

École polytechnique de Louvain

Denoising Monte Carlo doses for fast and accurate proton therapy treatment planning

Author: **Liliya SEMERIKOVA**

Supervisor: **John A. LEE**

Readers: **Gregory BUTI, Kevin SOURIS, Edmond STERPIN**

Academic year 2020–2021

Master [120] in Computer Science and Engineering

Abstract

Proton therapy is one of the main radiation therapy treatments nowadays. Using the spread out Bragg peak property, we can irradiate a tumor while saving healthy tissue around it. In order to design a radiation therapy plan, we need to simulate the dose distribution on a CT patient image. The algorithm that performs this task is based on the Monte Carlo method. However, Monte Carlo simulation suffers from statistical noise due to its stochastic nature. This noise can be reduced by simulating more protons. Unfortunately, there is a trade-off between noise reduction and duration of the simulation. Decreasing the noise significantly increases the computing time. In this thesis, we propose a solution to reduce the calculation time and maintain a low amount of statistical noise. In other words, we propose to simulate fewer protons and introduce a correction term to the optimisation algorithm.

Acknowledgements

I am glad to have had the opportunity to write my Master's thesis on such an important topic with such a nice team. It was a great adventure but it was not always easy and obstacle-free. Therefore, I would like to thank all people who were engaged in this process.

I want to sincerely thank the team that was there from the very first day and that guided me and helped me on this journey. Specifically, my promotor John Lee and supervisors Gregory Buti and Kevin Souris. Thank you for inviting me to your in-house seminars; they truly made me feel like part of the research group. I want to thank my promotor John Lee for all judicious and wise advice he provided during this year; Kevin, for helping me with all the installation and code problems; and Gregory, for taking time whenever I needed help and for giving me great advice.

I would also like to thank my friend Gilles Peiffer for always being there to discuss my progress, and for his advice and moral support.

Finally, this achievement would not be possible without the support of my family. Many thanks to my mom and my grandma who were always there to believe in me and encourage me in my endeavors.

Contents

1	Introduction	6
2	Context	9
2.1	Treatment planning process	9
2.1.1	Image generation	9
2.1.2	Tumor and OAR contouring	11
2.1.3	Plan generation	11
2.1.4	Plan evaluation	13
3	Methods	15
3.1	Optimisation Algorithm	15
3.2	Monte Carlo	17
3.2.1	Particle tracking	17
3.2.2	Proton physics	19
3.2.3	Uncertainties due to physics model	20
3.3	Deep Neural Networks	20
3.3.1	Feed-forward neural networks	22
3.3.2	Training	24
3.3.3	Back-propagation	24
3.3.4	Underfitting and Overfitting	25
3.3.5	Convolutional Neural Networks	26
4	Experimental setup	28
4.1	MCsquare	28
4.1.1	Input	28
4.1.2	Transport algorithm	29
4.2	Artificial Network simpleNet	30
4.2.1	Network training	32
4.2.2	Changes applied	32
4.2.3	Discarded layers	32
4.2.4	Results	34

5	Results	36
5.1	Dose simulation	36
5.2	Monte Carlo denoising	37
5.3	sNet denoising	40
5.4	Comparison	46
6	Conclusion	48

List of Figures

1.1	Energy released by proton versus conventional therapy	6
2.1	General planning process	10
2.2	X-ray beams for image creation	10
2.3	Thermoplastic mask to immobilize the patient's head during external beam radiotherapy	11
2.4	Schema of contouring of a treated volume and organ at risk	12
2.5	Dose-Volume histogram for a prostate	13
3.1	Time to solve the problem using Monte Carlo versus a deterministic analytic approach	18
3.2	The Monte Carlo method can be compared to the rolling of a die	19
3.3	Machine Learning is a subfield of Artificial Intelligence. Deep Learn- ing is a particularly complex part of Machine Learning	20
3.4	Representation of a Biological Neuron compared to the functioning of Neural Nets	21
3.5	Deep learning to automatically produce captions that accurately describe images	22
3.6	Deep Neural Network in a Multi-Layer Perceptron Layout	23
3.7	Example of under- and overfitting	25
3.8	Two fully connected layers	26
3.9	Two convolutional layers	27
4.1	Flowchart representing the main steps of the algorithm implemented in MCsquare	29
4.2	SimpleNet architecture	30
4.3	Illustration of the dilation layer	31
4.4	DVH of the target and OAR by applying sNet_50 on the same image but discarding different numbers of layers	33
4.5	Denoising with sNet_24 and sNet_50 respectively	34
4.6	DVH comparison graph between sNet_24 and sNet_50 models	35

5.1	Comparison of the results with and without applying the correction term using full Monte Carlo simulation by using 5×10^2 protons per beamlet	39
5.2	Comparison of the results with and without applying the correction term using full Monte Carlo simulation by using 1×10^3 protons per beamlet	39
5.3	Comparison results with and without applying the correction term using full Monte Carlo simulation by using 5×10^3 protons per beamlet	40
5.4	Comparison of results obtained with and without correction term using the sNet_24 architecture when using 5×10^2 protons per beamlet	42
5.5	Comparison of results obtained with and without correction term using the sNet_24 architecture when using 1×10^3 protons per beamlet	42
5.6	Comparison of results obtained with and without correction term using the sNet_24 architecture when using 5×10^3 protons per beamlet	43
5.7	Comparison of results obtained with and without correction term using the sNet_50 architecture when using 5×10^2 protons per beamlet	44
5.8	Comparison of results obtained with and without correction term using the sNet_50 architecture when using 1×10^3 protons per beamlet	45
5.9	Comparison of results obtained with and without correction term using the sNet_50 architecture when using 5×10^3 protons per beamlet	45

List of Tables

4.1	Influence of number of discarded layers on results	33
4.2	Overview of the effect of applying sNet denoising on target D_{95} and D_5 , as well as OAR D_{50}	35
5.1	Ground truth values computed by simulating 1×10^9 protons per beamlet without applying any correction	36
5.2	Comparison of the results by simulating different numbers of protons per beamlet without any denoising applied	37
5.3	Comparison of the results obtained by simulating different numbers of protons per beamlet when applying a denoising process using full Monte Carlo simulation	38
5.4	Comparison of the results obtained by simulating different numbers of protons per beamlet when applying a denoising process using the sNet_24 architecture	41
5.5	Comparison of the results obtained by simulating different numbers of protons per beamlet when applying a denoising process using the sNet_50 architecture	43
5.6	Computation time difference between sNet_24 and sNet_50	46
5.7	Difference in performance between Deep Learning and Monte Carlo	47
5.8	Computation time difference between Deep Learning and Monte Carlo	47

1 | Introduction

According to the World Health Organisation, cancer is currently the leading cause of death worldwide [1]. The main treatment methods are radiotherapy, chemotherapy, and surgery. This thesis mainly considers radiation therapy.

Radiation therapy consists in transferring energy to a tumor; one can express the absorbed dose in energy (J) absorbed per unit of mass (kg). This quantity has units measured in Gy. The goal of radiation therapy is to destroy cancer cells, and have as little impact as possible on nearby healthy tissues and vital organs.

There are three main types of radiation therapy: with photons, with electrons, and with protons. Nowadays, photon therapy, usually known as “conventional radiotherapy,” is the most used type. However, the low dose bath and high entrance dose make irradiation of the surrounding tissues unavoidable and that is why recent research has looked at the novel method of proton therapy. Indeed, the main advantage of protons is their energy release process, which allows for a more targeted form of therapy.

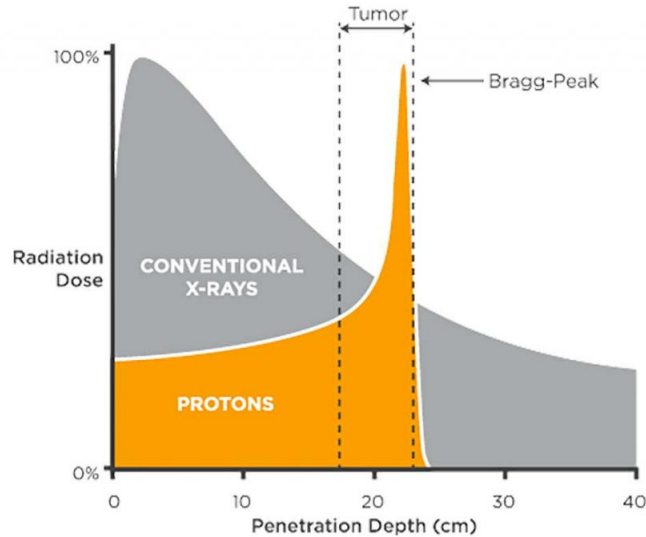


Figure 1.1: Energy released by proton versus conventional therapy. Figure from [2].

At the “Bragg peak,” which coincides with the end-of-range of the protons, a maximal amount of radiation is released in the tumor. Its peakedness allows us to maximally impact tumor cells, while avoiding nearby healthy tissues and vital organs. We can observe the difference between photon and proton energy release on figure 1.1. Compared to photon therapy, proton therapy has the potential to reduce the risk of side effects during and after treatment.

To evaluate the clinical optimality of the treatment plan, we need to compute a dose distribution. The optimal dose distribution is such that the tumor is fully covered and receives the prescribed dose, whereas the surrounding organs must be affected as little as possible. To compute the optimal dose for proton therapy, there are two main methods: analytical algorithms and the Monte Carlo method. Most clinical treatment planning systems use analytical algorithms. However, these methods can fail to predict proton ranges accurately in very heterogeneous geometries where the densities are small, such as in the lungs [3]. While the Monte Carlo method is the best way to achieve an accurate dose in such scenarios, its inherently computationally demanding process inhibits its clinical adoption. For intensity-modulated proton therapy (IMPT), a 3D map of the dose distribution for each beamlet needs to be computed. The computation time depends on the number of protons simulated when the beamlet dose is computed, where beamlets are the small beams that deliver the dose to the patient.

It is possible to reduce the number of protons simulated per beamlet and thus decrease the computation time, but this will result in a noisy image that can obviously not be used in clinical practice. The goal of this thesis is to reduce the number of protons needed in Monte Carlo simulations, while maintaining dose distribution quality.

The model must be fast and accurate:

- Fast, because it is a bottleneck that slows the treatment process. First of all, we need to care about the comfort of the patient, who needs to have the treatment at the right time. Additionally, if the time increases, the treatment cost would rise too. Expensive and time-consuming treatment leads to a lower number of patients that one could treat.
- The dose distribution must be accurate and respect all the prescribed limits and conditions, e.g. it must reach the minimum required dose for the target while not exceeding the maximum dose for the organs at risk (OAR).

In this thesis, two methods are proposed in order to compute a correction term. The correction term will be integrated into the optimisation algorithm, whose goal is to optimise the distribution dose. Thus, one could decrease the number of protons per beamlet simulated in order to compute the initial dose. However, the correction term provides us with a high quality dose distribution. Two methods

that were used to compute the correction term are detailed: a full Monte Carlo simulation and Artificial Neural Nets. The two models were tested and compared on images of patients suffering from prostate cancer.

The document is divided as follows: chapter 2 explains the details of the treatment planning process in proton therapy. In chapter 3, background knowledge about the two methods we use in order to compute the correction term (Monte Carlo and Artificial Neural Nets) is presented. In chapter 4, we define the Monte Carlo algorithm and architecture of the Neural Net we used. Finally, the document will present and discuss the results obtained in chapter 5; we then finish with a conclusion in chapter 6.

2 | Context

In this section, we present the general context of the thesis. We start by presenting a general treatment plan process, then continue by detailing each step of the treatment process and discussing the bottlenecks. We finish by presenting the Dose-Volume Histogram, which is the main method used to compare the results obtained in this thesis.

2.1 Treatment planning process

A general treatment planning process has four main steps as we can observe on figure 2.1. The process starts by producing a CT image of the patient, focusing on the tumor and nearby tissues and organs. After obtaining a 3D image, the physician needs to contour the tumor, the organs at risk, and define the dose prescription. Once all the parameters are set, the plan generation starts. This step consists in defining machine parameters in order to obtain a dose distribution that satisfies the prescription. Afterwards, one can finally start the treatment process.

2.1.1 Image generation

The image is generated by using a Computed Tomography (CT) scan that allows the reconstruction of the 3-D image by assembling 2-D images produced by X-ray as we can observe on figure 2.2.

