deviation of both times (0.665 and 0.638), we cannot ensure there is statistic differences between them.

Furthermore, we are going to compare both metrics in networks fed with patches of three slices. Theoretically, these two networks must provide a better performance but spending more time on filtering the dose distribution.



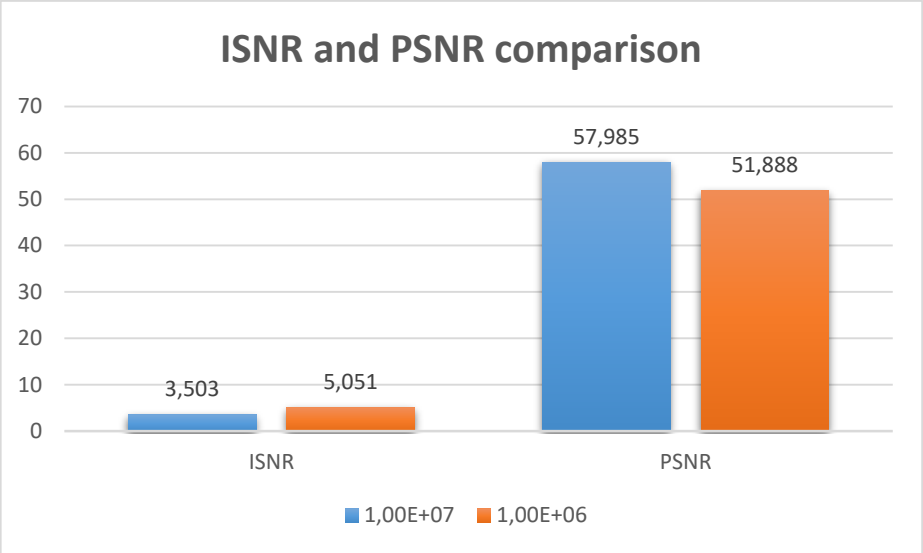
Graph 2. Comparison between the computational time and the ISNR of U-Net and DenseNet fed with patches of three slices. In this case, we realize the behavior is the same than in the previous one, DenseNet spends less time but provide a limited result. Nevertheless, here time difference is higher than the previous one, and the performances are more similar.

As we can see in the *Graph 2*, in this case we can consider to use DenseNet instead of U-Net, but the standard deviations (1.142 and 0.916) do not allow us to discern if there is an appreciable difference. Additionally, we must clarify that, this elevated standard deviation is due to the size variability of the different images, there are images that have 90 slices around, and there are others with 140 slices around.

Now, we are going to discuss about the differences between the images obtained with 1e7 particles and 1e6 particles.

First, we expect a better result on denoising 1e7 particles images because they are less noisy than the ones generated with 1e6. However, these last images are computed faster, so, if we achieve a result good enough on these data, useful images could be generated in a few seconds.

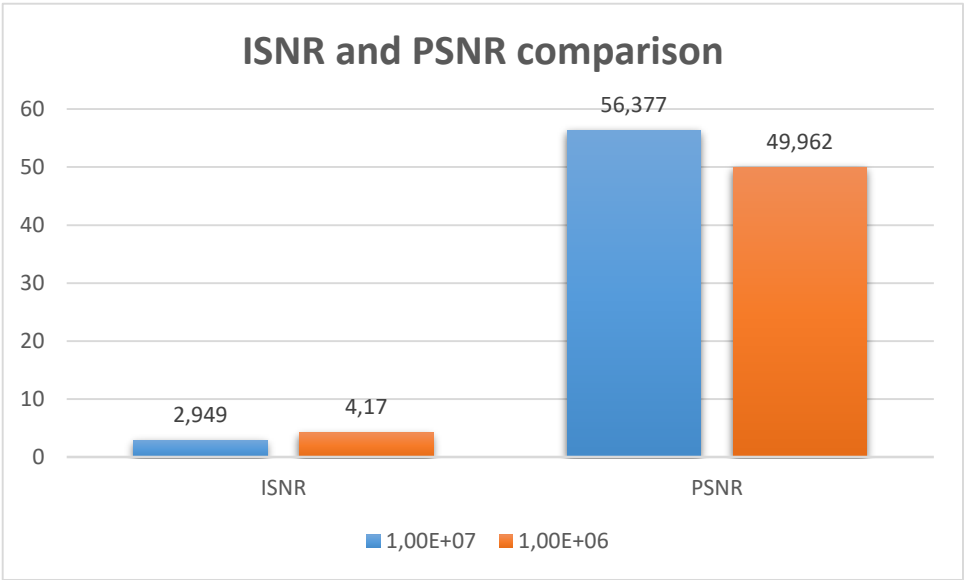
The following graph will compare ISNR and PSNR for outputs coming from 1e7 and 1e6 particles images. ISNR shows the improvement of the input, so, with the same ISNR, 1e7 particles images will be better, as they come from a better starting point. Nonetheless, PSNR shows the quality of the image by itself, so, the higher the PSNR, the better the image is.



