UNIVERSIDAD
DE GRANADA

MASTER'S THESIS

Mixed Monte Carlo - Convolutional Neural Network model for stereotactic radiosurgery with a dynamic micro-leaf collimator

Armando Delgado Fumero

Master's Degree in Physics and Mathematics

Academic Year 2021/2022

Supervisors:

- **Luis Díaz Angulo (UGR)**
- **Wilfredo González (UMA)**

Contents

List of Figures

List of Tables

1 Abstract

Based on the work of (Gonzalez et al. 2015), our goal is to develop a mixed Monte Carlo - Deep Learning based solution that allow us to predict the fluence outcome of a 6 MV Elekta Precise linac. This project goes through two main steps, first, we use Fippel et al. 2003 VSM (Virtual Source Model) to get a first approximation to the data, obtaining an equation composed of a Pearson VII distribution modified by another custom equation that defines a depression on the center of each profile. Monte Carlo step of this VSM is used to obtain 6, 5 coefficients that define the central depression present on the profile and a sixth one that defines how abrupt the fluence zone ends. After obtaining this function which approximates the fluence results, we train and test a 1 Dimensional Convolutional Neural Network, improving the results of the analytic function on a 20% basis.

2 Introduction

2.1 Linac

A linear acelerator (linac) is an artifact that uses microwave technology to initially accelerate electrons on a wave guide. These accelerated electrons hit a metal target, which produces x-rays. These photons product of the collision are directed towards the machine exit, shaped as the physician requires. These machines are used mainly on cancer treatments, delivering the radiation to the tumor.

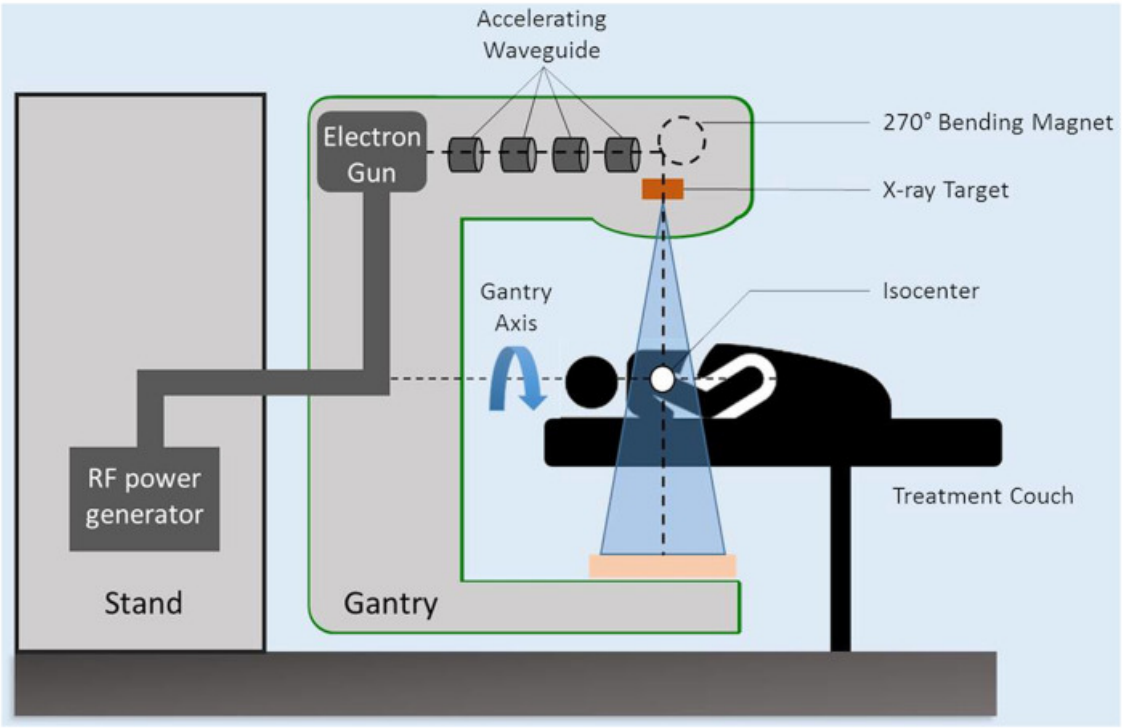

Figure 1: Schematic diagram of a linac (Jumeau et al. 2020)

2.2 Dataset

The data used in this work will be the product of a measuring of the radiation produced by the earlier mentioned Elekta MV 6 over a water mannequin, a quite common method when it comes to radiation procedures (Lutz and Larsen 1984; Benites R, Vega C, and Velazquez F 2012; Gonzalez et al. 2015; Tessonnier et al. 2017) . The result of these measurings is a total of 96 text files, each one of these correspond to a different combination of height, area and presence (or not) of a scattering part. We can see an example of one of these profiles on Figure 2. Text files shape are as shown on table 1.

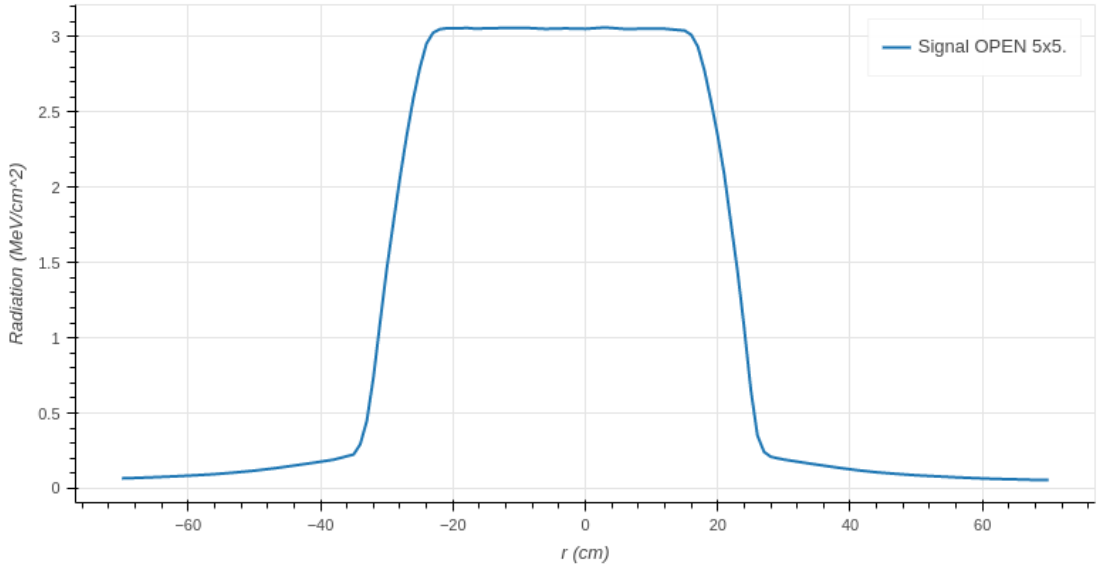

Figure 2: 5x5 open profile on y axis, z=100cm

	r	señal open 5x5	error open 5x5
0	-70.0	0.066602	0.31080
1	-68.0	0.069431	0.31110
2	-65.0	0.074360	0.31065
3	-62.0	0.080116	0.31080
4	-59.0	0.086831	0.31095
...	...	...	...
70	59.0	0.067845	0.31395
71	62.0	0.063037	0.31410
72	65.0	0.060057	0.31470
73	68.0	0.056351	0.31410
74	70.0	0.055396	0.31410

Table 1: Example of 5x5 profile txt

The files in this dataset comprise of all these possible combinations:

- **Area**: 5x5, 5x40, 10x10, 10x40, 20x20, 40x5, 40x10, 40x40
- **Z**: 85, 100, 115
- **FFF (scatter)**: FFF, OPEN

2.3 Goals

So, this study was focused on obtaining a Virtual Source Model for this linac based on the dataset we do have. Previous models (Benites R, Vega C, and Velazquez F 2012; Gonzalez et al. 2015; Tessonnier et al. 2017; W. González, A. M. Lallena, and Alfonso 2013; Yuan, Rong, and Chen 2015; Gonzalez 2015; Wilfredo González, Anguiano, and Antonio M. Lallena 2017) have been centered on obtaining an analytic equation that is able to mathematically define the reality. They rely on MonteCarlo or similar methods to obtain coefficients for the equation, however, this may be not the fastest nor the most effective method to obtain these results. With the current expansion of Artificial Intelligence and bayesian statistics, it may be interesting to look up those methods to solve this problem. Our objective however will be mixed. On one hand, we will obtain an analytic equation that defines our problem, obtaining the needed coefficients via MonteCarlo.

Once we have solved that part, we will look to improve the current result of the analytic equation feeding that data as input to several Neural Networks, which will try to create a prediction. Using Neural Networks in medical physics is not something new, as it is widely used in cancer recognition (Bottaci et al. 1997; N et al. 2010; Shen et al. 2019). However, there are not many examples of its use on dosimetry (Götz et al. 2020; Jiang et al. 2020; Lee et al. 2019) and most of them are quite new. As said on Lee et al. 2019, "The direct Monte Carlo simulation is considered as a state-of-art voxel-based dosimetry technique; however, it incurs an excessive computational cost and time. To overcome the limitations of the direct Monte Carlo approach, we propose using a deep convolutional neural network (CNN) for the voxel dose prediction". However, we mix both methods, trying to obtain best from both worlds.

3 Monte Carlo approximation

Following Fippel et al. 2003 and Gonzalez et al. 2015 VSMs, we develop one for this linac.
We can describe it as a two part equation, where the first leg of the equation corresponds to the main collimator and the second leg to the scattering.

$$\Phi_\gamma(x,y,z) = w_0\Phi_0(x,y,z)\Phi^\gamma_{horn}(x,y,z) + (1-w_0)\Phi_s(x,y,z) \tag{3.1}$$

Being more concise, w_0 is the fraction of the contribution from the primary (and therefore, 1 minus w_0 is the contribution from the scatter source), a normalized value. Then, Φ_0 and Φ_s are the fluence equations for both collimator and scatter source. Finally, Φ_{horn} is an equation that tries to define the central depression that we can see on the mayority of profiles. An example of this kind of profile can be seen on Figure 3.

3.1 Photon Primary Source Fluence

Fluence equations are defined as it follows:

$$\Phi_\alpha = Z(z; z^x_D, z^y_D, z_\alpha)T_\alpha(x^+_\alpha, x^-_\alpha)T_\alpha(y^+_\alpha, y^-_\alpha) \tag{3.2}$$

Out of these variables, Z (Eq 3.3) defines the reduction of the fluence based on the distance from the target to the source, while T defines the area.

$$Z(z; z^x_D, z^y_D, z_\alpha) = \frac{1}{4}\frac{(z^x_D - z_\alpha)(z^y_D - z_\alpha)}{(z - z_\alpha)^2} \tag{3.3}$$

Now, T refer to x or y, depending on what axis are we on, and it is defined by Pearson VII equation (3.4). This Pearson VII equation with X and Y as subject, with $\frac{t^\pm_0}{\delta_0}$ being v, as shown on equation (3.5). So, X and Y would be defined by equations (3.6) and (3.7).

$$Q_0(v) = \frac{v}{\sqrt{1+v^2}} \tag{3.4}$$

$$T_0(\frac{t^\pm_0}{\delta_0}) = \frac{\frac{t^\pm_0}{\delta_0}}{\sqrt{1+(\frac{t^\pm_0}{\delta_0})^2}} \tag{3.5}$$

$$X_0 = \frac{\frac{x^+_0}{\delta_0}}{\sqrt{1+\frac{x^+_0}{\delta_0}}} + \frac{\frac{x^-_0}{\delta_0}}{\sqrt{1+\frac{x^-_0}{\delta_0}}} \tag{3.6}$$

$$Y_0 = \frac{\frac{y^+_0}{\delta_0}}{\sqrt{1+\frac{y^+_0}{\delta_0}}} + \frac{\frac{y^-_0}{\delta_0}}{\sqrt{1+\frac{y^-_0}{\delta_0}}} \tag{3.7}$$

Variables on this equation need some explanation as well. x^+_0, x^-_0, y^+_0, y^-_0 are the outer bounds that delimit the collimation area. They are defined in equation (3.8).

$$t^\pm_\alpha = min[\frac{w^t_I z^t_U(z-z_0) \pm 2t * z_I(z_0 - z^t_U)}{2z_I(z-z^t_U)}, \frac{w^t_I z^t_D(z-z_0) \pm 2xz_I(z_0 - z^t_D)}{2z_I(z-z^t_D)}] \tag{3.8}$$

Now, we have finished defining new variables and these are actual data, being:

- w_I^t: Field size on the 'T' direction.
- z_U^t: and z_D^t: "Up" and "Down" limits of the collimation system for x and y directions.
- z: Position of the target on the z axis
- z_0: Position of the main fluence source
- t: Coordinate on x or y direction
- z_I: Z coordinate of the isocenter, which in this case is always 100.

Thereby, the photon primary source is defined in equation 3.9 (without incorporating the last leg which correspond to the central depression correction):

$$\Phi_0 = \frac{1}{4}\left(\frac{(z_D^x - z_\alpha)(z_D^y - z_\alpha)}{(z - z_\alpha)^2}\right) * \left(\frac{\frac{x_0^+}{\delta_0}}{\sqrt{1+\frac{x_0^+}{\delta_0}}} + \frac{\frac{x_0^-}{\delta_0}}{\sqrt{1+\frac{x_0^-}{\delta_0}}}\right) * \left(\frac{\frac{y_0^+}{\delta_0}}{\sqrt{1+\frac{y_0^+}{\delta_0}}} + \frac{\frac{y_0^-}{\delta_0}}{\sqrt{1+\frac{y_0^-}{\delta_0}}}\right) \tag{3.9}$$

However, we have not defined δ_0. This is one of the coefficients we have to find. This particular coefficient defines how abrupt is the decay from the actual fluence area to the non-fluence area.