The intensity of the gray colour on the image is correlated with the density of the tissue. For example, bones tend to be whiter in colour, as they are a high-density material absorbing a lot of radiation. In fact, the intensity of each voxel is proportional to the attenuation coefficient of the tissue. We can quantify the intensity by using the Hounsfield unit (HU). For a voxel with attenuation coefficient μ , the corresponding intensity is equal to:

$$\text{intensity} = \frac{\mu - \mu_{\text{water}}}{\mu_{\text{water}} - \mu_{\text{air}}},$$

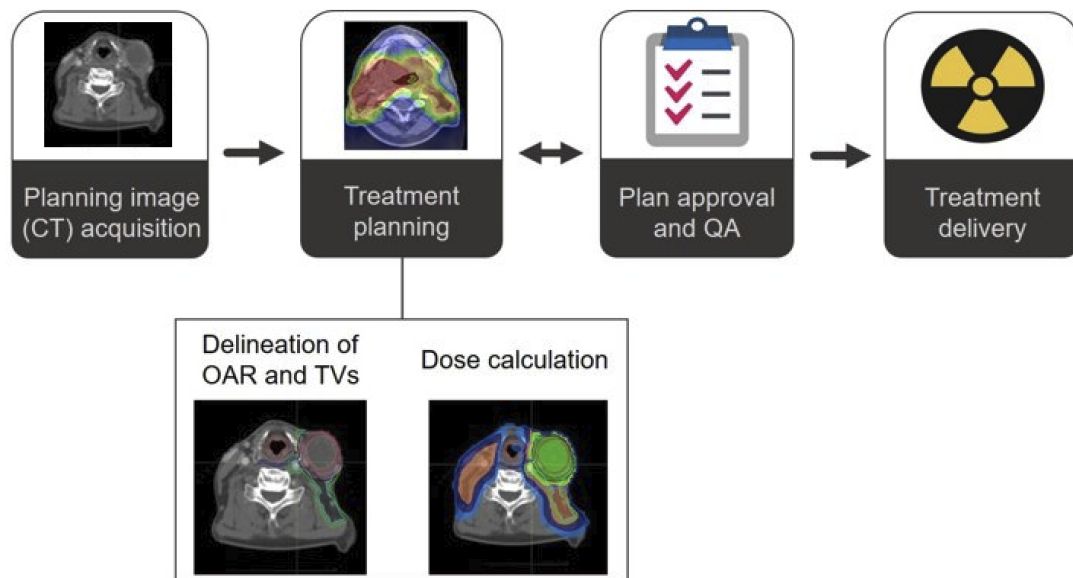


Figure 2.1: A general planning process consisting of four steps: production of a CT image, treatment plan generation, plan approval and treatment delivery. We can divide treatment plan generation into two sub-steps: contouring of the target/OAR and dose calculation.

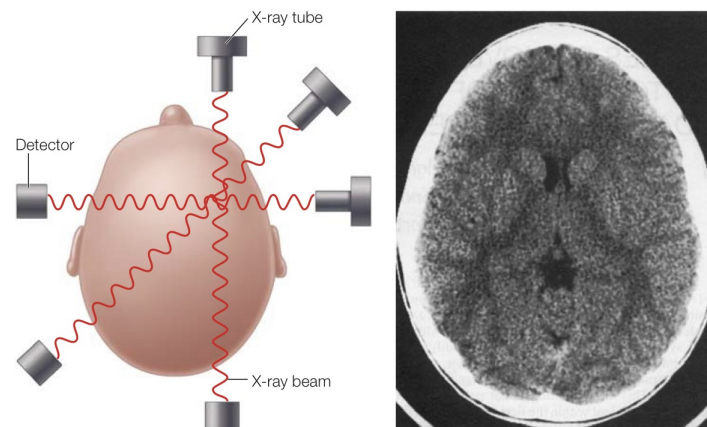


Figure 2.2: X-ray beams passing through the head to create a two-dimensional image [4]

where μ_{water} and μ_{air} represent the attenuation coefficients of water at standard temperature (equal to 0 HU) and of air (equal to -1000 HU), respectively.

The accuracy of the image is quite important for the treatment plan. Small movements or even breathing will introduce uncertainty in the image quality. To avoid this, devices, such as the thermoplastic mask shown on figure 2.3 are used to position the patient.

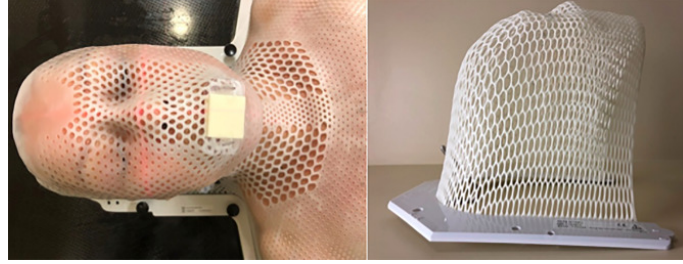


Figure 2.3: A thermoplastic mask to immobilize the patient's head during external beam radiotherapy [5].

2.1.2 Tumor and OAR contouring

After the image generation, the physician needs to manually set the contours of the tumor and organs at risk. There are a few essential types of contour: GTV, CTV and PTV. The gross tumor volume (GTV) contour is based on the visible contours of the tumor. However, one needs to take into account the cancer cells that might be found in the tissues surrounding the tumor. Thus, the clinical target volume (CTV) contour includes several millimeters behind the tumor, as well as the GTV contour. This is done based on the location of the tumor and any suspicion of cancer extension. Finally, the planning target volume (PTV) contour includes both the GTV and CTV, but also takes a margin due to setup errors, patient movements and other possible geometric variations. However, the PTV is mostly used with photon treatment. During proton treatment, we prefer to simulate the uncertainties due to e.g. setup errors, patient motion etc. directly during the optimization process. We call this “robust optimization.”

2.1.3 Plan generation

In this work we are interested in the treatment technique called *intensity-modulated proton therapy* (IMPT). For IMPT, a treatment plan is composed of the weights (intensity), position and energy of each beamlet. The goal is to optimize the beamlet intensities to obtain an optimal dose distribution. Plan generation consists of the following steps:

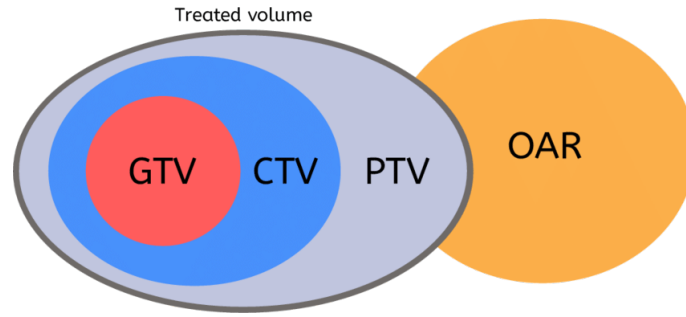


Figure 2.4: Schema of contouring of a treated volume and organ at risk [6].

1. define the treatment parameters (e.g. beams, treatment objectives etc.)
2. initialize the beamlet dose distribution;
3. optimize it; and
4. compute the final dose.

In order to find the best beam configuration, different numbers of beams, angles and intensities need to be tested on a dose engine. The final parameters of the machines depend on these results, and the simulator must be precise for this reason. The precision of the simulator depends on two things: the scanning image and the estimator algorithm. The patient planning CT image must be of good quality, with all regions (GTV, CTV, and PTV) properly selected. The plan optimisation algorithm is robust if it provides us with an accurate dose distribution based on the parameters that were set and the CT image with contours.

Nowadays, we have two main algorithms that one could use in order to compute the dose distribution: using a Monte Carlo algorithm or using analytic methods. As was said before, the analytic method is quite fast but in some situations we could observe a lack of precision due to the difficulty of appropriately modeling heterogeneous materials. Monte Carlo algorithms can provide us with an accurate simulation of the dose distribution, provided a sufficient number of particles are simulated. However, this takes too much time, preventing this method from being applied in clinical environments. One of the strategies to combine the efficiency of the analytical approach and the accuracy of the Monte Carlo simulation was discussed in [7]. Indeed, they investigated a hybrid optimisation strategy to achieve the best trade-off between these two approaches.

In the treatment planning process, the dose is evaluated as the sum of beamlet doses. The beamlet doses have to be pre-computed before the optimization process. The time bottleneck of the Monte Carlo simulation is proportional to the number of protons per beamlet that are used to pre-compute the dose distribution. In

order to compute the dose distribution of a single beamlet with sufficient accuracy (at most 1 % uncertainty), typically 10^6 protons must be simulated. Given that a typical therapy treatment plan is composed of more than 10^3 beamlets, one needs to simulate more than 10^9 individual protons. We can simulate fewer protons per beamlet, but the quality of the dose distribution would then not be sufficient for the treatment plan. The quality here is represented by the noise level of the dose distribution. Thus, the goal of this project is to simulate fewer protons per beamlet, thereby reducing the computation time of the beamlet dose calculation, but to introduce a correction term during the dose optimisation process that will allow us to maintain the required quality.

We adapt the similar approach that was proposed in [7], but instead of correcting the dose provided by an analytical algorithm, we propose two strategies to correct a noisy Monte Carlo dose: using noiseless Monte Carlo and using a deep learning model.

2.1.4 Plan evaluation

In order to judge the quality of the plan, a Dose-Volume Histogram is typically used (DVH). A DVH represents the dose in a given volume. On figure 2.5, one can observe a typical DVH curve, where the x -axis represents dose quantity and the y -axis gives the percentage of volume.

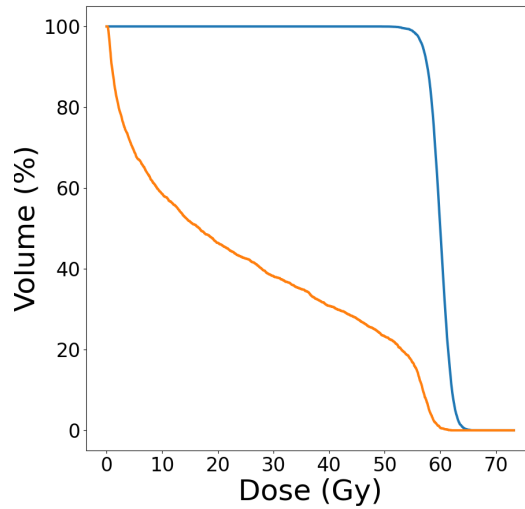


Figure 2.5: Dose-Volume histogram for a prostate, with treatment aiming to hit a dose of 60 Gy in the tumor. The curve in blue is a DVH of the target; the curve in orange is a DVH of the organs at risk in orange.

In particular, we can abstract some values from the histogram, that we will call D_X and which will represent the dose delivered to X % of the volume. Thus, D_{95} represents the minimum dose delivered to 95 % of the tumor volume. Similarly, a D_{mean} dose could be computed that represents the mean dose received by the tumor or by the organs at risk.

3 | Methods

As was said previously, the goal of this thesis is to be able to simulate fewer protons for the pre-computed beamlets while maintaining the quality of the optimized dose distribution. In order to do this, one can introduce a correction term in the optimisation algorithm. Our idea is to denoise the dose map several times during the optimisation process. In order to accomplish this, two methods were used: full Monte Carlo simulation and deep learning for the denoising process.

First, we will explain the theory behind the optimisation algorithm which was the main focus of improvement during this thesis. Then, we will present the basic principles of the two methods used in order to compute the correction term: Monte Carlo simulation and Neural Nets. We will detail the theory behind the Monte Carlo method, its particle tracking, the physics of protons and the uncertainties of the model. We finish with the theory about the deep learning method including feed-forward neural nets, training and its problems (gradient computation and under-/over-fitting) and conclude by briefly discussing convolutional nets.

3.1 Optimisation Algorithm

Once the minimum and maximum dose constraints are set for the tumor and OAR, we can define an optimisation problem. The optimisation problem in this case consists in finding the appropriate weight for each beamlet in order to satisfy the prescribed dose.

In the beginning all weights are initialized to 1. Thus, the objective function for this optimisation problem is a weighted sum of quadratic terms that penalize the voxels with a dose d_i , where i is the voxel index, which differs from the prescription ξ_r . The dose d_i is obtained as the weighted sum of the beamlets:

$$d_i = \sum_j (P_{i,j} \times w_j),$$

with $P \in \mathbb{R}^{n \times m}$ represents the dose influence matrix and j the spot index. From this, we can define the gradient as a vector containing the first partial derivatives

with respect to $w \in \mathbb{R}^m$, which is a vector containing the m beamlet weights. The gradient will help us minimize the objective function since it allows us to find the direction of fastest decrease of the latter. Therefore, we can define the objective function as in [8]:

$$f(\mathbf{w}) = \sum_{r \in \text{T}} \frac{a_r}{N_r} \sum_{i=1}^{N_r} \left(\max\{0, \xi_r - d_i(\mathbf{w})\} \right)^2 + \sum_{r \in \text{T, OAR}} \frac{a_r}{N_r} \sum_{i=1}^{N_r} \left(\max\{0, d_i(\mathbf{w}) - \xi_r\} \right)^2,$$

where r represents the region of interest that could be part of “T” for “target” and “OAR” for “organs at risk.” For each region of interest, the penalty function is multiplied by a weighting factor a_r that accounts for its relative importance with respect to the other penalties, and divided by the total number of voxels, N_r .

In order to attain our goal, i.e. simulating fewer protons and having the same quality for the result, we will introduce a correction term in the algorithm.

