UCLouvain

École polytechnique de Louvain

AutoML framework for dose prediction models in radiotherapy

Authors: **Emilyen Laffineur, Romain Loncelle**
Supervisors: **John A. Lee, Ana Barragan, Margerie Huet**
Reader: **Sébastien Jodogne**
Academic year 2021–2022
Master [120] in Computer Science

Abstract

Radiotherapy is one of the main types of treatment against cancer. The aims of any radiotherapy is to destroy the tumorous cell while sparing as much as possible the healthy tissues. When a patient needs radiotherapy, a plan must be established. To do so, one must first get the images of the patient, then physicians must contour the tumor and the different organs at risks, after that some thresholds have to be set for each organ and for the tumor, finally a dosimetrist have to generate the plan. This process takes many hours and computational resources. Artificial intelligence is nowadays used in these different steps in order to speed it up but also, reduce the amount of work needed from the specialists.

Automated machine learning (AutoML) is a new trend in machine and deep learning world. Its goal is to propose an easy way of making model selection and parameters tuning for any dataset. Thus, reducing the amount of time spent at manually doing these steps. A secondary goal is to have more people being able to use such software even if they are not specialists in the domain because the model will do the complicated steps by itself.

This thesis wants to reunite both worlds, by proposing an AutoML framework for the dose prediction problem in radiotherapy. It is based on previous work on HD-Unet and adapt it to use the principle of AutoML. It is easy to use, maintain and evolve. For a new dataset, it computes many parameters without the need for the user to do anything. To test this model, multiple datasets were used to show the adaptation of the model. Its results are better than the work it is based on concerning H&N cancer and even if not as good as the fine-tuned model on other datasets its results are still interesting.

Acknowledgements

We would like to thank our supervisors John A. Lee, Ana Barragan and Margerie Huet. Their help during the realization of this thesis and their recommendation for this manuscript were really precious.

We also want to thank Sébastien Jodogne who has accepted to be a reader and a jury's member for this thesis.

We want to insist on Margerie who helps us countless times with the model implementation but also providing us the needed dataset and the results of previous experiments.

Thank you Ana for all the comments and recommendations regarding this thesis.

Thank you Camille and Benjamin for providing us the results of your own dataset and for answering our questions on them.

Thank you Marie for your continuous reading of this thesis and your help on correcting and improving it.

Also, I, Emilyen, would like to personally thank Christelle, Francis and Hélène for their continuous support during the realization of this work and my studies.

For my part, I, Romain, would like to thank a little more personally our supervisors for the encouragement and the possibilities they try to offer me regarding my possible future career. I would also like to thank Marie and her family who gave me support and motivation during these years. I can never thank them enough.

Computational resources have been provided by the supercomputing facilities of the Université catholique de Louvain (CISM/UCL) and the Consortium des Équipements de Calcul Intensif en Fédération Wallonie Bruxelles (CÉCI) funded by the Fond de la Recherche Scientifique de Belgique (F.R.S.-FNRS) under convention 2.5020.11 and by the Walloon Region.

Contents

1 | Introduction

1.1 Motivation

Thanks to the emergence of medical imaging and related technologies, the medical sector has made tremendous progress. This is particularly the case for treatments against cancer, these technologies have significantly accelerated the speed of detection and diagnosis of cancer as well as the apparition of new treatments while being much less invasive than surgery. However, the fast progress of technology might be a double-edge sword for humans. Indeed, with the democratization of medical imaging, the number of images and clinical data to analyse increases exponentially, while the number of specialists able to analyze them is not[1][2]. Also, sometimes the amount of information conveyed in the images is too high to analyze it at the naked eye. Technological advances in the computer vision domain could be a plausible solution to this problem.

Among the available treatment modalities employed against cancer, radiation therapy is one of the most commonly used. While conventional radiotherapy, using X-ray beams, can and already helped a lot of people, it has some problems. Indeed, conventional radiotherapy encounters issues when the tumor is too deep in the body or even if it is close to critical structures and is therefore not optimal. For this type of case, we would rather use a more modern technology called proton therapy. The idea behind it is the same as before, but instead of using X-ray beams, it delivers protons. Indeed, with conventional radiotherapy, the photons deliver the dose in an exponentially decreasing way, with a peak of the dose as soon as they enter the body and slowly attenuating after that. As a result, a dose of radiation is delivered to the healthy tissue on the way out. The quantity of radiation absorbed by the healthy tissues after the radiation reaches the target is called exit dose. On the other hand, the greatest advantage of proton therapy is its ability to stop at a given point in the body, resulting, as shown in figure 1.1, in a peak of dose on this precise point (Bragg Peak), without affecting the healthy tissues that comes after

the target (i.e., proton therapy does not have an exit dose).

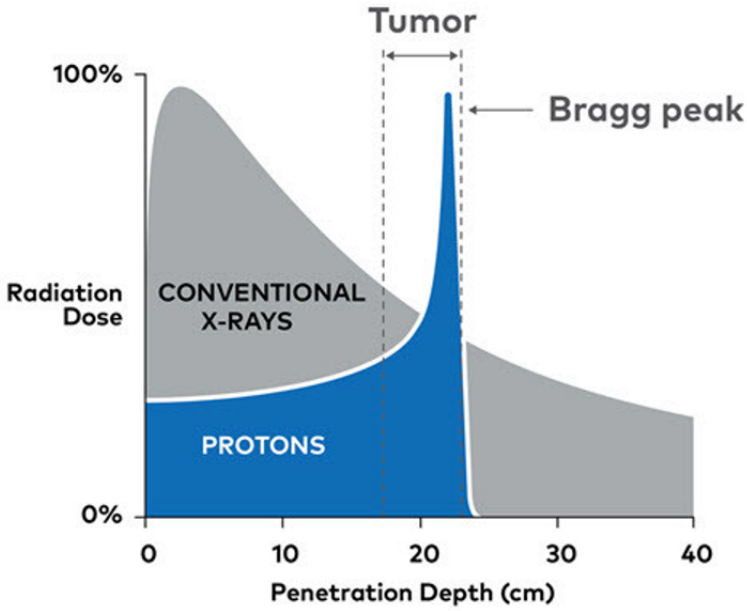

Figure 1.1: Comparison of the radiation dose received at different depths from conventional radiotherapy (using X-rays) and proton therapy. Credits : [3].

If any of these two modalities have to be employed, a treatment plan has to be established. This task is quite complex and time consuming (up to several hours), and it requires the intervention of different medical specialists and specialized tools. In addition to the machines and hardware required to acquire the images (e.g., Magnetic Resonance or Computed Tomography scans), the treatment planning process requires dedicated software to aid the delineation of the organs and tumor volume, as well as to optimize the machine parameters (e.g., number of radiation dose beams, intensity, etc.) that deliver the desired dose. Figure 1.2 describes the whole process.

1. Step one consists of getting images of the patient. To do that Computed Tomography (CT) and/or MRI scans are used;

2. Then, out of these scans, a physician outlines the organs at risk and the tumor contours. This step is also known as segmentation;

3. A medical dosimetrist is needed after the segmentation : using an optimization software they produce the plan to determine the appropriate external beam configuration required for the given patient;

4. Only after all these step the patient's treatment can start.

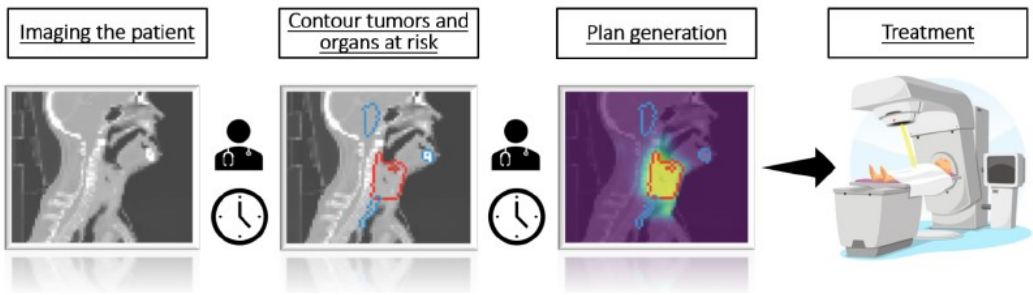

Figure 1.2: Actual treatment planning pipeline. The tumors (in red) and organs at risk (in blue) are contoured manually by a specialist and is a long process. The plan generation is also done manually by another specialist and takes time. Credits : [4].

The radiotherapy treatment is divided in several sessions (or fractions), delivered once or twice per day for a few weeks (up to 30 days usually). Typically, a plan is created at the beginning and used during all sessions. However, this is not ideal since the anatomy might change, and the plan would not be optimal anymore for the patient. Indeed, given the time and resources needed to establish a plan, the act of creating a new plan during the treatment (or re-planning) is only done in specific cases (i.e., when the current situation has changed too much).

It is for all these reasons and for the well-being of the patients that many research is trying to fully automate this process as much as possible. This would have many positive effects, not only directly on the process we have just discussed, but also because in doing so, the variability of medical specialists involved would be reduced. Moreover, those who still takes part in the process would be relieved of some of their workload. This would allow all these specialists to be able to make use of this time. And while some work to automate this process has already been done, there is still room for improvements. This thesis aims to improve an existing deep learning framework to predict the optimal dose for a given patient. As shown in figure 1.3, this thesis could be incorporated into a new, automated, pipeline.

The biggest advantage of this automated pipeline is the possibility to update the plan for each session. The patient's treatment can be adapted "in real time", this principle is called adaptive treatment, this has many advantages, the most interesting being that the plan would be optimal for each fraction[5][6][7].

Deep learning dose prediction models, such as the one aimed to be improved in this thesis, have thus the advantage of automating the planning process, reducing the

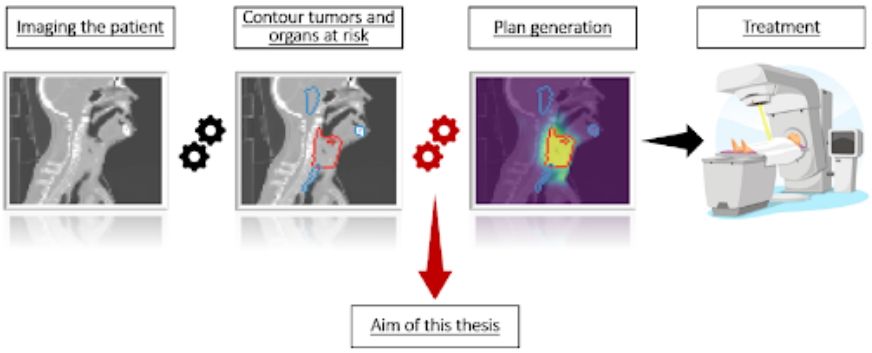

Figure 1.3: Automated treatment pipeline. The contouring and plan generation are done in a faster way thanks to automatization. Credits : [4].

human intervention to a minimum and speeding up the entire process. However, the biggest drawback of this approach is that individual models need to be trained for each specific treatment type and location. Since the models are specific to the dataset used for training them, a model trained for proton therapy in head and neck cancer will not be accurate for conventional radiotherapy treatments in prostate cancer patients, for instance. In addition, despite being entirely automatic once they are trained, data-specific operations and human decisions are still involved in the process of training a model. For example, the number of input images, type, size and resolution might influence the choice of certain parameters of the model architecture. Furthermore, in order to find the optimal value for those parameters (i.e., hyperparameter tuning), an empirical process is set up. Typically, humans run the model for different set of parameters and chose the combination that provides the most accurate results according to the dataset under study. However, it can be complex for someone who is not an expert in machine learning to know which parameter sets to use or within which limits it is interesting to modify the parameters (i.e., one could test too many different values for a given parameter). This could make the empirical process longer than necessary or even less efficient in the end. All these elements make the process of model training accessible for only machine learning experts. Auto machine learning or Automated Machine Learning (AutoML) is a recent concept that is gaining attention since it tries to automate the process of building a model, from the raw database to the final optimized model. Since it has a high degree of automation, developing an AutoML framework for dose prediction models will make the model generation process accessible for the medical community without imposing them to become machine learning experts. It will also ease the generation of multiple specific models for each cancer treatment and location. Indeed, the use of AutoML allows for faster generation of models which are capable of outperforming hand-designed ones.

Finally, this thesis has a second focus, following what has just been said, it aims to optimize the work already done in the automation of the plan generation by making it faster while remaining at least as accurate as before. The AutoML part will already partially take care of this by adapting to the received data. But the idea is to go further and to exploit the available hardware resources as much as possible in order to optimize the code that has to be executed by using particular data structures or by trying to limit as much as possible the hardware bottlenecks.

1.2 Objectives

This thesis aims to optimize the HD-Unet[4] by transforming it from a classic static model to an AutoML one. To tackle down this problem the following steps were planned:

- This thesis aims to create an AutoML framework for the dose prediction problem (this includes speeding up the process in a general way : set up, learning, training time, etc.);
- Completely understand nnU-Net and the classical HD-Unet;
- Convert HD-Unet from Tensorflow to Pytorch;
- Develop the AutoML framework;
- Test it against the current HD-UNet architecture.

Our model must also meet some requirements :

- At least as good as the original model;
- Easy to use;
- Adapt to the dataset of the user;
- Give good results on any dataset.

Here, a new HD-Unet model is proposed, an AutoML one which allows its users to give it a data set and directly have a usable model without having to optimize or modify the model by themselves. To develop it the main used data set is composed of patients suffering from head and neck tumors as it was also the one use for the original HD-Unet[4] model. Then the model was validated using datasets containing images of patients suffering from different cancer types and treated with

different techniques :

- Dataset 1) esophagus cancer patients treated with conventional radiotherapy;
- Dataset 2) esophagus cancer patients treated with proton therapy;
- Dataset 3) head and neck cancer patients treated with conventional radiotherapy;
- Dataset 4) head and neck cancer patients treated with proton therapy;
- Dataset 5) prostate cancer patients treated with conventional radiotherapy.

This thesis is divided as follows : Chapter 2 will expose the clinical context and planning process. Chapter 3 is the review of state-of-the-art models and literature review. Chapter 4 takes its interest by explaining how we have designed the AutoML model and Chapter 5 shows the results achieved by our model. This thesis is then completed by a discussion and the conclusion.

2 | Context

2.1 Clinical context

Once patients are diagnosed with cancer, a whole process starts in order to help them deal with the disease, beginning with the imaging of the patients and ending with their treatment. During this process a lot of specialists get involved for many tasks. Here is an introduction to the basic pipeline.

2.1.1 Patient imaging

First, specialists need to produce images of the patient. This is done using a Computed Tomography (CT) scan which allows the specialists to analyze the patient's body and allow them to detect some diseases. But more importantly, as a part of the basic pipeline presented here, it allows them to detect the tumor. CT scan works as follows : it takes different X-ray images of the patient using distinct angles and produces slices of images. Then it combines all these slices and produces a 3D image that can be analyzed by physicians. This process allows to grant great details to the final images.

For this step to be smooth, the patient must be as steady as possible. To achieve this goal, multiple techniques can be used either to immobilize the patient but also to ensure that the process is not too painful or uncomfortable. Every time an image is created this way, it must be recorded in the patient's file. Indeed, the beam intensity is calculated based on the patient's position with all accessories used (e.g., mask).

When the patients are being scanned, they go through an X-ray tube that rotates around them. Detectors are used to measure the amount of attenuation of X-ray that traverses the tissues. The Hounsfield Unit (HU) is used to determine the types of tissues traversed by the X-ray. It varies from a range of $[-1000 \rightarrow 3000]$ where

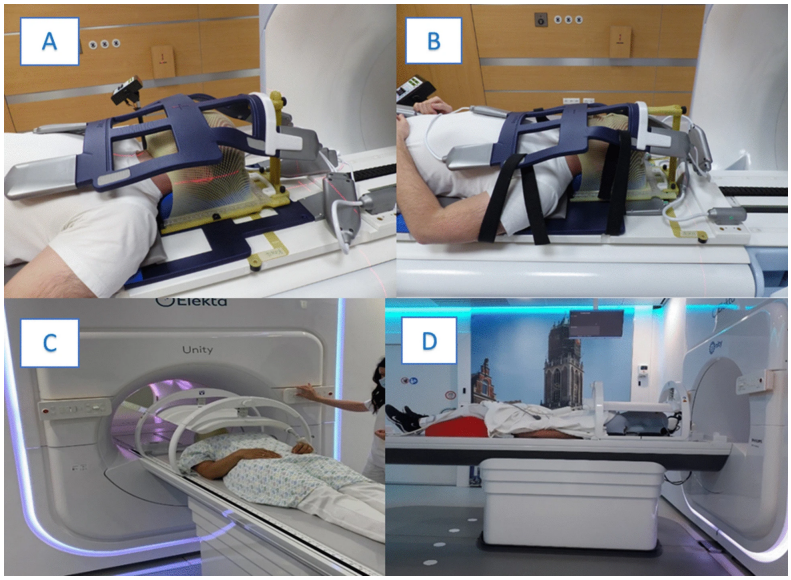

Figure 2.1: Examples of CT scans and masks used. A and C for brain, B and D for H&N using two different CT systems. Credits : [8].

-1000 HU is the measurement for air, 0 HU is for water for a given temperature and pressure, 1000 HU for bones, 2000 HU can be reached by denser bones and metals reach 3000 HU. It can be measured since the attenuation of the X-ray is proportional to the physical density of the tissues traversed following the given equation 2.1. Dense tissues have a higher HU and lead to brighter zones on the scans while the final image is computed while less dense tissues lead to darker zones.[9]

$$HU = 1000 * \frac{\mu - \mu_{water}}{\mu_{water} - \mu_{air}} \tag{2.1}$$

Equation 2.1 : Hounsfield equation where μ correspond the voxel attenuation, μ_{water} the attenuation coefficient for water and μ_{air} the attenuation coefficient of air.

Different techniques such as Magnetic Resonance Imaging (MRI) or Positron emission Tomography (PT) scans can be used together to find as precisely as possible where is located the patient's tumor. Different technologies give different images like the following 2.2.