Graph 3. ISNR and PSNR comparison between 1e6 and 1e7 particles images filtered with U-Net (patches of one slice). The ISNR is higher for 1e6 images, but the 1e7 images PSNR is 6 points over the first one.

As we can see in *Graph 3*, PSNR of 1e7 images is 6 points over 1e6 one. Maybe this difference is too much important. However, doctors must decide if these 6 points convolve an excessive decrease of quality for the clinical application.

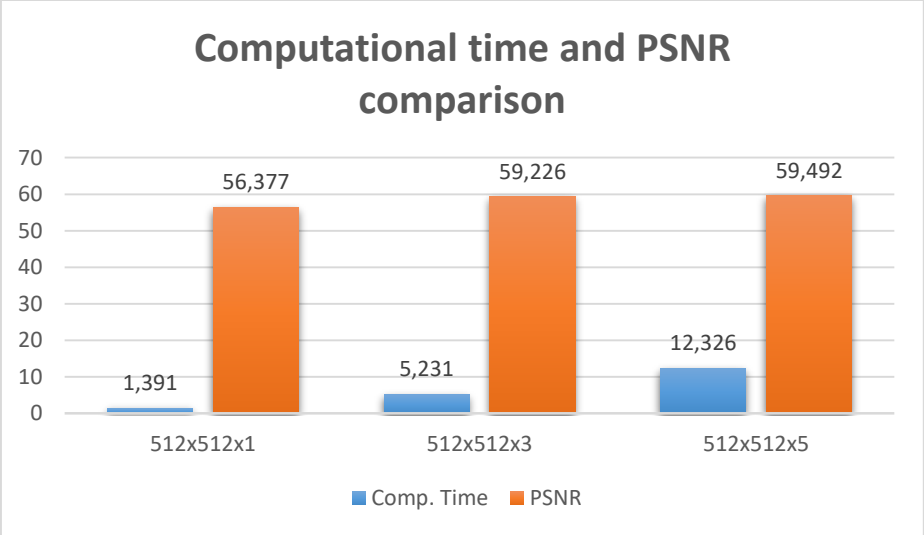
Secondly, we are going to analyze the performance of the Dense network on 1e6 and 1e7 particles images. We are comparing ISNR and PSNR as done before.



Graph 4. ISNR and PSNR comparison between 1e6 and 1e7 particles images filtered with DenseNet (patches of one slice). Here we can see an appreciable difference between the two performances. ISNR of 1e6 particles images is higher, but, PSNR is 6 points under the 1e7 one.

Graph 4 shows us a very similar information than the previous one. PSNR is 6 points lower for 1e6 particles images, and doctors show decide if that quality difference is too much for the clinical application.

Otherwise, we are going to discuss the performance of the different networks according to the patch size. Firstly, we are going to explain the results of DenseNets by comparing the computational time and the PSNR.

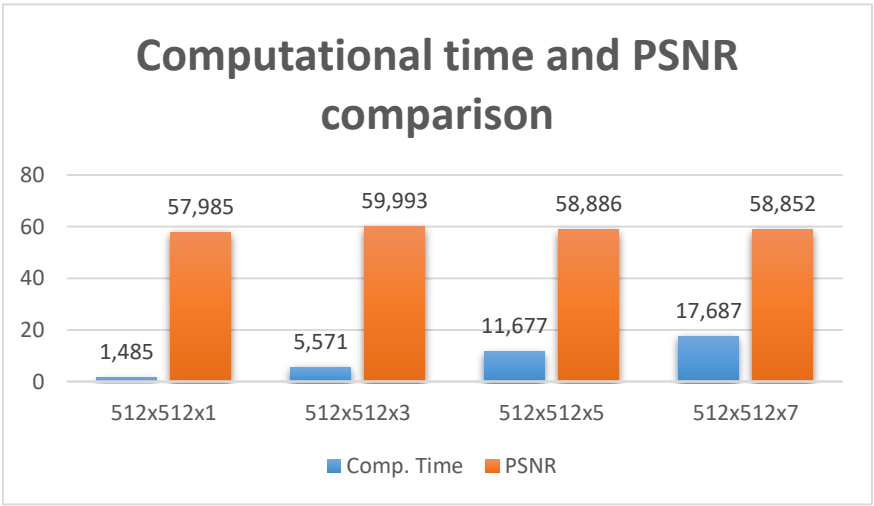


Graph 5. Computational time and PSNR comparison of DenseNets (1e7). There is an important increase of the computational time according to the patch size. However, the quality of the performances does not increase that much. We cannot appreciate any significant difference between the performance of the second and the third one.