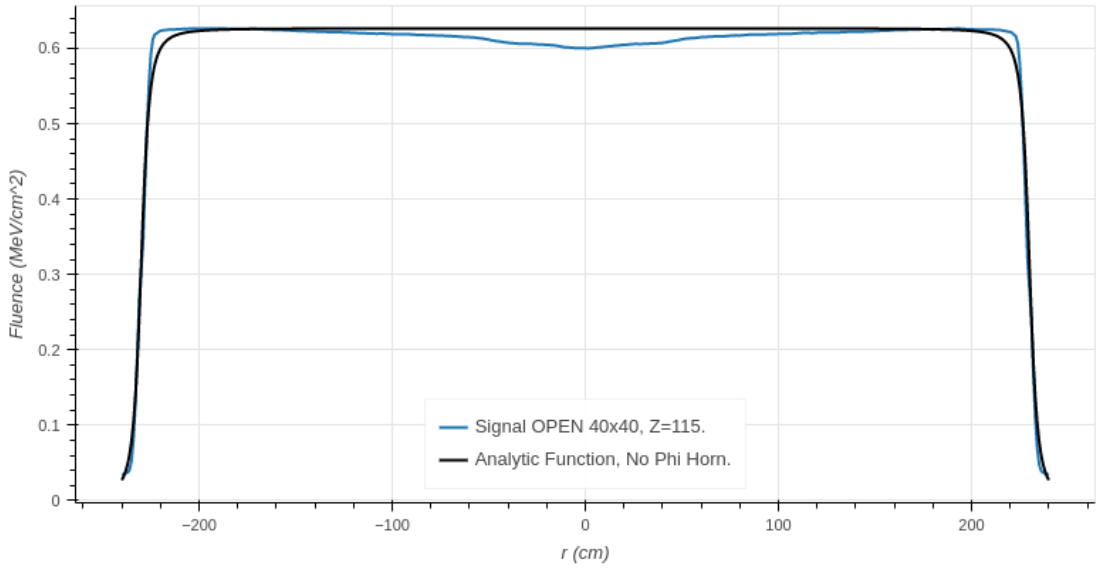

Figure 3: 'x' axis profile for 40x40, Z=115. Representation of the first leg of the equation vs reality using an already defined δ_0 value

3.2 Central depression correction

Since most profiles present a depression surrounding the center of the area, this is an added correction to the main fluence equation.

$$\Phi_{horn}(x, y, z) = 1 + \rho^2 \sum_{k=0}^{4} h_k \rho^k \tag{3.10}$$

Where h_k are coefficients we need to obtain and ρ is a measure of the distance covered by the radiation:

$$\rho = \frac{\sqrt{x^2 + y^2}}{z - z_0} \tag{3.11}$$

The so-called central depression can be seen on Figure 4.

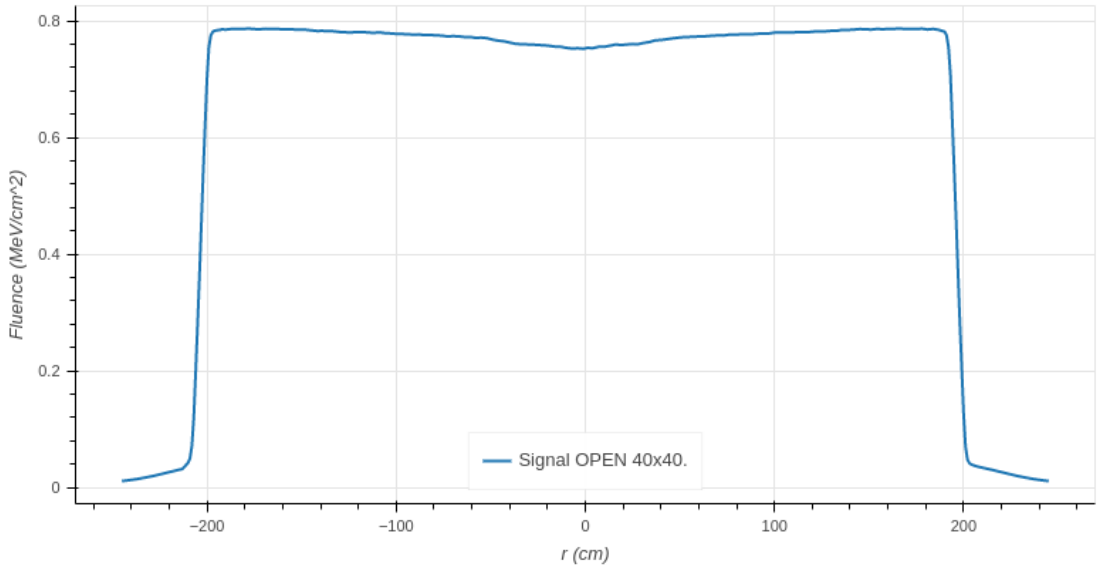

Figure 4: 'x' axis profile on the 40x40, Z=100 example. Central depression is more clear the larger the area is.

Now, in cases where we only have the main fluence source ($w_0 = 1$), our equation would be defined as in equation (3.12). First leg of the equation needs one coefficient, which is easily obtained, however, for the central depression correction we need 5 coefficients which is a more difficult task.

$$\Phi_\gamma = \frac{1}{4}\left(\frac{(z_D^x - z_\alpha)(z_D^y - z_\alpha)}{(z - z_\alpha)^2}\right) * \left(\frac{\frac{x_0^+}{\delta_0}}{\sqrt{1 + \frac{x_0^+}{\delta_0}}} + \frac{\frac{x_0^-}{\delta_0}}{\sqrt{1 + \frac{x_0^-}{\delta_0}}}\right) * \left(\frac{\frac{y_0^+}{\delta_0}}{\sqrt{1 + \frac{y_0^+}{\delta_0}}} + \frac{\frac{y_0^-}{\delta_0}}{\sqrt{1 + \frac{y_0^-}{\delta_0}}}\right) * \left(1 + \rho^2 \sum_{k=0}^{4} h_k \rho^k\right) \tag{3.12}$$

3.3 Monte Carlo method

Putting it simple, a Monte Carlo simulation (Kroese et al. 2014) is a process where we introduce random numbers as coefficients, obtaining a distribution out of them. Our desired VSM is composed at the second leg of 5 coefficients, but it is an arbitrary decision to stop at 5. We could create a more complex equation if we used more coffiecients, or less, if we used a smaller number of coefficients. As a example of this, you can see how we recreate the results using 3 coefficients (equation (3.13) and Figure 4).

$$\Phi_\gamma = \frac{1}{4}\left(\frac{(z_D^x - z_\alpha)(z_D^y - z_\alpha)}{(z - z_\alpha)^2}\right) * \left(\frac{\frac{x_0^+}{\delta_0}}{\sqrt{1 + \frac{x_0^+}{\delta_0}}} + \frac{\frac{x_0^-}{\delta_0}}{\sqrt{1 + \frac{x_0^-}{\delta_0}}}\right) * \left(\frac{\frac{y_0^+}{\delta_0}}{\sqrt{1 + \frac{y_0^+}{\delta_0}}} + \frac{\frac{y_0^-}{\delta_0}}{\sqrt{1 + \frac{y_0^-}{\delta_0}}}\right) * \left(1 + \rho^2 \sum_{k=0}^{2} h_k \rho^k\right) \tag{3.13}$$

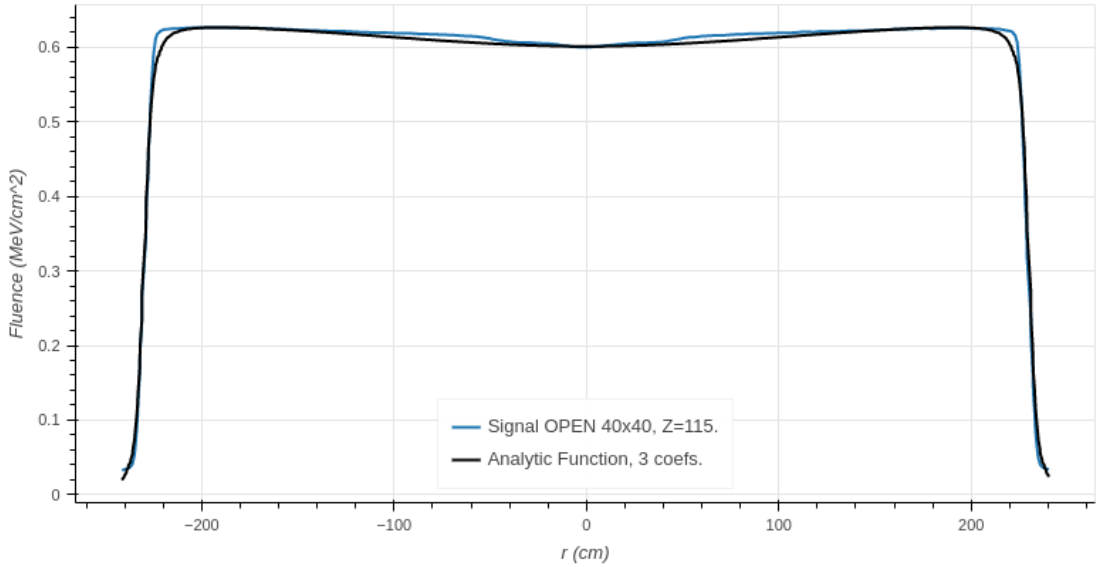

Figure 5: Signal and Analytic Function fitted using a 3 coefficient equation as Φ_{horn}. This is the plot of a case where there is no scatter signal, Z=115 and the area to be covered is 40x40 cm

VSMs on the bibliography present coefficients that are useful for both profiles ('x' and 'y'). However, during the course of this study, I realised that the behaviour of each axis was not the same, and coefficients that were good for 'x' were not so good for 'y', so it was decided to isolate each axis and study different models for each one.

As an example of this, if we use 'x' axis coefficients we obtained after approximately a million iterations on the 'x' axis (Table 2), we obtain a mean absolute error of $0.0142 MeV/cm^2$. If we use the same coefficients for the 'y' axis we obtain an error of $0.0245 MeV/cm^2$. Complete results are showed later, on Section 6.1

δ_0	h_0	h_1	h_2	h_3	h_4
2.659	0.0433	0.0114	-0.0383	0.0129	0.000053

Table 2: Best coefficients obtained for x axis

These are the results obtained at the end of the iterations. However, it was clear from the beginning that profiles for both axis were different, so it was decided to search separately for each axis.
Monte Carlo method followed in this case was built using Python. Mayority of the calculation was run on a laptop (which has an Intel® Core™ i5-8250U CPU @ 1.60GHz × 8, a not very fast CPU in nowadays standards), the development of it was done through several batches, which would last for approximately a week.

3.3.1 Random Monte Carlo search

Initially, the code was meant to create random numbers between two limits for each coefficient. Since δ_0 and the 'h' coefficients are independent between them, in order to speed up the search of ideal coefficients, 'h' coefficients are searched first using an standard value of $\delta_0 = 2$ and once found the best 'h' coefficients we do Monte Carlo again only on δ_0. Coefficients and errors were stored in lists. Each batch usually was composed of 100000 epochs, after which the results would be reviewed. Based on the lowest errors, a new iteration script would be created, adapting the limits of the random number generator for each coefficient. Initial limits were set between 10^{-7} and 10^{-1} for both negative and positive values. Randomization was done between '-7' and '-1' since the logarithmic scale suits better for this problem optimization. Code was similar to what is shown on appendix A. Table 3 contains results that serve as example of the first batch.

Results (as shown in Table 3), show a clear pattern on the h_0 coefficient. Limit was set on 1e-2, and best values surround that limit, so in the next iteration it was ideal to upgrade that limit to 1e-1 and close the bottom limit to 1-3. The other coefficients are not that clear, besides you can see that values for h and k tend to be quite low (all well below 1e-3). Color maps shown on Figures 6-9 confront h_0 (which has the easiest pattern) with the other 4 coefficients, allowing us to visually check patterns. Table show as that boundaries on 'l' can been stretched a lot, while the other ones are still quite doubtful. Next batch of iterations would start on h_0 limits between 1e-2 and 1e-3. Visually, we can see clear patterns on the graphics aswell, which help us to short the boundaries on the other coefficients, rather than letting the Monte Carlo method go on forever.