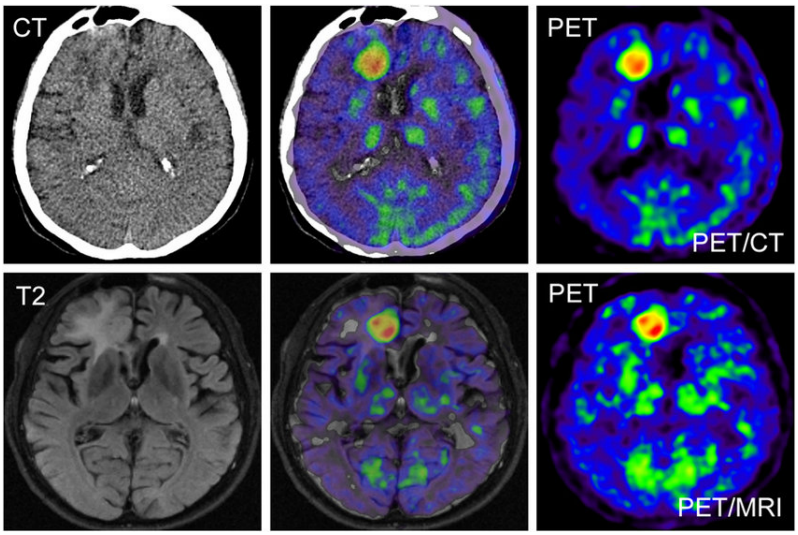

Figure 2.2: Different modalities (PET, CT, MRI) scan for a patient diagnosed with a glioblastoma multiforme. Credits : [10].

2.1.2 Patient's organs contouring

The next step includes the physician who, given the images of the patient's scans, will contour different volumes as well as the organs at risk on the images. These volumes are the following: Gross Target Volume (GTV), Clinical Target Volume (CTV) and Planning Target Volume (PTV).

- **GTV** : Also known as macroscopic volume, it contours the location, position and extension of the tumor. GTV usually highlights the part of the tumor that has the highest tumor cell density[11]. GTV can be produced by specialists via contouring or with the help of computers. In the latter case, the process is therefore called segmentation.

- **CTV** : The Clinical Target Volume consists of the GTV and an extension of it in order to include the spread of the disease. The part of the tumor highlighted in the CTV has a lower tumor cell density than the one in the GTV. This task is hard to do as the margin to add can not solely be based on the tumor, some anatomical barriers must also be taken into account. Hence, it is based in past time series, analyzing the risk and spread of the tumorous cells[11].

- **PTV** : The Planning Target Volume is a more abstract concept. It is used to take into account errors that may have occurred due to patient movement or positioning[11]. It encompasses the CTV and its purpose is to ensure that

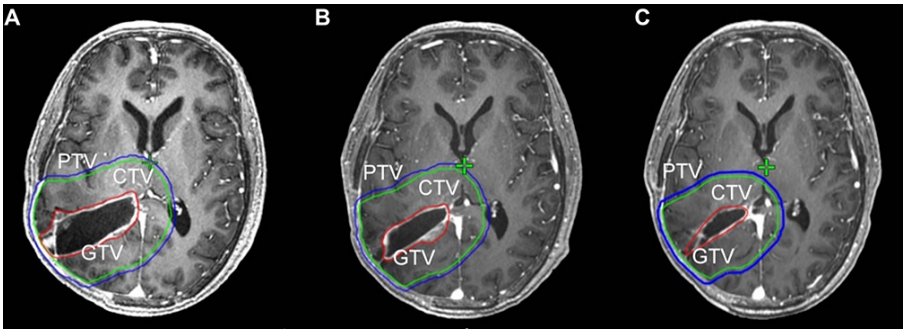

Figure 2.3: Examples of contouring. GTV is contoured in red, CTV is green and PTV is blue, the three images are gotten from different MRI technologies (MRI before RT, MRI during 10-fraction RT and MRI during 20-fractions RT. Credits : [12].

the CTV receives the intended dose.

This step takes a lot of time and effort from the specialists involved. Some automated software like nnU-Net[13] tries to automate and make this step much easier for everyone.

2.1.3 Dose constraints

Once the different contours have been drawn, the dosimetrist must prescribe the dose to each of the Target Volume (TV). This is a very important and difficult task as they must take into account not only the tumorous cells but also all the different organs near it and the amount of dose each of them can receive at maximum. To measure the dose, the Gray unit is applied :

$$1Gy = 1J/Kg \tag{2.2}$$

Equation 2.2 : Gray unit formula

2.1.4 Plan generation

Now that the initial prescription is done, software is used to compute and optimize the dose delivery. The specialist encodes all the information such as the maximum or minimum amount of dose the organs can receive. Then they optimize a first time the parameters used by the software so it can produce a dose map respecting the different constraints given by the specialist.

The goal of this operation is to ensure that the volume of Gray addressed to the tumor is sufficient while protecting as much as possible the surrounding tissues. This process can be repeated several times before the prescribed dose is delivered. The final step of this process is to produce the dose map for the different organs.

This task takes a lot of time since if the dose map is not good at the first time, one can add more constraints to guide the software through its optimization process. This thesis wants to optimize this process. Transferring the burden of parameters tuning to the model instead of relying on the specialists. The idea behind it would be to feed the network with the patients' images that are contoured, then all parameters will be automatically computed without the need for human intervention.

2.1.5 Evaluation

In order to evaluate the treatment accuracy, one can generate Dose Volume Histogram (DVH graph). These graphs allow the specialists to analyze the dose distribution and volume for each Organ At Risk (OAR), an example is given below in figure 2.4. A good DVH graph would have a steep line for the target volumes at the desired prescription, while the dose to the OAR should be as close as possible to the left of the plot (i.e., closer to a dose of 0 Gy).

These kinds of plots also allow specialists to extract different interesting metrics such as D_{95} which is the dose received to the 95% of the volume for a given organ/tumor. Different metrics can be used for different points of interest (e.g., D_{mean}, D_5).

2.1.6 Treatment

Once all these steps have been made, the patient treatment can start. It is a hard process for the patients as they undergo with cancer. Besides, during the court of treatment, there could be a need to adapt it. Having the possibility to adapt the treatment during the process is called adaptive radiotherapy. This is needed as the patients' anatomy can change during the course of the treatment. The plan is adapted the new anatomy to ensure a more precise treatment. In this case, extra images are taken at the beginning of each session.

Note that, besides having to deal with cancer, the treatment can have some side effects too. As an example, patients that suffer from head and neck tumors could develop xerostomia or dysphagia which are two complications that impairs swallowing and speech[14].

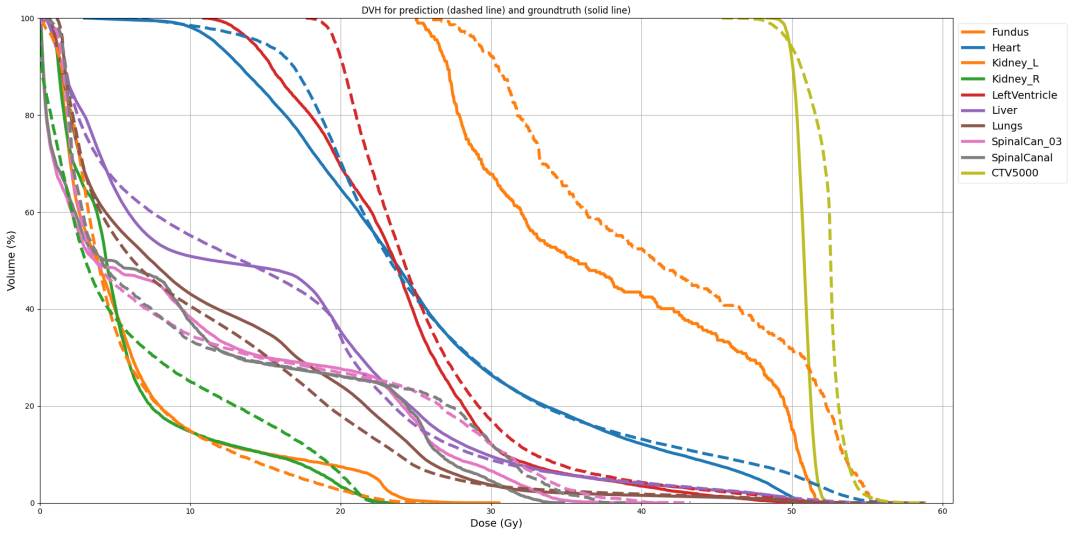

Figure 2.4: Example of a DVH graph generated by our model for a patient suffering of an Esophagus tumor. The x-axis shows the dose in Gray while the y-axis shows the percentage of volume. On this plot, the yellow line represents the dose that will be delivered to the tumor will the other lines represents the dose that the OAR will receive.

Besides, if they do not respond well to their treatment and it has to be adapted, the whole process must start again. Consequently, it consumes more time and more resources while it can discourage the patients and make them less responsive to their treatment[15].

2.2 Neural Networks

Neural networks (NN) take their origin decades ago, starting with a classical brain inspiration to nowadays complex models. This section will take a quick overview of neural networks and their use in radiotherapy and proton therapy.

Neural Networks also called Artificial Neural Networks (ANN) are inspired from the structure of the human brain as shown in figure 2.5 which represent one neuron. The different dendrites represent the inputs while the output is represented as the axon terminal. The intermediate steps "imitate" the transfer of information from the dendrites through the axon terminal passing by the myelin sheath, which allows electrical impulses to go along the cells.

A neural network is composed of a set of these neurons, also called units. These

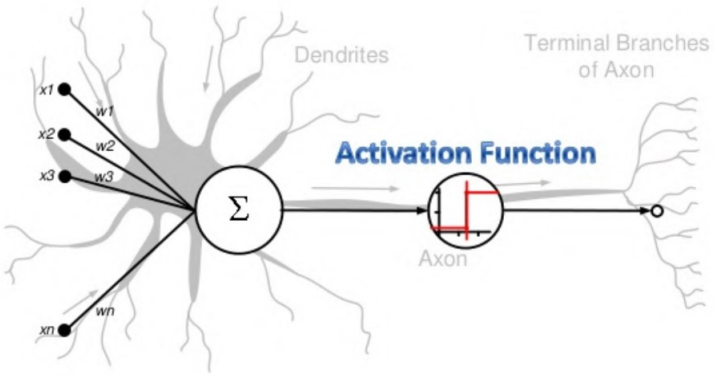

Figure 2.5: The original perceptron design inspired by a real neuron. Each x_n (the dendrites) correspond to one input, each w_i represents the associated weight, all these results are summed up and passed to an activation function to produce the output y (at the axon terminal). Credits : [16].

units are arranged by layers, the first layer is the input layer. Then come one or several hidden layers and finally the output layer. Deep Neural Networks (DNN) are networks that have multiple hidden layers. A hidden layer is simply a set of neurons connected to the previous and the next set of neurons. Figure 2.6 shows the difference between a simple neural network and a deep neural network.

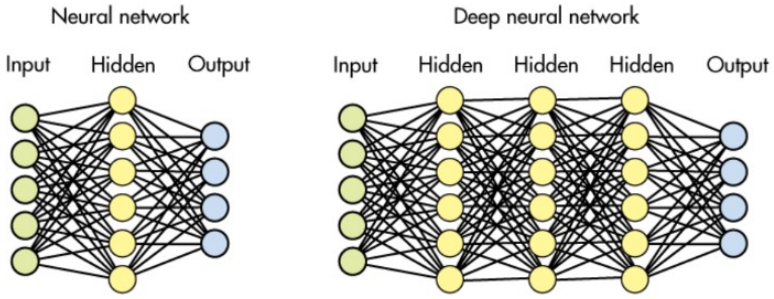

Figure 2.6: On the left, a simple neural network composed of an input layer, a hidden layer then the output layer. On the right side is a more complex deep neural network where multiple hidden layers are in presence. Credits : [16].

In NN, the layers and the units are connected to each other through links. Every link has its own weight. Then, one can add a bias, which is generally a small constant value that will be added to each inputs neurons in order to shift the activation function. By fine-tuning the network using training data (e.g., updating the weights and biases), the network is able to distinguish relevant features and to use them for classification or regression tasks.

Among the different types of other neural networks existing, Convolutional Neural Networks (CNN) are widely used for images processing. Figure 2.9 shows an example of CNN. Such networks are composed of different layers and operations :

- Kernel : a kernel can be compared to a filter that is used in the convolutional and pooling layers. Kernels are represented by a matrix (x, y). The size of the kernel represents the portion of the input that will be analyzed at each time. As an example, a common size for analyzing 2D images is a kernel size of $(2, 2)$. Figure 2.7 shows an example of a kernel used in a CNN;

- Stride : stride is the shift by which the kernel will be moved through the inputs. Depending on the selected stride, the new matrix analyzed can also contains information about the previous one (i.e., some pixels are in common) or being completely new information;

- Convolution : the goal of convolutional layers is to extract abstraction of the input using feature maps. The size of the extracted feature maps can be either higher or smaller than the input size. To do so, a kernel and a stride are used, these are the parameters that will influence the size of the feature maps. Feature maps allow to extract information from their input. In general, the first convolution learns the global input information (i.e., colors, edges, etc.) while the deeper convolution allows the model to have a global understanding of the images[17]. Each information extracted with the convolution is called a feature. Thus, the different convolution steps are performing feature extraction. This step is very important in machine and deep learning models since it allows the model to know which features are important and which are not. By extracting features, the network can also create new ones from what it has extracted thanks to another principle called feature construction (i.e., retrieve the missing information between features)[18];

- Subsampling/pooling : this operation is used to reduce the input size. This reduction is very interesting since the smallest the image is, the less computational power it needs. For each feature maps pooled, a new and smaller feature map is produced. Furthermore, pooling operation uses a kernel and a stride to know how to reduce the input. Two main types of pooling exists : Max Pooling and Average Pooling. Figure 2.8 shows these two pooling operations. For each analyzed matrix, the Max Pooling will keep the highest value, while the Average Pooling will keep the average value of the analyzed matrix. It is known that Max Pooling should be better to keep the intensity (e.g., the brightness of an image) of the analyzed input while Average Pooling

should be better to keep the context information of the input[19].

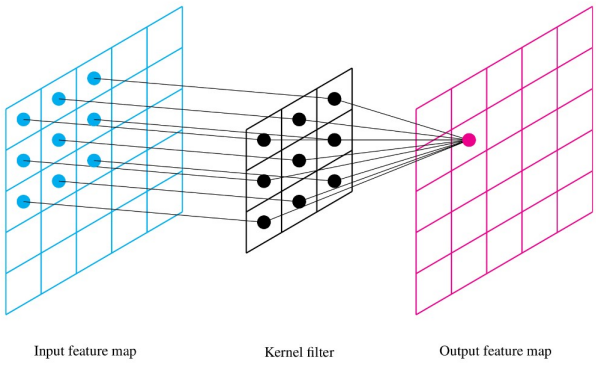

Figure 2.7: Example of a kernel with shape (3, 3). Due to its size, the kernel will parse the input feature map by squares of (3, 3) then will be strided to analyze the next portion of the image. Credits : [16].

Thus, in comparison with the networks presented in 2.6 where every neuron is connected to every other neurons in the previous and next layer, the different units of a CNN will not be connected to all units of the next layer. Each unit is connected locally with other units and learn its own features. By learning these different features and merging them, the network is able to use all of them with fewer connections, this principle is called weight sharing. There are different types of CNNs, in the field of conventional radiotherapy, a specific CNN architecture has become popular : U-Net[20] which is designed for image segmentation and has been extended over the years.

Otherwise, a CNN is not different from another NN, it must also be trained. Once the model is designed, some parameters have to be defined. First of all, one will choose a number of epochs which correspond to how many times the model will see the entirety of the dataset. A good practice is to run a k-fold validation (e.g., 5-fold validation) which consists of training k models to limit the randomness that might have arisen during training.

In addition, one must define what is called a loss function. The goal of this function is to compute mathematically the proximity between the model prediction and the real data. In general, the model will try to minimize this score during the training. For instance, Mean square error (MSE) is a well-known loss function. These will be discussed later in chapter 4.

To perform that kind of model training and validation, the dataset will be split in three categories : training, testing and validation. These three categories of dataset

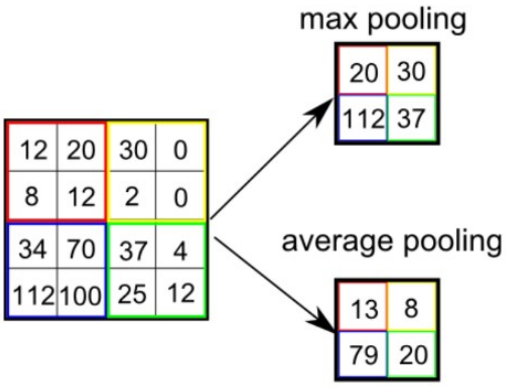

Figure 2.8: Example of average versus max pooling. One can see that with a kernel size of (2,2), max pooling takes the maximum value of the analyzed feature map, while average pooling takes the average of the analyzed feature map. Credits : [16].

will be used to respectively train, test and validate the model. Some good practice should be done to limit overfitting[1]. At each fold, a certain number of data will be set aside and not used at all during the training. These data will compose the validation dataset. Its goal is to validate the performance of the model when it receives brand new data. What is left from the data will be split again in two datasets : a training set and a test set. At each epoch, the model will be trained using the training dataset and tested with the test dataset. Once all the epochs are done, the model will be validated using the validation dataset. Each fold will repeat these steps. This is a first way to limit overfitting. A second way is to shuffle the dataset between folds (i.e., in order to be sure that the split is not always the same) but also to shuffle the training dataset between each epoch (i.e., to be sure that the model will not always see the same data in the same order).

Note that, even at this step, some more operation could have been performed. Indeed, if one wants their model to be accurate, then the dataset itself must be accurate, thus it should have been cleaned of incomplete data (i.e., some fields are empty, inexact). Besides, one could also think about data augmentation, it consists of generating new data from the existing ones. For instance, if one is working with a dataset of 40 images, by simply adding the mirrored images to the dataset, it becomes a dataset of 80 images. One could also rotate the data, scale them, etc. Data augmentation is a robust technique that improves the models

[1]Overfitting is one of the biggest disadvantages of neural networks. When a model suffers from overfitting, it means that it has learned too much from the training data and is not able anymore to generalize what it has learned. It leads to very good training performances but poor validation performances.