Graph 5 show us that the second network provides a better performance than the first one. Additionally, the last network does not improve the performance, but requires more time. The main explanation for this is the lack of data. Lack of data impedes us to use more complex networks, so, models are not specific enough. More complex models allow us to improve our performances, but there is a threshold we cannot cross due to our issue.

Furthermore, we cannot use larger networks due to the lack of memory, so the network dimension has two limits: the memory and the amount of data.

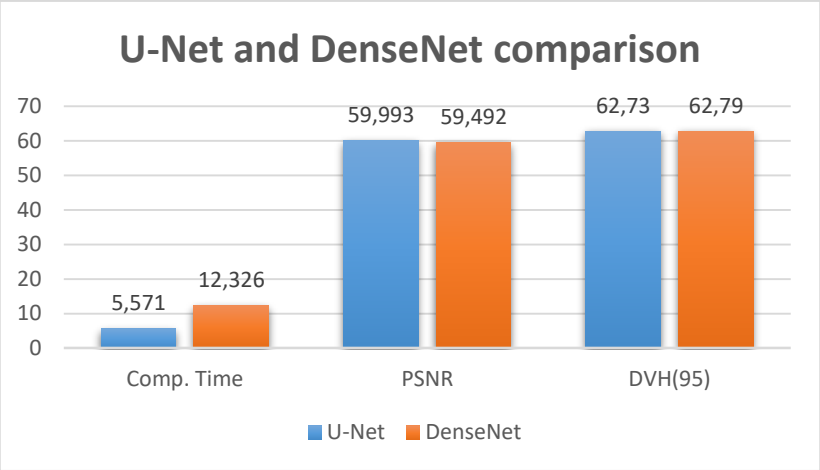
On the other hand, we have the summary of the U-Net performance.



Graph 6. Computational time and PSNR comparison of U-Nets (1e7). As we can see, computational time increase significantly due to the size of the patch. However, PSNR does not increase so significantly, achieving the best performance in the second case.

As we can see in *Graph 6*, the best performance is achieved in the second network, and this is due to the fact that third and fourth networks are very little complex to the amount of information that 5 and 7 slices patches provide. This could be result increasing the complexity of those networks, nevertheless, considering the amount of data we have, this will result in overfitting. We considered those networks, and we added Dropout and Batch Normalization layers for keep the overfitting at bay, but these layers added noise to the final result, so, the final performance was worse than the one we show.

Now, we are going to discuss about the best networks trained, the best U-Net and the best DenseNet. We are comparing the computational time, the PSNR and the DVH_{95} for 512x512x3 U-Net and 512x512x5 DenseNet, both of them fed with 1e7 particles dose simulations.

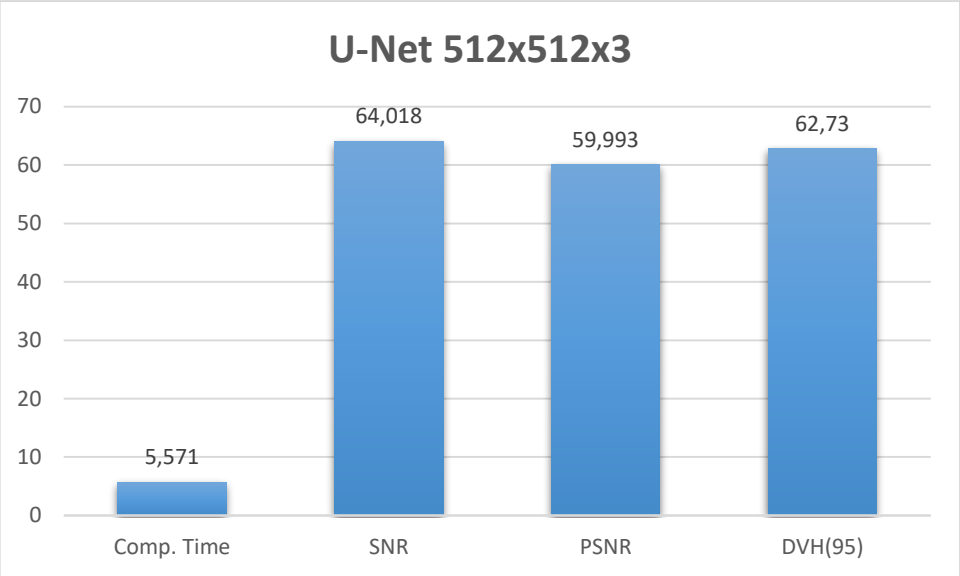


Graph 7. U-Net and DenseNet global comparison. We can observe, DenseNet requires more time and provides a worse PSNR and DVH_{95} than U-Net. DVH_{95} is worse because we want the closer one to the reference (62.45).