Mean Absolute Error	l	k	j	h	g
0.067715	0.009776	1.013672e-03	5.200028e-04	-8.364561e-07	1.674677e-06
0.067739	0.009648	4.834647e-03	2.763648e-05	-5.879016e-04	-1.031359e-04
0.067758	0.008969	2.270703e-03	-1.487425e-07	1.349939e-04	3.512052e-05
0.067759	0.009235	-7.479714e-06	1.295526e-03	-2.128316e-07	-6.476427e-06
0.067791	0.007948	3.669876e-03	-9.849719e-04	2.654817e-07	1.495117e-05
0.067799	0.009047	-2.627768e-04	3.961705e-04	3.323590e-04	5.140993e-06
0.067803	0.009623	-2.660140e-03	2.280633e-03	9.482199e-06	-5.843446e-06
0.067807	0.007860	5.550067e-07	-1.400713e-05	8.618755e-07	2.529700e-04
0.067811	0.007967	-5.448018e-06	2.537317e-03	-5.988936e-04	-2.827558e-07
0.067822	0.009885	-1.369494e-06	1.699570e-03	1.661274e-06	-4.148678e-06

Table 3: Random search of coefficients, results after the first batch of iterations on the 'x' axis

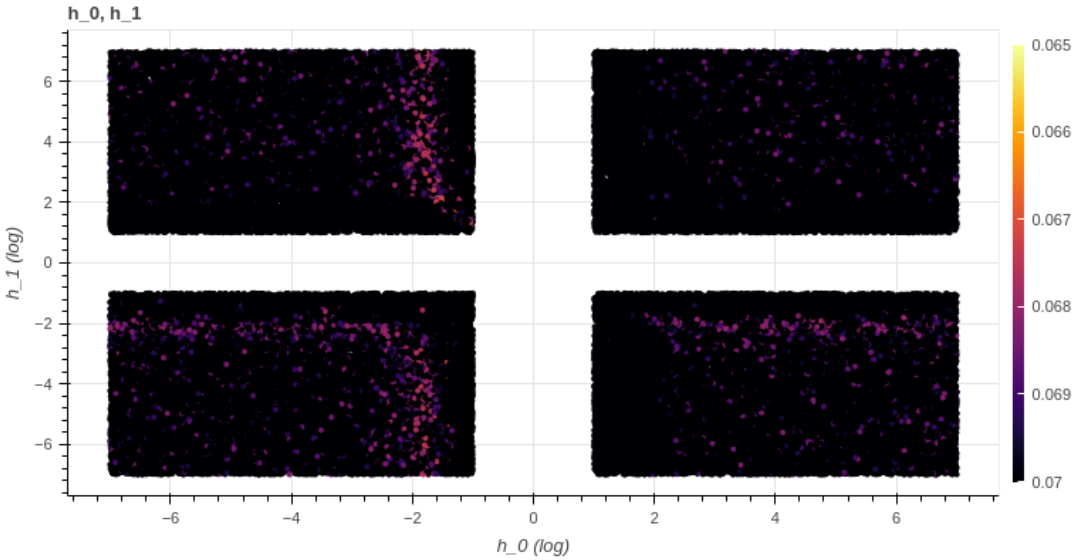

Figure 6: h_0 vs h_1

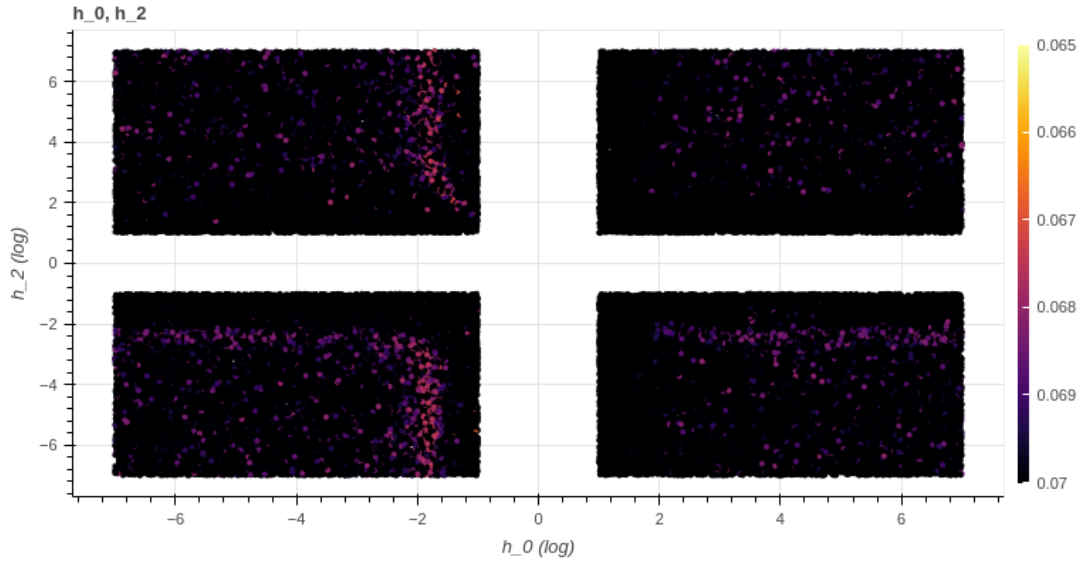

Figure 7: h_0 vs h_2

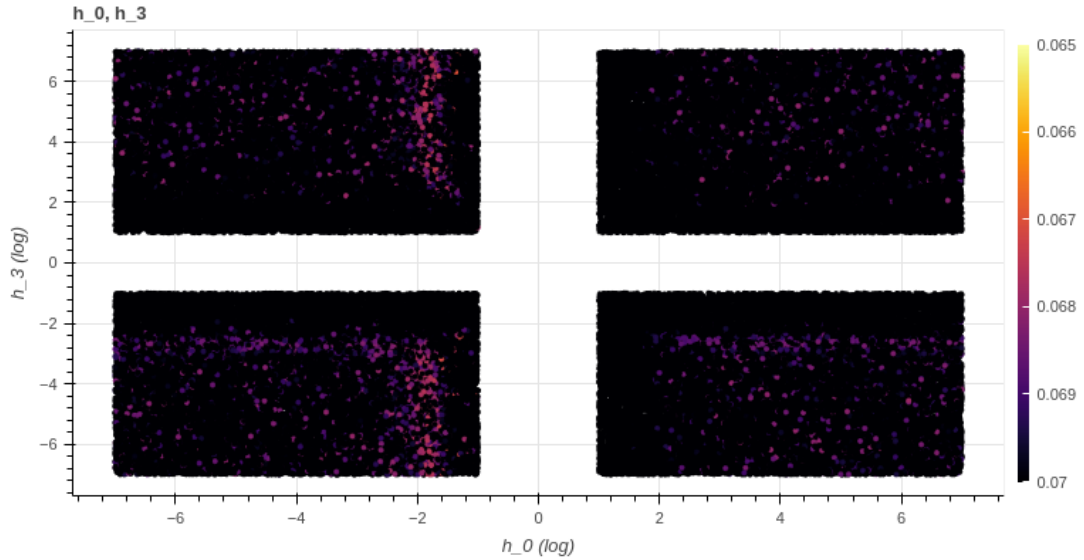

Figure 8: h_0 vs h_3

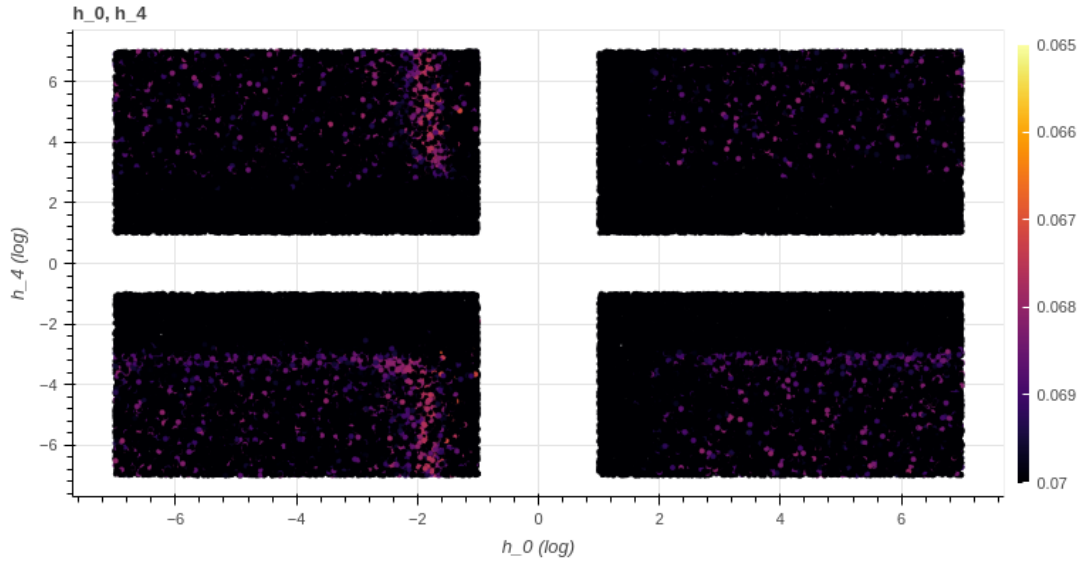

Figure 9: h_0 vs h_4

3.3.2 Non randomized Monte Carlo method

Besides the pure Monte Carlo method is meant to be totally random, the presence of 5 coefficients, the selection of python as engine (high-level, interpreted language, easy to use but not very efficient resource-wise) and lack of powerful hardware makes it difficult to achieve a great result using pure randomness. So, it was decided to create a set of given variables (10 possibilities, 5 negative and 5 positives, growing on a logarithmic way from 10^{-7} to 10^{-1}) and try all the possible combinations between them. This method allowed us to obtain more clear results in a faster way. Table 4 shows results for first batch of iterations on x axis using non randomized way (and their coefficients). On Figure 10 we can see a comparative error distributions for first iteration (100000 trials) for both methods. Both methods achieve quite similar results, with the randomized one obtaining slightly better error on its peak, but the non randomized method offers an easier look at what are the best combinations. Since we are not looking for the best overall result (since this is the first iteration of many to come), the objective of the iterations at this point is to set the next limits. This s better achieved through the non randomized method, which is followed for the rest of the study.

Error	h_0	h_1	h_2	h_3	h_4
0.068006	0.003162	3.162278e-03	1.000000e-04	0.0001	0.000100
0.068040	0.003162	3.162278e-03	3.162278e-06	0.0001	0.000100
0.068042	0.003162	3.162278e-03	1.000000e-07	0.0001	0.000100
0.068042	0.003162	3.162278e-03	-1.000000e-07	0.0001	0.000100
0.068043	0.003162	3.162278e-03	-3.162278e-06	0.0001	0.000100
0.068063	0.003162	3.162278e-06	3.162278e-03	-0.0001	-0.000003
0.068063	0.003162	1.000000e-07	3.162278e-03	-0.0001	-0.000003
0.068063	0.003162	-1.000000e-07	3.162278e-03	-0.0001	-0.000003
0.068063	0.003162	-3.162278e-06	3.162278e-03	-0.0001	-0.000003
0.068064	0.003162	1.000000e-04	3.162278e-03	-0.0001	-0.000003

Table 4: Best results obtained on the non-randomized Monte Carlo

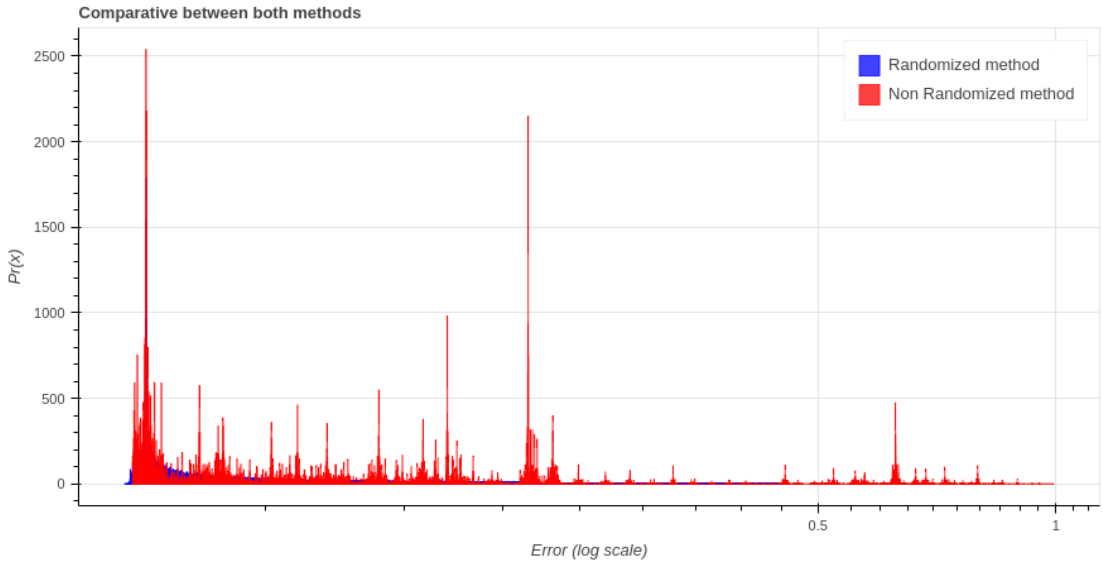

Figure 10: Histograms showcasing error for both methods

4 Artificial Intelligence

4.1 Introduction

Science by definition is a way of mathematizing reality, the study and work that tries to represent reality in an organized and predictable way. However, problems on reality are not isolated like they are on laboratories and the attempts to predict in a scientific way any kind of 'open' problem is just an approximation. This is esentially the case here. Besides the trials are made on a laboratory, there are variables that we cannot take into account. In this case, we have already seen that profiles on the x axis are not exactly the same than profiles on the y axis, and there are some drops in different zones. Those are problems that come with the machine we are using, and they will replicate, but obtaining analytical approximations that define these problems is a kind of study that would take tremenduous work for not so much reward. That is a great point for the use of Artificial Intelligence on Science, it does not substitute the equations, they simply work as a way of smoothing and approximating results when the work needed to achieve a significant improvement is not worth, which is the main part of this study.

Artificial Intelligence (AI from now on), is a broad concept that unites several fields. Monte Carlo method for example is also defined as AI, since usually, AI is defined as a computerized way of applying statistics and maths on a automatized way. AI development in the last 50 years has gone through 3 eras (Ongsulee 2017; Bini 2018):

- Artificial Intelligence: Primitive AI, automatized statistics and maths, Monte Carlo method is a classic example of this field. However, following development is still included as AI.
- Machine Learning: First proper development of what AI is usually understood as. Examples of this field are Tree Decision algorithms (Palmer and Schwenk 1979; Batra and Jawa 1975) and Gradient Boosters (Freund and Schapire 1997).
- Deep Learning: Neural Networks on all its variaties

On the previous section we have defined what Monte Carlo method is, so there is no need for further introduction. However, Machine Learning and Deep Learning will be introduced as they were both used for this project.

