University Degree in Biomedical Engineering
2018-2019

Bachelor Thesis

IMPLEMENTATION OF AN AUTOMATIC CONTOURING WORKFLOW FOR RADIATION THERAPY PLANNING BASED ON DEEP LEARNING

Inés Muñoz Arnau

Javier Pascau González Garzón

Manuel Concepción Brito

Leganés, 2019

ABSTRACT

In the past few decades, the era of artificial intelligence has revolutionized the field of image analysis, reaching medical image analysis. In this line, automatic segmentation of regions of interest in medical images has appeared as an alternative to conventional segmentation methods (manually delineating the regions by an expert) with the aim of accelerating clinical processes. However, the need of numerous libraries and frameworks to run a neural network and the difficulty of handling them, difficult the implementation of such innovative techniques inside a real clinical workflow.

In this bachelor's final thesis, an intuitive user interface was created through 3D Slicer to automate the communication with a Docker Image that stored an already-trained neural network that performed the segmentation of 6 of the main organs at risk in head and neck CT images. Although the final setup in the Radiotherapy Service for which the application was designed was not accomplished, the steps and guide to do it were described. Besides, the deep learning based tool for automatic segmentation was proved capable of generalizing in other environments.

This project constitutes an approach on how to integrate inside a clinical workflow a tool for automatic segmentation of very concrete images (head and neck images). Nevertheless, this will lay the foundations for further research in this field to reach the final objective, which is to shorten times of radiotherapy treatment planning to provide better health care for cancer patients.

AGRADECIMIENTOS

Me gustaría agradecer este trabajo a mi familia y amigos, por apoyarme y animarme siempre; a los técnicos del servicio de radioterapia, por el cariño con el que siempre me trataron; y, en especial, a mi tutor Javier Pascau, por su inestimable ayuda y por darme la oportunidad de participar en este proyecto.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS AND ABREVIATIONS

HNC:	Head and Neck Cancer
HPV:	Human Papilloma virus
CT:	Computed Tomography
OAR:	Organs at Risk
MAS:	Multi Atlas Segmentation
ANN:	Artificial Neural Networks
ReLU:	Rectified Linear Unit
HGUGM:	Hospital General Universitario Gergorio Marañón
MRI:	Magnetic Resonance Imaging
PET:	Positron Emission Tomography
VM:	Virtual Machine
OS:	Operating System

1. INTRODUCTION

1.1 Head and neck cancer

Cancer continues to be one of the leading causes of morbidity in the world, with approximately 14 million new cases in 2012 [1] . As stated in [1], it is expected to increase by 70% in the coming decades, reaching up to 24 million cases by 2035, due to the increase in the life expectancy, among other factors. According to this, an estimation of 315.535 new cases will be diagnosed in Spain by the year 2035.

Head and neck cancer (HNC) includes a wide variety of malign neoplasms that arise in the head or neck region. Therefore, they include numerous anatomical areas with multiple sub-locations (Fig. 1.1), among which are: oral cavity, pharynx, larynx, paranasal sinuses, nasal cavity and salivary glands. Esophageal, skin and brain tumors are excluded from this definition [2].