We write $\tilde{d}$ to denote a dose obtained as a weighted sum of the pre-computed beamlets. Let $\bar{d}$ be the dose computed either by using a full Monte Carlo simulation on the fly or by denoising the noisy 3-D dose with the deep learning model. The correction term can be expressed as a difference between $\tilde{d}$ and $\bar{d}$. Hence, the corrected dose that would be used during the optimisation process can be expressed as follows:

$$\begin{aligned} c &= \bar{d} - \tilde{d}, \\ d^c &= \tilde{d} + c, \end{aligned}$$

where d^c is a corrected dose. Knowing that the correction term is computed every x iterations, one could write the pseudocode of algorithm 1:

Algorithm 1: Optimisation Algorithm

```

Initialize  $c = 0$ ;
 $i = 0$ ;
 $x = 20$ ;
while  $\text{diff}_{\text{obj}}$  is not small enough and  $i < i_{\text{max}}$  do
    Compute  $\tilde{d}$  based on the current weights;
    if  $i \% x == 0$  and  $x \neq 0$  then
        Compute  $\bar{d}$  by using a full Monte Carlo or deep learning model;
         $c = \bar{d} - \tilde{d}$ ;
    end
     $d^c = \tilde{d} + c$ ;
     $i = i + 1$ ;
end

```

In this algorithm, diff_{obj} is the relative variation of the objective function between two iterations, i_{max} is the maximum number of iterations allowed and x is the precise iteration when we compute the correction term. In order to compute the correction term, we have explored two methods: Monte Carlo and Deep Learning. These methods and their results are presented below.

3.2 Monte Carlo

The Monte Carlo method is based on the idea of repeating a stochastic process so many times that the average result starts to converge to the expected value with a sufficiently small statistical variance. A key feature of the Monte Carlo method is the number of times that we repeat the process: the accuracy of the result depends on the number of times that one simulates the event. The desired accuracy thus affects the calculation time [9].

As was discussed previously, the Monte Carlo method is much slower than the analytic approach. This is due to the fact that we need to simulate the process exhaustively to decrease statistical variance. On the other hand, for the analytic approach, we need to compute a set of operations only once. However, the computation time increases with the complexity of the model that we need to construct using the analytic approach. The Monte Carlo method is not affected by the macroscopic complexity of the problem; this is why deterministic analytical methods can fail to predict proton ranges accurately in very heterogeneous geometries where the densities are small, such as in the lungs [3]. Actually taking into account the micro complexities while using analytical algorithms would be even more time consuming than using a Monte Carlo approach, as can be observed on figure 3.1.

3.2.1 Particle tracking

The Monte Carlo method is also the most well-known method to simulate particle interactions within a medium. Actually, Monte Carlo is well adapted to simulate particle interactions because at this scale (quantum) physics is a non-deterministic process. Particle movement is divided into several steps, and at each step, the movement of the particle based on the statistical probability of the physical event is determined. The tracking of one particle until its escape from the zone of interest is called a history. A simulation of multiple histories is called a run [9].

To simulate the history of the particles, we start by sampling a number of events from a starting source distribution. In our case, this would be a flux from an accelerator. The next step is to simulate the passage of particles through a geometry, one by one. Step by step, taking the statistical probability and physical

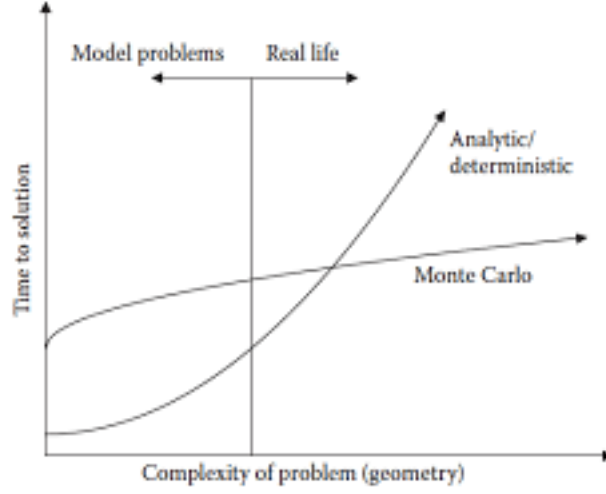


Figure 3.1: Time to solve the problem using Monte Carlo versus a deterministic analytic approach [9].

laws into account, particle interaction will be simulated. The geometrical model in our case is a patient geometry [9].

At each step, the distance to the next interaction is randomly sampled according to interaction probabilities (cross-section). It has to be determined in 3-dimensional space by sampling direction cosines. The maximum allowed step is a parameter that needs to be chosen by the user. The step size must be small to ensure a small difference of the cross sections at the beginning and the end of the step. However, bigger step sizes could decrease the computation time of the Monte Carlo simulation.

The type of the interaction is randomly picked based on their probability. Electromagnetic and nuclear interactions thus induce a change of the particle direction and/or energy. There could also be emission of secondary particles. Those types of particles are stored in memory. However, not every secondary particle is taken into account. For example, we do not need to store particles with low energy, as their range would be negligible. This is actually the very reason why Monte Carlo is more precise than the analytical approach: the former takes into account the interactions on a particle-by-particle basis.

The simulation ends if the particle energy reaches the low threshold set by the user or if it leaves the region of interest. The process would be repeated until the desired statistical accuracy is reached.

3.2.2 Proton physics

As we try to track a particle through a medium, some physical interactions could occur as shown on figure 3.2. If the particle is uncharged, the finite number of all physical interaction could be simulated. It is not the case for protons, which are charged particles. In fact, it would be impossible to simulate all the possible outcomes as charged particles interact very frequently. For that reason, we would just condense many discrete events in larger simulation steps [10]. Therefore, all the small deflections can be grouped into one scattering event.

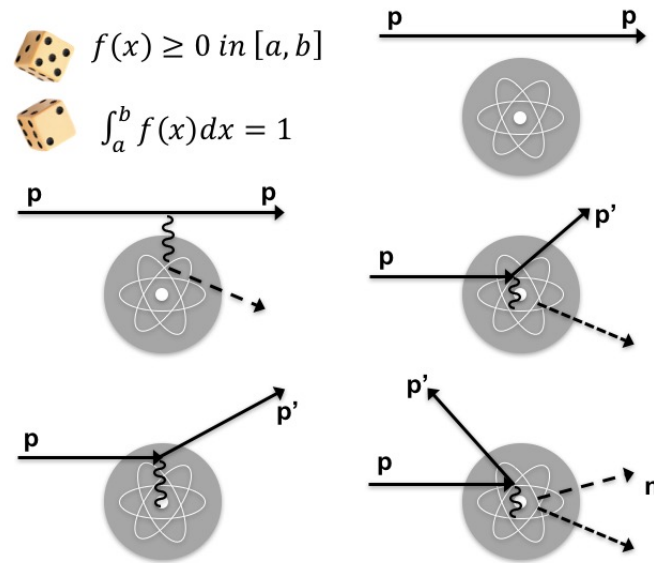


Figure 3.2: The Monte Carlo method can be compared to the rolling of a die. This would determine which of all the possible interactions might happen for a proton with matter. Each of these interactions has finite probability. The sum of the probabilities of all the possible interactions is equal to 1 [9].

Generally, we identify two different condensed history techniques [9]:

- Class I: we group all the interactions of a particular type in one condensed history step. In this case, we use a predetermined set of path lengths where the simplest choice is to take a constant path length.
- Class II: we will group only minor collisions but we consider the individual sampling of major events. This type of major interaction is called catastrophic and needs to be treated individually. In this case, we will define a maximum step size.

In general, class II is more accurate than I. Therefore, most Monte Carlo simulations use a combination of combined history and discrete processes where we consider each interaction. In proton therapy, discrete processes are nuclear interactions, secondary particle production and large angle Coulomb scattering.

3.2.3 Uncertainties due to physics model

The Monte Carlo method is sensitive to the accuracy of the parameters and physical assumptions. Thus, different Monte Carlo models could produce different results. For example, uncertainty in stopping powers can cause the computation to be different. The mean excitation energy might have uncertainties on the order of 5% to 15%. Moreover, the differences in mean excitation energies for given elements can exceed 10%. Also, the uncertainty on nuclear cross sections is about 10%. If we take this into account, it can result in up to 1 mm uncertainty in the predicted beam range in water [9]. Therefore, a lot of experiments must be done to ensure the accuracy of the Monte Carlo method.

3.3 Deep Neural Networks

Deep learning is a subset of machine learning, which is itself a form of artificial intelligence 3.3.

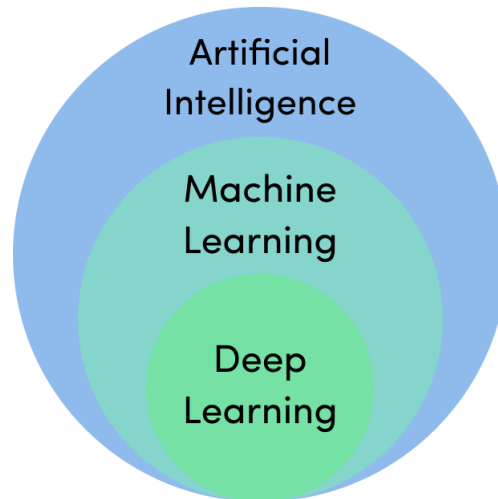


Figure 3.3: Machine Learning is a subfield of Artificial Intelligence. Deep Learning is a particularly complex part of Machine Learning [11].

In the beginning of artificial intelligence research, researchers tried to model artificial neurons based on biological neurons. The idea was to compare the input of the neural nets to the input signal of dendrites. Indeed, the signals of dendrites are combined and they go through something very similar to the **ReLU** activation function that we use in deep learning, as can be observed on figure 3.4.

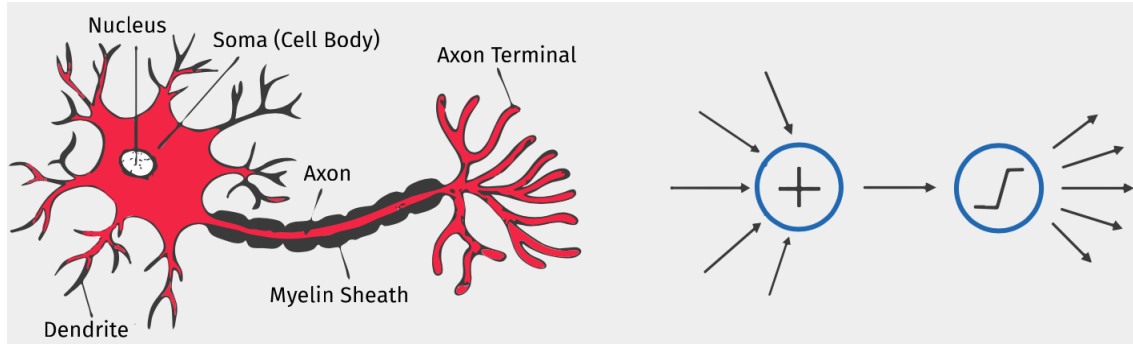


Figure 3.4: Representation of a Biological Neuron compared to the functioning of Neural Nets [12].

However, even if deep networks are a reasonable approximation of the functioning of biological neurons, at the end of the day, they are only a graph of differentiable operations that allows one to define a whole range of functions.

There are three main types of environments in machine learning:

- Supervised learning is the most used technique nowadays. In order to illustrate it, we can imagine collecting some features for n observations (e.g. images). For each observation vector $\mathbf{x}_i$, we also know the outcome (e.g. y_i is equal to 1 if it is a dog, 0 if it is a cat). For this data set, we would like to estimate the weights ($\mathbf{w}$) in order to construct a predictive model $f(\mathbf{x}_i, \mathbf{w}) = y_i$.
- Unsupervised learning is based on collecting n observations and trying to learn something from this data set. Here, we do not give any supervision labels, so the model does not know the “right” answer.
- Reinforcement learning differs from others in that it can interact with its environment to get the maximum reward. The most common example of such an environment is that of playing games. We would let the trained model play the game several times in order to improve its result.

Deep learning performs pretty well in all these types of learning. For example, Google research scientists developed a machine learning system that can automatically produce captions to accurately describe images the first time it sees them.

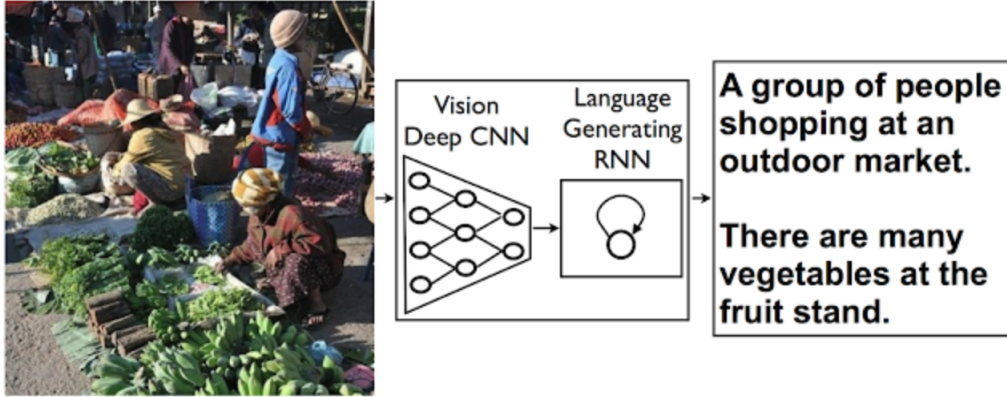


Figure 3.5: The model combines a vision CNN with a language-generating RNN so it can take in an image and generate a fitting natural language caption [12].