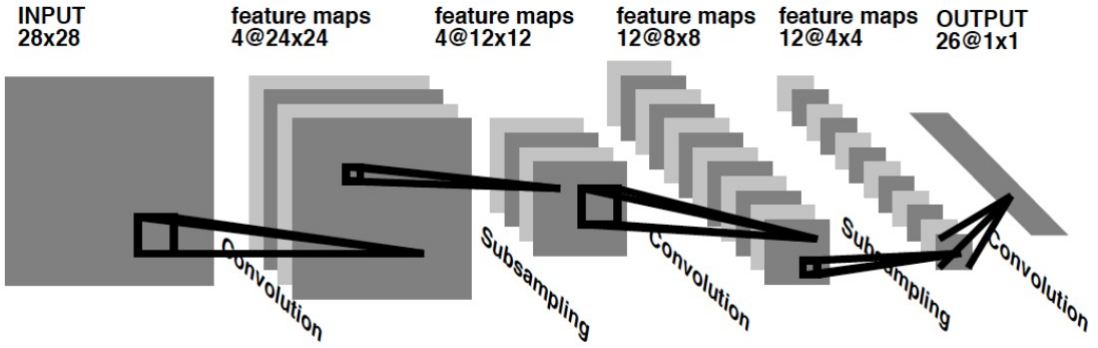

Figure 2.9: A CNN network example used for handwriting recognition. The first layer is the input layer of size 28x28, then a first convolution is used resulting in 4 features maps of size 24x24. After that a subsampling (also called pooling) is used with a kernel size of (2,2) and a stride of 2. These operations are repeated until the final layer which is a vector composed of 26 neurons of size 1 that allow the prediction. Credits : [17].

performances[21].

These steps may already be complex enough, but there is more. Indeed, at this point the model has not learned anything yet. In order to learn, the model should be able to use the information it has produced and learn from it. This is where backpropagation comes in. The backpropagation algorithm allows the model to compute the gradients of the loss function in regards to the weights of the links between each unit. The gradients are then propagated back to the network allowing every weight to be updated. Some optimizer can be used to optimize this gradients descent like the Adam optimizer which is discussed later in chapter 4.

Other techniques can be used to improve the network accuracy, one of them is regularization. The goal of regularization is also to minimise the overfitting of the network. One well-known regularization technique is the dropout. The idea behind it is to randomly drop neurons from the network during training. Thanks to that, the units can not co-adapt between themselves too much[22]. Indeed, the co-adaptations between the units lead them to be useful only in the presence of the other units with which they have adapted. However, one wants each neuron present to be as helpful as possible independently of the others. The figure 2.10 illustrates the comparison between a fully connected neural network and a neural network with some of its neurons dropped with a probability of 0.5.

The last step concerning neural network is maybe the most time consuming one.

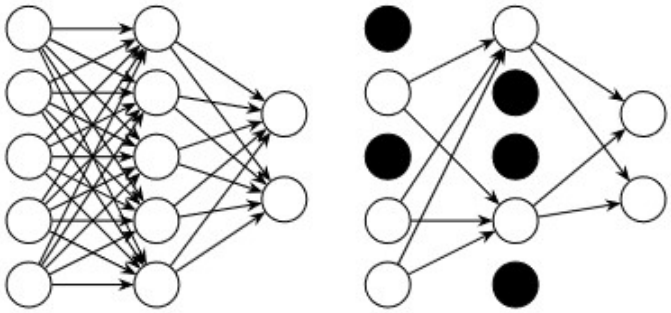

Figure 2.10: The left model is a standard fully connected one while the right model has some of its neuron dropped ($p = 0.5$). Credits : [23].

For every parameter that is needed through the network (i.e., the kernel size, the stride, the dropout probability, etc.), one must set these parameters. In the classical workflow of machine learning and deep learning, this step is usually done using what is called "grid search". To perform this search, the possible values for each parameter should be determined. For each combination of parameters and values, a new model should be trained. Then, at the end, the best model is kept. This process takes a lot of time since one model is run for every parameters combination. Furthermore, it assumes that one knows in advance some of the possible values for each parameter. Moreover, this process can be applied not only to the parameter of a network, but also for the different types of models one wants to try. A solution to this parameter tuning problem is automated machine learning (AutoML).

2.3 AutoML

AutoML is a relatively new way of making machine learning and deep learning models. The first time this notion was mentioned was by C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown in 2013 [24] when writing about Auto-WEKA (Waikato Environment for Knowledge Analysis). The goal of Auto-WEKA is to tackle the problem of algorithm selection and parameters settings as a common task instead of doing them separately. In this regard, one can say that AutoML is a way of doing machine learning where the whole classical pipeline of machine learning and deep learning (i.e., data preparation, feature selection, model selection, etc.) is considered as one big problem instead of a combination of smaller problems.

Nowadays many enterprises take interest in AutoML[2][3][4]. Seeing these big companies take an interest and invest money in AutoML, one can find obvious that it is a very interesting topic and maybe even the future of machine learning and deep learning techniques.

However, business is not the only area that takes interest in AutoML, many researchers also try to develop new models or adapt some existing one to this AutoML style[25]. Among these, some are well known and used like nnU-Net[13] which is an image segmentation algorithm that already helps a lot in the automation for a particular task of the basic pipeline presented before in 2.1.

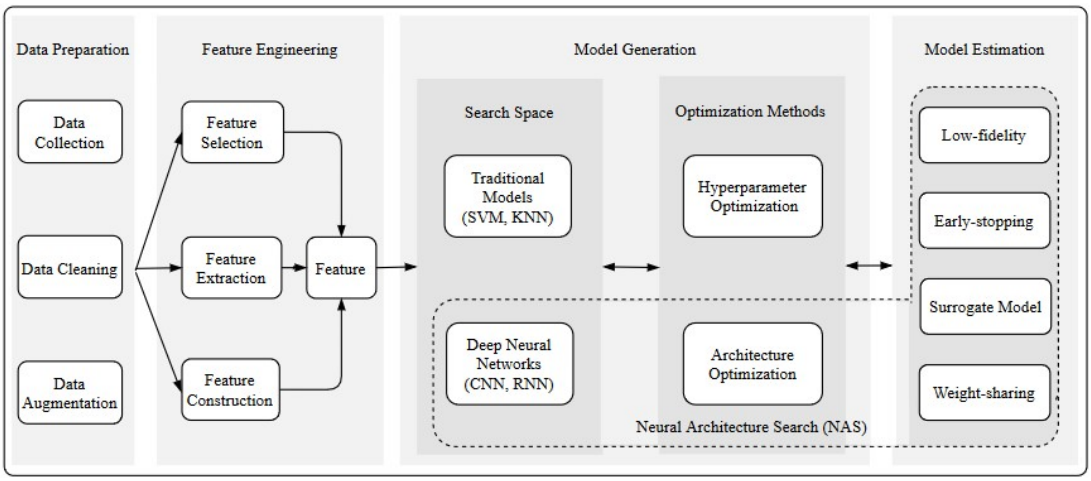

Figure 2.11: AutoML pipeline as shown in *Automl: A survey of state-of-the-art*[25].

An ideal AutoML framework should probably do all the steps referenced in the figure 2.11, but for now, due to a lack of computational resources, it is only realistic to choose a subset of these. The reason why AutoML is useful is that the vast majority of nowadays machine learning and deep learning models are task-specific, and to be even more precise : data specific. As a result, even the smallest error can lead to a drastic loss of performances of the model. Thus experts, experimentation and time are needed every time one of those models have to be adapted to a new set of data. This is even more true when the model has to handle 3D images where you have different images sizes, voxel sizes, modalities (CTV, PTV, MRI, etc.).

Going back to the principles of AutoML, this thesis focus on the following ones : Data Augmentation, Deep Neural Network and Architecture Optimization. Some

[2] `https://cloud.google.com/automl`
[3] `https://azure.microsoft.com/en-us/services/machine-learning/`
[4] `https://aws.amazon.com/fr/sagemaker/autopilot/`

other concepts are also used :

- Dataset fingerprint : "*a standardized dataset representation comprising key properties such as image size, voxel spacing information or class ratios*"[13];
- Pipeline fingerprint : "*which we define as the entirety of choices being made during method design*"[13].

Concerning the hyper-parameters of the model, there is a distinction between three types. They are defined as follows :

- Fixed parameters : common robust parameters settings which are coming from previous experimentation in the domain;
- Rule-based parameters : as many of the remaining parameters as possible, the idea is to try to find heuristics to define them according to both fingerprints mention earlier;
- Empirical parameters : as few as possible, it represents the remaining of the parameters. They are set after experimentation.

3 | State of the art

Neural Networks, or more generally machine learning and deep learning are seeing more and more uses lately. From MuZero[26] in board and video games to nnU-Net[13] in medical image segmentation passing through ResNet[27] for image recognition.

The evolution of hardware allowing more powerful models is part of the reason. This section will present the state-of-the-art models and the framework that were used during this master thesis.

3.1 U-Net

U-Net networks were designed by Ronneberger, Fischer, and Brox (2015)[20]. The name comes from the particularity of this kind of network which is its U shape. An example of this network can be found in figure 3.1. They are characterized by three main parts :

- **Encoder** : also called downsampler, the purpose of the encoder is to decrease the image size, using stride or pooling operation during the convolution while the number of channels increases. This process allows features extraction. The original encoder is a ensemble of convolutional layers followed by a ReLU activation function using max pooling. This part is the descending part of the U;

- **Bottleneck** : represented by the lower horizontal part of the U and following the encoder it receives only compressed data. Having only this kind of data allows it to extract the most important information from the data in order to help provide a reliable output;

- **Decoder** : the decoder, sometimes named upsampler, is the ascending part

of the U. It is the part responsible to construct the output. At each step, it makes the reverse operation of the encoder at the same level. Thus increasing the image size back while decreasing the number of channels to produce the final image. Some more steps can be added to the decoder, like concatenations of previous skipped connections. Having these concatenations allows to gain back the information that was lost because of the pooling.

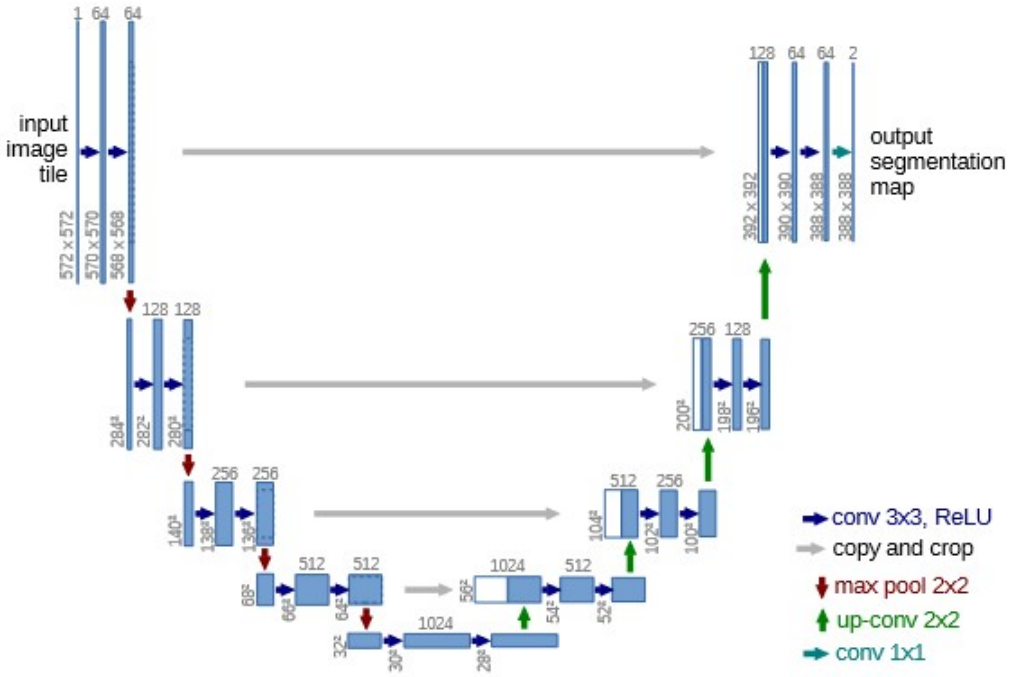

Figure 3.1: Original U-Net network architecture. Credits : [20].

3.2 nnU-Net

nnU-Net is the state-of-the-art AutoML segmentation model presented by Fabien Insensee in 2020[13]. It combines a U-Shape network as proposed by Ronneberger and al. [20] but it includes an AutoML part which makes it user-friendly for new datasets. It includes dataset conversion, plan generation for the dataset, different possible training (i.e., 2D, 3D full resolution, 3D cascade full and low resolution), model selection and inference. It also incorporates many pre-trained models to test or use on common datasets. nnU-Net[13] uses the structure shown in figure 3.2 for its parameters.

While solving the problem of model designed with only one dataset in mind, it also outperforms such models. The results are available in Fabien Insensee's thesis [13].

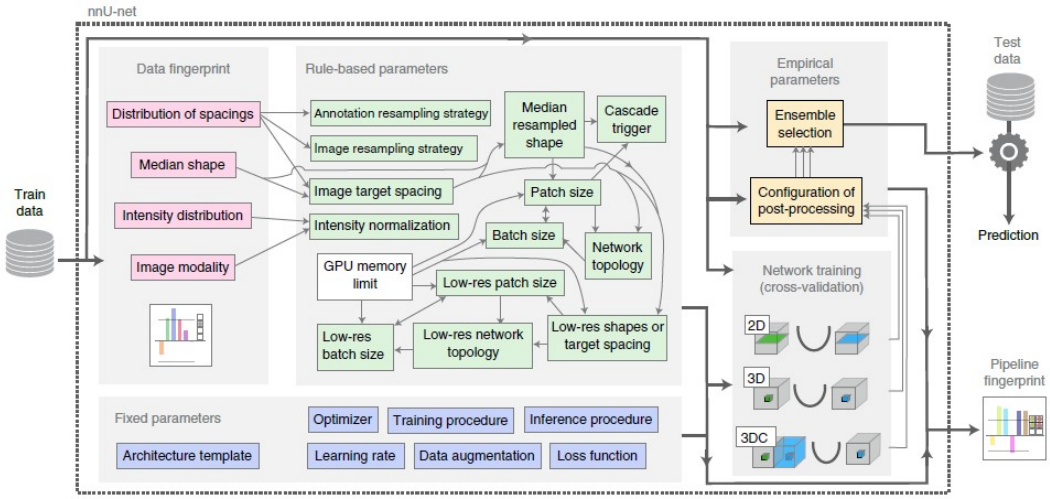

Figure 3.2: Overview of nnU-Net parameters. The arrows represent dependencies. Credits : [13].

3.3 HD-UNet

HD-Unet stands for Hierarchically Densed U-Net as proposed by Nguyen, Jia, Sher, Lin, Iqbal, Liu, and Jiang (2019) [28]. As shown in figure3.3, it is an extension of the classical U-Net architecture[20]. The difference with the classical U-Net[20] architecture is the presence of dense connections which are dense convolutions, dense downsampling and dense upsampling. In these connections, the input of the layers are concatenated with the feature maps produced by the convolutional layer.

During the academic years 2018-2019, as part of her thesis[4], Margerie Huet Dastarac has designed an extension of the classical HD-UNet[28], using Keras and Tensorflow, specialized for dose prediction in proton therapy. This version allows the 3D dose prediction of patients while the input is the set of dose prescriptions on target volumes, the organs at risks as a binary mask and some information on tissues density. At that time, it was the only deep learning model able to do dose prediction for proton therapy and although the research is still ongoing in this field we did not find any other model specialized in proton therapy during our research. Nowadays, this version of HD-Unet is used and maintained in the MIRO's lab.

The most important parameters of the modified HD-Unet[4] are the following : MSE as loss function, ReLU as activation function, Max pooling as part of the dense convolution operation, Dropout and Adam optimizer.

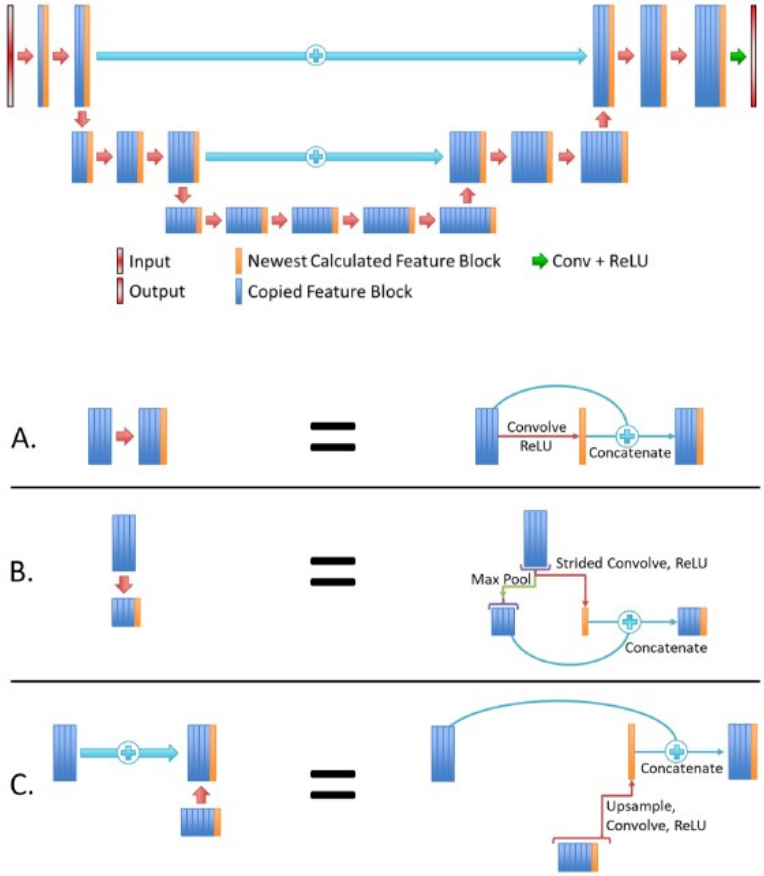

Figure 3.3: HD-Unet's architecture. On the upper side, it is its global structure. The bottom side represents from top to bottom : dense convolution operations (red arrow), dense downsampling (orange arrow), dense upsampling with connection from the skip connection concatenated. Credits : [28].