Graph 7 shows that DenseNet is more limited than U-Net. The time it takes to denoise the image is more than twice the DenseNet takes. However, DenseNet PSNR is 0.5 points under the U-Net one, and DVH_{95} is also worse.

We expected than DenseNet provided a little worse performance but reducing the computational time, but results have shown than it is worse than U-Net in all the aspects. Nevertheless, these limited results could be caused by the reduced complexity of the network, which is also caused by the lack of data, which would impede the overfitting issue. We considered DenseNet as faster because it has a much smaller number of parameters, but all the concatenations result in an increase of the memory requirement and, consequently, an increase of the spent on denoising the image.

Finally, we are going to discuss about the best network, U-Net fed with 512x512x3 slices per patch. We will analyze the *Mean Squared Error*, the *Signal-to-Noise ratio*, the *Peak-Signal-to-Noise ratio*, and the DVH_{95}.



Graph 8. U-Net 512x512x3 analysis. The first bar corresponds to the average of the computational time, the second bar corresponds to the average of the Signal-to-Noise ratio, the third one corresponds to the one of the Peak-Signal-to-Noise ratio and the last one to the DVH_{95}.

According to the results, this network is the best one. It achieved the higher SNR, which means that the noise has the lower level compared with the signal, in this case, the dose distribution. The PSNR achieved is also the highest one, that, instead of comparing the noise with the signal, compares it with the highest pixel value. The difference between the DVH_{95} from the reference and the one from the denoised image is 0.28 Grays, and this is not the lowest difference. *Figure 53* shows the DVH of the reference and the denoised image together.

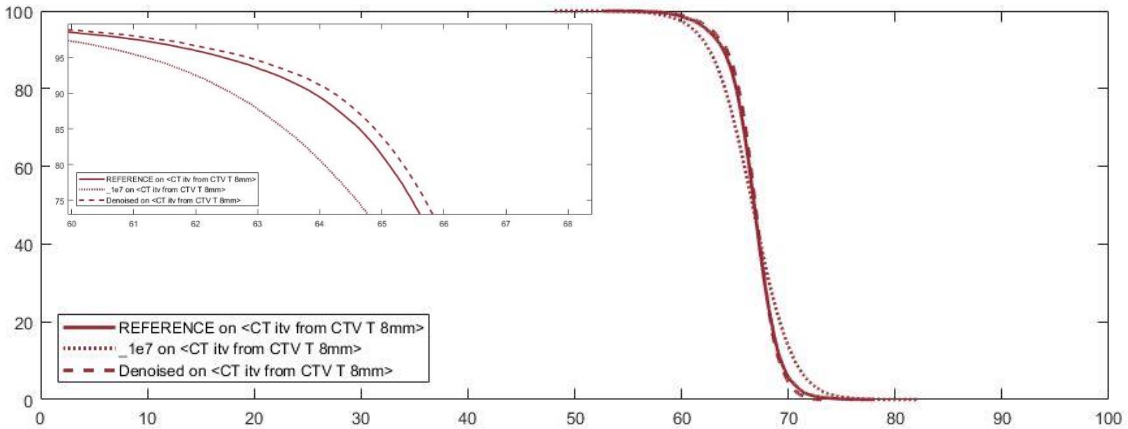
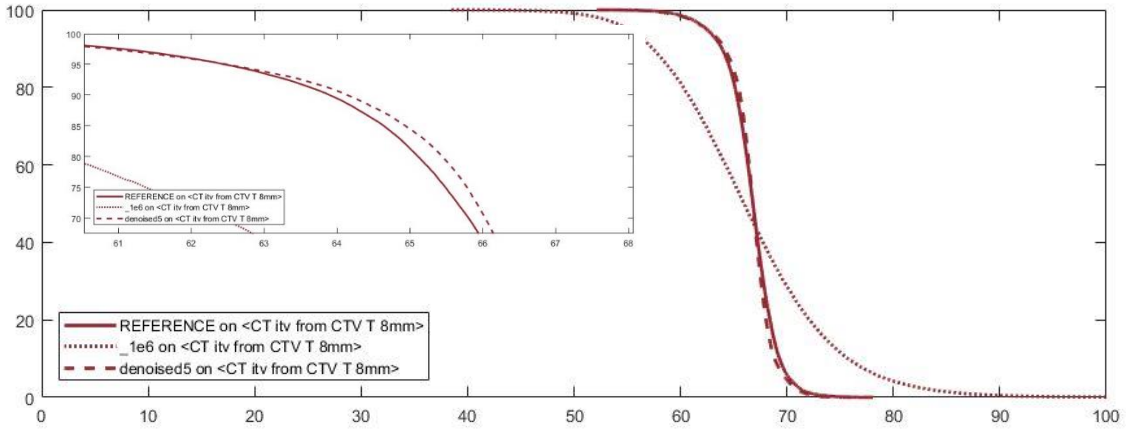