The system is quite complicated, as it combines computer vision methods with language processing, as shown in figure 3.5.

Moreover, on the medical image-understanding front, recent advances in deep learning have made a big leap to help identify, classify, and quantify patterns in medical images.

3.3.1 Feed-forward neural networks

In this particular work, we used only feed-forward neural network models. Feed-forward means that we can represent the deep learning model as a directed acyclic graph as represented in figure 3.6.

On the figure above we can observe various types of neural units, such as input units, output units, and intermediate units produced by the activation function. The neural network takes as input a linear combination of the input units and applies a nonlinear activation function to it. The activation function simulates an activation threshold like we can observe with biological neurons. There exist various activation functions used in practice. One of the most used ones is the **ReLU** function, which we will use in our models. The **ReLU** function and its derivatives are preferred because half of the range has a nonzero gradient which helps to slightly avoid the problem of vanishing gradients that we will discuss later.

Neural networks are typically organised into layers as we can observe on figure 3.6. One layer represents a function of the preceding layers. We can observe that every node of one layer is connected to all the nodes of the preceding layer. This type of network is called **fully connected**.

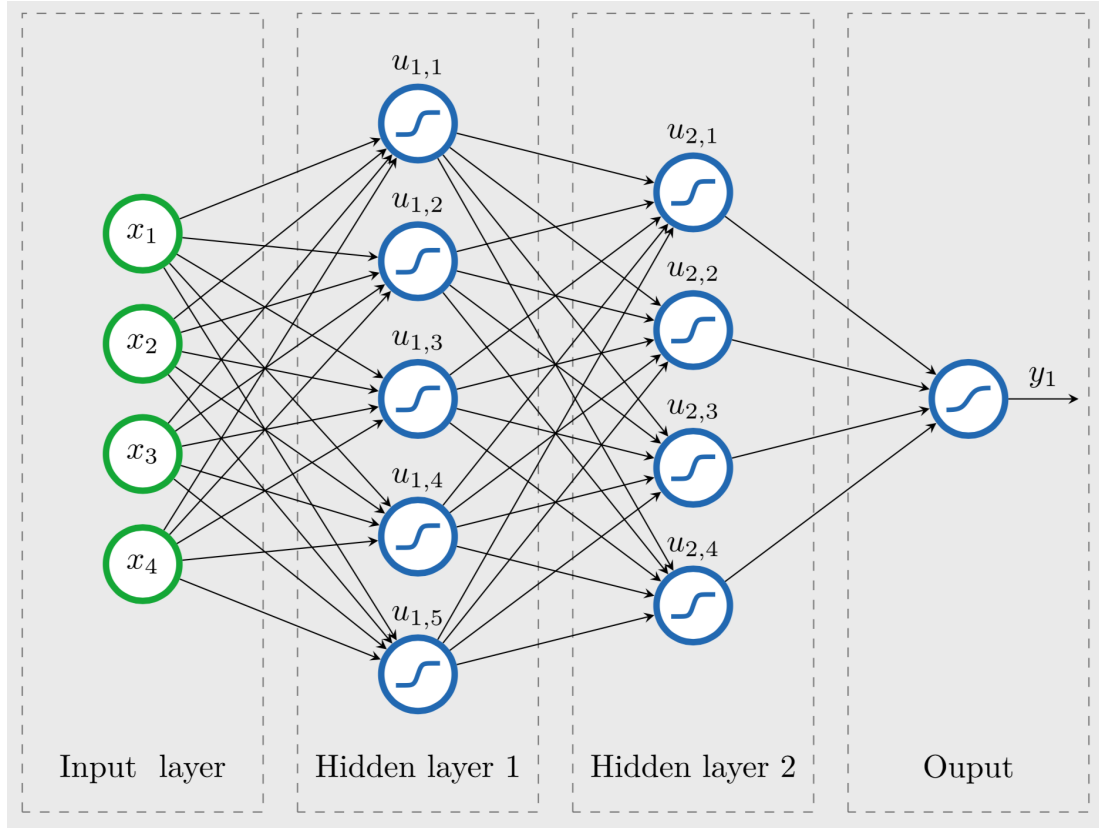


Figure 3.6: Deep Neural Network in a Multi-Layer Perceptron Layout where x_i is an input value, y_1 is an output value, and $u_{n,m}$ represents other values corresponding to activation function outputs [12].

3.3.2 Training

In order to train the model, we will evaluate a function f that takes as input $(x_1, x_2, \dots, x_n)$ and weights $(w_1, w_2, \dots, w_m)$ and produces an output $(y_1, y_2, \dots, y_p)$. We can write the following equation:

$$f(x_1, x_2, \dots, x_n; w_1, w_2, \dots, w_m) = (y_1, y_2, \dots, y_p)$$

The goal is to optimize the weights, which are chosen randomly in the beginning. In order to obtain a good result, we need to evaluate the output value of $f(x_i; w)$ and compare it to the observed result y_i . To do this, we use a loss function that sums the results over all the observations and represents the global error. In our case, we used the mean squared error (MSE) as a loss function. To optimize the weights, we compute the gradient of the loss function in order to move in the direction of a low loss (which corresponds to a low error rate) [12].

3.3.3 Back-propagation

The algorithm that we will use in order to compute the gradient of the loss function is called back-propagation. In fact, if we simply compute the gradient by numerical evaluation of each parameter, it will increase time exponentially. For example, if the neural net has 100 million parameters then we would need to do 100 million evaluations of the network.

Back-propagation solves this problem. When propagating information forward through the network graph, we accumulate a scalar error $e(w)$. Afterwards, we compute the derivative of the obtained error and apply the chain rule of derivation backwards from the output layer to every single weight in the network. The worst-case complexity of the back-propagation algorithm is $\mathcal{O}(n^2)$, but in practice, for most architectures, it tends to be linear [12].

One of the major problems of computing the gradient is the so-called **vanishing gradient**. We have a high risk that an input could produce a near zero derivative and gradient descent will get stuck making negligible steps. There are several ways to avoid this problem, for example:

- Using a **ReLU** activation function because half of the domain has a nonzero gradient.
- Computing the gradient over batches of the data. This process is called **mini-batch gradient descent**. Small batch sizes imply that the computed gradient is a little bit noisier and likely to change from batch to batch. The latter could help us to avoid the local minima. However, it could lead to poor convergence.

3.3.4 Underfitting and Overfitting

During the training phase, the problems known as underfitting or overfitting could occur as we could observe it on figure 3.7.

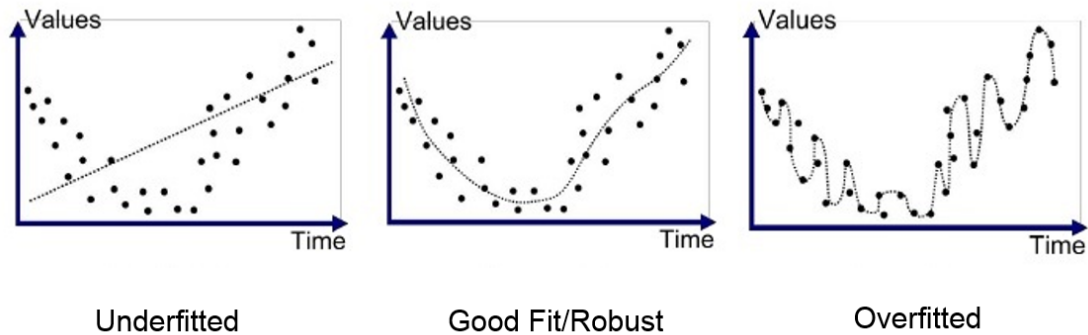


Figure 3.7: Example of different models on the same data set. A) Underfitting is observed, a model is of a lower order and does not model the data accurately. B) The model is well-chosen and correctly approximates the data. C) Overfitting is observed, the model is wrong in between the observed data [13].

- Underfitting: this problem occurs when the model cannot fit the training data and testing data. Underfitting is easy to detect as the value of the loss function would not be low enough. This problem is also easy to correct: one needs to increase the complexity of the neural net, for example by increasing the number of neurons or even changing the layer configuration.
- Overfitting: this problem occurs when the model perfectly fits the training data but actually does not perform that well on the testing dataset. This problem is also easy to detect by comparing the loss functions of both the training and the validation set. There are several techniques to avoid the problem. First of all, adding more data helps, as it allows one to better understand the distribution of the data. This will help the model to generalise better. Other ways to obtain more data could be achieved by taking the original observations and adding some noise to the features (e.g. adding some Gaussian noise). Another way is to add noise to the hidden unit itself, by using a data augmentation. The most common examples include rotation, scaling, translation etc. of existing samples, and we also use **dropout**, which can be seen as a process of multiplying existing data by noise.

3.3.5 Convolutional Neural Networks

As we saw previously, convolutional neural networks are a type of feed-forward neural network. However, convolutional net layers are not fully connected layers as opposed to, for example, dense layers. In dense layers, every input unit is connected to every output unit in the preceding layer, as we can see on figure 3.8.

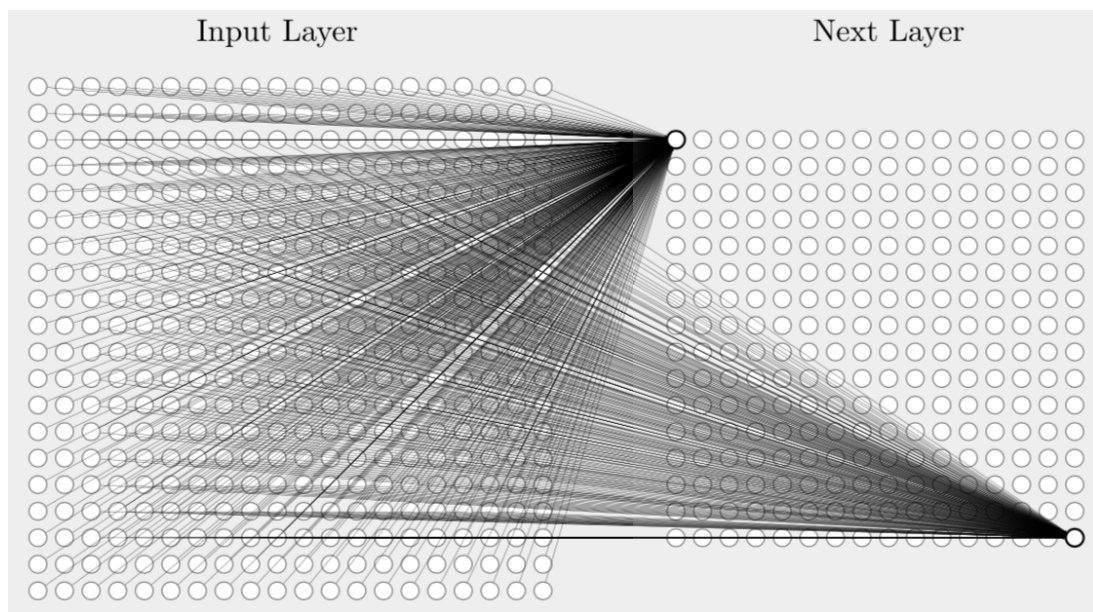


Figure 3.8: Two fully connected layers. Every input unit of the layer has to be connected to every output unit of the next layer [12].

Assume that we take an image of size $3hw$, where the 3 corresponds to three color channels. If the next layer is of the same size, we have to train $(3hw)^2$ weights. The number of weights becomes large very quickly. Moreover, by fully connecting all the units, we do not take advantage of the structure of the image, which usually has some level of spatial locality: pixel values tend to be related to their neighbors. This is what convolutional layers are doing, as we can observe on figure 3.9. The convolutional net replaced the individual scalar weights by a matrix that we call a **kernel**. This means we use convolution, rather than general matrix multiplication. The weights are no longer associated to precise units, as they are shared across all pixels.