The present thesis implementation was based on this model, first by converting it in Pytorch, then adding modularity to it and adding the AutoML part.

3.4 Pytorch

Pytorch[5] is one of the biggest machine learning and deep learning framework available nowadays. Released in 2016, it quickly gains respectability among the machine and deep learning community and the research field. Alongside libraries

[5] `https://pytorch.org/`

like Tensorflow[6] and scikit-learn[7], it allows its users to design models, train and test them.

As many others, Pytorch interprets designed models as directed acyclic graph (DAG). However one of its biggest strengths on its release time was the fact that this DAG is built during runtime : nodes can be defined, changed and executed while running. Some kind of models (Recall RNNs, Recursive RNN, tree-RNNs) takes a great advantage of that dynamic approach. Since then, other frameworks have been updated themselves and now allow dynamic DAG construction. This dynamic approach also allows a nice and easy way to debug code using classical python debugger without having to use two separated debuggers, one for the model and one for the code.

Besides, Pytorch has a very "pythonic" way of doing different stuff, it does not feel like using a framework since it does everything like you would do in classical python. Thanks to this "pythonic" approach and its easy debugging, Pytorch allows a great control over its parameters and the model, this is very convenient when it comes to AutoML. Indeed, being able to modify parameters easily is a great advantage for an AutoML framework where some of them will be dependent on heuristics. Furthermore, being able to modify the network architecture at run time allows the needed flexibility in the parameter choice but also allows developers to try parameters "on the fly" using the debugger.

Nowadays, we can see a global shift from the scientific community from Keras and Tensorflow to Pytorch, mainly thanks to the said control allowed over the parameters. The original HD-Unet[4] code was written in Keras and Tensorflow, after some discussions and research the best option would be to implement again HD-Unet but switching its implementation to Pytorch. There are two main reasons for that shift :

- Many researchers have already switched to Pytorch, therefore extending the possibilities proposed by the library;
- nnU-Net[13], which is a state-of-the-art AutoML model, is also written in Pytorch. It would be more convenient to adapt what makes it an AutoML model if both software were written in the same way.

[6] `https://www.tensorflow.org/`
[7] `https://scikit-learn.org/stable/`

4 | Methods

4.1 Network Architecture

The basis of the network architecture is directly taken from the work that has been done on HD-UNet[4] and the default parameters used in the network presented in this thesis are those that have been used before. HD-UNet is an adaptation of a hierarchical density U-Net that aims to perform 3D dose prediction in the context of radiotherapy. While most of the parameters rest unchanged, some of them can be tweaked thanks to the AutoML adaptation. For instance, originally HD-UNet is a 4-layer (also called stage in the context of this thesis) U-Net[20] and with the AutoML framework, the number of layers used is one of the parameters which could be adapted. A graphic representation of the parameters and how they are defined is presented in figure4.3.

4.1.1 Fixed and empirical parameters

The fixed parameters stay the same regardless of the dose predictions tasks and for all datasets. They were chosen with respect to the work and experimentation that have been done on HD-Unet. The most important parameter that stays the same is the Convolutional Neural Network (CNN) architecture which is itself an adaptation of a U-Net[20]. Note that, other types of architecture could be possible like it is done in nnU-Net[13](i.e., 2D, 3D, cascade, etc.). On the other hand, empirical parameters are the ones learned thanks to experimentation and analysis of different results.

- **Number of initial filters** : it refers to the number of features map that are produced by the convolutional layer initially. Feature maps contain local features/information (e.g., presence or not of dark edges) of a given input. This number can change from one stage to another depending on the expansion rate which has been defined. The value will be updated according

to the following equation 4.1. In this implementation the number of initial filters is fixed to 16.

$$\text{num\_filter} = \text{round}((\text{expansion\_rate}^{\text{stage}}) * \text{initial\_num\_filter}) \tag{4.1}$$

Equation 4.1 : Expansion rate formula

- **Loss function** : given that HD-Unet exists to solve a dose prediction problem, it means that the output values are continuous and therefore, it also means that this is a regression problem. Mean Square Error (MSE, see 4.2) and Mean Absolute Error (MAE, see 4.3) are the more commonly used cost function for regression problems in Deep Neural Network (DNN). However they both have their disadvantages. The first will give too much importance to the outliers since the larger the error is, the more it will be valued. This is a consequence of the squared operation applied in the equation 4.2. The latter can face convergence problems due to the fact that the gradient magnitude will be large even when the error is small. Consequently, MSE will perform best when outliers have to be accounted with a great importance or if they can be removed beforehand. And MAE will be best with outliers that should be ignored. The best lost function for this application seems to be MSE [4].

- **Activation function** : Rectified Linear Units (ReLU) are the activation functions used in HD-Unet[4]. The only major difference with the basic linear units is that the ReLU do not take into account negative entries by having half of its domain returning zero. It has been shown empirically that network using ReLU tend to have a more reliable and faster training than networks with tanh activation [29].

- **Pooling** : in a U-Net, the purpose of a pooling layer is to reduce the dimensionality of the data given as input in order to pass it to the next level (also called stage for the purpose of this thesis). One of the most frequently

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{4.2}$$

Equation 4.2 : MSE equation, Y_i represents the true value and $\hat{Y}_i$ represents the predicted value.

$$MAE = \frac{\sum_{i=1}^{n} |Y_i - \hat{Y}_i|}{n} \tag{4.3}$$

Equation 4.3 : MAE equation, Y_i represents the true value and $\hat{Y}_i$ represents the predicted value.

$$f(x) = \max(0, x) \tag{4.4}$$

Equation 4.4 : ReLU activation function equation. Where x is the input of a neuron.

used methods for CNN is Max pooling. Max pooling runs a filter over an image and extract the highest value from the said filter. While Average pooling, another frequently used pooling, would provide a better idea over the context of the input by taking the average of all values present in the filters. This context's information could be thought necessary, however, it has been shown[4] that the use of an average pooling does not bring any real improvement in the proposed architecture.

Given that convolutional layers allow to produce maps and the use of pooling layers between some of them (i.e., between each stage see 4.1), the network is able to produce feature maps and reduce their dimensions right away. Meaning that the number of channels (i.e features) is increasing while the dimension is decreasing.

There is a direct link between the number of pooling applied during the downsampling and the number of scaling factors used by the upsampler in the upsampling.

- **Regularization** : the goal of the regularization is to improve the network by reducing overfitting. The dropout layer[22] will randomly drop (i.e., deactivate) neurons creating a smaller version of the original network. The proportion of neurons that are dropped can be defined within a range of [0->1], 0 means that no neurons are dropped and 1 that all of them are dropped. Thus, each training will try to train a new random architecture.

  Concerning the value of the dropout rate, we have tested it empirically ranging from a probability of $[0 \rightarrow 0.5]$. It seems that a dropout rate of 0 works best

with our configuration. The results of the tests can be found in the chapter 5.

- **Optimizer** : optimizers are algorithms used to minimize the loss function. To do so, they update the trainable parameters such as the weights and the biases of the network's units. Nowadays, one of the most commonly used algorithms is Adaptive Moment estimation (Adam)[30]. Adam is an adaptive learning rate method, meaning it computes different learning rates for each neuron and hidden layers. Adam takes the best of what has been done by using dynamic learning rate and storing the decaying average of the past gradients like Momentum[31] would do, but also the decaying average of the past squared gradients in a similar way to Adadelta[32].

4.1.2 Rule-based parameters

The Rule-based parameters are defined using heuristics that have been formulated according to the relations between the dataset fingerprint and the pipeline fingerprint. These heuristics can be more or less complex. For example, the number of pooling layers depends on the resolution of the image or the patch.

- **Batch size** : the number of examples used for training at each step of an epoch. There is a known trade-off with the batch size, the memory consumption and the accuracy of the gradient. More often, one would set the batch size to a small value because the memory can not handle the full dataset, henceforth the training is less accurate. Meaning that the ideal case is to have a batch size equals to the size of the dataset in order to have the best accuracy. However it is not achievable yet. Furthermore, given that we should use all the data available (i.e., the one from the train set) at each epoch of the training, the batch size has a direct impact on the number of steps per epoch. Thus the smaller the batch size, the greater the number of steps needed to complete a single epoch of the training.

- **Number of convolutional layers per stage** : each convolutional layer will increase the depth of the image as the features maps it produces is concatenated to its input and is passed to the following layer. By default and as shown in 4.1, this number is set to 2 in order to mimic the original U-Net and HD-Unet architecture.

- **Number of stages** : the number of successive stages present in both the encoder and decoder of the architecture. Since a pooling operation is applied after each of the decoder's stages, it also impacts the number of pooling layers present in it. Meaning that if there are more stages, the dimension of

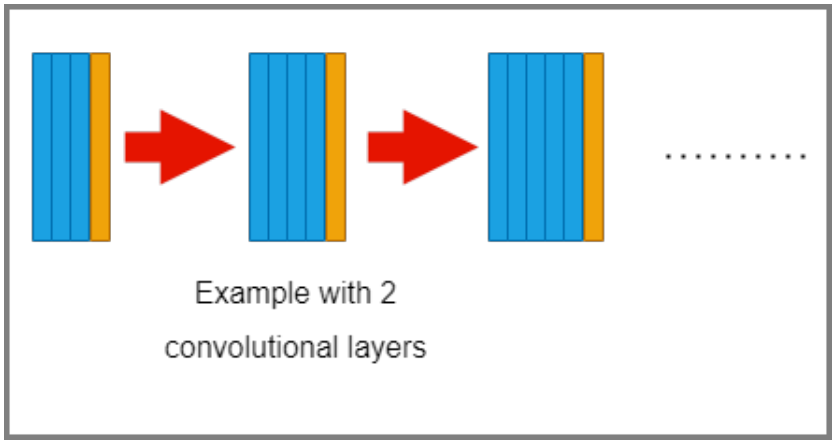

Figure 4.1: Illustration of a single stage with 2 convolutional layers (same convention as figure 3.3). A stage can have more convolutional layers if necessary.

the input will be lower before going into the bottleneck. As shown in 4.2, the upsampling part possesses the exact same number of stages than the downsampling, there are no pooling layers in this last part, but there are upsampling layers which are the one also influenced.

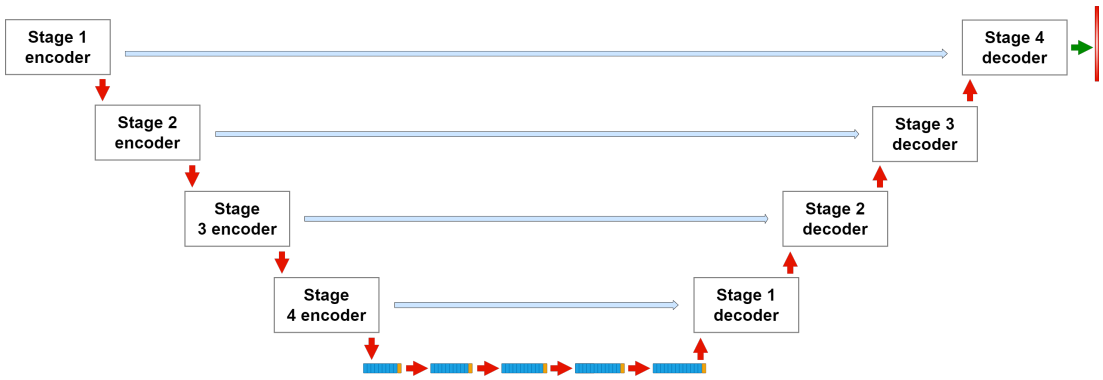

Figure 4.2: Illustration of the architecture using 4 stages as described in 4.1 (same convention as figure 3.3). The number of stages in the encoder and decoder are the same. The number of stages can be adjusted easily, allowing to have more or less than 4 stages.

- **Kernel size** : this is the size of the filter matrices applied over an input to produce a feature map. Note that by default the padding is set to 'same'. The kernel size is dependent on the resolution of the image.

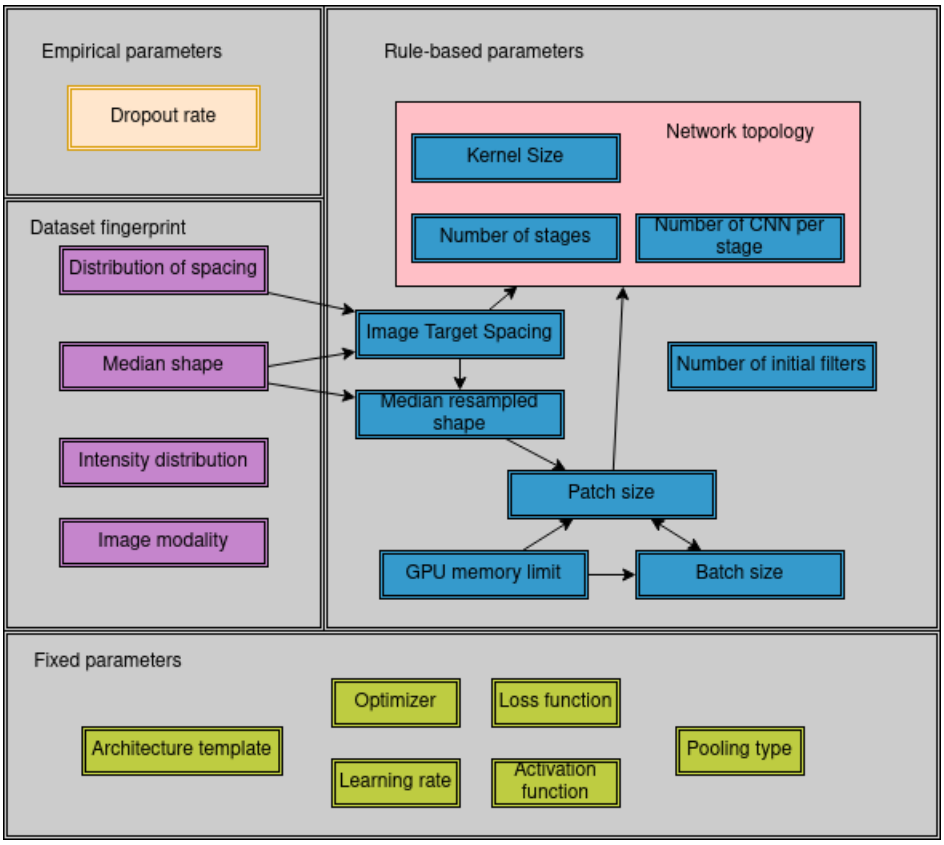

Figure 4.3: AutoML HD-Unet parameters. Each box represents a part of the model. Green boxes are used for fixed parameters which are from state-of-the-art literature or initial HDU-Net[4]. The dataset fingerprint box in purple shows the information gets by the plan operation. In blue boxes there are the rule based parameters with the network topology that is used to group some of them, finally in orange there are the empirical parameters. The arrows represent the influence of each parameter on the others.

4.2 Data

Several different datasets were used during this thesis, the first one is the one used to train the original HD-Unet model[4]. It has been used to develop and to adapt the whole base of the AutoML model. This dataset contains 60 patients suffering from Head And Neck (H&N PBS) cancer.

The data were initially pre-processed and stored in a specific way (MIRO format) and were loaded from different files as needed. This is due to the fact that the memory resources of that time did not allow to load all the data in their totality at the same time. Some pytorch libraries already use datasets in order to handle their data. While the said libraries are not used in this thesis, when the code has been converted from keras to pytorch, the idea of using dataset has been borrowed from them. Hence, a way to transform the data in a single dataset that could be loaded only once before training has been figured out. This is a part of the pre-processing that is done in our pipeline and it is applied to all new dataset as needed.

Afterwards, several other datasets have been used whether to validate our model or to define certain values empirically. Theses dataset uses different treatment technologies, this is interesting as it allows to verify how the model performs with them. These datasets use multiple technologies :

- **Intensity-Modulated Radiotherapy (IMRT)** : "*A type of 3-dimensional radiation therapy that uses computer-generated images to show the size and shape of the tumor. Thin beams of radiation of different intensities are aimed at the tumor from many angles. This type of radiation therapy reduces the damage to healthy tissue near the tumor.*"[33];
- **Volumetric Modulated Arc Therapy (VMAT)** : VMAT is a more advanced IMRT technique as the treatment machine, which delivers the dose continuously, rotates around the patient;
- **Pencil Beam Scanning (PBS)** : Uses a beam directly pointing to the tumorous cells.

Finally, here are the datasets :

- H&N PBS (60 patients, 40 train, 10 test, 10 val);
- esophagus PBS (40 patients, 30 train, 5 test, 5 val);
- esophagus IMRT (40 patients, 30 train, 5 test, 5 val);

- H&N VMAT (50 patients, 5 test, 5 val);
- Prostate IMRT (45 patients, 30 train, 10 test, 5 val).

4.3 Dataset analysis

One part of the AutoML pipeline that was chosen is to automatically adapt the model to the dataset given by the user. The segmentation model nnU-Net[13] has a whole part of their project for that specific task. After some discussions with the supervisors of this thesis, it was stated that segmentation and dose prediction problems can be approached in the same way for the following parameters :

- Batch size;
- Convolution kernel size;
- Bottleneck kernel size;
- Number of stages;
- Pooling kernel size.

Thus, for the dataset analysis part, this implementation uses the software from nnU-Net[13].

However, nnU-Net[13] uses its own data format[8]. Thus, it was needed to convert the data from the MIRO format to the one used by nnU-Net[13]. In order to do that, tools were developed. A converter goes through all the data, extract the properties of the dataset (creating a `dataset.json` file) and convert every patient files to the nifty format. For this to work, a pull request to the nnU-Net repository had to be made since its original code had a bug during the generation of the dataset properties. The bug had an impact during the rest of the process[9]. In order to use this converter, one can simply call the command `hdunet_convert_dataset_MIRO_to_nnUNet` which allows the user to convert its data in a smooth manner.