4.2 Machine Learning

There was no doubt of the utilities of basic AI, so, the first steps after classic AI were moved towards developing automatized AI that needed less information from the user, keeping the iterative algorithm that tries to solve problems. There came Random Forests (decision trees algorithms) and Gradient Boosters. For classic AI, like Monte Carlo, we define an equation and try different coefficients. There are different ways to develop it, using random sets of data, randomizing only per coefficient, using given sets of data, etc. but in all cases you need a defined equation. First big difference with machine learning (ML from now on): it does not need an equation. ML needs a 'x' array and 'y' array. 'x' array correspond to the number of variables we have, and 'y' is the result we are trying to achieve (Equations 4.1, 4.2). Sets of data are usually divided in two, a training dataset from which the algoritm tries to learn the correlations, and then it is validated on a test/validation dataset. If we tried to obtain a result out of machine learning for our case, we could use all 'x' variables we have for each 'y', without producing a proper equation earlier. In our case, each single point on the dataset is randomly assigned to either training or validation (on a 75-25% proportion).

$$\begin{pmatrix} x_0^0 \\ x_1^0 \\ x_2^0 \\ ... \\ x_n^0 \end{pmatrix} \rightarrow y^0$$

(4.1)

Machine learning work iterating through multiple samples, until it obtains the combination of operations and coefficients that approach the most to the desired results.

$$\begin{pmatrix} x_0^0 & ... & x_0^t \\ ... & ... & ... \\ x_n^0 & ... & x_n^t \end{pmatrix} \rightarrow \begin{pmatrix} y^0 & ... & y^t \end{pmatrix}$$

(4.2)

Here, ML algorithms (Figures 11, 12), try to obtain a correlation between those variables and fluence. However, it does not work out great. ML presents a problem, it work looking for simple correlations. In other words, they do not understand arrays as a whole cohesive set of data, but single points. In our case, this is key, fluence for each single point is not defined individually, we are looking to obtain the complete figure of the fluence profile, and using singular data is not enough, since each point has an structural relevance.

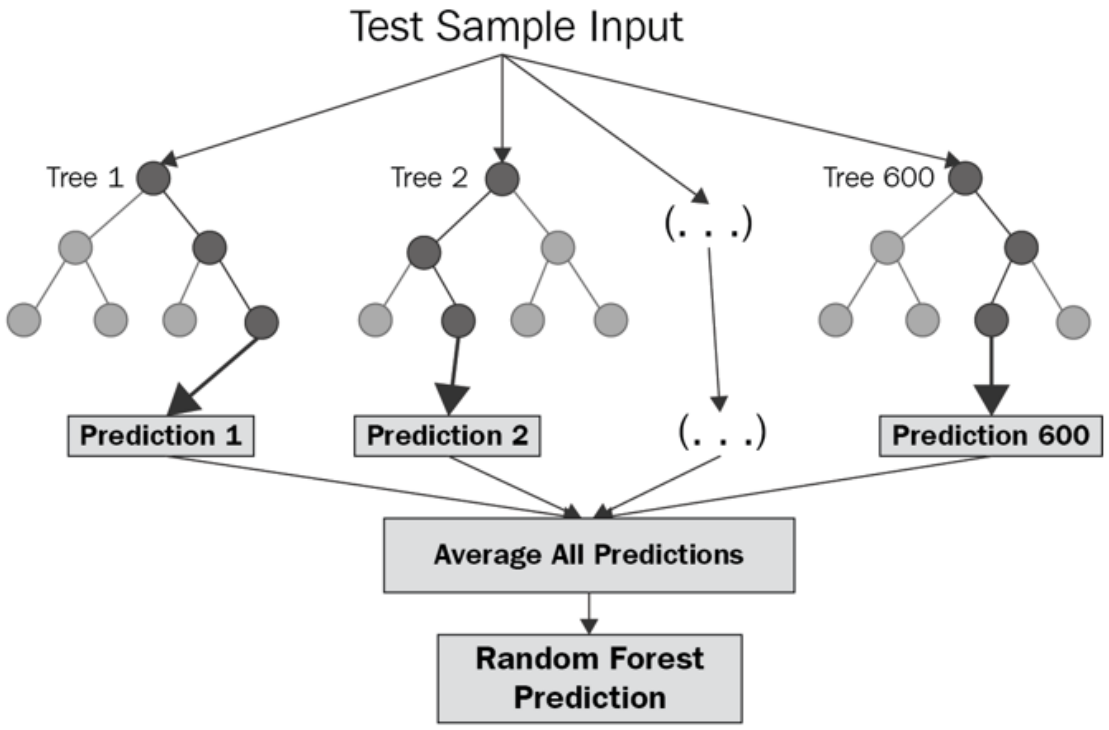

Figure 11: Random Forest schema. It works as a decision tree, averaging results from every tree at the end of the process

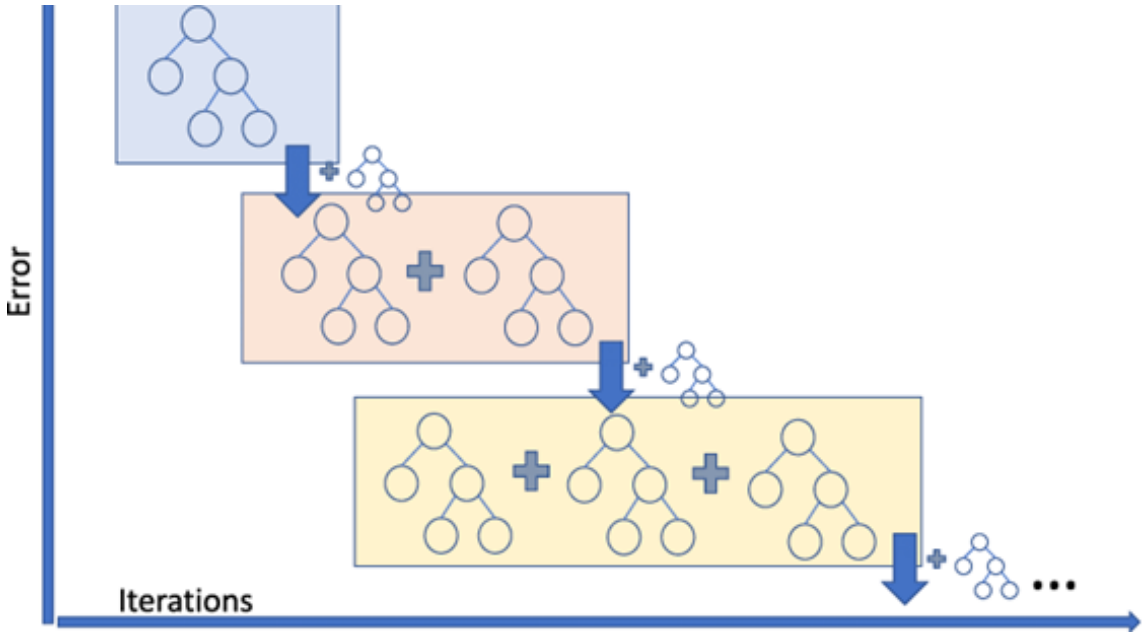

Figure 12: Gradient Boosting schema. It works similar to a random forest, but instead of averaging results at the end, each tree is made out of a previous one for each iteration, this way doing a 'real time' average

4.3 Deep Learning

So, latest installment of AI is Deep Learning (DL). DL adresses previously mentioned problems, they are able to understand complete arrays as data, using data in more than 2 dimensions iterating until it obtaines the best possible result. The biggest improvement on this line of work can be seen easily on matters like image detection. ML works on flattened 2-dimension arrays. So, in order to apply this to image classification, how could you work it out? You would have to convert 3-dimension arrays (height, width, number of channels hannels) to a 1-dimension array, stacking height, width and number of channels (if an image is RGB, is composed of 3 different channels), obtaning a number of variables (height*width*number_channels) = (result). Each pixel would be a variable. If we converted that array back to an image, we would have an image of height 1 and width equal to the total number of pixels there is in the image. This may be hard to comprehend but on figures 13 and 14 (P. Sharma 2019) the example is self-explanatory.

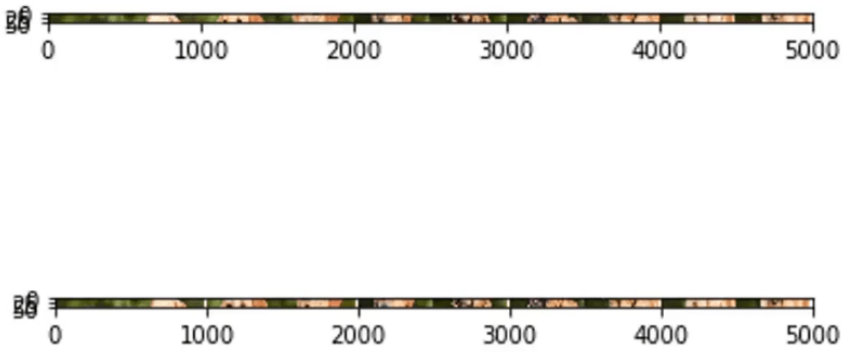

Figure 13: These are two images of dogs, flattened arrays as ones that a ML algorithm can read, one of them is true and the other has been modified

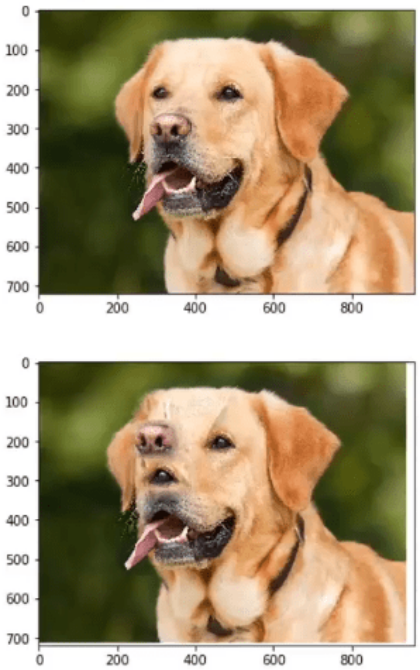

Figure 14: These are the same pictures showed on Figure 12. A simple demostration of how important the spatial configuration of an array may be

However, DL solves that problem and allow us to feed algorithms using other kind of dimensions. Probably the two most used Neural Network types are 'LSTM' (Long Short Term Memory) Neural Networks (Greff et al. 2015) and CNN (Convolutional Neural Networks) (O'Shea and Nash 2015). LSTMs are ideal for sequencial timeline-related data, they are fed with 3-dimension arrays, where each point is a time step. These kind of models are designed to learn that time steps are correlated, and, variables effect to a result on a time 't' are more related to how variables affected to a result on a time 't-1' than how variables affected to a result on 't-500'.
CNNs are ideal to image, video, or audio classification, because they understand arrays as a whole. A visual example: if we convert a set of videos to arrays we obtain a 5-dimensional array (number of samples, height, width, number of channels, time). The coherence between each one of those dimensions is key, and it is something not possible to achieve using ML or classic AI.
CNNs have 3 main types:

- Conv1D: 1 added dimension to a simple array. Examples: Audio signals (3 dimensions)
- Conv2D: 2 added dimensions to a simple array. Examples: Pictures (4 dimensions)
- Conv3D: 3 added dimensions to a simple array. Examples: Video (5 dimensions)

Our problem is not a video , but we could compare it with an audio signal. Profiles have much in common between them and some variables make them change, but out of data exploration we can see easily that they should be understood as a whole. We can adress that kind of problem with Conv1D NN. Our data may be presented as shown on equation 4.3 (assuming n varibles, m samples and a width for each profile that goes from -r to r).

$$\begin{pmatrix} \begin{pmatrix} {x_0}^0_{,-r} & \dots & {x_0}^n_{,-r} \\ \dots & \dots & \dots \\ {x_0}^0_{,r} & \dots & {x_0}^n_{,r} \end{pmatrix} \\ \begin{pmatrix} {x_1}^0_{,-r} & \dots & {x_1}^n_{,-r} \\ \dots & \dots & \dots \\ {x_1}^0_{,r} & \dots & {x_1}^n_{,r} \end{pmatrix} \\ \dots \\ \begin{pmatrix} {x_m}^0_{,-r} & \dots & {x_m}^n_{,-r} \\ \dots & \dots & \dots \\ {x_m}^0_{,r} & \dots & {x_m}^n_{,r} \end{pmatrix} \end{pmatrix} \rightarrow \begin{pmatrix} \begin{pmatrix} y_{0,-r} \\ \dots \\ y_{0,r} \end{pmatrix} \\ \begin{pmatrix} y_{1,-r} \\ \dots \\ y_{1,r} \end{pmatrix} \\ \dots \\ \begin{pmatrix} y_{m,-r} \\ \dots \\ y_{m,r} \end{pmatrix} \end{pmatrix} \tag{4.3}$$

Here, another small problem appear for us. Data fed to the Neural Networks must have always the same shape, something that it does not happen in our case, since data for each profile only cover the parts of the data where there is significant signal, plus a few centimeters more. Therefore, profiles on the 5x5 area have way less 'width' than the 40x40 profiles. To solve that problem, we 'resample' data, obtaining the mean value for each point (this resample is done obtaining data for each 0.5cm). If there is no value surrounding that point, we interpolate the data through a polynomial equation, and limits of the data are set as the minimun value obtained.

5 Modelization

5.1 Settings

So, once we know that we are using Conv1D Neural Networks, we need to create that model. While ML or AI do not need much configuration, with the complexity of Neural Networks comes a way higher amount of configuration possibilites. Some of them may be known from theoretical knowledge, but many of them need to be tried before discarded. Neural Networks may be created manually, since they are basically a iterator that uses certain activation functions, but it is easier to use predefined frameworks created from the main software companies of the world. Both Meta and Google have their own branches of DL. Tensorflow (Google) (Abadi et al. 2016) is the most broadly used DL framework nowadays and will be the one that we will be using on this project.