By doing this, we encounter the problem that the border pixels do not have the number of neighbours required by the kernel size. To deal with this, a **padding** term is introduced. For example, in Keras, two paddings are possible: “**same**” and “**valid**.” While **same** means that we put all the “missing” neighbours to 0, **valid** means that we do not take into account pixels that do not have a large enough

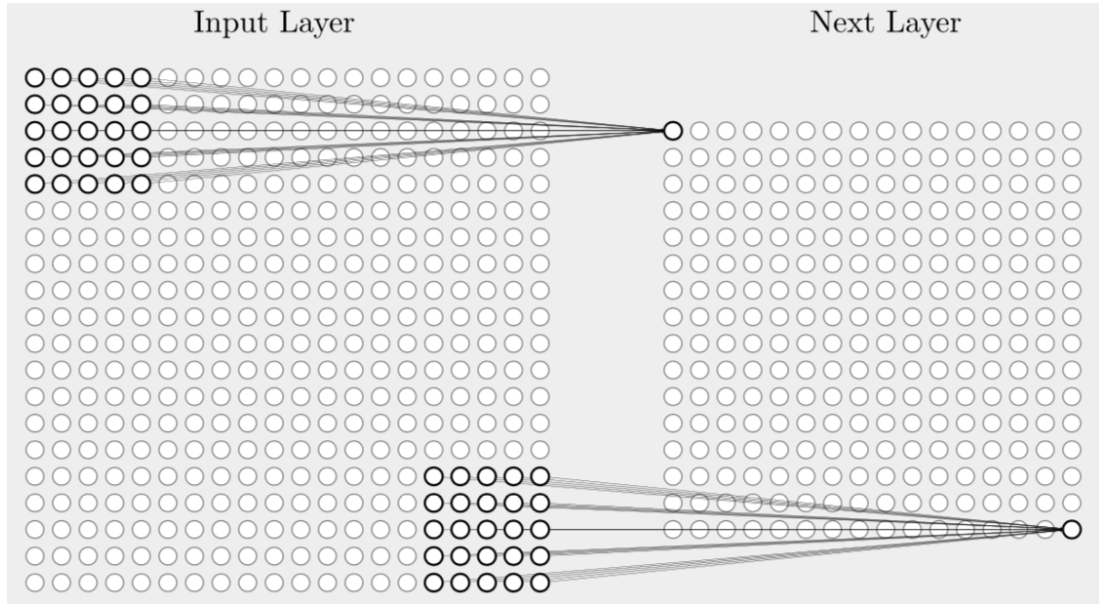


Figure 3.9: Two convolutional layers. A unit from the next layer is only connected to the neighbors of the same unit in the preceding layer [12].

number of neighbors. In the latter case, the image would be a little bit cropped. This has a direct implication on the denoising of the images in this particular work. Due to this, we need to perform multiple inferences in order to avoid the contamination of the slices. This slows down the denoising process.

4 | Experimental setup

In this section, we will describe all the experimental setup that allowed us to introduce and test the correction term in the process. We start by explaining the functioning of the MCsquare implementation. The two developed methods were implemented in the in-house treatment planning system **OpenTPS** implemented by Kevin Souris, a Python based research tool [14]. **OpenTPS** uses the Monte Carlo dose engine MCsquare to perform the dose calculations. This open-source project can be found in [15]. All the changes and simulations were made inside this tool. We then introduce the deep learning architecture **simpleNet**, developed by Umair Javaid [16], and discuss our improvements to it, comparing them to the existing model.

4.1 MCsquare

For this thesis, a Monte Carlo algorithm called MCsquare was used. The implementation fully makes use of the CPU, so the calculation could be made on a computation server or even a laptop, being easy to use in research as well as allowing optimal integration in clinics. The source code is open source on openmcsquare.org. MCsquare was adapted to provide the dose information and uncertainty model required by the robust optimizer. MCsquare stands for “multi-core Monte Carlo.” The accuracy of the Monte Carlo method is crucial for appropriate further use. For this reason, instead of applying a correction to water data, stopping powers and nuclear cross sections are computed from a database of multiple materials [17]. One could observe the summary of the main steps of the MCsquare algorithm on image 4.1.

4.1.1 Input

We need to provide a CT image of the patient, usually in MHD binary file format. The latter is a special format for medical images. Moreover, a calibration curve is also provided in order to convert the Hounsfield Units of the CT image into mass

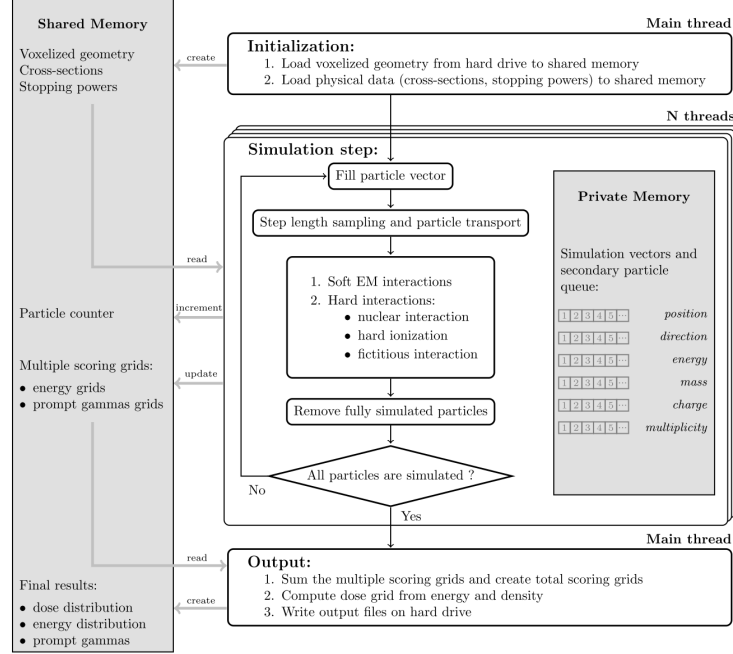


Figure 4.1: Flowchart representing the main steps of the algorithm implemented in MCsquare [17].

densities and type of materials. The simulation parameters can easily be modified by the user depending on their needs. The speed and accuracy would also depend on the entered parameters.

4.1.2 Transport algorithm

Simulating every particle interaction is a time-consuming procedure. As was seen in chapter 3 above, we can combine the discrete events into one scattering event. Thus, we will use the condensed history. There are two ways to simulate the particle interaction as was discussed in previous sections: class I and class II. In MCsquare, we will use class II. The second type of condensed history involves grouping the minor events and individually computing the big ones. In this case, only nuclear interactions or collisions with orbital electrons involving energy transfers higher than a set threshold are simulated individually. The stopping criterion is defined by the kinetic energy. When the latter falls under a certain threshold defined by the user, the remaining energy is locally absorbed and the process stops.

The interaction probability of the voxel varies depending on the material and the density. Thus, if the particle crosses more than one voxel, the error probability increases. There are two methods implemented in MCsquare to deal with interface

crossing:

- The most accurate method is to stop the simulation step at each voxel.
- The faster method is to not stop it at each voxel. In this case, if there is a change of the voxel interface, the step length is scaled by the local density and the mass stopping power ratios. This method allows us to take into account the different materials and thus respectively update the loss energy without losing time on starting a new simulation. However, this leads to incorrect sampling of nuclear reactions near interfaces. Therefore, it adds some slight uncertainties in the simulation process.

4.2 Artificial Network simpleNet

For the neural network part of this work, a simpleNet architecture (denoted hereinafter as sNet) was used [16]. The sNet is a fully convolutional neural network with a stack of 7 convolutional layers as we can observe on figure 4.2.

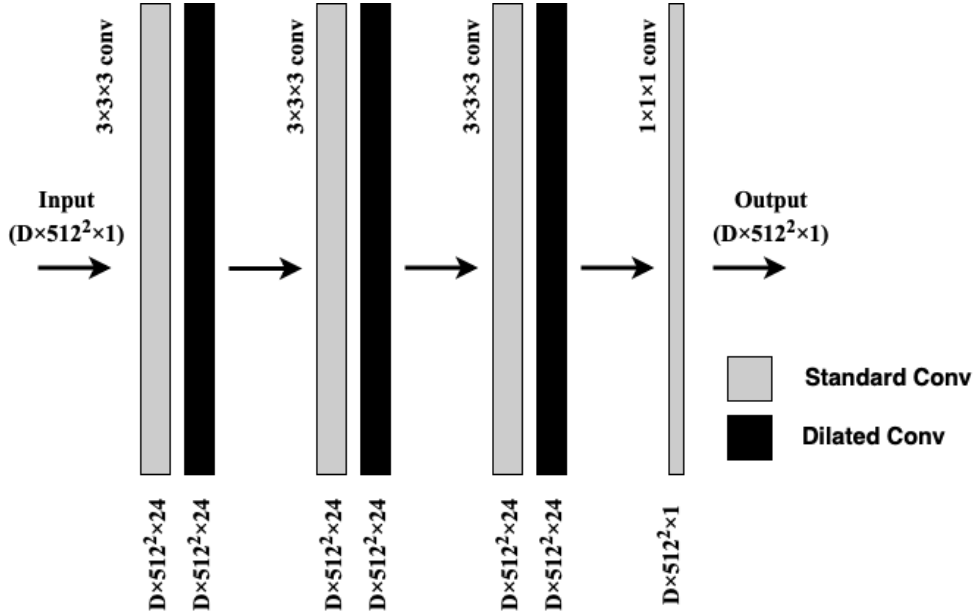


Figure 4.2: SimpleNet architecture. This architecture contains seven convolutional layers, each of them having a $3 \times 3 \times 3$ kernel size with ReLU as activation function, except for the last layer which has a $1 \times 1 \times 1$ kernel. Inputs of dimension $D \times 512 \times 512$ are given to the network, where D corresponds to the depth (or number of layers), either 7 or 24. Images of dimension $D \times 512 \times 512$ are produced as output.

All convolutional layers are 3-D, with a kernel of size $3 \times 3 \times 3$ and are followed by a nonlinear activation function, **ReLU**. Input of dimensions $D \times 512 \times 512$ is given, where D is the depth (or number of layers), which is either 7 or 24 depending on the training setup, and represents the number of possible layers.

A dilated convolution was applied to every second convolution. Let $F: \mathbb{Z} \rightarrow \mathbb{R}$ be a discrete function. Let $\Omega_r = [-r, r]^2 \cap \mathbb{Z}^2$ and $k: \Omega_r \rightarrow \mathbb{R}$ be a discrete filter of size $(2r + 1)^2$. The discrete convolution operator $\star$ can be defined as:

$$(F \star k)(\mathbf{p}) = \sum_{\mathbf{s}+\mathbf{t}=\mathbf{p}} F(\mathbf{s})k(\mathbf{t}).$$

Compared to standard convolution, dilated convolution can be defined as:

$$(F \star_l k)(\mathbf{p}) = \sum_{\mathbf{s}+l\mathbf{t}=\mathbf{p}} F(\mathbf{s})k(\mathbf{t}),$$

where l is a dilation factor and $\star_l$ is a dilated convolution or an l -dilated convolution. One needs to note that $\star$ is simply the 1-dilated convolution [18]. One can observe the geometric transformation performed by the dilation operation on figure 4.3 presented below.

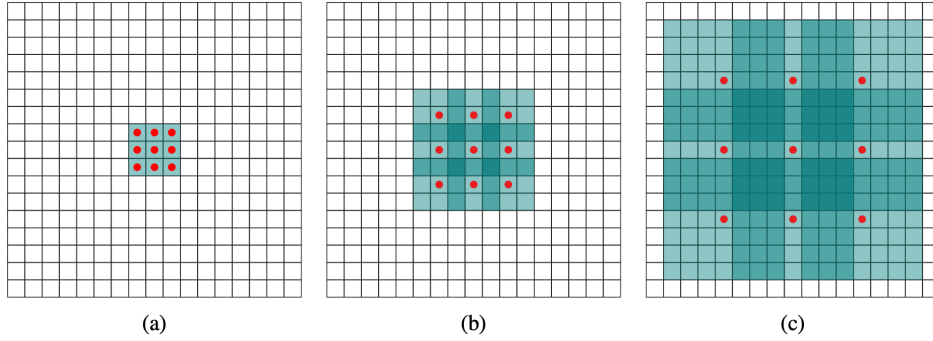


Figure 4.3: Illustration of the dilation layer. Systematic dilation supports exponential expansion of the receptive field without loss of resolution or coverage. (a) F_1 is produced from F_0 by a 1-dilated convolution; each element in F_1 has a receptive field of 3×3 . (b) F_2 is produced from F_1 by a 2-dilated convolution; each element in F_2 has a receptive field of 7×7 . (c) F_3 is produced from F_2 by a 4-dilated convolution; each element in F_3 has a receptive field of 15×15 . The number of parameters associated with each layer is identical. The receptive field grows exponentially while the number of parameters grows linearly [18].

Compared to normal convolution, dilated convolution enables network operation on a coarser scale by aggregating additional spatial context. For a convolutional $k \times k \times k$ kernel, the size of the resulting dilated filter is $kd \times kd \times kd$, where $kd = k + (k - 1)(r - 1)$ and r is the dilation rate. In our case, the kernel size is $k = 3$ and $r = 2$.