[8] `https://github.com/MIC-DKFZ/nnU-Net`
[9] `https://github.com/MIC-DKFZ/nnunet/pull/1005`

4.4 Dataset generation

As stated earlier in point 4.2, the format used by the original data was a combination of files for each patient, containing its different information: arrayshape (shape of array used for the patient), organs at risks (where they are and how many they are), CT (CT scans), CTV (Clinical Target Volume), dose (the ground truth dose), structure and voxel size.

At each iteration of the training, when the information of patients was needed, the original HD-Unet[4] was retrieving the data from the files, loading them each time. Again, this is due to memory limitations at that time. The model could not load the entire dataset and deal with it as it comes, instead they needed to load the patients' files, convert them to the correct structure used by the model, make the needed operation (e.g., data augmentation) and then delete the variables from the memory.

This model optimizes this part of the process. Indeed the available hardware is much better now than what it was when HD-Unet[4] was originally designed. A much higher amount of RAM and VRAM are now available. This leads to the creation of one dataset containing the information of every patient.

The command `hdunet_convert_dataset_MIRO_to_HDUnet` will automatically go to the patient folder, load every patient only once and add each patient to a dictionary using the following structure dict<patient_key, dict<property, value> >. It is then saved, creating the dataset in a folder specified by the user.

This dataset is then used in the model. When a patient's information is needed, the code can access them in $\mathcal{O}(1)$. Thanks to the double dictionary structure, it is also the same complexity for accessing patient properties. This has significantly reduced the training time, as the model is no longer dependent on disc reading speed anymore. Besides, the patients' information can be accessed at any time in $\mathcal{O}(1)$ as the dataset is loaded once at the beginning of the training. This allows a gain of 33% for the training time. This can be summed up as one more model run every day as the time decrease from ±9H30 of training to ±6H30 (using the H&N PBS dataset).

4.5 Batch and patch generation

Ideally, one should feed the entire image to the model in order to have as much context as possible. However, it is not possible due to a lack of resources. This

is why patches are useful. A patch is a subset of an image and by merging these given sets of patches, we could recreate the original image. Meaning that we could train on each of these patches and it would be like training on the whole image.

There is a compromise to make between the number of batches (or batch size) and the patch size (dimension of the patch). When one has more computational resources, he or she could exploit them in several ways, including increasing the number of batches use at each step. But he or she could also keep the same batch size and increase the size of the patch that will be fed to his or her model; thus the image fed is bigger and the model has more context.

4.6 Data Augmentation

The lack of data is a recurrent problem in the domain of machine learning, indeed, training a model with only a few data will not allow it to generalize really well. Medical images are not an exception to this problem which directly involves our dose prediction problem. Optimally, the model should be fed with thousands (or even more) data.

While some techniques to face the problem of generalization already had been explained in 4.1, none of them are directly facing the number of data given to the model. This is where data augmentation comes. As its name suggests, the purpose of the data augmentation is to increase the amount of data available. To do so, it creates brand new modified data from the existing ones by applying some transformations to them. These transformations could be spatial (or geometric) transformations such a translation, rotation, cropping and so on or act as filters applied to the data in order to add some noise, contrast or blur to the image as an example.

As always, data augmentation should be used sparingly in order to keep the produced images as realistic as possible. Especially when working with images of the human's body. For example, one could not always mirror every image, it depends a lot on which part of the body is concerned. To give a more precise and concrete example, in this thesis, a dataset containing images of heart was used. In this case, mirroring should be used with caution; as the processed image will not be anatomically correct (for most of the patients at least).

Therefore, the model will have more data to train and will not overfit as much as before on the train set. In case of this thesis, the data augmentation is applied to all the batches (if there are more than one) meaning that the original input is not directly kept, it could still exist but with a low probability. However, data

augmentation has some disadvantages, one of them being that more computations have to be done and it can become computationally heavy and take more time to train the model.

To tackle this problem, the used of multithreading has been proposed. Indeed, since data augmentation is mostly computation over matrices, one could use multiple threads/CPUs which are convenient for this type of task. Allowing more threads to work on this task permits to remove some workload of the GPU and let the GPU focus more on the training part. For example, if the data augmentation is done with only one thread, the GPU will have to wait for the CPU to do his job, meaning it won't work 100% of the time. This is a waste of resources, one wants the GPU to run all the time, this is how the use of multi-threading could help to solve this issue. Having more threads working on the data augmentation allows the GPU to work, in the best cases, all the time, leading to a huge gain of time in the end.

The parameters do not change from one dataset to another. The data augmentation pipeline is part of the fixed parameters just like the network architecture described earlier. The data augmentation has been implemented using the code from the *batchgenerators*[10] framework with the only exception of the Gaussian Noise that has been adapted to suit our problem. Here are the data augmentation techniques applied on the dataset :

1. Rotation and Scaling : Rotation and scaling are applied at the same time with an independent probability to be applied of 0.2. The scaling and rotation are applied independently on each axis, meaning that each axis (X, Y, Z) can have different values for scaling and rotation angle. The angle of rotation for each axis is picked at random in $[-11,46° \rightarrow 11,46°]$, on the other hand, the scaling factor is picked at random in [0.9 (zoom out) $\rightarrow$ 1.1 (zoom in)];

2. Mirror : The mirroring is applied with a probability of 0.2 and is applied over all the axis;

3. Gaussian Noise : Gaussian Noise is applied with a probability of 0.2. The variance of the noise is picked at random in $[0 \rightarrow 0.1]$. Unlike the one proposed in *batchgenerator*, it is not applied to all the channels of the input, but only the CT channel.

[10] `https://github.com/MIC-DKFZ/batchgenerators`

4.7 Training

The CISM/CECI was used to run the training, this cluster has multiples available GPUs and a total of 7072 CPUs[11]. Each GPUs have access to only a certain number of CPUs so the choice of the GPU must be in accordance to the CPUs needed. Here is the list of available graphical hardware:

- 2 Nvidia Tesla A100 80 Gb having access to 64 CPUs and ±257 Gb of RAM;
- 6 Nvidia Tesla A100 40 Gb they are grouped by two and each group has access to 64 CPUs however two groups have access to ±515 Gb of RAM while the third one has access to ±257 Gb of RAM;
- 4 Nvidia GeForceRTX 3090 having access to 96 CPUs and ±515 Gb of RAM;
- 6 Nvidia GeForceRTX 2080Ti having access to 32 CPUs and ±385 Gb of RAM;
- 4 Nvidia Tesla V100 they are grouped by two, one group having access to 32 CPUs and the other one has access to 16 CPUs while both have access to ±386 Gb of RAM.

These GPUs and CPUs are organized as room, each of them containing a certain number of GPUs and CPUs. For example, the two A100 80 Gb are contained in one room having access to 64 CPUs in total. If one need to request more CPUs, more room will be reserved for the job. Thus, the more computation power is requested, the more waiting time one would have for his or her job to be launched. That's why, in the context of this thesis, in the end only one GPU was requested (whatever the one it is). 16 CPUs were also requested which is the number found to be a great compromise between requested resources and computational power.

A part of AutoML evolves with the available hardware. The big advantage of this is that the model can take advantage of whatever power is in his presence. However, this thesis implements a limited hardware scaling as it uses the CESI/CISM cluster and most people that will use the developed model are also gonna use it. Indeed, despite the powerful hardware available on the cluster some dataset where simply too memory dependent using the computed patch size, thus the request of some specific GPU was needed. A more robust hardware scaling could be an evolution of the model and it is discussed in chapter 6.

[11] The exact number of CPUs changes a lot. New ones are added regularly. This number is the one of late May 2022.

When a job is submitted to the CESI/CISM cluster one can select what are the resources needed, so one can assume, at least in first time that the model will always, at least have the needed calculation power. Indeed, by default only one node is rejected in the submission script, the "least good" GPU it can get is an Nvidia RTX 2080Ti. It is already a very powerful piece of hardware and sufficient for some datasets. If more resources are needed, one can modify the submission script and request a more powerful GPU. Thus, it was decided to let the hardware scaling part as a possible evolution of our model.

4.8 Pipeline

The AutoML model developed in this thesis allows a smooth pipeline execution. It consists of six steps where four are needed only once, even if one wants to train the model multiple times. To make it easier some command line scripts were developed and are usable when HD-Unet[4] is installed with pip. Examples of these instructions can be found in appendix A.1. Figure 4.4 is a quick view of the pipeline execution. It is also explained from point 4.8.1 to 4.8.7.

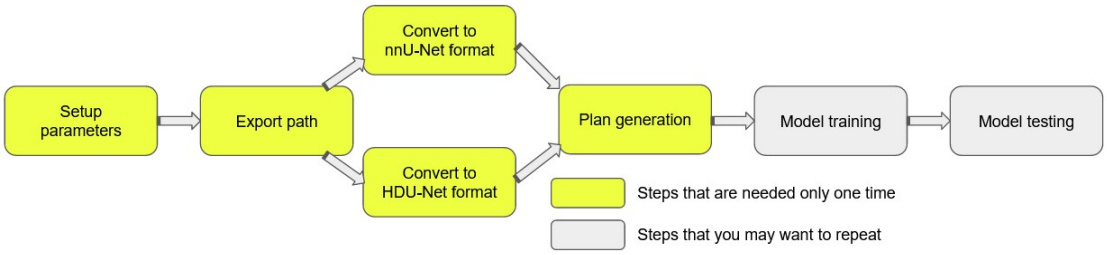

Figure 4.4: Pipeline execution. Yellow boxes represent steps needed the first time the pipeline is run. Grey boxes are steps that can be repeated. When boxes are on the same vertical level, one can do these steps in any order. Arrows represent the order in which the steps must be done.

4.8.1 Parameters set in the file `hdunet/parameters.py`

There are several parameters you must set:

- `datafolder` : The path to the dataset processed in the manner described in 4.4 is expected. This parameter is used during training;
- `output_folder` : The path where the results of cross-validations and testing will be stored. This is used during training and testing;

- `task_name` : The task name, it must also contain the task id. nnU-net[13] advises to start numbering at 500 to avoid any possible conflict with the pre-trained model. Also, the same task id must not be used repeatedly for different planning. Used for the plan generation;

- `images`, `labels`, `ctv` : All three parameters are used for the conversion to the nnU-Net[13] format. They are classical boolean variables, having the value "True" to these parameters will allow the conversion for that particular parameter. Defaults values are respectively "True", "True" and "False";

- `n_cpus` : How many CPUs will be used during the training. It is used during the data augmentation process, more CPUs will lead to a faster data augmentation. Ideally, the number of CPUs should match the GPU power, that way neither one will bottleneck the other during the training. The default number is 16;

- `nnUNet_raw_data_base` : The path that points to the raw database used by nnU-Net[13]. It must be the same as the one used in the export function (see 4.8.4). It is used for converting the data from a custom format to the nnU-Net's[13] format and in the plan generation;

- `nnUNet_preprocessed` : The path used to reference the nnUNet[13]'s pre-processed folder. It must be the same as the one used in the export function (see 4.8.4). Default is the combination of `nnUNet_raw_data_base` and `'nnUNet_preprocessed'`. It is used when converting the data from a custom format to the nnU-Net's[13] format and in the plan generation;

- `db_dir` : The path where the original, unprocessed data are stored. In the experiments ran for this master thesis, it is where the data using the MIRO's format were stored. Used by the two converters and the test model;

- `ct_file` to `patient_list` : these are thirteen parameters that must point to file names used by the custom format. The default ones are the file names used by the MIRO's format.

The other parameters can be set. If they are not set, the model will use the default ones that were chosen during the experimentation or that are coming from the original HD-Unet[4] implementation :

- `model_param` : A dictionary used for the model parameters (i.e., dropout rate, padding, activation function, etc.);

- `training_param` : A dictionary used for the training parameters (i.e., loss function, batch size, epochs, etc.).

These two are explained in more details in subsection 4.1.1. If the plan is used from `hdunet_plan_and_preprocess` (described in 4.8.5) it will overwrite the following parameters: batch size, bottleneck kernel size, plans for each stage, number of stages, pooling kernel size and patch size.

4.8.2 Convert data from MIRO's format to the one used by nnU-Net

Using the command `hdunet_convert_dataset_MIRO_to_nnUNet`, the developed solution allows the conversion from the MIRO's format to the one used by nnU-Net[13].

One parameter is available, which is `verify_dataset_integrity` it will verify if the converted dataset is usable by nnU-Net[13]. It should be used once when the new dataset is converted for the first time. If the dataset was already validated this parameter should not be used anymore.

4.8.3 Convert data from MIRO's format to the one use by HD-Unet

A second conversion is needed, from the original data format to the one used by this implementation of HD-Unet. This is the conversion which allows this implementation to load patients in $\mathcal{O}(1)$. The model developed for this thesis allows the following command : `hdunet_convert_dataset_MIRO_to_HDUNet` for this process.

It has one parameter, which is `verify_dataset <patient_id>`. It will ensure that, after the conversion, the dataset can be loaded. The `patient_id` given must be the ID of a patient present in the dataset.

4.8.4 Export folders

The next step is to export folders as shown in the appendixA.1. This step is mandatory as this implementation use some part of nnU-Net[13] using these exports. Exported folders are used to store data generated by nnU-Net[13] during the `plan_and_preprocess` operation. The plan will also be generated there, in the preprocessed folder.

4.8.5 Plan generation

Now it is time to generate the plan. This step analyzes the dataset and compute optimal parameters for it. For this process, the `task_name` parameter set as explained in 4.8.1 is important since the task id will be retrieved from it. To do that, the command line `hdunet_plan_and_preprocess` is available.

Only one parameter can be used here, available in both short and long version in order to stay consistent with nnU-Net[13]. To plan the training, the parameter `pl3d` or `planner3d` allow the selection of the planner that will be used for the plan generation. The default one is a planner assuming that 11 Gb of VRAM will be available from the GPU. This choice was made regarding the available hardware. Indeed, all but one GPU on the cluster has at least 11 Gb of VRAM. All other available planners can be used. In order to use them, the class name is needed after the parameter. Other planners can be found in nnU-Net[12]

4.8.6 Training

Once all other steps have been made, the network can be trained. To train the model, the command `hdunet_train_modular` can be used. Three parameters can be set when calling this command line :

- `num_threads` : This parameter allows the selection of threads that will be used by the program. For this thesis the number of 16 was always chosen;
- `not_use_plans` : By default, this implementation wants to use the generated plan. If for any reason the plan is not wanted, one can pass this parameter; thus it will use the default parameters as they are in the file `parameter.py`;
- `model_output_folder` : This parameter allows to select the folder that will be used to save the results of the training.

4.8.7 Testing

Finally, the model needs to be tested in order to validate its performances. To do so, the `hdunet_test_modular_model` command is helpful. Two parameters can be passed along the command. These two parameters are common ones with the training as described in 4.8.6. Thus, they must have the same value.

- `model_output_folder` : This refers to the folder where the test results will be saved. In order for the test to work, this folder must also contain the

[12] `github.com/MIC-DKFZ/nnUNet/tree/master/nnunet/experiment_planning`

training results. If this requirement is not met, the test will not work;

- `not_use_plans` : By default, this implementation wants to use the generated plan. If this parameter was used during the training, it must be used during the test.

Note that the last two steps can be slightly modified if the CISM/CECI cluster[13] is used. In A.2 and A.3 examples of scripts used on the clusters can be found.

Also, if for any reason there is a need to train again the model, the whole pipeline must not be used again. Only the last two steps, training and testing, have to be run again. Parameters settings, dataset conversion, path export and plan generation have to be done only once for each different dataset used.

[13]https://www.cism.ucl.ac.be/doc/

5 | Results

This chapter presents the results obtained during the training of the model and the selection of its empirical parameter. First, every dataset used will be presented as well as an explanation to how the model validation has been done. Then different results will be displayed, starting with the sanity check of the, converted, Pytorch model, following with the result of the empirical process concerning the dropout probability. Finally, the model proposed in this thesis will be compared with other models on different datasets.

5.1 Dataset presentation

For both the empirical parameters selection and the model evaluation, several dataset were used. These datasets are described in the following sections as well as the training, test and validation split that they respectively use.

5.1.1 H&N PBS

This is the first dataset used during the development of the model. It is also the one used by the HD-UNet[4]. The dataset is composed of 60 patients' images with a voxel median size of (183, 183, 97). It is split as 40, 10, 10 for training, test and validation respectively.

The different organs present in this dataset are : spinal cord, mandible, parotid right, parotid left, brainstem, oral cavity, larynx, pharynx's constrictor, submandibular gland right, submandibular gland left, skin left, skin right and trachea.

5.1.2 H&N VMAT

This dataset is composed of images of 50 patients suffering from head and neck cancer. The difference with the previous one (see 5.1.1) is the usage of Volumetric

Modulated Arc Therapy (VMAT). The 50 patients are split as follows : 40 for training the model, 5 for testing it and 5 to validate it. The median size of the images in voxels is (183, 183, 98). Note that due to some issues linked to the format, one of the patients is not used. The validation and test sets are not impacted by this removal.

The different organs present in this dataset are the same as in 5.1.1 : spinal cord, mandible, parotid right, parotid left, brainstem, oral cavity, larynx, pharynx's constrictor, submandibular gland right, submandibular gland left, skin left, skin right and trachea.

5.1.3 Esophagus IMRT

In the esophagus photon dataset, there are a total of 40 images of patients. The train, test and validation split is divided as follow : 30 patients for training the model, 5 for testing it and finally 5 for the model validation. The median image size in voxels is (167, 167, 158).

The different organs present in this dataset are : fundus, heart, kidney left, kidney right, ventricle left, liver, lungs, spinal canal (03) and spinal canal.

5.1.4 Esophagus PBS

This is a second esophagus dataset. However, in contrast to the first one using photons as explained in 5.1.3, this one is using protons. It is composed of 40 images of patients. The train, test and validation split are selected as follows : 30, 5 and 5 respectively. The median image size in voxels is (167, 167, 158).

The different organs present are the same as in 5.1.3 : fundus, heart, kidney left, kidney right, ventricle left, liver, lungs, spinal canal (03) and spinal canal.

5.1.5 Prostate IMRT

This dataset is a set of 45 patients suffering from prostate cancer. The different splits are respectively 30, 5, 10 for training, validation and testing. The median image size in voxels is (183, 183, 133).