Base CNN model is composed of a number of entry convolutional layers that usually increase periodically the number of filters they are applying, which then get translated to 'Dense' (also called 'MLP', Multi Layer Perceptron) layers, which are a type of layer that work similar to ML. We will use 3 Convolutional layers linked to 3 Dense layers. The list of settings (which will be further explained later) to configure is:

- Number of filters on Convolutional layers
- Kernel size on Convolutional layers
- Number of neurons on layer 4 (Dense)
- Number of neurons on layer 6 (Dense)
- Activation function on layer 1 (Conv1D)
- Activation function on layer 2 (Conv1D)
- Activation function on layer 3 (Conv1D)
- Activation function on layer 4 (Dense)
- Activation function on layer 5 (Dense)
- Activation function on layer 6 (Dense)
- Dropout
- Optimizer
- Loss function

Iterator code is shown on appendix C and will be further explained now.

5.2 Hyperparameter tuner

In order to obtain the best possible configuration, we use a tool called keras tuner (O'Malley et al. 2019). Tensorflow is the DL engine used, while keras is the framework that works as frontend for Tensorflow, both developed by Google. Keras tuner is an iterator that may go through 3 methods (Random Search, Bayesian Optimization or Hyperband) to try different combination of the selected hyperparameters (the previously mentioned settings to configure).

- Random Search: a pure random search that combines selected settings in any possible way, keeping the best result
- Bayesian Optimization: It starts the iterations using random combinations. Once it gets results out of these combinations, it starts choosing new combinations based on the previous results, discarding the settings that did not work earlier, iterating until the chosen maximun number of trials.

- Hyperband (Li et al. 2018): This model tries several predefined configurations based on previous studies, iterating over early stops. On other words, if you want a certain Neural Network to iterate for 60 epochs, this iterator model will start iterating over just 2 or 4 epochs, comparing different configurations and developing further only the ones that present good results. This method is an order of magnitude faster than Bayesian Optimization, obtaining similar results, so it was the used method for this study.

5.3 Hyperparameters

Filters

Dimension space of our data after it has gone through the convolutional layer.

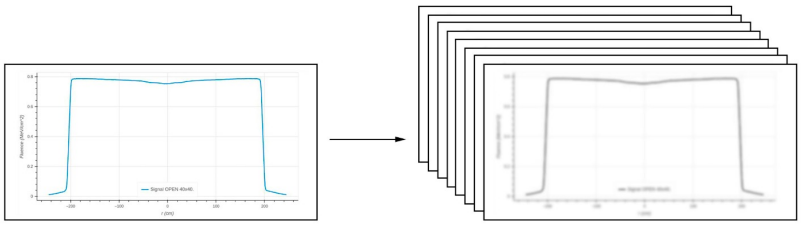

Figure 15: Schematic representation of convolutional network layers. In this case we would have 8 filters, which is a pretty common number, normally, convolutional layers are stacked in geometric progressions of 2.

Kernel

Convolutional layers go through the data learning in batches. When we are using one-dimensional data, kernels have one dimension as well

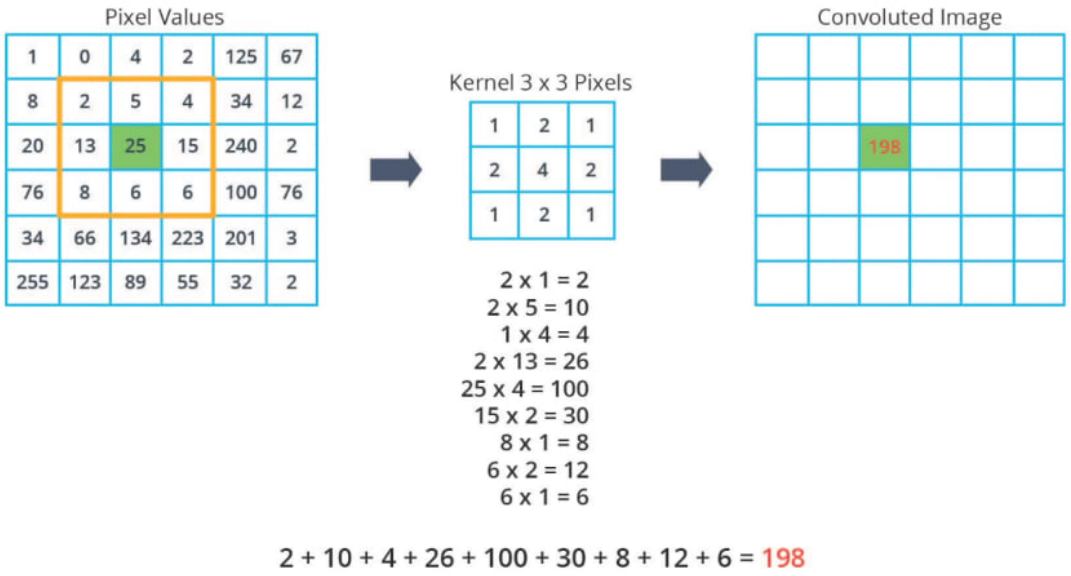

Figure 16: Besides we are using 1D kernels, it is easier to understand examples with 2D kernels like the one on this figure

Activation function

Non linear functions that allow linear information processed from one layer proceed to the next layer (Sagar Sharma, Simone Sharma, and Athaiya 2017). On regression problems like the ones we are facing, 'relu' and its derivates are the best options, so we kept 3 possibilites: Relu (5.1), elu (5.2) and selu (5.3). On the mid layers we use those 3 layesr, while on the output layer we may use 'relu' or 'linear' (which means there is no activation function and we obtain a linear result as result of the last layer).

$$\begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases} \quad (5.1) \qquad \begin{cases} \alpha(e^x - 1) & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases} \quad (5.2) \qquad \lambda \begin{cases} \alpha(e^x - 1) & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases} \quad (5.3)$$

With $\lambda = 1.0507$ and $\alpha = 1.67326$ for Selu and α to be found on Elu.

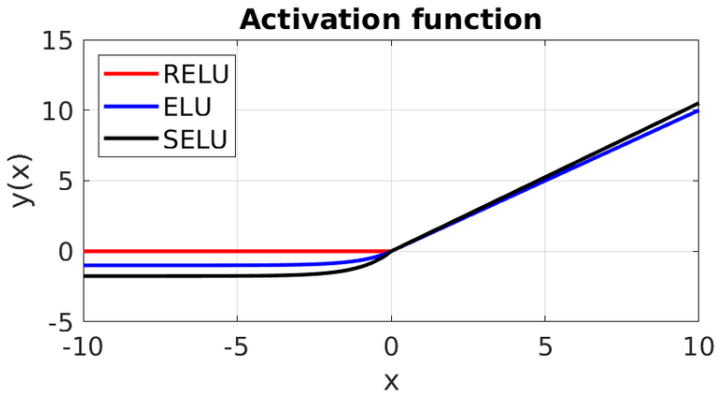

Figure 17: Graphic representation of our activation functions

Neurons

Number of neurons on a layer, similar concept than filters, but in this case the spatial dimension is diferent, and they are named this way for 'Dense' (MLP) layers.

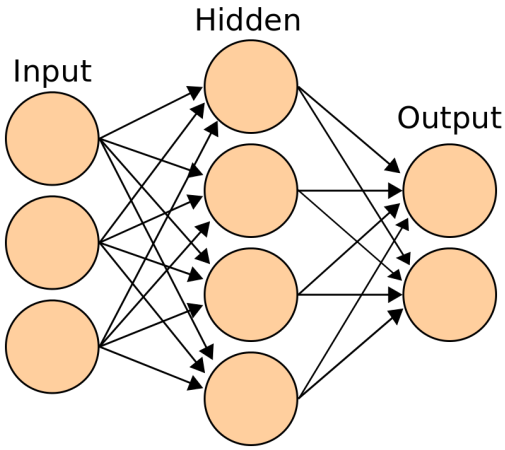

Figure 18: Really basic scheme of a NN. Input layer is defined by the dimension of our data, hidden layer is the actual number of neurons we have to define and output is the dimension we want as output (in our case, 1)

Dropout

Percentage of the neurons not used in each epoch of the model. This is made to avoid overfitting, this way neurons will learn assimetrically in a temporal matter, so the results don't converge so fast and converge from different angles.

Optimizer

The type of algorithm used to evolve the weights ('coefficients') through every epoch. We try 3 different types on this study, RMSprop (Tieleman, Hinton, et al. 2012), Adam (algorithm obtained as a kind of combination of RMSprop and Adagrad) (Duchi, Hazan, and Singer 2011) and Adamax (which is basically an Adam modification using infinite 'p') (Kingma and Ba 2014). Optimizers look to minimize error, reaching a global minimum. A classic problem on Deep Learning is setting the learning rate in a proper way to not get stuck on a local minimum, that's why these optimizers and their evolving learning rate (all these 3 have dynamic learning rate) are so important. We can picture this on Figure 19 (Amini et al. 2018).

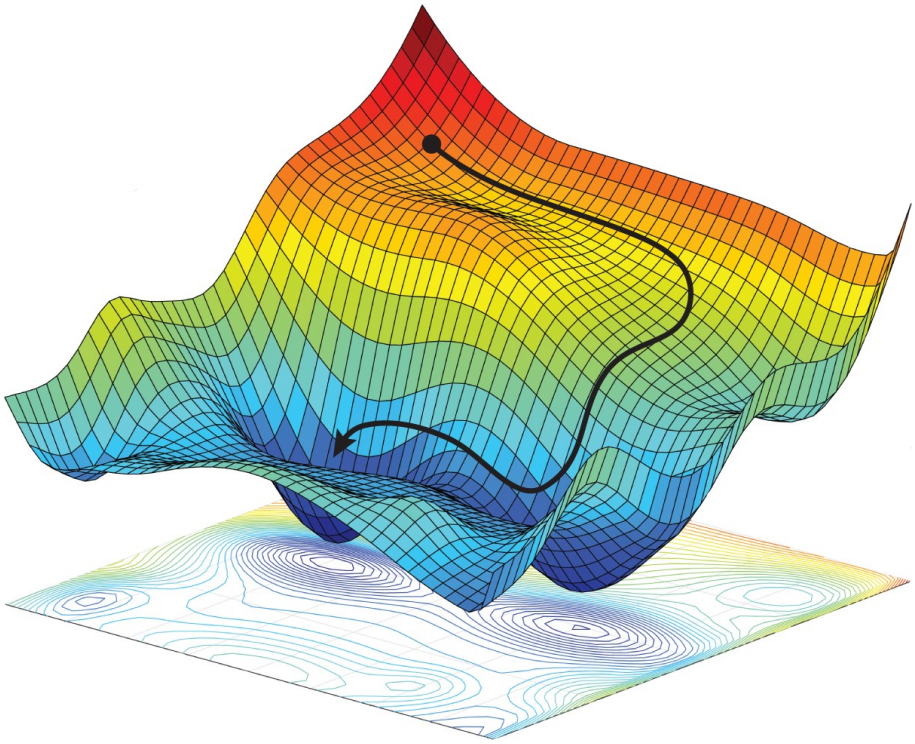

Figure 19: Schematic figure of the evolution of Neural Networks results. Picture tries to recreate the evolution of weights (black arrow) going back and forth. Color and height represent loss measure, we can see how the Neural Network finds a local minimum. However, keeps searching and after increasing error for while finds the global minimum.

- Adam

Figure 20: Caption

- Adamax
- RMSprop

Loss function

Neural Networks iterates the previously mentioned algorithms looking to minimize a loss function. Loss functions for regressions cases are usually Mean Absolute Error (MAE) and Mean Squared Error (MSE). In this case we add a function called logarithm of the hyperbolic cosine (log_cosh), which approximates very much to mean squared error but is less affected by outliers, obtaining an overall better results if there are many outliers.

- MAE

$$Loss = \sqrt{(Prediction - y_{test})^2} \tag{5.4}$$

- MSE

$$Loss = (Prediction - y_{test})^2 \tag{5.5}$$

- log(cosh(f(x))

$$Loss = log(\frac{e^x + e^{-x}}{2}) \tag{5.6}$$

As an example of how useful may log(cosh(f(x)) be compared to MAE can be seen on Figure 20.

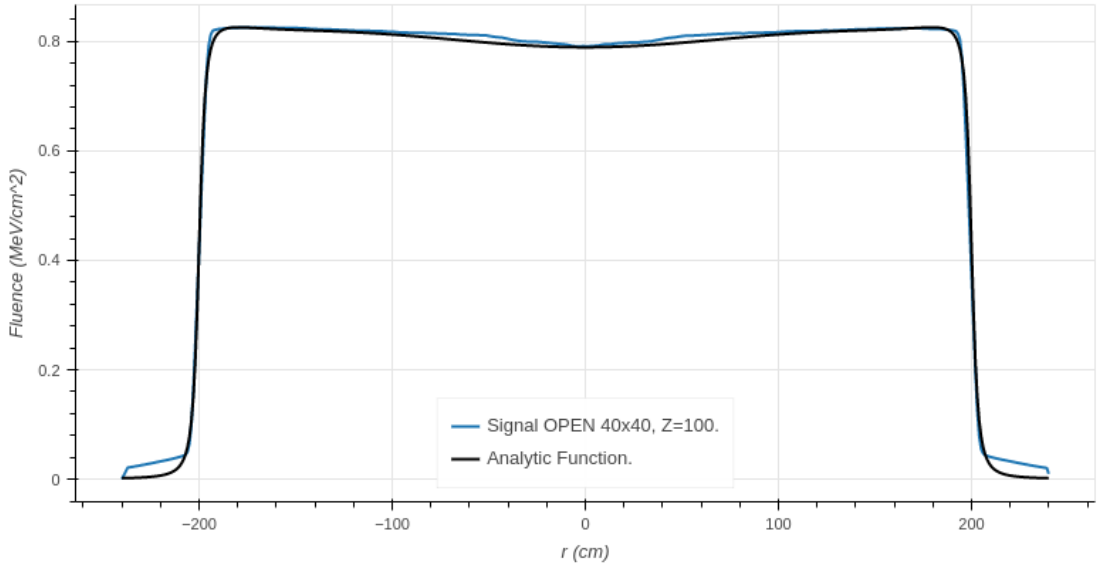

Figure 21: Monte Carlo result in contrast to original data for the 40x40 no scattering x profile, Z=100

We can see that both arms of the profile present big outliers, since the equation turn the result close to 0 after the defined zone is over, while the actual data present that there are some fluence leftovers. We cannot achieve to solve those leftovers using none of our coefficients. 'h' coefficients refer to the central depression, while the δ_0 coefficient refer to how abrupt the slope is, which we can see that are quite well fitted. However, MAE loss will be relatively high due to those outliers, and when the Neural Network tries to solve the problem and minimize the loss, if we used MAE, it might worsen the result on the fluence zone while improving the outliers. Using log(cosh) we can solve this particular problem, improving the overall result while not overfitting on the outliers.