4.2.1 Network training

Training the network took up to four hours. The TensorFlow25 libraries (a deep learning platform) with Keras26 interface were used for all experiments. The training data consisted of 31 patients with different tumor locations: brain, head and neck, liver, lungs, and prostate. To form the validation data, we used the data from four patients that have cancer in the head and neck, liver and lungs. For a more robust training process, a noisy dose map per patient was added. The mean squared error was used as the loss function. The artificial network was trained for 500 epochs, using the Adam algorithm with learning rate 0.001 as optimiser. If, during training, no improvement was observed during 10 epochs, the learning rate was put to 0.1. The same is true for the stopping criterion: if no changes were observed during 15 epochs, the training was stopped.

4.2.2 Changes applied

In order to improve the given architecture and adapt it more to our needs, we decided to change the number of slices for the 3D image. Given the fact that during development of sNet, research was limited by GPU capacity, we decided to explore this aspect and see if the results could be even better.

The explored depth was $D = 50$. As input we thus have images of size $50 \times 512 \times 512$. For those images that we could not “cut” up to 50 slices, we added zeroes on the border of the image.

4.2.3 Discarded layers

As was mentioned before, when we are using Convolutional Neural Nets, the borders could become noisy and thus not exploitable in our case. Therefore, to properly denoise the image we need to discard certain layers. The theory tells us we have

$$k = 2h + 1$$

where k is the area covered in the input image and h is the number of contaminated layers to discard. In our case, as we have 3 layers of 3×3 convolutional kernels, and thus cover an area of size 7 in the input image, this becomes

$$7 = 2 * h + 1 \iff h = 3$$

This means that in theory, we have to discard 3 layers. In order to check this in practice, we tested different numbers of discarded layers in order to see the impact. On figure 4.4, we show the obtained results.

On figure 4.4, one can easily see that discarding one slice on each side is not enough, and that the resulting image is noisy. To more clearly compare the other

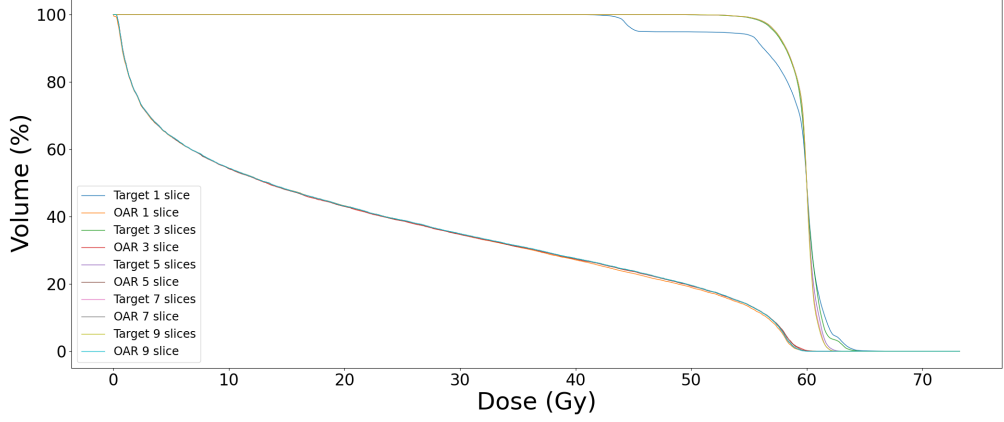


Figure 4.4: DVH of the target and OAR by applying `sNet_50` on the same image but discarding different numbers of layers. The legend shows the number of discarded layers for each curve. We can directly observe that only discarding one slice is not enough, as the result is even noisier than without applying any correction term. On the right lower corner, we can observe that the curve representing 3 discarded slices is also quite noisy compared to the remaining ones. The curves with 5, 7 and 9 discarded slices visually look very similar.

values, we look at the target D_{95} and D_5 values for the different simulations, which we tabulated in table 4.1.

Discarded slices	Target D_{95}	Target D_5
1	45.51 Gy	62.25 Gy
3	57.19 Gy	61.75 Gy
5	57.27 Gy	61.40 Gy
7	57.29 Gy	61.27 Gy
9	57.29 Gy	61.21 Gy

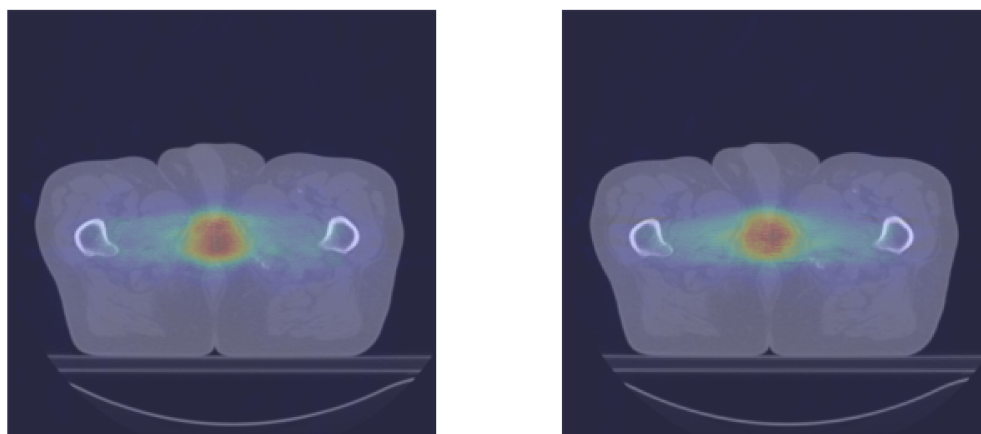
Table 4.1: Effect of discarding a certain number of slices due to the fact that CNN affects borders. One can see that if we discard only one slice of each side, the resulting image is too noisy and does not give us any exploitable results. If we discard three layers on each side, the result is actually already good compared to the ground truth 5.1 but we still could do better. We also observe that the result starts to stabilise after discarding 5 slices or more.

The values in the table confirm that discarding just one slice is not the best idea. One could also see that by discarding only 3 layers on each side, the target D_5 value is not optimal. The remaining results are very similar. For further simulations, we decided to discard 7 slices from each side, thus 14 slices in total.

4.2.4 Results

In this section, we will present the comparison between sNets with depth 24 and 50 (which hereinafter we will refer to as **sNet_24** and **sNet_50** respectively) and discuss the obtained performance. First of all, we produced a dose map obtained by using 10^3 protons per beamlet without any correction term but we applied the optimisation process.

Then, we applied the denoising process of the two sNet architectures on that same image. The result can be seen on figure 4.5.



Denoising with **sNet_24**.

Denoising with **sNet_50**.

Figure 4.5: Denoising with **sNet_24** and **sNet_50** respectively.

In order to judge the result properly, as we previously mentioned, we use dose-volume histograms and the target D_{95} and D_5 values. For consistent comparison of the dose distributions, we normalized the dose such that D_{50} is equal to $D_{\text{prescribed}}$. On figure 4.6, one can observe the DVH of the initial dose map and the resulting dose map after applying **sNet_24** and **sNet_50** respectively. We can easily see that both methods performed pretty well: they improved, and thus “denoised,” the initial dose map.

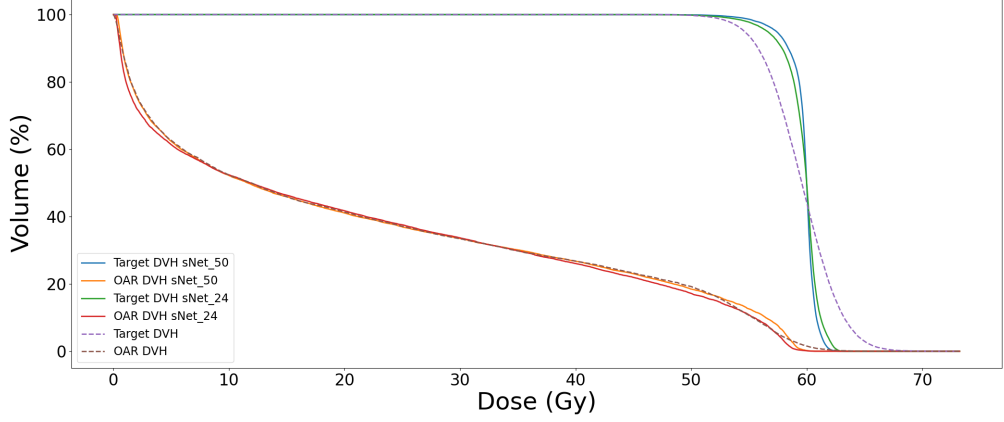


Figure 4.6: DVH comparison graph. The dotted curve represents the DVH of the dose map before applying a deep learning process on it. The solid curve represents the DVH of the target and OAR of the `sNet_24` and `sNet_50` models, respectively.

To compare the performance of both architectures, table 4.2 presents the important target D_{95} and D_5 values, and D_{50} for OAR.

Denoising applied	Target D_{95}	Target D_5	OAR D_{50}
None	54.61 Gy	64.29 Gy	11.85 Gy
<code>sNet_24</code>	56.60 Gy	61.72 Gy	11.84 Gy
<code>sNet_50</code>	57.37 Gy	61.22 Gy	11.63 Gy
Ground Truth	57.53 Gy	61.32 Gy	11.97 Gy

Table 4.2: Overview of the effect of applying sNet denoising on target D_{95} and D_5 , as well as OAR D_{50} . The first line represents the values before any denoising process is applied. Ground truth was obtained by simulating 1×10^9 particles. We see that `sNet_50` outperforms `sNet_24` by quite a bit.

By comparing the two architectures, we observe that `sNet_50` performs better than `sNet_24`. In this particular case, the difference in D_{95} is equal to 0.77, which is an improvement of more than 1%. Results are similar for the difference in D_5 . Overall, the performance of the method is mostly seen in the target coverage metrics. The ground truth value was obtained by simulating 1×10^9 protons per beamlet. We obtain nearly the same results with `sNet_50` as we had in the ground truth.

5 | Results

In this section we will first present the results obtained without introducing the correction term, then we will apply the correction term using full Monte Carlo simulation. Next, we will do the same but with neural nets. Finally, we will compare the two methods.

All the simulations were done on a prostate cancer patient. The prescribed dose is set to 60 Gy. The ground truth was simulated using 1×10^9 protons per beamlet and the results are summarised in table 5.1.

Target D_{95}	Target D_5	OAR D_{50}
57.53 Gy	61.32 Gy	11.97 Gy

Table 5.1: Ground truth values computed by simulating 1×10^9 protons per beamlet without applying any correction term.

5.1 Dose simulation

The following results were obtained by simulating 10^3 beamlets on the CT image of the patient with prostate cancer. Table 5.2 represents the improvement that we obtained by increasing the number of protons per beamlet. The results obtained below are an average over 10 simulations for each case.

Protons/beamlet	Target D_{95}	Target D_5	OAR D_{50}	Time
10^2	54.06 Gy	64.45 Gy	9.11 Gy	150.96 s
10^3	56.40 Gy	63.05 Gy	12.24 Gy	308.01 s
10^4	56.58 Gy	62.87 Gy	11.90 Gy	1751.15 s
10^5	57.06 Gy	61.88 Gy	11.52 Gy	4831.73 s
10^6	57.26 Gy	61.58 Gy	11.70 Gy	11 006.20 s

Table 5.2: This table represents the average results obtained by simulating different numbers of protons per beamlet without any denoising applied. We compare these various simulations on their target D_{95} and D_5 values, as well as the OAR D_{50} and the average computation time of each simulation. We observe that increasing the number of protons per beamlet causes an increase of the target D_{95} value and a decrease of the target D_5 value. This is what one would expect to see, as our goal is to respect the prescribed dose for the target (which is 60 Gy) but still affect OAR as little as possible. We would like our denoising results to reach at least the same performance as when using 10^6 protons per beamlet, while reducing the computation time.

When observing the values presented above, the target D_{95} increases and the target D_5 decreases with the number of protons per beamlet, as one would expect. Computation time also drastically increases with the number of protons per beamlet. Our goal is to respect the prescribed dose for the target, that is, deliver 60 Gy to the tumor, while affecting OAR as little as possible. By simulating more protons per beamlet, we decrease the noise on the result, which explains the fact that with more protons we obtain something more accurate.

We also observe diminishing returns in accuracy: the difference between simulating 10^3 protons per beamlet compared to the simulation of the 10^6 protons per beamlet is a little bit more than 1 %, but getting this improvement takes 30 times as long. Our goal with the correction term is to obtain results that reach the ground truth of table 5.1 but considerably faster than it takes to simulate 10^6 protons per beamlet.

5.2 Monte Carlo denoising

Table 5.3 shows the results obtained when applying a correction term during the optimisation algorithm using full Monte Carlo simulation. The corrected term was applied twice during the optimisation process.