Figure 54. DVH comparison of the reference dose, the 1e7 dose and the denoised one. The DVH_{95} corresponds to the value of the histogram for the volume 95.

As we can see in this figure, the three curves are similar, but the reference and the denoised are almost the same curve. Nevertheless, the results show that the doses

Figure 53. DVH comparison of the reference dose, the 1e6 dose and the denoised one. The DVH_{95} corresponds to the value of the histogram for the volume 95.

generated with 1e6 particles and denoised are more similar than the ones generated with 1e7 particle in terms of DVH. The *Figure 54* shows that result.



This DVH shows that the denoised image coming from the Monte Carlo simulation, made with 1e6 particles is almost the same than the one generated with 1e7 particles and denoised. There is a huge difference between the one made with 1e6 and 1e7 particles before the denoising, so, maybe, considering this metric there is not difference between filter those two simulations for the clinical application.

The following figure shows the difference between a 1e6 and a 1e7 denoised image.

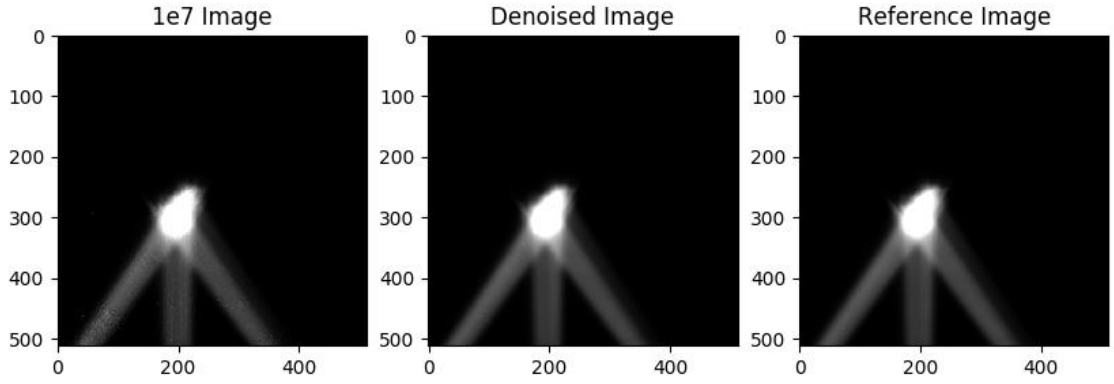


Figure 55. Comparison of the 1e7 simulation (a), the denoised 1e7 simulation (b) and the reference dose (c). These images correspond to the 50 slice of a lung tumor.

Figure 55 shows the image before denoising, the denoised one and the reference. Visually, images (b) and (c) are very similar, what is confirmed for the results of the metrics like DVH. Furthermore, we are going to compare the pixel intensities of the row 350.