5.4 Preprocessing data

Normally, a neural network problem is composed of several 'x' variables that compose a sort of equation, like we mentioned earlier. On this case the inmediate variable we want to use as input for the neural network is the VSM that we developed through Monte Carlo. There are other variables that may help us (some of them mentioned earlier on subsection 3.1 Photon Primary Source Fluence).

- w_I^t
- z_U^t
- z
- t
- z_I
- ρ

Information obtained through these variables is a bit redundant (for example Z and t are correlated to Rho). To adress that and clean data, we use Principal Component Analysis (PCA) (Tipping and Bishop 1998). This mathematical method comprises data on a lower dimentional space. That is, if you have a 280x6 array, you can transform it into a 280x2 shaped array for example. That is one of the methods, selecting a desired shape. The other method, which we will use, is comprising a % of the information on the data in the lowest shape possible. We use a 99% data conservation for our case.

Showing a visual representation of this process is not easy since representation of more than 3 dimension is not really comprehendable for human perception, but we can show a simpler example with 2 of those variables

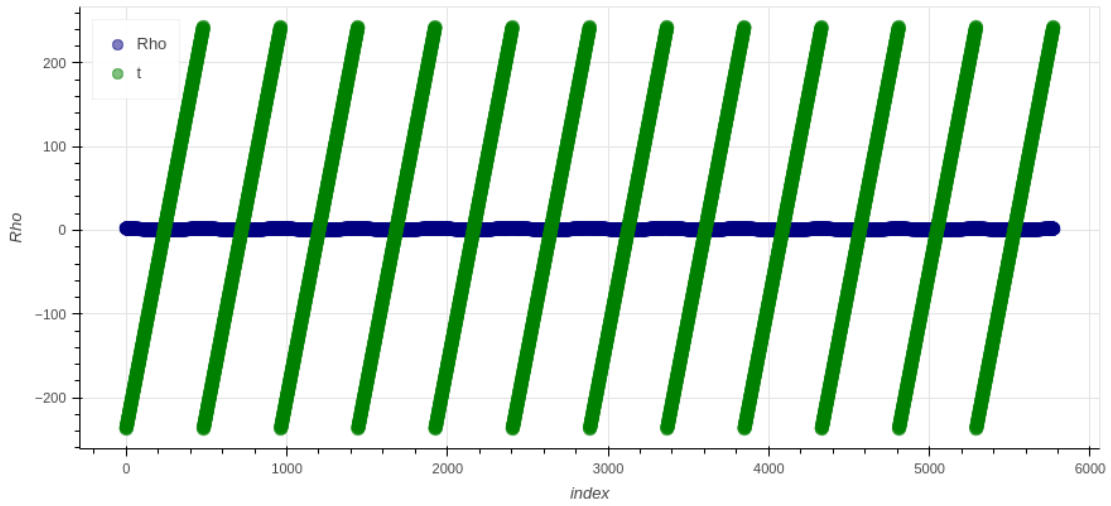

Figure 22: 't' and 'Rho' representation for all our data, horizontally stacked (index represent each one of the values for all singular cases)

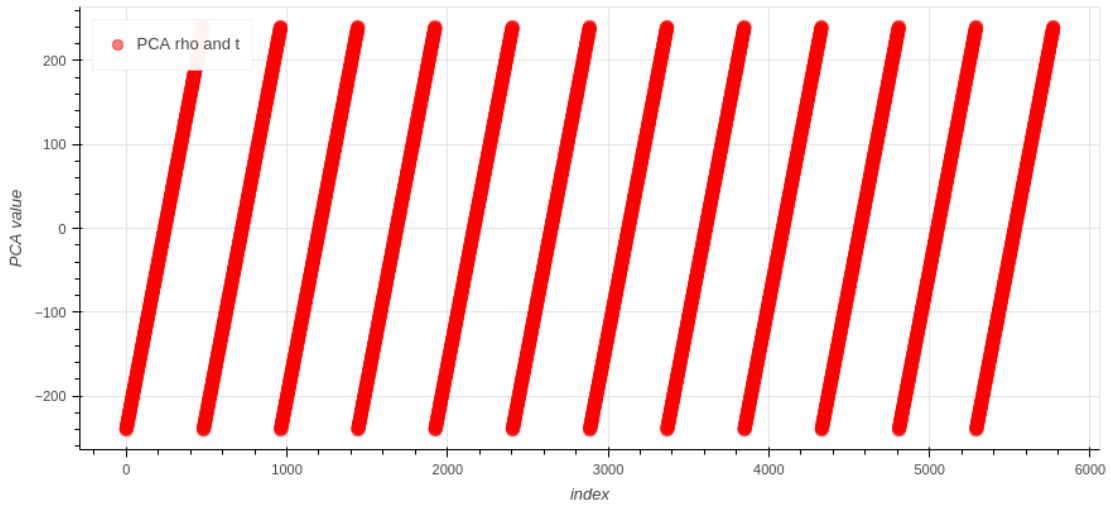

Figure 23: Representation of the new variable obtained by PCA. It is really similar to 't' value, but it is not exactly the same. On this particular case, 99.997266 % of the information was kept after the PCA was done.

For further clarification, if we obtain the difference between t and the PCA result, we obtain Figure 23, which is really similar to the visual representation of ρ on figure 24 (although it is in a different scale, which is the pursued point).

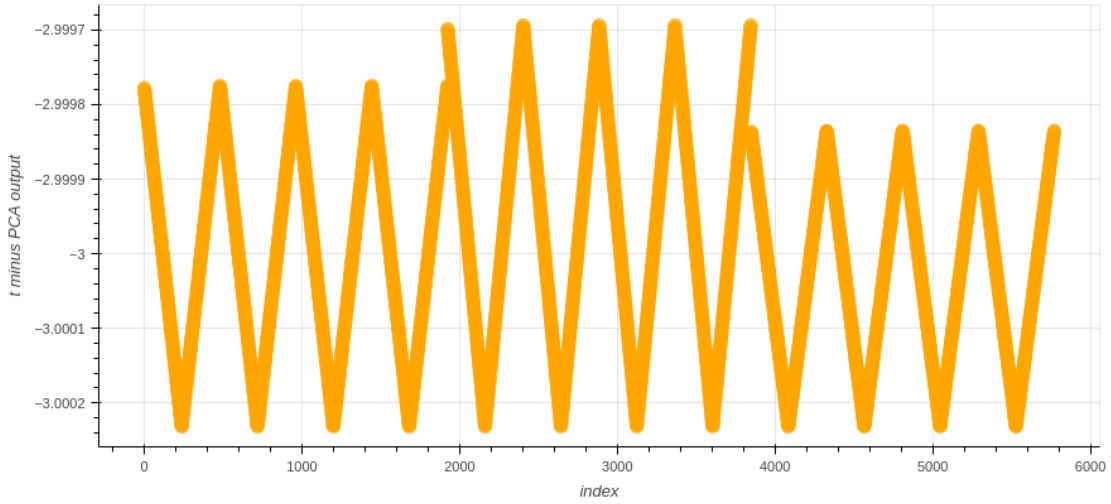

Figure 24: PCA minus T

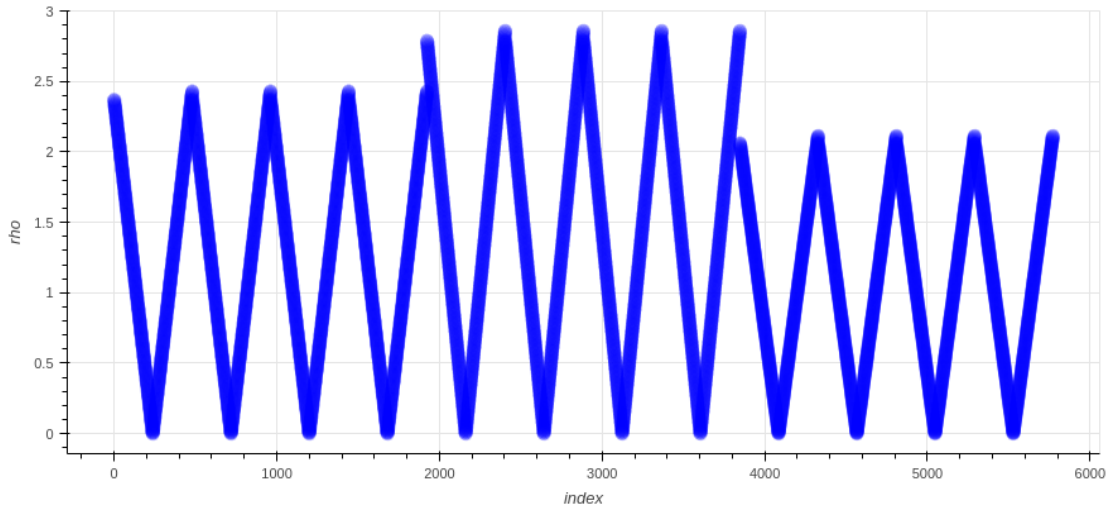

Figure 25: Rho values

After PCA is applied, data meant to be used on Neural Networks (or in general, IA algorithms) are usually normalized (Sola and Sevilla 1997). This operation speeds up computing of the results (lower number are easier to manipulate by the process) and usually improves results. If we apply PCA to all external variables (except our analytic function obtained through Monte Carlo) before normalizing, we end up having a 5772x2 matrix. If we apply PCA after applying normalization to said values, we obtain a 5772x5 matrix. To feed to the Neural Network we try three sets of data.

- PCA of external variables and then normalization of PCA output and Analytic Function (Will be referred later as Combination 1)
- Normalized analytic function (Will be referred later as Combination 2)
- Normalization of external variables, PCA and then normalized PCA output and normalized analytic function as data (Will be referred later as Combination 3)

6 Results

Due to the previously mentioned hardware and software limitations, results have not been equal in both axis. Also, due to the desired delivery dates, there was not enough time to get the same number of iterations for x and y profiles. However, both results on the Monte Carlo part are fine enough to get an approximation and to prove our method (using them as input for a Neural Network). This divergence on the sample, however, present an opportunity to us. This project's point is to prove the suitability of using a Convolutional Neural Network on a given analytic approximation to obtain an improved result. Having different reliability on the approximations serve us a proof of the NN working on different environments. Another problem is that y axis present some irregularities that are not present on the x axis, which lead us to higher errors.

6.1 Monte Carlo results

We can see on Table 5 and Table 6 that results improve quite slowly over each iteration. Each iteration of the Monte Carlo algorithm is composed of 100000 trials and last 8 days of uninterrumpted work on a laptop. These results were obtained through a span of approximately two months (taking into account unexpected problems like typos on the code or the laptop getting unplugged).

	δ_0	h_0	h_1	h_2	h_3	h_4	error
First iteration	2.551	0.001000	0.001000	0.001000	0.001000	-0.000100	0.015100
Second iteration	2.533	0.010000	0.001000	-0.000100	-0.000001	0.000010	0.014700
Third iteration	2.533	0.017783	0.017783	-0.017783	0.003162	0.000100	0.014040
Fourth iteration	2.755	0.046416	0.004642	-0.031623	0.010000	0.000316	0.014212
Fifth iteration	2.666	0.043333	0.011365	-0.038312	0.012915	0.000053	0.014019

Table 5: Results obtained through iterations on the 'x' axis

	δ_0	h_0	h_1	h_2	h_3	h_4	error
First iteration	1.857	-1.000000e-05	0.001	0.000010	0.001000	0.00100	0.027306
Second iteration	1.653	1.000000e-06	0.010	0.000032	0.000794	0.00010	0.024800
Third iteration	1.612	2.511886e-07	0.010	0.000068	0.000527	0.00001	0.023591

Table 6: Results obtained through iterations on the 'y' axis

6.2 Algorithms results

Now we start out of the latest result (fifth iteration on the x axis, third iteration on the y axis) as input for the Neural Network. As said earlier, we tried 3 types of datasets (combinations 1, 2 and 3), depending on the use of external variables and when were they normalized. Keras hyperparameter tuner is able to automatically iterate over the mayority of settings. However, loss functions and optimizers are not included among those settings, so we create a single hyperparameter tuner for each combination of settings. Algorithm results are measured over the test data (3 random profiles selected over the complete set of data), so the analytic error in the following tables are not exactly the same to the analytic error showed earlier. Every complete iteration of the hyperparameter tuner (which obtains a total of 1270 trials) last approximately 30 minutes, so trying the set of 11430 neural networks is completed in under 5 hours. Having into account we try 3 different datasets, we try 34290 different combinations for each axis. We can see in both tables that results are best on Combination 2, which is the only dataset that consistently obtains better results than the analytic equation ones (incluided aswell in the tables for the sake of comparison). Tables are sorted by Combination 2 results for simplicity.