Protons/beamlet	Target D_{95}	Target D_5	OAR D_{50}	time
5×10^2	56.54 Gy	62.35 Gy	12.11 Gy	523.28 s
1×10^3	57.00 Gy	62.11 Gy	11.87 Gy	1294.19 s
5×10^3	57.23 Gy	62.17 Gy	11.85 Gy	1537.75 s

Table 5.3: This table shows the results obtained when simulating different numbers of protons per beamlet by applying a denoising process using a full Monte Carlo simulation. We compare these various simulations on their target D_{95} and D_5 values, as well as the OAR D_{50} and the average computation time of each simulation. When decreasing the number of protons per beamlet, we observe the target D_{95} increasing towards the ground truth. Also, we observe a slight decrease of the target D_5 value.

One could see in table 5.3 above, that with the increase of the number of protons per beamlet, we observe an increase of the target D_{95} and a slight decrease of the target D_5 . As one would expect, with a higher number of protons per beamlet, we decrease the amount of noise. This time, by applying the correction term, we observe that we have even less noise than was previously observed with the same amount of protons per beamlet but without applying a correction term.

If we compare the computation time, we get the same result. Figures 5.1, 5.2 and 5.3 show the difference by applying the correction term as computed through Monte Carlo denoising. The dotted lines represent the dose-volume histogram of the simulation without any correction term applied; the full lines represent the simulation done with the same number of protons per beamlet but by applying a correction term using a full Monte Carlo simulation.

Note that applying the correction term increases the computation time for a same number of protons, but also drastically improves the performance of the simulation. Thus, by comparing the obtained results with the results obtained without any correction term in table 5.2, we can see that with 10^3 protons per beamlet and the correction term being applied, we can reach the same result as by simulating 10^5 protons per beamlet but without the correction term. Moreover, the same quality result is obtained four times faster.

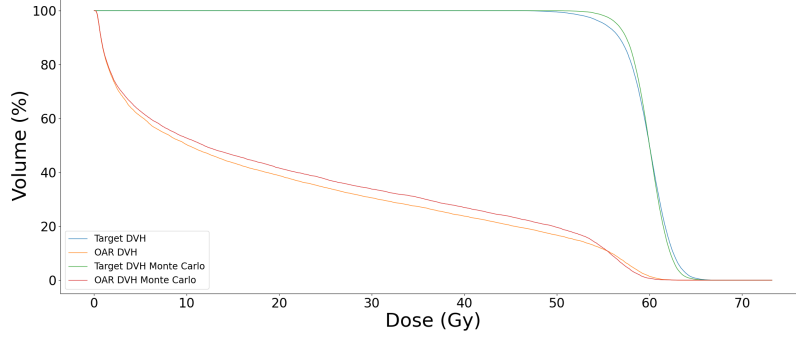


Figure 5.1: Comparison of the results with and without applying the correction term using full Monte Carlo simulation by using 5×10^2 protons per beamlet. Dotted DVH curves stand for the image resulting from not applying any correction term. The full curves stand for the simulation where the correction term was applied. The target D_{95} is a value very sensitive to the noise. We could observe for the DVH of the target that the values of D_{95} with the correction term are better than the ones without it. The same observation could be made for the D_5 value.

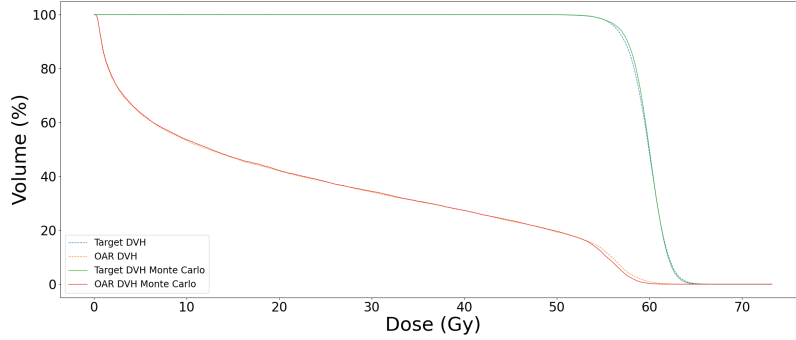


Figure 5.2: Comparison of the results with and without applying the correction term using full Monte Carlo simulation by using 1×10^3 protons per beamlet. Dotted DVH curves stand for the image resulting from not applying any correction term. The full curves stand for the simulation where the correction term was applied. The D_{95} is a value very sensitive to the noise. We could observe for the DVH of the target that the values of D_{95} with the correction term are better than the ones without it. The same observation could be made for the D_5 value.

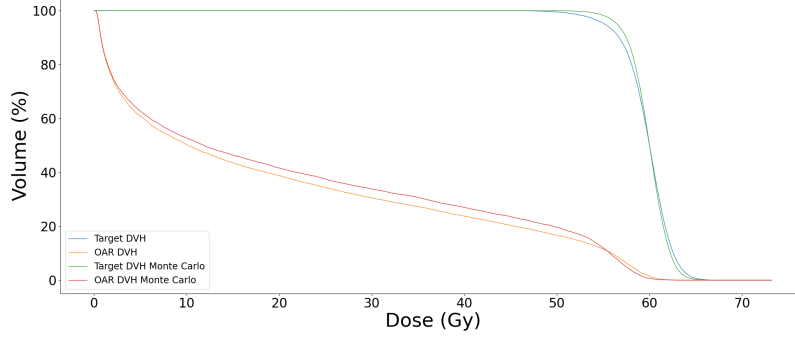


Figure 5.3: Comparison of the results with and without applying the correction term using full Monte Carlo simulation by using 5×10^3 protons per beamlet. Dotted DVH curves stand for the image resulting from not applying any correction term. The full curves stand for the simulation where the correction term was applied. The D_{95} is a value very sensitive to the noise. We could observe for the DVH of the target that the values of D_{95} with the correction term are better than the ones without it. The same observation could be made for the D_5 value.

5.3 sNet denoising

In this section we will look at the results obtained by the `sNet_24` and `sNet_50` neural network architectures. We start by looking at them individually, then compare them.

Table 5.4 shows the results obtained when applying a correction term to the optimisation algorithm using `sNet_24`. The correction term was applied twice during the optimisation process.

Protons/beamlet	Target D_{95}	Target D_5	OAR D_{50}	time
5×10^2	57.20 Gy	61.83 Gy	12.02 Gy	415.60 s
1×10^3	57.42 Gy	61.76 Gy	11.89 Gy	629.16 s
5×10^3	57.54 Gy	61.02 Gy	11.93 Gy	870.38 s

Table 5.4: This table shows the results obtained when simulating different numbers of protons per beamlet when applying a denoising process using the `sNet_24` architecture. We compare these various simulations on their target D_{95} and D_5 values, as well as the OAR D_{50} and the average computation time of each simulation. We can observe that we have less noise with a higher number of protons. This can be seen in the increase of the target D_{95} and decrease of the target D_5 .

Using the table above, one could observe the decrease of the noise that occurs when simulating more protons per beamlet. Therefore, we can confirm that the correction term helps us to decrease the noise. Actually, the values in the table 5.4 show that we are very close to the ground truth of table 5.1. Even if the time increases with the number of simulated protons, it is still much better than what one could do without applying the correction term. Thus, if we refer to table 5.2, we can see that we obtain the same result with 5×10^2 protons per beamlet as by simulating 10^6 protons per beamlet in the undenoised case. However, thanks to the denoising processing, we do it more than 25 times faster.

On figures 5.4, 5.5 and 5.6, we show the difference between simulations using the same number of protons per beamlet, but with or without a correction term computed using `sNet_24`. By comparing the values of the target values such as D_{95} and D_5 using tables 5.2 and 5.4, we can observe a change of more than 1 %. This is visible on the figures as well. Thus, one can observe the deviation of the denoised DVH curve in the upper left corner and lower left corner on all three graphs due to this.

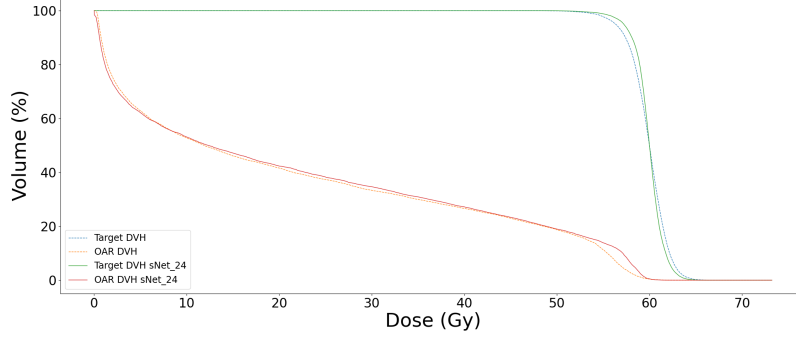


Figure 5.4: Comparison of the results with and without applying the correction term obtained using the `sNet_24` architecture by using 5×10^2 protons per beamlet. Dotted DHV curves stand for the image resulting from not applying any correction term. The full curves stand for the simulation where the correction term was applied. The D_{95} is a value very sensitive to the noise. We could observe for the DVH of the target that the values of D_{95} with the correction term are better than the ones without it. The same observation could be made for the D_5 value. It is due to the fact that we decreased the noise by applying the correction term.

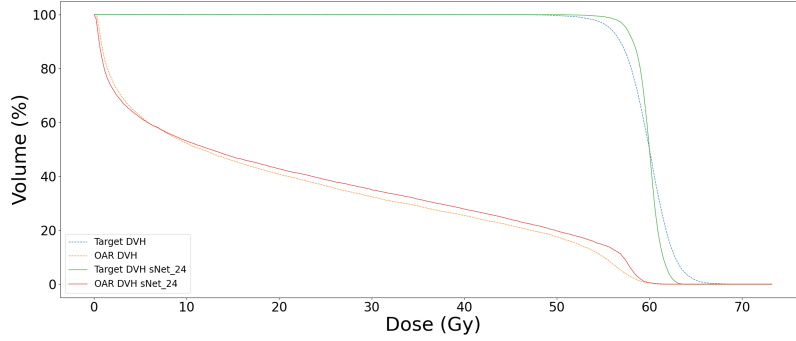


Figure 5.5: Comparison of the results with and without applying the correction term obtained using the `sNet_24` architecture by using 1×10^3 protons per beamlet. Dotted DHV curves stand for the image resulting from not applying any correction term. The full curves stand for the simulation where the correction term was applied. The D_{95} is a value very sensitive to the noise. We could observe for the DVH of the target that the values of D_{95} with the correction term are better than the ones without it. The same observation could be made for the D_5 value. It is due to the fact that we decreased the noise by applying the correction term.

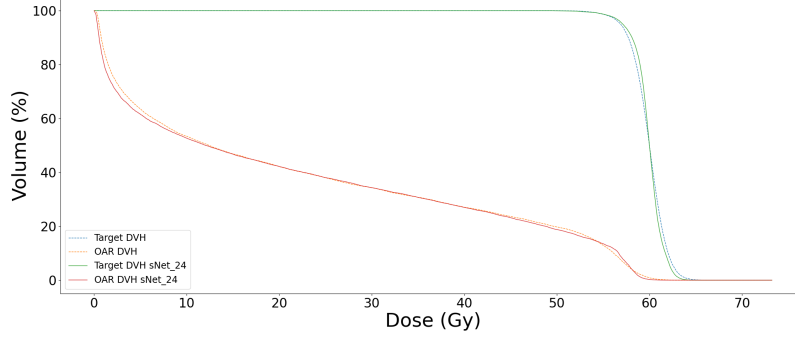


Figure 5.6: Comparison of the results with and without applying the correction term obtained using the **sNet_24** architecture by using 5×10^3 protons per beamlet. Dotted DHV curves stand for the image resulting from not applying any correction term. The full curves stand for the simulation where the correction term was applied. The D_{95} is a value very sensitive to the noise. We could observe for the DVH of the target that the values of D_{95} with the correction term are better than the ones without it. The same observation could be made for the D_5 value. It is due to the fact that we decreased the noise by applying the correction term.

In table 5.5, we show the results when applying a correction term during the optimisation algorithm, but this time using the **sNet_50** architecture. The correction term was applied twice during the optimisation process.

Protons/beamlet	Target D_{95}	Target D_5	OAR D_{50}	time
5×10^2	57.07 Gy	61.55 Gy	12.10 Gy	321.55 s
1×10^3	57.47 Gy	61.31 Gy	11.83 Gy	409.87 s
5×10^3	57.49 Gy	61.27 Gy	11.95 Gy	604.55 s

Table 5.5: This table shows the results obtained when simulating different numbers of protons per beamlet when using the **sNet_50** architecture. We compare these various simulations on their target D_{95} and D_5 values, as well as the OAR D_{50} and the average computation time of each simulation. The D_{95} is a value very sensitive to the noise. We could observe for the DVH of the target that the values of D_{95} with the correction term are better than the ones without it.

The same observation could be made for the D_5 value.