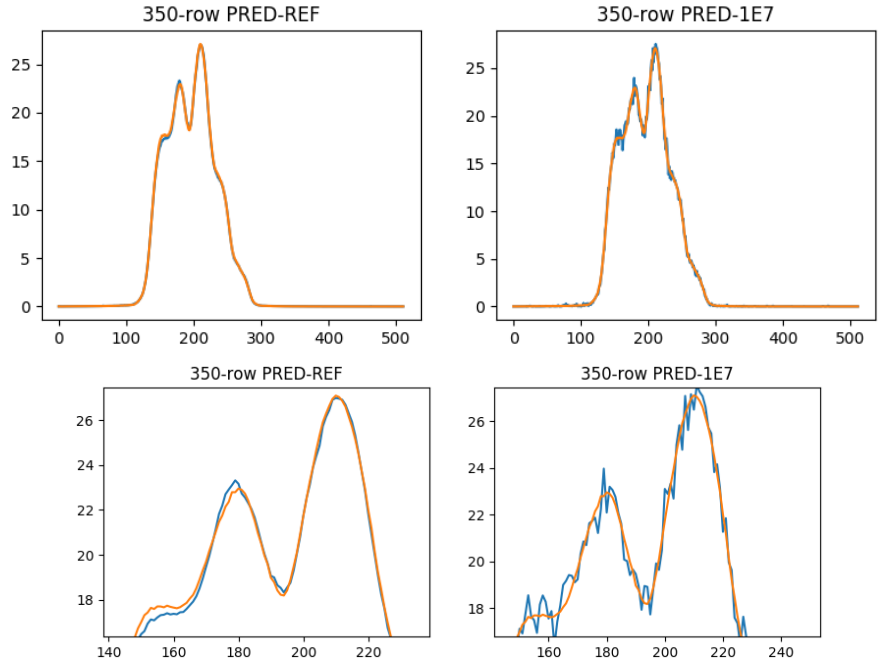


Figure 56. Comparison between the denoised image, the reference and the 1e7. Image (a) compares the 350 rom from the denoised distribution (orange) and the reference (blue). Image (c) is the first one aggrandized. Image (b) is the comparison between the denoised image and the 1e7. Image (d) is image (b) aggrandized.

Figure 56 shows the difference between the signal of the 350 row from the 1e7 distribution, the reference and the denoised image. We can see in these profiles that the noise is significantly reduced.

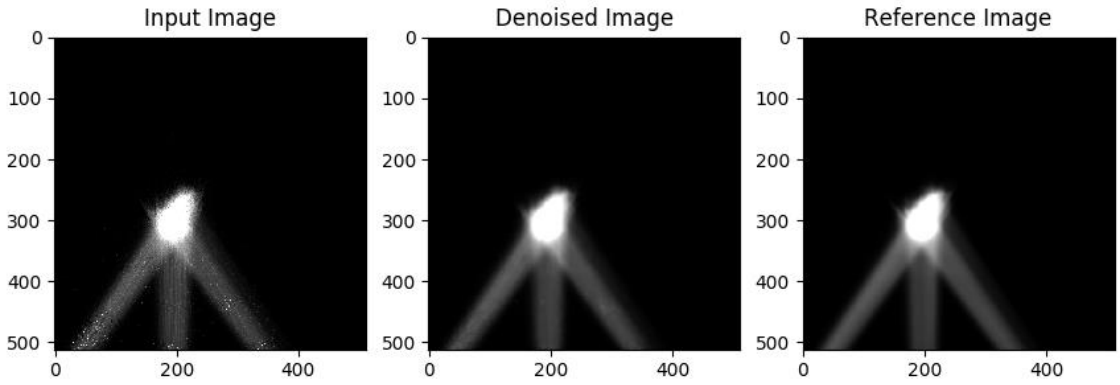


Figure 57. Comparison of the 1e6 simulation (a), the denoised 1e7 simulation (b) and the reference dose (c). These images correspond to the 50 slice of a lung tumor.

Figure 57 shows the image before denoising, the denoised one and the reference. There is a larger difference between this performance and the last one, due to the noise of the 1e6 simulations, which is higher here. We can see a small difference between the denoised and the reference image.