	Combination 1	**Combination 2**	**Combination 3**
MSE RMSprop	0.079930	**0.011795**	0.041807
MSE Adam	0.021779	0.012387	0.020336
log cosh Adam	0.024581	0.013169	0.048212
MAE Adam	0.017286	0.013528	0.042306
MAE RMSprop	0.080163	0.013833	0.057472
log cosh Adamax	0.026564	0.014159	0.051768
log cosh RMSprop	0.064878	0.014343	0.059584
MSE Adamax	0.026564	0.014786	0.094382
Analytic	**0.014996**	0.014996	**0.014996**
MAE Adamax	0.017975	0.016372	0.052906

Table 7: Results for the 'y' axis, MAE

	Combination 1	**Combination 2**	**Combination 3**
MSE Adamax	0.070065	0.017253	0.042311
log cosh Adamax	0.047978	0.017364	0.037958
MAE Adam	**0.017714**	0.019160	0.024568
MAE RMSprop	0.041807	0.020069	0.064813
Analytic	0.020128	0.020128	**0.020128**
MAE Adamax	0.048893	0.020321	0.033222
log cosh RMSprop	0.061308	0.021524	0.064187
MSE RMSprop	0.064243	0.021808	0.061596
log cosh Adam	0.047339	0.022317	0.037502
MSE Adam	0.066479	0.022322	0.041273

Table 8: Results for the 'x' axis using only analytic function as input

Results comparing best Combination 2 algorithm and the analytic equation can be seen on Figures 26 to 31. It is kind hard to differentiate between analytical and CNN approximation, both approximations are shown mainly to illustrate how close are the approximations to reality.

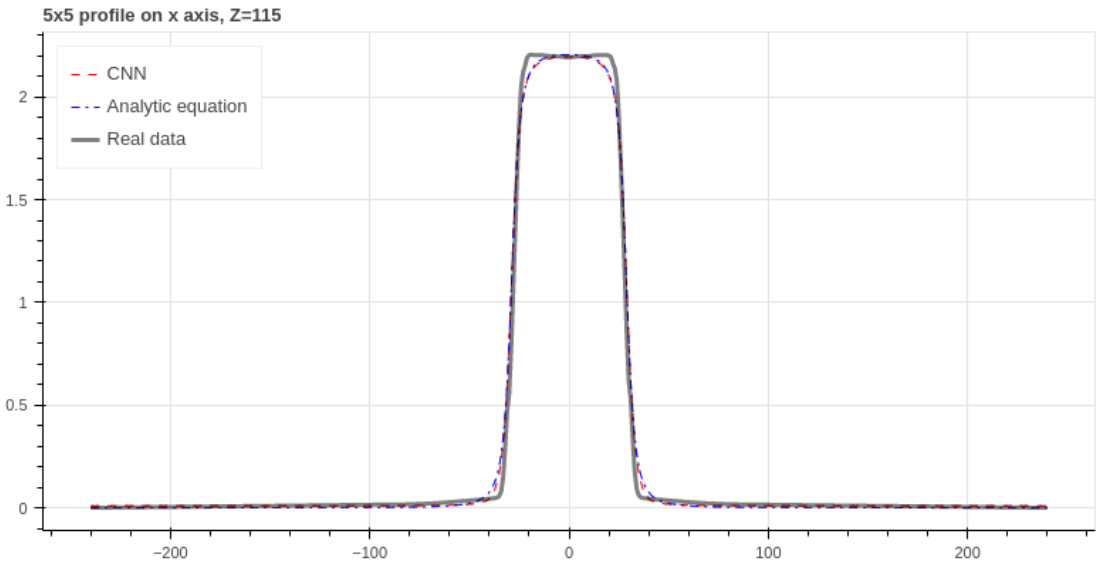

Figure 26: 5x5 profile, x axis

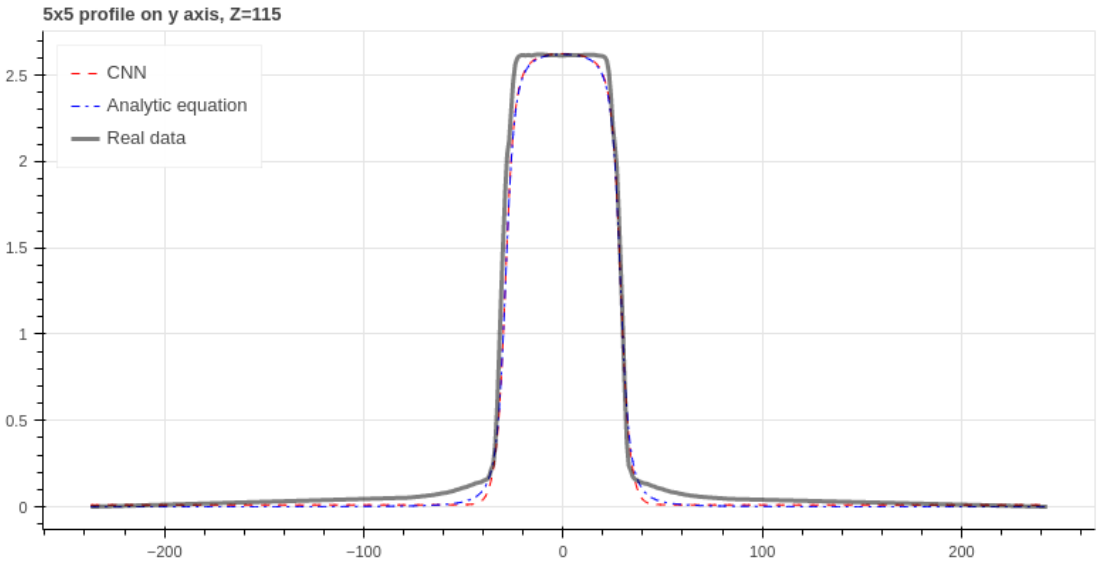

Figure 27: 5x5 profile, x axis

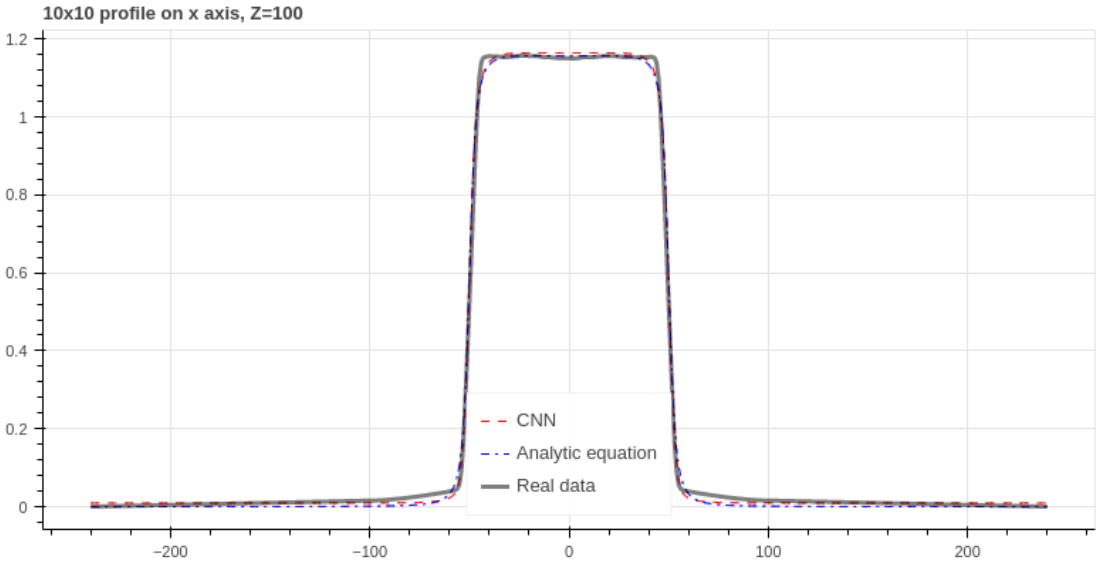

Figure 28: 10x10 profile, x axis

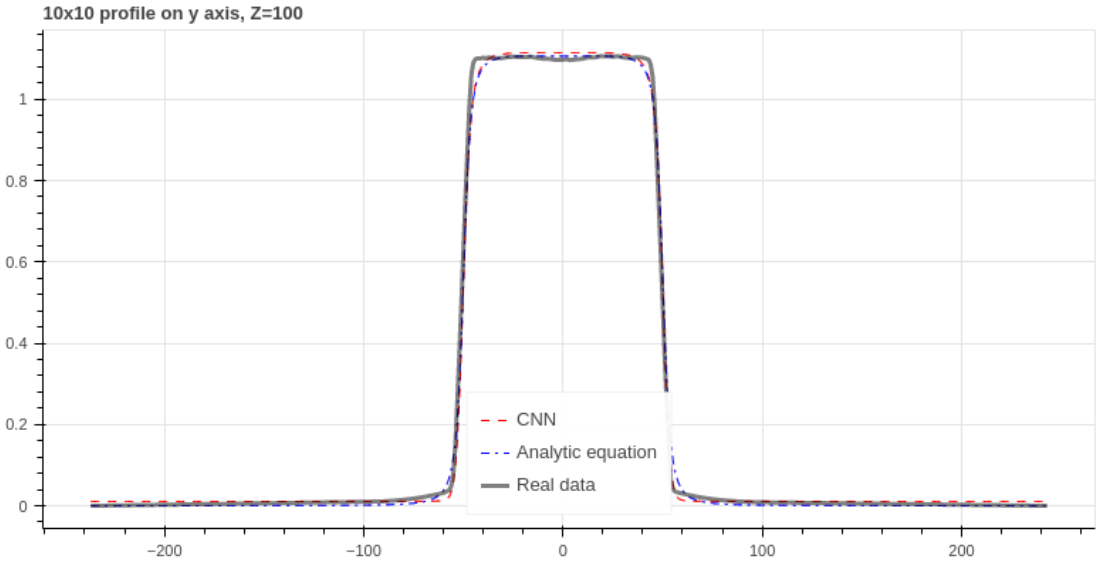

Figure 29: 10x10 profile, y axis

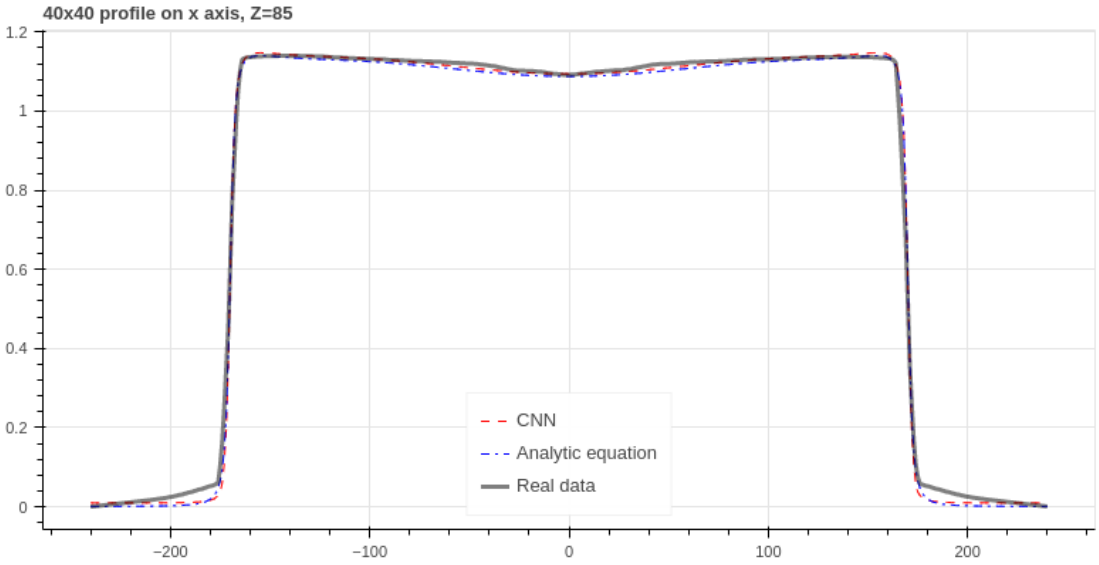

Figure 30: 40x40 profile, x axis

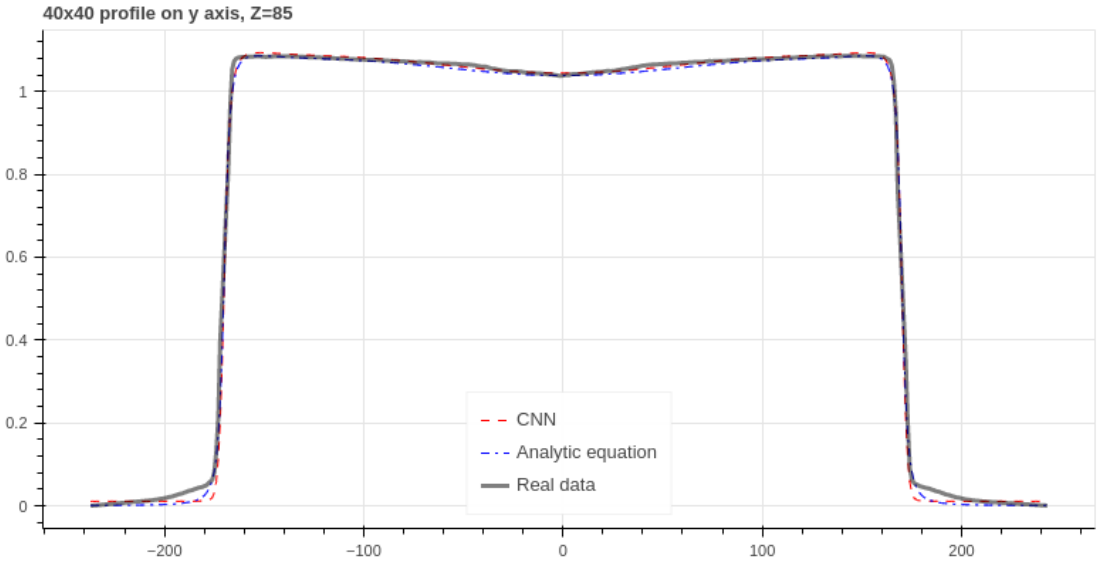

Figure 31: 40x40 profile, y axis

7 Conclusions

Out of these results, we can confirm our initial thesis. Using Neural Networks to obtain an approximation analytic approximation is possible and actually improves the results, with no dependance on the quality of the approximation. Trials on regular Neural Networks did not work out, however, CNNs did work great. Usually, algorithms work best using different variables so the algorithm can use the relationships between them to obtain an overall better result. The fact that in this it did not work out indicates us that the rest of variables only add noise to the algorithm. Obtaining alternative approximations out of this data, adding it to the analytic equation that we already used as variable could work out great. Other possible paths to improve the result is using 2 dimensional Neural Networks.