On this table we observe an increase of the target D_{95} value towards the ground truth values 5.1 by increasing the number of protons. The same could be said

about the target D_5 value, a slight decrease is happening when we increase the number of protons per beamlet. Therefore, we can notice the result of the correction term which significantly decreases the noise. Moreover, by comparing the obtained results with those of table 5.2, we can observe that with only 5×10^2 protons per beamlet we can reach a better result than by simulating 10^6 protons per beamlet. Also, it is done more than 25 times faster with a correction term than without one.

On figures 5.7, 5.8, and 5.9, we visualise the difference between simulations using the same number of protons per beamlet, but with or without a correction term computed using `sNet_50`. By comparing the values of the sensible values such as D_{95} and D_5 using tables 5.2 and 5.4, we can observe changes of more than 1 %. Thus, one could observe the deviation of the denoised DVH curve in the upper left corner and lower left corner on all three graphs due to this.

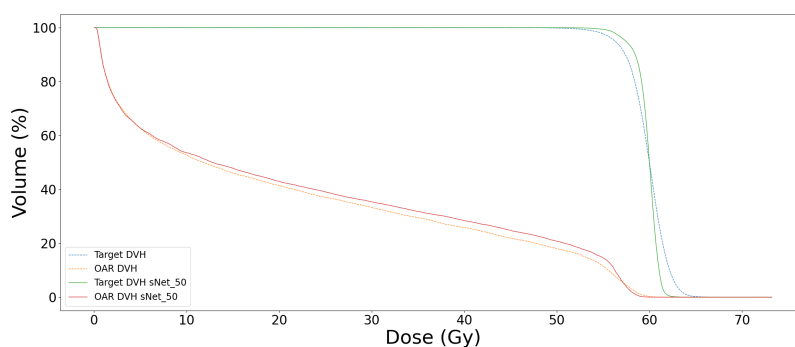


Figure 5.7: Comparison of the results with and without applying the correction term obtained using the `sNet_50` architecture by using 5×10^2 protons per beamlet. Dotted DHV curves stand for the image resulting from not applying any correction term. The full curves stand for the simulation where the correction term was applied. The D_{95} is a value very sensitive to the noise. We could observe for the DVH of the target that the values of D_{95} with the correction term are better than the ones without it. The same observation could be made for the D_5 value.

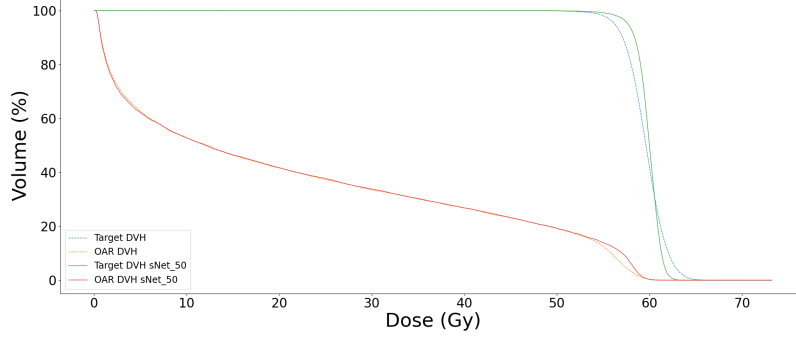


Figure 5.8: Comparison of the results with and without applying the correction term obtained using the **sNet_50** architecture by using 1×10^3 protons per beamlet. Dotted DHV curves stand for the image resulting from not applying any correction term. The full curves stand for the simulation where the correction term was applied. The D_{95} is a value very sensitive to the noise. We could observe for the DVH of the target that the values of D_{95} with the correction term are better than the ones without it. The same observation could be made for the D_5 value. It is due to the fact that we decreased the noise by applying the correction term.

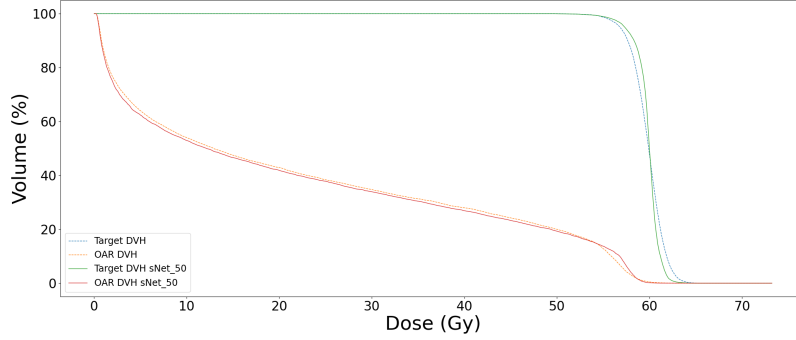


Figure 5.9: Comparison of the results with and without applying the correction term obtained using the **sNet_50** architecture by using 5×10^3 protons per beamlet. Dotted DHV curves stand for the image resulting from not applying any correction term. The full curves stand for the simulation where the correction term was applied. The D_{95} is a value very sensitive to the noise. We could observe for the DVH of the target that the values of D_{95} with the correction term are better than the ones without it. The same observation could be made for the D_5 value. It is due to the fact that we decreased the noise by applying the correction term.

Comparing these two neural network-based methods for denoising, we saw

in chapter 4 that **sNet_50** performs slightly better than **sNet_24** when denoising the same image once. When using these methods to compute the correction term, no significant differences in the results were observed. The difference is too small to affirm that either one performs better than the other. However, **sNet_50** requires less time to achieve the same results.

5.4 Comparison

As was discussed in the previous section, **sNet_50** performs slightly better than **sNet_24** in terms of computation time and we can observe it in table 5.6. For that reason, we will just use this architecture to compare the full Monte Carlo simulation to the Deep Learning denoising.

Protons/beamlet	sNet_24 time	sNet_50 time
5×10^2	415.60 s	321.65 s
1×10^3	629.16 s	409.87 s
5×10^3	870.38 s	604.55 s

Table 5.6: Computation time difference between **sNet_24** and **sNet_50**. One could observe that **sNet_50** performs better than **sNet_24**.

If one compares the results of tables 5.5 and 5.3, it is clear that deep learning methods outperform their Monte Carlo counterparts both in terms of time and in terms of accuracy. Quantitatively, Monte Carlo denoising takes two times longer than the Deep Learning approach. Moreover, with the Deep Learning techniques, we obtain values very similar to the ground truth. Table 5.7 contains a summary of the differences between the two methods regarding the DVH curves values and the table 5.8 shows the time differences.

Protons/beamlet	D_{95} MC	D_5 MC	D_{95} DL	D_5 DL
5×10^2	56.54 Gy	62.35 Gy	57.07 Gy	61.55 Gy
1×10^3	57.00 Gy	62.11 Gy	57.47 Gy	61.31 Gy
5×10^3	57.23 Gy	62.17 Gy	57.49 Gy	61.27 Gy

Table 5.7: We computed the correction term using the two methods: Monte Carlo full simulation and Deep Learning. MC stand for Monte Carlo and DL for Deep Learning. One could see by the results that Deep Learning performs much better than Monte Carlo.

Protons/beamlet	Monte Carlo time	Deep Learning time
5×10^2	523.28 s	321.65 s
1×10^3	1294.19 s	409.87 s
5×10^3	1537.75 s	604.55 s

Table 5.8: Here we can observe the computation time using Deep Learning and Monte Carlo methods. One could conclude, that Deep Learning is much faster than the Monte Carlo methods.

By analysing these tables, we conclude that Deep Learning performs better than the Monte Carlo method, both in accuracy and time.

6 | Conclusion

The goal of this thesis was to propose a technique to denoise Monte Carlo doses for fast and accurate proton therapy treatment planning. To achieve this, we proposed two methods: Monte Carlo simulation and Deep Learning. Both methods achieve good results compared to the ground truth dose, i.e. when we simulate 1×10^9 protons per beamlet.

To implement both methods, different tools were used. First of all, all implementations and simulations were done using the **opentps** tool. In order to implement the Monte Carlo simulation process, the Monte Carlo algorithm called **MCsquare** was used. For the deep learning approach, we used an **sNet** network architecture. In order to improve the latter, we modified the architecture by increasing the number of layers. The new architecture, called **sNet_50**, showed good improvement compared to the **sNet_24** architecture, mainly at the level of the computation time.

The first method used to compute the correction term was through full Monte Carlo simulation. We achieved accurate results four times faster by applying the correction term using Monte Carlo methods than we did by simulating the dose map without introducing any correction.

The second method used was Deep Learning. The results were quite promising, as they achieved the ground truth by simulating only a small amount of particles. Although the introduction of the denoising step increases computation time for a same number of protons per beamlet, it also gives incredible improvements in results compared to the ground truth.

Finally, we compared the two methods used: Deep Learning and full Monte Carlo simulation. The Deep learning denoising performs better than Monte Carlo one, not only by achieving a dose similar to the ground truth, but also by performing better in terms of computation time.

Nevertheless, there could be other ways to explore and improve the computational time or/and denoising process. For example, it would be interesting to consider the following ones:

- Try to develop an autoencoder Deep Learning architecture. As it is a specific architecture made to denoise pictures, it would be interesting to explore this idea.

- Play with more parameters of the given **sNet** architecture as, for example, the kernel size and number of features. However, for better generalization, one could potentially need more data to obtain satisfactory results.
- Explore the number of different possible layers. For this thesis we chose to explore 50. It would be interesting to explore other values, and see how they stack up.

We hope to have shown through this work the benefits of denoising proton therapy treatment plans using either Monte Carlo simulations, or deep neural network-based approaches.

Bibliography

- [1] World Health Organisation. *Cancer*. <https://www.who.int/health-topics/cancer>. Accessed on 2021-31-03.
- [2] Provision Cares. *What is Proton Therapy for Cancer Treatment?* <https://provisionhealthcare.com/about-proton-therapy/advantages-of-proton/>. Accessed on 2021-31-03.
- [3] H. Paganetti. “Range uncertainties in proton therapy and the role of Monte Carlo simulations”. In: *Physics in Medicine and Biology* 57 (2012), pp. 99–117.
- [4] M.S. Gazzaniga, R.B. Ivry, and G.R. Mangun. *Cognitive neuroscience: the biology of the mind*. W. W. Norton & Company, 2014.
- [5] K. Petras et al. “Basic principles of brain tumor radiotherapy”. In: *Oligodendroglioma* (2019), pp. 245–262.
- [6] T. Pavel. “Feasibility of magnetic resonance imaging-based radiation therapy for brain tumour treatment”. In: *Master thesis* (2017).
- [7] A.M. Barragan Montero et al. “Performance of a hybrid Monte Carlo-Pencil Beam dose algorithm for proton therapy inverse planning”. In: *Medical Physics* (2017), pp. 846–862.
- [8] A.M. Barragan Montero. “Robust, accurate and patient-specific treatment planning for proton therapy”. In: *Dissertation* (2018), pp. 35–36.
- [9] H. Paganetti. *Proton therapy physics: second edition*. CRC Press, 2019.
- [10] M.J. Berger. *Monte Carlo Calculation of the penetration and diffusion of fast charged particles*. Inspire, 1963.
- [11] Arne Wolfewicz. *Deep learning vs. machine learning – What’s the difference?* <https://levity.ai/blog/difference-machine-learning-deep-learning>. Accessed on 2021-08-06.
- [12] François Pitié. *Deep Learning and its Applications*. <https://frcs.github.io/4C16-LectureNotes/index.html>. Accessed on 2021-26-05.

- [13] Anup Bhande. *What is underfitting and overfitting in machine learning and how to deal with it*. <https://medium.com/greyatom/what-is-underfitting-and-overfitting-in-machine-learning-and-how-to-deal-with-it-6803a989c76>. Accessed on 2021-26-05.
- [14] K. Souris, J. A. Lee, and E. Sterpin. “Fast multipurpose Monte Carlo simulation for proton therapy using multi- and many-core CPU architectures”. In: *Medical Physics* (2016).
- [15] *Fast Monte Carlo simulations for proton PBS*. <http://www.openmcsquare.org/>.
- [16] Umair Javaid. “Mitigating inherent noise in Monte Carlo dose distributions using dilated U-Net”. In: *Medical Physics* (2019), pp. 5790–5798.
- [17] K. Souris. “Accurate assessment of proton therapy treatments : fast Monte Carlo dose engine and extensive robustness tests”. In: *PHD thesis* (2018).
- [18] Fisher Yu and Vladlen Koltun. “Multi-Scale Context Aggregation by Dilated Convolutions”. In: *International Conference on Learning Representations (ICLR)*. May 2016.
- [19] M. Urie et al. *Degradation of the Bragg peak due to inhomogeneities*. Phys Med Biol, 1986.
- [20] Oriol Vinyals et al. *A picture is worth a thousand (coherent) words: building a natural description of images*. <https://ai.googleblog.com/2014/11/a-picture-is-worth-thousand-coherent.html>. Accessed on 2021-08-06.
- [21] D. Shen, G. Wu, and H.I. Suk. “Deep Learning in Medical Image Analysis. Annual Review of Biomedical Engineering”. In: *Annual Review of Biomedical Engineering* (2019), pp. 221–248.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | www.uclouvain.be/epl