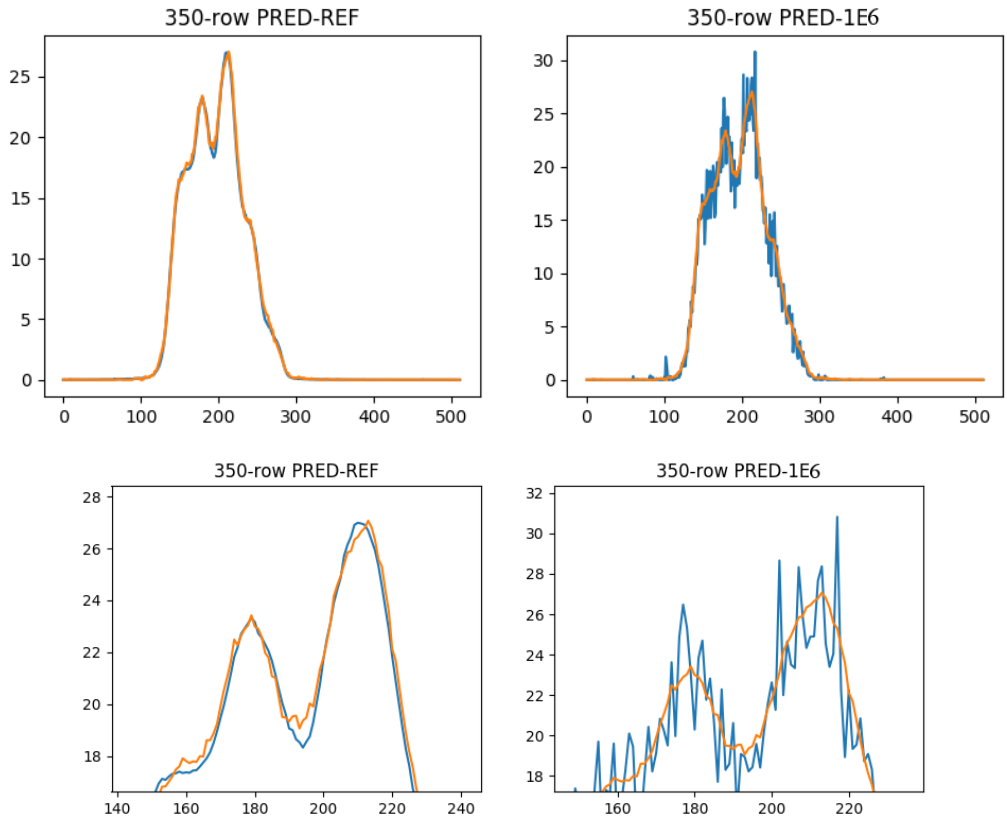


Figure 58. Comparison between the denoised image, the reference and the 1e6. Image (a) compares the 350 rom from the denoised distribution (orange) and the reference (blue). Image (c) is the first one aggrandized. Image (b) is the comparison between the denoised image and the 1e6. Image (d) is image (b) aggrandized.

As we can see in *Figure 58*, the noise of the original signal is higher than the one of the 1e7 particles simulation, and, the filtered image has some differences with the reference, but it is quite similar.

Conclusion

The current thesis has expose two different methods based on deep learning for denoising fast Monte Carlo dose distribution simulations, achieving distributions similar to the 1e9 particles ones. These two networks have largely improved the denoising state-of-the-art, and they are fast methods which takes no more than 14 seconds to filter a whole 3D image.

Nevertheless, these networks could be improved with several changes we are going to describe in the following lines, including training different networks, more complex ones or using different data.

Firstly, we discovered the great variability due to the tumor location. The anatomy implies different sizes according to the exactly place in the human body. Lung and liver tumors, for instance, are larger than brain ones, and the 3D image has more than 110 slices, implying an increase of the computing time. Additionally, larger tumors are treated with larger beams which are more complex to model and are noisier. On the other hand, neck or prostate tumors are smaller and less noisy, and the 3D images have around 90 slices, which implies a smaller computational time. We could build different organ-specialized networks, reducing the variability in the input dataset and improving the general performance, but the amount of data is a huge problem, because we need enough data of each tumor, which implies 5-10 time the data we currently have.

Secondly, there are more complex networks that could provide better performances, as Generative Adversarial Networks, where two different networks are trained, one is the denoiser, which takes the input and produces a free-noise image, and the other one must discover if the output of the first one is an original or a denoised image. Both networks are connected to ease information to the other network. At the end, the first network will produce almost free-noise images that the Adversial Network cannot recognize. Recently, researchers discovered Capsule Networks, which have shown a huge power, reducing the needed data. These networks are based on detecting the image objects, their location and their rotations, so we do not need to provide the network with different rotated images.

Thirdly, we can feed the networks with different information. We can work with the frequency domain instead of spatial domain, or even both of them. There are a lot of domains that could be used for improving the results. Several current denoisers use parameters like albedo to enhance the performance, making uncommon connections in their networks. All these parameters could perform together to find the best approximation of the noise model.

Fourthly, we could build very simple networks for making softs improvements. Assuming each output has a random noise, we can calculate the average of all those outputs for enhance the final image. Those networks must be simple and fast for having a large number of outputs to calculate a robust average.

Finally, we could create a special network, that, instead of denoising a Monte Carlo dose distribution, generates that distribution. This would be the hardest

improvement to do, but, as neural networks are applied on several issues, we could provide the system with all the required parameters (beam angles, beam energy, beam position, CT image, etc.). Those networks will achieve a good result in terms of computational time, but it would be hard to achieve a quality performance as the one obtained by Monte Carlo algorithms.

To conclude, there are a lot of techniques that could improve the images generated by a Monte Carlo algorithm. Reducing the computational time of dose distributions to a few seconds, joint to the automatized segmentation of tumors and organs, is going to increase the radiation therapy process.