References

[1] Martin Abadi et al. "TensorFlow: A system for large-scale machine learning". In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 2016, pp. 265–283. URL: https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf.

[2] Alexander Amini et al. "Spatial Uncertainty Sampling for End-to-End Control" (2018). DOI: 10.48550/ARXIV.1805.04829. URL: https://arxiv.org/abs/1805.04829.

[3] Anish Batra and Vibhu Jawa. "Classification of arrhythmia using conjunction of machine learning algorithms and ECG diagnostic criteria". *Training Journal* (1975).

[4] J. Benites R, H. R. Vega C, and J. Velazquez F. "Spectra and absorbed dose by photoneutrons in a solid water mannequin exposed to a Linac of 15 MV". In: Espectros y dosis absorbida por fotoneutrones en un maniqui de agua solida expuesta a una Linac de 15 MV. NUCLEAR PHYSICS AND RADIATION PHYSICS. Mexico: Sociedad Mexicana de Irradiacion y Dosimetria, 2012. URL: http://inis.iaea.org/search/search.aspx?orig_q=RN:44026241.

[5] Stefano A. Bini. "Artificial Intelligence, Machine Learning, Deep Learning, and Cognitive Computing: What Do These Terms Mean and How Will They Impact Health Care?" *The Journal of Arthroplasty* **33** (2018), pp. 2358–2361. DOI: https://doi.org/10.1016/j.arth.2018.02.067. URL: https://www.sciencedirect.com/science/article/pii/S0883540318302158.

[6] Leonardo Bottaci et al. "Artificial neural networks applied to outcome prediction for colorectal cancer patients in separate institutions". *The Lancet* **350** (1997), pp. 469–472. DOI: https://doi.org/10.1016/S0140-6736(96)11196-X. URL: https://www.sciencedirect.com/science/article/pii/S014067369611196X.

[7] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization." *Journal of machine learning research* **12** (2011).

[8] Matthias Fippel et al. "A virtual photon energy fluence model for Monte Carlo dose calculation". *Medical physics* **30** (2003), pp. 301–11. DOI: 10.1118/1.1543152.

[9] Yoav Freund and Robert E Schapire. "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting". *Journal of Computer and System Sciences* **55** (1997), pp. 119–139. DOI: https://doi.org/10.1006/jcss.1997.1504. URL: https://www.sciencedirect.com/science/article/pii/S002200009791504X.

[10] Wilfredo Gonzalez. "Virtual source model for Monte Carlo calculations in external radiotherapy with photon beams". PhD thesis. 2015.

[11] Wilfredo Gonzalez et al. "A general photon source model for clinical linac heads in photon mode". *Radiation Physics and Chemistry* **117** (2015). DOI: 10.1016/j.radphyschem.2015.08.006.

[12] W. González, A. M. Lallena, and R. Alfonso. "Monte Carlo Simulation of a Micro-multileaf Collimator". In: *V Latin American Congress on Biomedical Engineering CLAIB 2011 May 16-21, 2011, Habana, Cuba*. Ed. by José Folgueras Méndez et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 1252–1255.

[13] Wilfredo González, Marta Anguiano, and Antonio M. Lallena. "Abstract ID: 79 Virtual source model for stereotactic radiosurgery with a dynamic micro-multileaf collimator". *Physica Medica* **42** (2017). Abstracts of the MCMA2017, p. 16. DOI: https://doi.org/10.1016/j.ejmp.2017.09.039. URL: https://www.sciencedirect.com/science/article/pii/S1120179717303587.

[14] Th I Götz et al. "A deep learning approach to radiation dose estimation". *Physics in Medicine &amp Biology* **65** (2020), p. 035007. DOI: 10.1088/1361-6560/ab65dc. URL: https://doi.org/10.1088/1361-6560/ab65dc.

[15] Klaus Greff et al. "LSTM: A search space odyssey". *IEEE transactions on neural networks and learning systems* **28** (2015). DOI: 10.1109/TNNLS.2016.2582924.

[16] Dashan Jiang et al. "Convolutional Neural Network Based Dosimetry Evaluation for Esophageal Radiation Treatment Planning". *Medical Physics* **47** (2020). DOI: 10.1002/mp.14434.

[17] Raphaël Jumeau et al. "Stereotactic Radiotherapy for the Management of Refractory Ventricular Tachycardia: Promise and Future Directions". *Frontiers in Cardiovascular Medicine* **7** (2020), p. 108. DOI: 10.3389/fcvm.2020.00108.

[18] Diederik Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". *International Conference on Learning Representations* (2014).

[19] Dirk P. Kroese et al. "Why the Monte Carlo method is so important today". *Wiley Interdisciplinary Reviews: Computational Statistics* **6** (2014).

[20] Min Sun Lee et al. "Deep-dose: a voxel dose estimation method using deep convolutional neural network for personalized internal dosimetry". *Scientific Reports* **9** (2019), p. 10308. DOI: 10.1038/s41598-019-46620-y. URL: https://doi.org/10.1038/s41598-019-46620-y.

[21] Lisha Li et al. "Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization". *Journal of Machine Learning Research* **18** (2018), pp. 1–52. URL: http://jmlr.org/papers/v18/16-558.html.

[22] W R Lutz and R D Larsen. "Breast phantom for ion chamber dosimetry". en. *Int J Radiat Oncol Biol Phys* **10** (1984), pp. 933–934.

[23] Ganesan N et al. "Application of Neural Networks in Diagnosing Cancer Disease using Demographic Data". *International Journal of Computer Applications* **1** (2010). DOI: 10.5120/476-783.

[24] Tom O'Malley et al. *KerasTuner*. https://github.com/keras-team/keras-tuner. 2019.

[25] Keiron O'Shea and Ryan Nash. "An Introduction to Convolutional Neural Networks". *CoRR* **abs/1511.08458** (2015). arXiv: 1511.08458. URL: http://arxiv.org/abs/1511.08458.

[26] Pariwat Ongsulee. "Artificial intelligence, machine learning and deep learning". In: *2017 15th International Conference on ICT and Knowledge Engineering (ICTKE)*. 2017, pp. 1–6. DOI: 10.1109/ICTKE.2017.8259629.

[27] Edgar M Palmer and Allen J Schwenk. "On the number of trees in a random forest". *Journal of Combinatorial Theory, Series B* **27** (1979), pp. 109–121.

[28] Pulkit Sharma. *Build an Image Classification Model using Convolutional Neural Networks in PyTorch*. https://www.analyticsvidhya.com/blog/2019/10/building-image-classification-models-cnn-pytorch/. Accessed: 11-07-2022. 2019.

[29] Sagar Sharma, Simone Sharma, and Anidhya Athaiya. "Activation functions in neural networks". *towards data science* **6** (2017), pp. 310–316.

[30] Li Shen et al. "Deep Learning to Improve Breast Cancer Detection on Screening Mammography". *Scientific Reports* **9** (2019), p. 12495. DOI: 10.1038/s41598-019-48995-4. URL: https://doi.org/10.1038/s41598-019-48995-4.

[31] J. Sola and J. Sevilla. "Importance of input data normalization for the application of neural networks to complex industrial problems". *IEEE Transactions on Nuclear Science* **44** (1997), pp. 1464–1468. DOI: 10.1109/23.589532.

[32] Thomas Tessonnier et al. "Dosimetric verification in water of a Monte Carlo treatment planning tool for proton, helium, carbon and oxygen ion beams at the Heidelberg Ion Beam Therapy Center". *Physics in Medicine and Biology* **62** (2017). DOI: 10.1088/1361-6560/aa7be4.

[33] Tijmen Tieleman, Geoffrey Hinton, et al. "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude". *COURSERA: Neural networks for machine learning* **4** (2012), pp. 26–31.

[34] Michael E Tipping and Christopher M Bishop. "Mixtures of probabilistic principal component analysers" (1998).

[35] Jiankui Yuan, Yi Rong, and Quan Chen. “A virtual source model for Monte Carlo simulation of helical tomotherapy”. *Journal of Applied Clinical Medical Physics* **16** (2015), pp. 69–85. DOI: https://doi.org/10.1120/jacmp.v16i1.4992. eprint: https://aapm.onlinelibrary.wiley.com/doi/pdf/10.1120/jacmp.v16i1.4992. URL: https://aapm.onlinelibrary.wiley.com/doi/abs/10.1120/jacmp.v16i1.4992.

A Monte Carlo

A.1 Random Monte Carlo

```
error = []
usedl = []
usedk = []
usedj = []
usedh = []
usedg = []

for i in range(0, 100000):
    l = float(random.choices([-1, 1])[0])*10**(np.random.uniform(low=-7, high=-1))
    k = float(random.choices([-1, 1])[0])*10**(np.random.uniform(low=-7, high=-1))
    j = float(random.choices([-1, 1])[0])*10**(np.random.uniform(low=-7, high=-1))
    h = float(random.choices([-1, 1])[0])*10**(np.random.uniform(low=-7, high=-1))
    g = float(random.choices([-1, 1])[0])*10**(np.random.uniform(low=-7, high=-1))


...

    error.append(momerror)
    usedl.append(l)
    usedk.append(k)
    usedj.append(j)
    usedh.append(h)
    usedg.append(g)
```

A.2 Directed Monte Carlo

```
error = []
usedl = []
usedk = []
usedj = []
usedh = []
usedg = []

ls1 = 10**np.linspace(-1, -7, 5)
ls2 = -10**np.linspace(-1, -7, 5)
ls = np.hstack([ls1, ls2])

ks1 = 10**np.linspace(-1, -7, 5)
ks2 = -10**np.linspace(-1, -7, 5)
ks = np.hstack([ks1, ks2])

js1 = 10**np.linspace(-1, -7, 5)
js2 = -10**np.linspace(-1, -7, 5)
js = np.hstack([js1, js2])

hs1 = 10**np.linspace(-1, -7, 5)
hs2 = -10**np.linspace(-1, -7, 5)
hs = np.hstack([hs1, hs2])


gs1 = 10**np.linspace(-1, -7, 5)
gs2 = -10**np.linspace(-1, -7, 5)
gs = np.hstack([gs1, gs2])
```

```
for l in ls:
    for k in ks:
        for j in js:
            for h in hs:
                for g in gs:


...

                error.append(momerror)
                usedl.append(l)
                usedk.append(k)
                usedj.append(j)
                usedh.append(h)
                usedg.append(g)
```

B Keras tuner example

```
def model_builder(hp):

    hp_units1 = hp.Choice('units Conv', values=[4, 8, 16, 32, 64])
    hp_units3 = hp.Int('units 3', min_value=200, max_value=2000, step=50)
    hp_units4 = hp.Int('units 4', min_value=10, max_value=200, step=10)
    kernel1 = hp.Choice('kernel 1', values=[1, 3, 5, 7, 9])
    hp_learning_rate = hp.Choice('learning_rate', values=[1e-2, 1e-3])
    dropout = hp.Float('Dropout', min_value=0, max_value=0.2, step=0.05)
    activation1=hp.Choice('dense_activation 1',values=['relu', 'relu', 'elu', 'selu'])
    activation2=hp.Choice('dense_activation 2',values=['relu', 'relu', 'elu', 'selu'])
    activation3=hp.Choice('dense_activation 3',values=['relu', 'relu', 'elu', 'selu'])
    activation4=hp.Choice('dense_activation 4',values=['relu', 'relu', 'elu', 'selu'])
    activation5=hp.Choice('dense_activation 5',values=['relu', 'relu', 'elu', 'selu'])
    activation6=hp.Choice('dense_activation 6',values=['relu', 'linear'])


    model = Sequential()
    model.add(Conv1D(hp_units1, (kernel1), padding='same', activation=activation1))
    model.add(Dropout(dropout))
    model.add(Conv1D(hp_units1*2, kernel_size=(kernel1), padding='same', activation=activation2))
    model.add(Dropout(dropout))
    model.add(Conv1D(hp_units1*4, kernel_size=(kernel1), padding='same', activation=activation3))
    model.add(Dropout(dropout))
    model.add(Dense(hp_units3, activation=activation4))
    model.add(Dense(hp_units4, activation=activation5))
    model.add(Dense(1, activation=activation6))
    model.compile(optimizer=tf.keras.optimizers.RMSprop(learning_rate=hp_learning_rate)
                                , loss='mae', metrics = ['MAE'])
    return model

tuner = kt.Hyperband(model_builder,
                     objective='val_loss', max_epochs=150, hyperband_iterations=3,
                     directory='my_dir',
                     project_name='Conv1D_IN0_RMSprop')


stop_early = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=3)

tuner.search(x_train, y_train, epochs=100, validation_data=(x_test, y_test),
                     callbacks=[stop_early], batch_size=x_train.shape[0])
```