Several organs are considered into account in this dataset : anal canal, bladder, colon, femur left, femur right, penile bulb, rectum, sigmoid and small bowel.

5.2 Model validation

In order to validate the model, 5-fold cross validation is used. Each fold consists of 150 epochs and each epoch is itself decomposed in training and validation steps. Each fold has a different train, validation and test sets. The number of steps per epoch is described thanks to the following equation 5.1. Once the training steps are made, the validation set is used to validate the model trained before starting the next epoch. The scoring function used to check whether the model performs better or not is the same as the loss function (4.1.1), Mean Squared Error (MSE). If the mean of the MSE provided by the validation steps is lower than the one stored before, the new model is saved as the best one for its fold.

$$number\_of\_steps = \frac{number\_of\_epochs}{batch\_size} \tag{5.1}$$

Equation 5.1 : Number of steps by epochs computation.

Finally, for each fold, its patients test set is used to run inferences on its best model. When all fives models had predicted the dose for the test set and every MSE computed have been stored, a mean operation is then made, leading to the generalisation error E_{gen}.

$$E_{gen} = \frac{\sum_{j=1}^{n} MSE_{m_{i,j}}}{5n} \forall_i = 1, ..., 5 \tag{5.2}$$

Equation 5.2 : Generalisation error equation, where m_i represents the model of the i^{th} fold. j represents the j^{th} patient of the test set and n represents the total number of patients in the test set.

5.3 Sanity checks

By "Sanity checks" this thesis aims to verify its implementation in the same conditions that were used by the original HD-Unet[4] model which was written using Keras. The purpose is to check if the model converted from Keras to Pytorch has a comparable accuracy. Thus, it means, no data augmentation other than mirroring, no AutoML part and the same parameters used by both models. It is

expected that this implementation's performances will be as good as the original one or, at least, in an acceptable margin. The default parameters are the following : 4 stages, a pooling kernel size of (2, 2, 2), a convolution kernel size of (3, 3, 3), the max pooling and ReLU activation function and a batch size of 1.

Concerning the dataset, the original HDU-Net[4] used the H&N PBS one. Thus, the same one is selected to perform these checks. A split of 40, 10, 10 is selected for training, validation and test respectively. The patch size used is (96, 96, 64). Results of this experiment can be found in table 5.1.

Results sanity checks, using H&N PBS dataset	
Dataset	E_{gen}
Keras/Tensorflow	2.53
Pytorch	2.12

Table 5.1: Table of results sanity checks

5.4 Empirical parameters selection

Within the context of this thesis, only one empirical parameter has been met, the dropout probability. In order to find the best suiting value, the model ran through several databases (H&N PBS, H&N VMAT, Esophagus IMRT, Esophagus PBS and Prostate IMRT), with a dropout probability varying in the range : $[0 \rightarrow 0.5]$. This section will present the results of these models. For each of the model trained, the planned parameters are used. Some of these parameters are always planned with the same values. Therefore, they are presented here : a batch size of 2, (3, 3, 3) as convolution kernel size everywhere in the model, a pooling kernel size of (2, 2, 2) and a number of 5 stages.

For the last parameter, the patch size, it is fixed at a value known to be good. A value "known to be good" is a value that was used by a previous model designed for that particular dataset. The results can be observed in figure 5.1.

5.4.1 H&N PBS

The fixed patch size used is (96, 96, 64). The model was run using different GPUs depending of which ones were available at that time. Therefore, it is a combination of Nvidia RTX 2080Ti, Nvidia Tesla V100 and Nvidia RTX 3090Ti. For each of the run, 16 CPUs were used. The training time is varying between $[3h47 \rightarrow 5h50]$. The results can be found in a more precise table A.4.1.

5.4.2 H&N VMAT

The fixed patch size chosen for this dataset is (96, 96, 64). The model was run using Nvidia TeslaV100 GPUs and 16 CPUs. The training time was in the range $[3h51 \rightarrow 4h52]$. The results can be found in a more detailed way in table A.4.2.

5.4.3 Esophagus IMRT

A fixed patch size of (128, 128, 96) is used for this dataset. Every model was run using an Nvidia RTX 3090Ti and 16 CPUs. The training time is varying in the range $[5h35 \rightarrow 6h36]$. Results in more details can be found in table A.4.3.

5.4.4 Esophagus PBS

The fixed patch size used is (128, 128, 96). For training this dataset three GPUs were used, depending on the available hardware. A brand new Nvidia Tesla A100 80 Gb was used for the dropout probability of 0. A Nvidia Tesla V100 was used for the dropout probability ranging from $[0.1 \rightarrow 0.2]$ while a Nvidia RTX 3090 was used for the last two runs (dropout probability of 0.4 and 0.5). Concerning the training time it is almost consistent, except when using the Nvidia Tesla A100, varying from $[3h44 \rightarrow 6h07]$. Results in more details can be found in table A.4.4.

5.4.5 Prostate IMRT

For this dataset, the fixed patch size that was selected is (128, 128, 96). Every model was run using a Nvidia RTX 3090Ti and 16 CPUs. The table A.4.5 shows the results of this experiment in a detailed way.

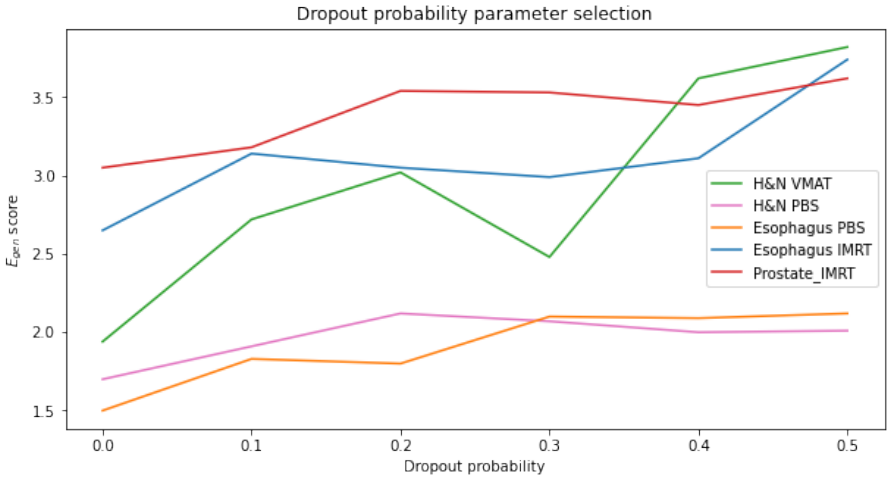

Figure 5.1: Results of the empiric process concerning the dropout probability. Every dataset is represented with its own color. Green, pink, orange, blue and red represent H&N VMAT, H&N PBS, esophagus PBS, esophagus IMRT, prostate IMRT respectively.

5.5 Results with complete pipeline

This section will present the results obtained using the complete pipeline. For each of the model trained, the planned parameters are used. Some of these parameters are always planned with the same values. Therefore, they are presented here : a batch size of 2, (3, 3, 3) as convolution kernel size everywhere in the model, a pooling kernel size of (2, 2, 2) and a number of 5 stages. Note that the patch size generated by the plan is also used for the Full AutoML models.

5.5.1 H&N PBS

For the H&N PBS dataset, the plan proposed a patch size of (160, 192, 96), which is transformed in (160, 160, 96) for memory reason as the computed patch size was too large even for some powerful GPUs. This will be discussed later in chapter 6. Figure 5.2 shows the comparison between three models : the original one, this thesis's model with the fixed patch size of (96, 96, 64) as explained in 5.4.1 and finally the complete AutoML model. For each of them, the E_{gen}, the standard deviation and the variance are analyzed.

The AutoML model was trained using a Nvidia Tesla V100 GPU for a total time of 12h33. The final E_{gen} of this model is 1.48. The figure 5.3 shows the computed DVH for a given patient in the H&N PBS dataset.

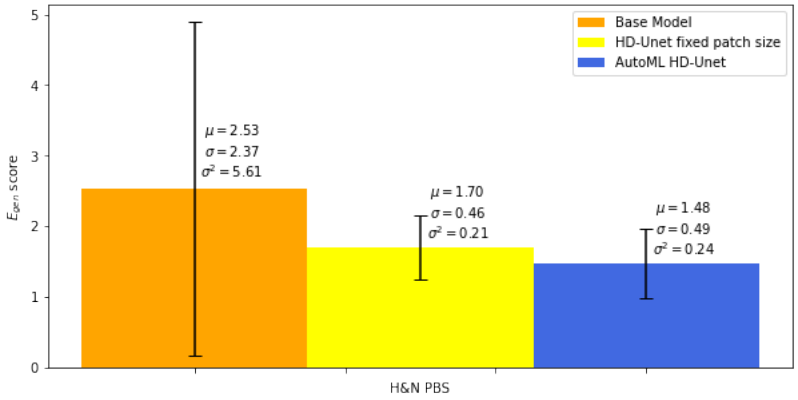

Figure 5.2: Model comparison for the H&N PBS dataset. The orange bar corresponds to the base model, the yellow one to this thesis model with a fixed patch size and finally the blue one is for this thesis model with full plan. For every model, μ is the E_{gen}, σ is the standard deviation and σ^2 is the variance.

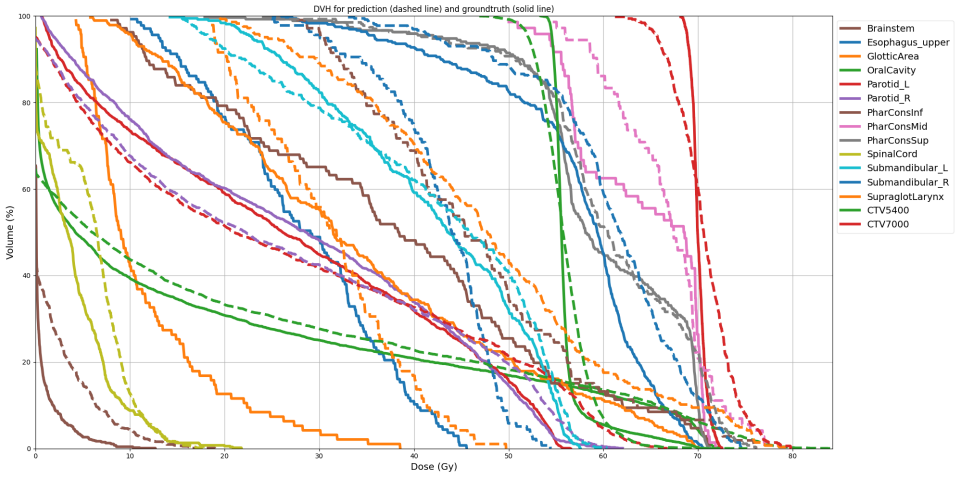

Figure 5.3: DVH for a given patient in the H&N PBS dataset. Every dashed line corresponds to the prediction given by the model while the solid line is for the groundtruth. Every organ is represented with its own color. The CTV values correspond to the volume that will be received by the tumorous cells.

5.5.2 H&N VMAT

Concerning this dataset, the computed patch is (160, 192, 96) which is rescaled to (160, 160, 96). This transformation is done for memory reasons that will be discussed in chapter 6. Figure 5.4 shows the comparison between three models : the original one, this thesis's model with the fixed patch size of (96, 96, 64) as explained in 5.4.1 and finally the complete AutoML model. For each of them, the E_{gen}, the standard deviation and the variance are analyzed.

The complete AutoML model was run with a Nvidia Tesla V100 GPU in 18h13. The score for this model is a E_{gen} of 1.71. The figure 5.5 shows the computed DVH for a given patient in the H&N VMAT dataset.

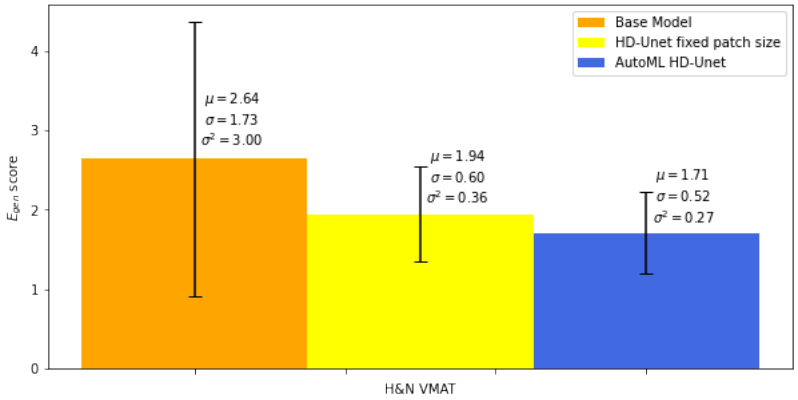

Figure 5.4: Model comparison for the H&N VMAT dataset. The orange bar corresponds to the base model, the yellow one to this thesis model with a fixed patch size and finally the blue one is for this thesis model with full plan. For every model, μ is the E_{gen}, σ is the standard deviation and σ^2 is the variance.

5.5.3 Esophagus IMRT

For this dataset, the computed patch size is (128, 160, 128). Figure 5.6 shows the comparison between three models : the original one, this thesis's model with the fixed patch size of (128, 128, 96) as explained in 5.4.1 and finally the complete AutoML model. For each of them, the E_{gen}, the standard deviation and the variance are analyzed.

The complete AutoML model is trained using an Nvidia Tesla V100 GPU in 9h41. Having a E_{gen} of 2.25. The figure 5.7 shows the computed DVH for a given patient in the Esophagus IMRT dataset.

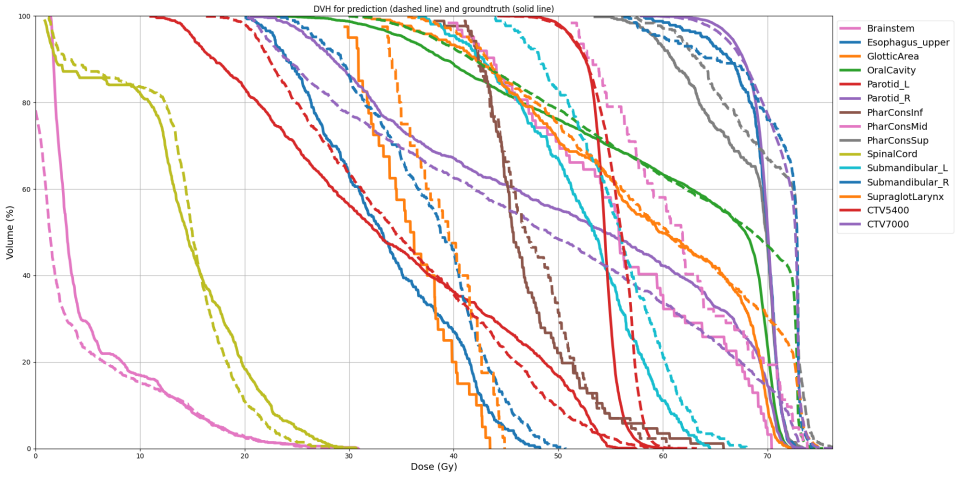

Figure 5.5: DVH for a given patient in the H&N VMAT dataset. Every dashed line corresponds to the prediction given by the model while the solid line is for the groundtruth. Every organ is represented with its own color. The CTV value correspond to the volume that will be received by the tumorous cells.

5.5.4 Esophagus PBS

The second dataset for the esophagus cancer, the same patch size was computed than the one in the previous point : (128, 160, 128). Figure 5.8 shows the comparison between three models : the original one, this thesis's model with the fixed patch size of (128, 128, 96) as explained in 5.4.1 and finally the complete AutoML model. For each of them, the E_{gen}, the standard deviation and the variance are analyzed.

The AutoML model was trained using a Nvidia V100 GPU in 9h35. Achieving a E_{gen} of 1.38. The figure 5.9 shows the computed DVH for a given patient in the Esophagus PBS dataset.

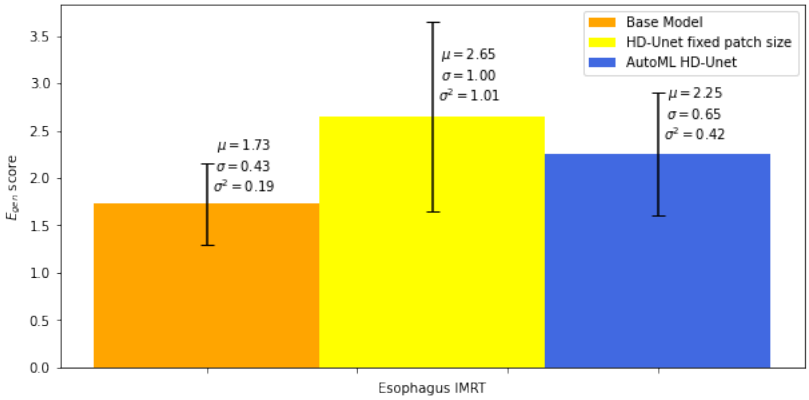

Figure 5.6: Model comparison for the Esophagus IMRT dataset. The orange bar corresponds to the base model, the yellow one to this thesis model with a fixed patch size and finally the blue one is for this thesis model with full plan. For every model, μ is the E_{gen}, σ is the standard deviation and σ^2 is the variance.

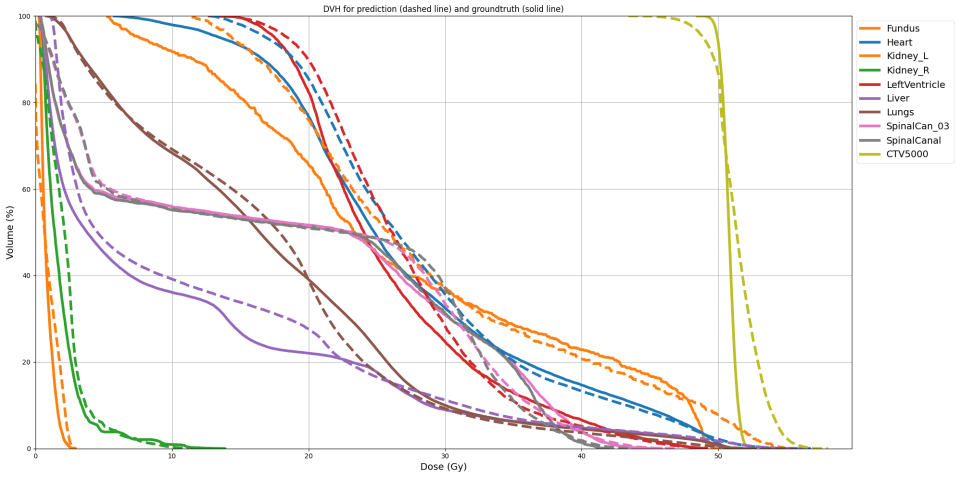

Figure 5.7: DVH for a given patient in the Esophagus IMRT dataset. Every dashed line corresponds to the prediction given by the model while the solid line is for the groundtruth. Every organ is represented with its own color. The CTV value correspond to the volume that will be received by the tumorous cells.