References

[1] R. L. Siegel, K. D. Miller and A. Jemal. Cancer Statistics. January 2017.

[2] M. Malvezzi, G. Carioli, P. Bertuccio, P. Boffetta, F. Levi, C. La Vecchia and E. Negri. European cancer mortality predictions for the year 2017, with focus on lung cancer. May 2017.

[3] V. Verma, C. Shah, J. M. Rwigema, T. Solberg, X. Zhu,C and B.Simone II. Cost-comparativeness of proton versus photon therapy. February 2016.

[4] I. Kawrakow, M. Fipple and K. Friedrich. 3D electron dose calculation using a Voxel based Monte Carlo algorithm. April 1996.

[5] A. Krizhevsky, I. Sutskever and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. 2012.

[6] R. Symonds, C. Deehan, C. Meredith and J. Mills. Walter and Miller's Textbook of Radiotherapy: Radiation Physics, Therapy and Oncology, pages 311-313. May 2012.

[7] M. L'Annunziata. Radioactivity: Introduction and History, pages 55-58. July 2007.

[8] R. N. Kjellberg, T. Hanamura, K. R. Davis, S. L. Lyons, et al. Bragg-Peak Proton-Beam Therapy for Arteriovenous Malformations of the Brain. August 1983.

[9] O. Ronnenberger, P. Fischer and T. Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. May 2015.

[10] C. Geng, J. Daartz, K. Lam-Tin-Cheung, M. Bussiere, HA. Shih, H. Paganetti and J. Schuemann. Limitations of analytical dose calculations for small field proton radiosurgery. January 2017.

[11] K. Souris. Accurate assessment of proton therapy treatments: Fast Monte Carlo dose engine and extensive robustness tests, pages 22, 77 and 105.

[12] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. 1958.

[13] H. Ramchoun, M Amine, Y. Ghanou and M. Ettaouil. Multilayer Perceptron: Architecture Optimization and Training. 2016

[14] S. Sabour, N. Froost and G. E. Hinton. Dynamic Routing Between Capsules. November 2017.

[15] Y. le Cun. A Theoretical Framework for Back-Propagation. 1988.

[16] D. P. Kingma and J. Lei Ba. Adam: A Method for Stochastic Optimization. 2015.

[17] N. Srivastava, G. Hinton, A. Krizhevshy, I. Sutskever and R. Salakhutdinov. Dropout: A Simple Way to Preven Neural Networks from Overfitting. June 2014.

[18] S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. March 2015.

[19] S. Quo, X. Xu and B. Cai. FReLU: Flexible Rectified Linear Units for Improving Convolutional Neural Networks. January 2018.

[20] K. He, X. Zhang, S. Ren and J. Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. February 2015.

[21] J. Long, E. Shelhamer and T. Darrell. Fully Convolutional Networks for Semantic Segmentation. 2015.

[22] P. F. Christ, M. Ezzeldin, A. Elshaer, F. Ettlinger, S. Tatavarty, et al. Automatic Liver and Lesion Segmentation in CT Using Cascaded Fully Convolutional Neural Networks and 3D Conditional Random Fields. October 2016.

[23] G. E. Hinton and R. R. Salakhutdinov. Reducing the Dimensionality of Data with Neural Networks. July 2006.

[24] P. Baldi. Autoencoders, Unsupervised Learning and Deep Architectures. 2012.

[25] B. Kayahbay, G. Jensen and P. van der Smagt. CNN-based Segmentation of Medical Imaging Data. July 2017.

[26] G. Kaur, R. Choudhary and A. Vats. A Wavelet Approach for Medical Image Denoising. October 2017.

[27] H. M. Ali. MRI Medical Image Denoising by Fundamental Filters. 2017.

[28] J. Bai, S. Song, T. Fan and L. Jiao. Medical image denoising based on sparse dictionary learning and cluster ensemble. 2018.

[29] W. Jifara, F. Jiang, S. Rho, M. Cheng and S. Liu. Medical image denoising using convolutional neural network: a residual learning approach. 2017

[30] Q. Yang, P. Yan, Y. Zhang, H. Yu, Y. Shi, X. Mou et al. Low Dose CT Image Denoising Using Generative Adversarial Network with Wasserstein Distance and Perceptual Loss. April 2018.

[31] C. Dong, C. C. Loy, K. He, and X. Tang. Image Super-Resolution Using Deep Convolutional Networks. July 2015.

[32] G. Huang, Z. Liu, L. van der Maaten and K. Q. Weinberger. Densely Connected Convolutional Networks. January 2018.