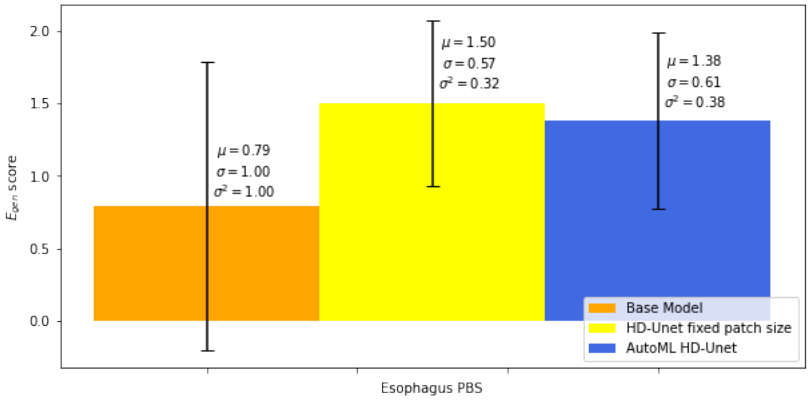

Figure 5.8: Model comparison for the Esophagus PBS dataset. The orange bar corresponds to the base model, the yellow one to this thesis model with a fixed patch size and finally the blue one is for this thesis model with full plan. For every model, μ is the E_{gen}, σ is the standard deviation and σ^2 is the variance.

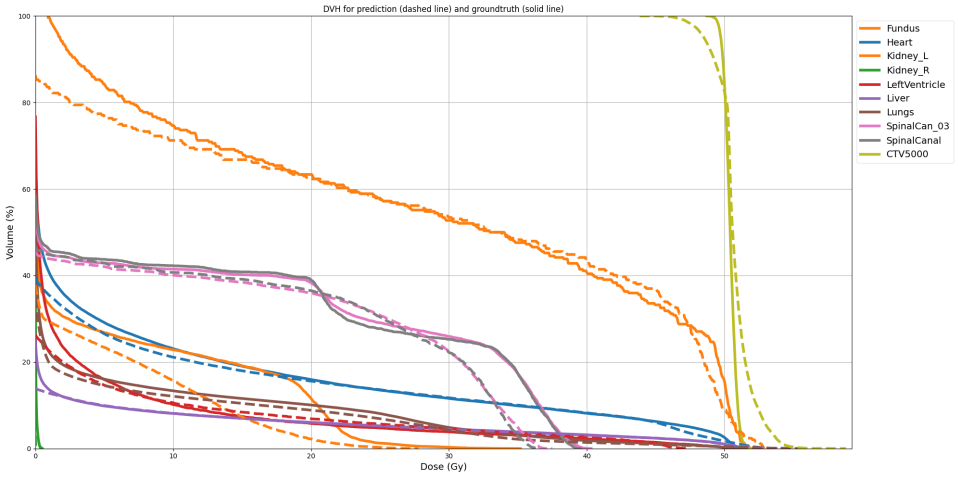

Figure 5.9: DVH for a given patient in the Esophagus PBS dataset. Every dashed line corresponds to the prediction given by the model while the solid line is for the groundtruth. Every organ is represented with its own color. The CTV value correspond to the volume that will be received by the tumorous cells.

5.5.5 Prostate IMRT

Finally, the Prostate IMRT Dataset had a computed patch size of (160, 160, 112) which was transformed to (160, 160, 96) for memory reasons that will be discussed in chapter 6. Unfortunately for this dataset it was impossible to find comparable results for the E_{gen}, standard deviation and variance thus, there is no plot as opposed to the other dataset.

The complete AutoML model was trained in 4h on a Nvidia A100 GPU achieving a E_{gen} score of 1.79. The figure 5.10 shows the computed DVH for a given patient in the Prostate IMRT dataset.

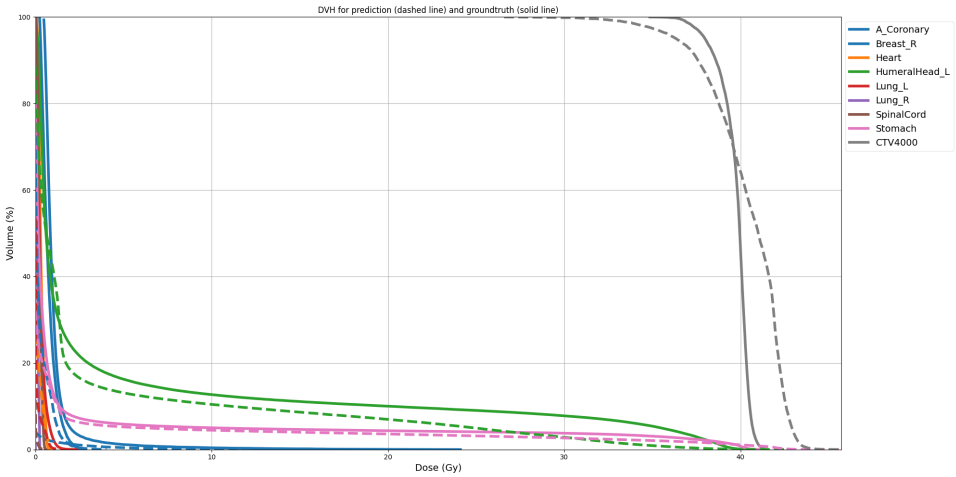

Figure 5.10: DVH for a given patient in the prostate IMRT dataset. Every dashed line corresponds to the prediction given by the model while the solid line is for the groundtruth. Every organ is represented with its own color. The CTV value correspond to the volume that will be received by the tumorous cells.

5.5.6 Sum up

To conclude this chapter the figure 5.11 is a sum-up of every model performance for each dataset.

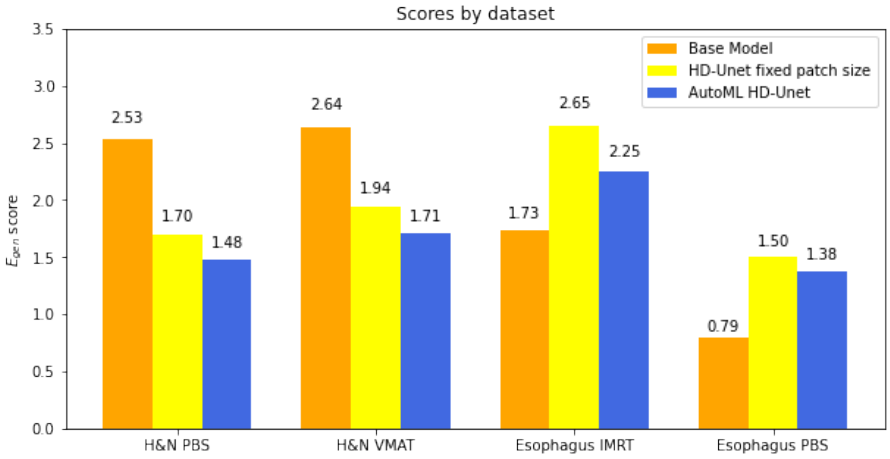

Figure 5.11: Model comparison. Each orange bar is the results of the actual model. Each yellow bar corresponds to the model developed in this thesis with the fixed patch size. Each blue bar is the result of the model developed in this thesis using the full AutoML pipeline. Every dataset is presented, from right to left : H&N PBS, H&N VMAT, Esophagus IMRT, Esophagus PBS.

6 | Discussion

6.1 Over results

Many discussions happened on how to prove that the model proposed in this thesis is relevant in comparison to other existing deep-learning models. Indeed, there are no other models that tackle the same problem, in an AutoML way. Thus the choice was made to compare it to models that are also doing dose prediction, in a more classical way, by optimizing the model for their specific dataset.

However, one should keep in mind that this thesis is based on the previous work on HDU-Net[4], thus the comparison with that model is a first interesting step. Nonetheless some optimizations have been done (i.e., more advanced data augmentation techniques). In consequence, when the models are compared these differences should be considered. Indeed, it is expected that a model being an optimization of another is better for the same given task. This comparison is still interesting to do considering this thesis model is somewhat of a conversion of the base model in another framework.

To complete the comparisons, the model proposed was run using nothing else than its own hyper-parameters tuning for different datasets and then, compared to models specialized for the given dataset. Thanks to that, one could observe how the proposed model does adapt itself to new datasets. Of course, in order for the comparison to be more accurate, more than one model should have been taken into account, each of them running on multiple datasets. More models with more dataset could have given a ranking of the models for each dataset in the same way that nnU-Net[13] did. However, as the number of models performing dose prediction is limited, it was not possible to do that.

Still, this thesis want to compare the results in the most factual way and it is the goal of this section to present such results. To pursue it, this section is divided as

follows : a discussion about the obtained results, followed by the limitations of the proposed model before being concluded by the possible evolution of this thesis.

6.1.1 Sanity checks

The first results obtained are the one of the sanity check. Indeed, as one of the first steps of this thesis was to convert the original HDU-Net[4] from Keras to Pytorch, the main goal at this step was to have a model that is as least as powerful as the original one. The results of this first step can be found in table 5.1.

One can see that the results were promising since the model converted in Pytorch was already providing a 2.12 E_{gen} while the Keras/Tensorflow model provided a 2.53 E_{gen}. Thus, the model converted in Pytorch could already outperform the original one.

6.1.2 Empirical parameters selection

Figure 5.1 presents the results of the empirical process implemented to define which dropout probability best fits the model.

Observing the results of the said figure, one can see that, in every single case, having a dropout probability set to 0 leads to better performance of the model. Therefore the dropout probability kept is 0. As already explained in 4.1.1, it means that the model will never drop a single neuron.

6.1.3 Model performances

Figure 5.2 shows the results of the comparison for the H&N PBS dataset. Considering the H&N PBS dataset, the results are the one expected, both of the models proposed perform better than the original one. Not only is the MSE better, but also the results tend to vary less. The full AutoML model performs the best and has the second-best stability behind the model using a fixed patch size. Therefore, in this case, having a smaller patch tend to provide a better stability with the trade-off being a lower MSE.

Figure 5.4 shows the results of the comparison for the H&N VMAT dataset. This thesis's model was not directly designed for this dataset but to a very similar one (H&N PBS). As expected, the results are better than the one provided by the base model with once again more stability. In contrast to the model designed for H&N PBS, the model which has the best stability is the one full AutoML one.

Figure 5.6 shows the results of the comparison for the Esophagus IMRT dataset.

The Esophagus IMRT dataset is the first dataset that really departs from what our model has been confronted with so far. The MSE and the stability of the full AutoML model are worse than the base model. The best case with the proposed model seems to be an average case for the base model. The full AutoML still proposed a better MSE and provide results that tend to vary less than the one using a fixed patch size.

Figure 5.8 shows the results of the comparison for the Esophagus PBS dataset. This dataset is similar to the previous one, Esophagus IMRT. One can see that the base model perform better, but it has a very bad stability. The base model worst case performs worse than this thesis AutoML model average case. However, the worst case of the latter is still worse than the limit case of the base model. Despite that, the best case of the AutoML model is still the average result of the base model. On the other, the stability of the proposed model is much better than the one of the base model. One interesting fact is the difference of stability between the AutoML model and the model that use a fixed patch size is that standard deviation and the variance are lower with a smaller patch size. Thus, it is a common point to both PT dataset.

Figure 5.11 shows the results of the comparison of all the models. In the end, as mentioned at the beginning of this section, it is hard to say if the results are really good or really bad. In the case of H&N, one could only say that the AutoML model performs better than its original implementation, which was expected and it does not mean that the model can not be outperformed by a new one sooner or later. On the other hand, concerning the esophagus dataset, one can say that the base model performs better than the AutoML framework proposed in this thesis, maybe the latter outperforms other models specialized in Esophagus dose prediction. But in either case, one could not say that it is good or bad for a given dataset.

Nevertheless, there are a few interesting things to be raised :

- The first observation is that, in the case of a PBS dataset, using the proposed model with a smaller patch size seems to provide slightly more stability in the dose predicted while offering, as expected, worse results on average;

- Another observation is that the AutoML model reaches a lower E_{gen} on Esophagus PBS than on the H&N PBS dataset which was used to design it. This could be due to a difference in the composition of the dataset;

- Finally, Esophagus IMRT reaches the worst E_{gen} while having a totally different way/technique of delivering the dose. The original model was designed for PBS dose prediction and, among the techniques used for delivering

the dose proposed in this thesis, IMRT is the furthest from PBS (i.e., not the same type of beams and way to deliver them). The architecture of the network could not be the best suited for IMRT and should maybe be adapted when facing datasets that use techniques which are that different.

6.2 Limitations

6.2.1 Dataset size

As already stated earlier in this thesis, one of the biggest problems in machine learning is the number of data provided to the model. Optimally, a model would be fed with thousands and thousands of data. Although Convolutional Neural Networks (CNN) have better performances with small datasets than typical Neural Networks, they also benefit from a larger dataset. However, in this thesis, the number of patients' images was limited. In the best case (H&N PBS), 60 patients were provided, giving respectively 50, 10 and 10 for training, validation and testing. And that is the best case which can benefit from the mirroring applied during data augmentation. In the worst case (Esophagus PBS), this mirroring is not ideal (but it has still been done), and there are only 40 patients. Thus, the train, validation and test set are reduced to 30, 5 and 5 respectively. On the other hand, it was also stated during the discussion over results that the best E_{gen} was reached with this particular dataset. This may be a consequence of a difference in homogeneity between the dataset [34]. It could be interesting to provide datasets that are bigger or with a different degree of homogeneity in order to see if the model would be able to reach better accuracy.

6.2.2 Regularization

Since it appears that no dropout (probability of 0) is needed for the model proposed, no regularization is done. However, it is a well-known fact that regularization is an important component of deep learning [35], also, no regularization could lead to overfitting. Using another type of regularization technique could maybe have an impact on the model and thus yield better results.

6.2.3 Hardware scaling

The planner used in the AutoML pipeline is not really dynamic. Indeed, the planner is not executed in relation to the memory available to the GPU which is allocated to it. It will use a fixed value as the available memory. Since the CESI/CISM cluster was used and the hardware available on it is shared between different users.

It was decided to take a smaller plan (i.e., a smaller value) to be able to run it on most of the GPUs available and to avoid queues as mush as possible. This means that in some cases the model was executed by a GPU that could have accepted a larger patch size and therefore perhaps given better performance. In the case of this thesis, this is not such a serious problem since the results should be comparable, but under real conditions, it could become more problematic.

6.3 Perspectives

6.3.1 More robust hardware scaling

AutoML includes heuristics about available hardware, this thesis includes only a limited part of hardware scaling. Indeed, as explained in chapter 4.8.5 the plan takes into account that a GPU with 11 Gb of VRAM will be available. This is not perfect as the planner cannot know in advance how much Gb of VRAM will be consumed during the training, it tries to estimate it. This can lead to errors when running the code because of the GPU can run out of memory with a bad estimation. On the other hand, in some cases, the GPU will still have some unused VRAM available. Multiple solution could be possible :

- The first one could be to detect how much Gb of VRAM are exactly available from the GPU using Nvidia's software and call the according planner;
- With an empirical process, one could try to estimate how much Gb are needed for their dataset and use the appropriate planer;
- Write a more accurate planner that could be more precise for the dose prediction task instead of using the one from nnU-Net[13].

GPUs are not the only part of the hardware that could be scaled, CPUs are also part of it. At the moment, the CPUs are used during the data augmentation part as explained in 4.6. The exact number of CPUs that will be used is set as a parameter that has to be set when running the program. This allows some control over it as one could choose exactly the number of CPUs that they want to use for the model. However, a possible solution could be to retrieve the number of cores directly in the model code and use that number of CPUs. This can be a great solution as-is if a particular cluster is used (like CESI/CISM), but need some careful considerations to be run on someone else computer. To be able to run on anyone computer's the number of used CPUs should probably be : $cpu\_core - \{1, 2\}$ in order to let some CPUs core to the operating system.

6.3.2 GAN

Generative Adversarial Networks (GAN) are increasingly used in the field of imaging whether for analysis or image processing. It is therefore not surprising that this method is adapted to make dose prediction [36]. A GAN trains two networks, the first one is a generator, its goal is to generate new data which in the case of this thesis is a dose prediction, but could be another task such as an image segmentation. This creates a second dataset composed of only created images. These two datasets are then used in alternations to feed the second network, the discriminator. This second network goal is to predict if the image is coming from the real dataset or the fully created one. In this context, the generator could be this thesis model and another CNN as discriminator.

6.3.3 Adapted data augmentation

During discussions with this thesis's supervisors, it appears that the mirroring part of the data augmentation could lead to some problem for particular datasets. It was originally done in HD-Unet[4] as the H&N body region is perfectly symmetrical, so by using mirroring, it leads to twice as much data. This thesis is based on the explained implementation, therefore, the mirroring was kept.

However, once different datasets have been explored, it appears that some of them are not symmetrical. For instance, in the Esophagus PBS and XT dataset one of the organs at risk is the heart of the patient as shown in 5.1.3 and 5.1.4. It is a well-known fact that (most) humans have their heart on the left side of the body. Now, if one mirror these images, the heart appears to be on the right side.

This is only one example, but probably the most obvious one. Nevertheless, this shows the need to think more about the data augmentation techniques that are used. In this thesis, the choice was made to propose only one kind of data augmentation which uses all the steps as explained in 4.6.

The model was developed to be as modular as possible, thus it would not be difficult to create more data augmentation functions that could be used by the model. A possible evolution of this project would be to include such subsets of the default data augmentation and propose a way to select the one that will be used.

6.3.4 More rule-based parameters

In this implementation, there are a few parameters that are rule based (i.e., depending on the dataset). One possible evolution would be to add even more rule

based parameters in order to adapt the network as much as possible to the dataset. Here are some examples of possible evolution :

- Rule-based on loss function : it appears that for some dataset a better loss function than the MSE could be used. To give a more concrete example, if some organs are very small, one could want to increase their weight in the equation;

- Rule-based patch size vs batch size : the batch size is determined by the plan but also the patch size. Ideally, one would want to feed the complete image to the network and not only patches of it. The hardware limitation as explained in 6.2 had forced the choice to keep the patching system. Thus, there is a preference for the patch size over the batch size. One could want to change the priority between these two;

- Initial filter number : as explained in chapter 4, the model proposed in this thesis uses an initial number of filters of 16. An improved solution could be to adapt the number of initial filters to the number of organs at risks present in the dataset. A simple threshold could be applied, if there are more than X organs, than the initial number of filters is set to Y, otherwise it is set to Z. It was not applied in this thesis. Indeed, as all the datasets used have a number of organs superior to 8 (which was the initial threshold idea), it was impossible to test it. Still, it is a possible evolution of the software.

6.3.5 Regularization

The dropout was used as a regularization technique. It is also the one that was used for the original implementation of HDU-Net[4], since this thesis is based on it, the choice to keep it was made. However, as discussed before in 6.1 the best models are with a dropout probability of 0.

Having a dropout probability set to 0 means that all the neurons are kept during the training. It means that the network needs all its connection to perform well. This can happen and is not a bad thing. Yet, the model does not take advantage of the dropout's ability to reduce its overfitting. Hence, one possible upgrade to this model would be to explore diverse regularization techniques and select the one that makes the model even better.

6.3.6 Ensembling

In nnU-Net[13], they have explored some techniques where the final model can be a combination of their different models (2D, 3D full resolution, 3D low resolution and 3D cascade). The final ensemble can be any of the possible one, but it is the one that leads to the best scores.

In this thesis, there is only a single model. However it could be possible to train multiple times the model, leading to have 1, 2 or 3 (or more) models were each of them would have run through some k-fold validation and see what is the best ensemble of them for prediction.

7 | Conclusion

The aim of this thesis was to optimize the HD-Unet[4] by transforming it from a classic static model performing 3D dose prediction for head and neck cancer to an auto machine learning (AutoML) framework. The development of an AutoML framework for dose prediction models allows for the medical community to be able to generate models without having to become machine learning experts.

The conversion of the original model from Keras/Tensorflow to Pytorch was not only a great success, but also allows to adapt what has already been done in Pytorch. Thus saving a lot of time, mainly by allowing adapting the work already done on another framework of this type for image segmentation[13]. In the end, the time spent converting the model from Keras/Tensorflow to Pytorch turned out to be smaller than the time it would have taken to rewrite the entire AutoML pipeline.

The model proposed in this thesis provides results that are better than the original model for head and neck cancer while being able to adapt itself to new datasets. The results on these new dataset may not be as good as the ones proposed by the models fine-tuned for them. Indeed, in the best scenario, the results on the H&N PBS dataset is a E_{gen} of ± 1.48 when the original model has a E_{gen} of ± 2.53. Whereas on the worst scenario using the esophagus IMRT dataset, the proposed model has a E_{gen} of ± 2.25 while the fined-tuned model has a E_{gen} of ± 1.73. However, the model still provides interesting results (i.e., not absurd) and many improvements are possible and some have already been proposed.

Finally, results put aside, the AutoML pipeline is created, from data conversion to dose prediction passing to data augmentation and other optimizations of the base model. Furthermore, everything has been designed to be simple not only to use, but also to maintain and evolve.

Better results could, perhaps, be achieved by evolving the pipeline in different ways.

Among the possible evolution, the most promising ones seem to be :

- The addition of new rule-based parameters. For instance, an adaptive loss function that, depending on the dataset, could, for example, give more weight to smaller organs than others. Or even the application of a simple threshold to define the initial number of filter produced according to the number of organs in the dataset;

- The use of an adapted data augmentation that takes into account the organs present in the dataset. Thus, it is possible to avoid using certain types of transformations if they lead to the new image produced being anatomically incorrect.

A | Appendix Title

A.1 Pipeline execution

1. Set your data input/output folders path and the name used by your files in `hdunet/parameters.py`
2. Export the different folders for nnU-Net:
   - `export nnUNet_raw_data_base= "path/nnUNet_raw_data_base"`
   - `export nnUNet_preprocessed= "path/nnUNet_raw_data_base/nnUNet_preprocessed"`
   - `export RESULTS_FOLDER= "path/nnUNet_raw_data_base/nnUNet_trained_models"`
3. Convert data from MIRO's format to the one use by nnU-Net: `hdunet_convert_dataset_MIRO_to_nnUNet`;
4. Convert data from MIRO's format to the one use by our HD-Unet: `hdunet_convert_dataset_MIRO_to_HDUNet`;
5. Execute the plan generation `hdunet_plan_and_preprocess -t <task_id>`;
6. Start training `python3 hdunet/run/modular_training.py`;
7. Testing and generating results `hdunet/run/testr_modular_model.py`;

A.2 Model submit on CESI/CISM cluster

Listing A.1: Model submit

```
#!/bin/bash
# Submission script for Manneback
# Resource request steps
#SBATCH --job-name=job_name
#SBATCH --time=18:00:00 # hh:mm:ss
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=16
#SBATCH --mem-per-cpu=8192 # megabytes
#SBATCH --exclude=mb-mil[102]
#SBATCH --partition=gpu
#SBATCH --gres="gpu:1"
#SBATCH --mail-user=your_mail
#SBATCH --mail-type=ALL
#SBATCH --output="outputcode.txt"
#SBATCH --error="error.txt"
#SBATCH --comment=hdunet

#Job steps
module load releases/2019
module load releases/2019b
module load Pillow/6.2.1-GCCcore-8.3.0
module load releases/2020b
module load Python/3.8.6-GCCcore-10.2.0
pip install --user --upgrade pip
pip install --user --upgrade Pillow
pip install --user scipy
pip install --user numpy
pip install --user matplotlib
pip install --user pandas
pip install --user torch==1.10.1+cu113 \
torchvision==0.11.2+cu113 torchaudio==0.10.1+cu113 \
-f https://download.pytorch.org/whl/cu113/torch_stable.html
pip install --user pytorch-ignite
pip install --user torchsummary
pip install --user tqdm
pip install --user batchgenerators
pip install --user setuptools
```

```
pip install --user -e .
python hdunet/run/modular_training.py --num_threads=16 \
--model_output_folder=output_folder_name
```

A.3 Model test submit on CESI/CISM cluster

Listing A.2: Submit model test

```
#!/bin/bash
# Submission script for Manneback
# Resource request steps
#SBATCH --job-name=job_name
#SBATCH --time=06:00:00 # hh:mm:ss
#SBATCH --ntasks=1
#SBATCH --mem-per-cpu=8192 # megabytes
#SBATCH --partition=gpu
#SBATCH --gres="gpu:1"
#SBATCH --mail-user=your_mail
#SBATCH --mail-type=ALL
#SBATCH --output="outputcode_test.txt"
#SBATCH --error="error_test.txt"
#SBATCH --comment=hdunet_test_model

#Job steps
module load releases/2019
module load releases/2019b
module load Pillow/6.2.1-GCCcore-8.3.0
module load releases/2020b
module load Python/3.8.6-GCCcore-10.2.0
pip install --user --upgrade pip
pip install --user --upgrade Pillow
pip install --user scipy
pip install --user numpy
pip install --user matplotlib
pip install --user pandas
pip install --user torch
pip install --user pytorch-ignite
pip install --user torchvision
pip install --user torchsummary
pip install --user tqdm
pip install --user batchgenerators
```

```
pip install --user setuptools
pip install --user -e .
python hdunet/tests/test_modular_model.py \
--model_output_folder=output_folder_name
```

A.4 Results

A.4.1 H&N PBS

Results HAN PBS			
Dataset	GPU	Training time	E_{gen}
Dropout 0	2080Ti	3h53	1.7259228
Dropout 0.1	2080Ti	4h00	1.910648
Dropout 0.2	TeslaV100	05h50	2.1181936
Dropout 0.3	3090	3h47	2.0660923
Dropout 0.4	3090	3h53	2.0081654
Dropout 0.5	2080Ti	3h52	2.010742

Table A.1: Table of results HAN PBS

A.4.2 H&N VMAT

Results HAN VMAT			
Dataset	GPU	Training time	E_{gen}
Dropout 0	TeslaV100	4h52	1.9398503
Dropout 0.1	TeslaV100	3h51	2.7223697
Dropout 0.2	TeslaV100	3h52	3.0184138
Dropout 0.3	TeslaV100	3h51	2.4770713
Dropout 0.4	TeslaV100	3h52	3.6213844
Dropout 0.5	TeslaV100	3h53	3.8234189

Table A.2: Table of results HAN VMAT

A.4.3 Esophagus IMRT

Results Esophagus IMRT fixed patch size of (128,128,96)			
Dataset	GPU	Training time	E_{gen}
Dropout 0	3090	5h35	2.65
Dropout 0.1	3090	5h36	3.14
Dropout 0.2	3090	5h37	3.05
Dropout 0.3	3090	6h23	2.99
Dropout 0.4	3090	5h50	3.11
Dropout 0.5	3090	6h36	3.74

Table A.3: Table of results Esophagus IMRT with fixed patch size of (128,128,96).

A.4.4 Esophagus PBS

Results Esophagus PBS			
Dataset	GPU	Training time	E_{gen}
Dropout 0	A100	3h44	1.5073699
Dropout 0.1	V100	6h03	1.8281184
Dropout 0.2	V100	6h08	1.8053001
Dropout 0.3	3090	6h09	2.1025574
Dropout 0.4	3090	6h06	2.0910032
Dropout 0.5	3090	6h07	2.1174562

Table A.4: Table of results Esophagus PBS

A.4.5 Prostate IMRT

Results Prostate IMRT fixed patch size of (128,128,96)			
Dataset	GPU	Training time	E_{gen}
Dropout 0	3090	6h	3.05
Dropout 0.1	3090	5h59	3.18
Dropout 0.2	3090	6h07	3.54
Dropout 0.3	3090	6h10	3.53
Dropout 0.4	3090	5h42	3.45
Dropout 0.5	3090	5h48	3.62

Table A.5: Table of results Prostate IMRT with fixed patch size of (128,128,96).

Bibliography

[1] R. C. of Radiologists. Clinical radiology uk workforce census 2020 report. https://www.rcr.ac.uk/system/files/publication/, April 2021.

[2] R J M Bruls and R M Kwee. Workload for radiologists during on-call hours: dramatic increase in the past 15 years. *Insights Imaging*, 11(1):121, November 2020.

[3] Varian | Benefits of Intensity Modulated Proton Therapy.

[4] Margerie Huet Dastarac. Dose prediction for protontherapy using neural networks. Master's thesis, Ecole polytechnique de Louvain, Université catholique de Louvain, 2019. Prom. : Lee, John Aldo ; Sterpin, Edmond.

[5] Di Yan, Frank Vicini, John Wong, and Alvaro Martinez. Adaptive radiation therapy. *Physics in Medicine and Biology*, 42(1):123–132, jan 1997.

[6] Michel Ghilezan, Di Yan, and Alvaro Martinez. Adaptive radiation therapy for prostate cancer. *Seminars in Radiation Oncology*, 20(2):130–137, 2010. Adaptive Radiotherapy.

[7] Emma Colvill, Jeremy Booth, Simeon Nill, Martin Fast, James Bedford, Uwe Oelfke, Mitsuhiro Nakamura, Per Poulsen, Esben Worm, Rune Hansen, Thomas Ravkilde, Jonas Scherman Rydhög, Tobias Pommer, Per Munck af Rosenschold, Stephanie Lang, Matthias Guckenberger, Christian Groh, Christian Herrmann, Dirk Verellen, Kenneth Poels, Lei Wang, Michael Hadsell, Thilo Sothmann, Oliver Blanck, and Paul Keall. A dosimetric comparison of real-time adaptive and non-adaptive radiotherapy: A multi-institutional study encompassing robotic, gimbaled, multileaf collimator and couch tracking. *Radiotherapy and Oncology*, 119(1):159–165, 2016.

[8] Francesco Cuccia, Filippo Alongi, Claus Belka, Luca Boldrini, Juliane Hörner-Rieber, Helen Mcnair, Michele Rigo, Maartje Schoenmakers, Maximilian Niyazi, Judith Slagter, Claudio Votta, and Stefanie Corradini. Patient positioning and immobilization procedures for hybrid mr-linac systems. *Radiation Oncology*, 16, 09 2021.

[9] T D Denotter and J Schubert. Hounsfield unit. *StatPearls. Treasure Island*, 2022.

[10] Andreas Boss, Sotirios Bisdas, Armin Kolb, Matthias Hofmann, Ulrike Ernemann, Claus Claussen, Christina Pfannenberg, Bernd Pichler, Matthias Reimold, and Lars Stegger. Hybrid pet/mri of intracranial masses: Initial experiences and comparison to pet/ct. *Journal of nuclear medicine : official publication, Society of Nuclear Medicine*, 51:1198–205, 08 2010.

[11] Neil G Burnet, Simon J Thomas, Kate E Burton, and Sarah J Jefferies. Defining the tumour and target volumes for radiotherapy. *Cancer Imaging*, 4(2):153–161, October 2004.

[12] Ying Cao, Du Tang, Yining Xiang, Li Men, Chao Liu, Qin Zhou, Jun Wu, Lei Huo, Tao Song, Ying Wang, Zhanzhan Li, Rui Wei, Liangfang Shen, Zhen Yang, and Jidong Hong. Study on the appropriate timing of postoperative adaptive radiotherapy for high-grade glioma. *Cancer Manag. Res.*, 13:3561–3572, April 2021.

[13] Fabian Isensee, Paul Jaeger, Simon Kohl, Jens Petersen, and Klaus Maier-Hein. nnu-net: a self-configuring method for deep learning-based biomedical image segmentation. *Nature Methods*, 18:1–9, 02 2021.

[14] Garrett Anderson, Maryam Ebadi, Kim Vo, Jennifer Novak, Ameish Govindarajan, and Arya Amini. An updated review on head and neck cancer treatment with radiation therapy. *Cancers (Basel)*, 13(19):4912, September 2021.

[15] Michelle B. Riba, Kristine A. Donovan, Barbara Andersen, IIana Braun, William S. Breitbart, Benjamin W. Brewer, Luke O. Buchmann, Matthew M. Clark, Molly Collins, Cheyenne Corbett, Stewart Fleishman, Sofia Garcia, Donna B. Greenberg, Rev. George F. Handzo, Laura Hoofring, Chao-Hui Huang, Robin Lally, Sara Martin, Lisa McGuffey, William Mitchell, Laura J. Morrison, Megan Pailler, Oxana Palesh, Francine Parnes, Janice P. Pazar, Laurel Ralston, Jaroslava Salman, Moreen M. Shannon-Dudley, Alan D. Valentine, Nicole R. McMillian, and Susan D. Darlow. Distress management, version

3.2019, nccn clinical practice guidelines in oncology. *Journal of the National Comprehensive Cancer Network J Natl Compr Canc Netw*, 17(10):1229 – 1249, 2019.

[16] Michel Verleysen John A. Lee. Lelec2870 machine learning : regression, deep networks and dimensionality reduction. `https://uclouvain.be/en-cours-2021-lelec2870`, 2020-2021.

[17] Yann Lecun and Yoshua Bengio. *Convolutional Networks for Images, Speech and Time Series*, pages 255–258. The MIT Press, 1995.

[18] Hiroshi Motoda and Huan Liu. Feature selection, extraction and construction. *Communication of IICM (Institute of Information and Computing Machinery, Taiwan)*, 5:67–72, 01 2002.

[19] Dingjun Yu, Hanli Wang, Peiqiu Chen, and Zhihua Wei. Mixed pooling for convolutional neural networks. pages 364–375, 10 2014.

[20] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015.

[21] Luis Perez and Jason Wang. The effectiveness of data augmentation in image classification using deep learning. *CoRR*, abs/1712.04621, 2017.

[22] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[23] Alex Labach, Hojjat Salehinejad, and Shahrokh Valaee. Survey of dropout methods for deep neural networks. *CoRR*, abs/1904.13310, 2019.

[24] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *Proc. of KDD-2013*, pages 847–855, 2013.

[25] Xin He, Kaiyong Zhao, and Xiaowen Chu. Automl: A survey of the state-of-the-art. *CoRR*, abs/1908.00709, 2019.

[26] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy P. Lillicrap, and David Silver. Mastering atari, go,

chess and shogi by planning with a learned model. *CoRR*, abs/1911.08265, 2019.

[27] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[28] Dan Nguyen, Xun Jia, David Sher, Mu-Han Lin, Zohaib Iqbal, Hui Liu, and Steve Jiang. 3d radiotherapy dose prediction on head and neck cancer patients with a hierarchically densely connected u-net deep learning architecture. *Physics in Medicine & Biology*, 64(6):065020, mar 2019.

[29] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[30] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[31] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1):145–151, 1999.

[32] Matthew D. Zeiler. ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701, 2012.

[33] National Cancer Institute. Nci dictionary of cancer terms. https://www.cancer.gov/publications/dictionaries/cancer-terms.

[34] Ana M Barragán-Montero, Melissa Thomas, Gilles Defraene, Steven Michiels, Karin Haustermans, John A Lee, and Edmond Sterpin. Deep learning dose prediction for IMRT of esophageal cancer: The effect of data quality and quantity on model performance. *Phys. Med.*, 83:52–63, March 2021.

[35] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016. `http://www.deeplearningbook.org`.

[36] Rafid Mahmood, Aaron Babier, Andrea McNiven, Adam Diamant, and Timothy C. Y. Chan. Automated treatment planning in radiation therapy using generative adversarial networks. *CoRR*, abs/1807.06489, 2018.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**