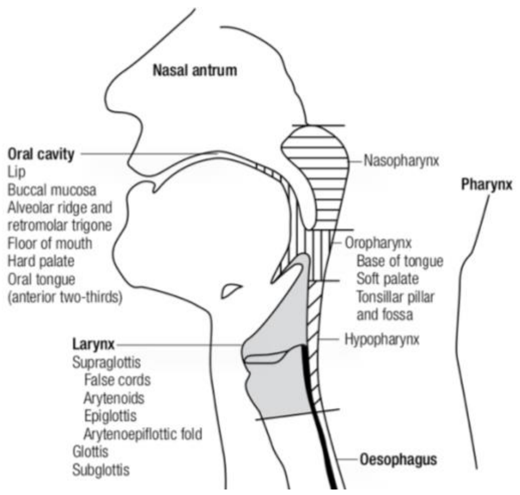
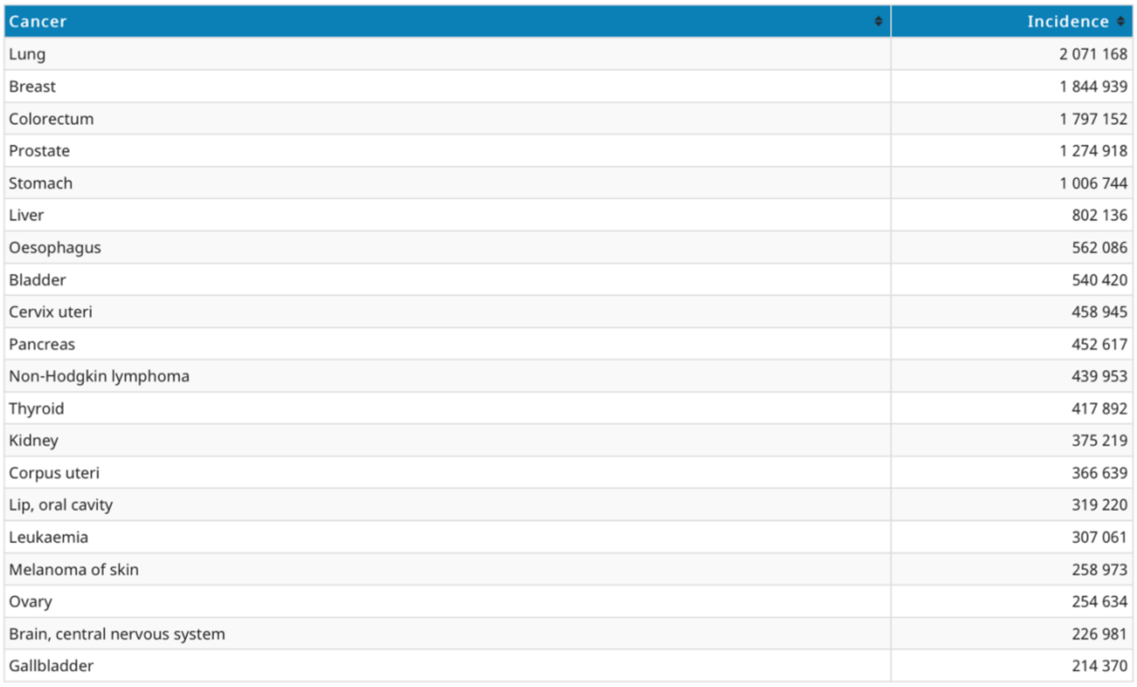


Fig. 1.1 Anatomical sites and subsites for the head and neck cancer [2]

The primary risk factors for this group of cancers are tobacco, alcohol consumption and human papillomavirus (HPV). The prevalence of HNC is related to frequency, intensity and duration of tobacco consumption, and it has a multiplicative effect when combined with alcohol. The HPV infection is the cause of HNCs that occur primarily in the oropharynx [3, 4].

Globally, HNC accounts for approximately 6% of all cancers, being oral cavity and laryngeal cancers the most common head and neck cancers. From [5] it can be concluded that these cancers are way more common among men than they are among women, and that its prevalence is increased with age. The estimated number of incidences of lip and oral cavity cancers in 2018 for both males and females aged over 40 worldwide was 319.220 [5] (see Table 1.1. Estimated number of incident cases worldwide, both sexes, ages 40+ [5] below). In Spain, this number was estimated to be 3.878 [5] .

Table 1.1. Estimated number of incident cases worldwide, both sexes, ages 40+ [5]



Cancer	Incidence
Lung	2 071 168
Breast	1 844 939
Colorectum	1 797 152
Prostate	1 274 918
Stomach	1 006 744
Liver	802 136
Oesophagus	562 086
Bladder	540 420
Cervix uteri	458 945
Pancreas	452 617
Non-Hodgkin lymphoma	439 953
Thyroid	417 892
Kidney	375 219
Corpus uteri	366 639
Lip, oral cavity	319 220
Leukaemia	307 061
Melanoma of skin	258 973
Ovary	254 634
Brain, central nervous system	226 981
Gallbladder	214 370

Treatment decisions are very complicated as they involve many specialists and they need to address the specific needs of each case: the location and stage of the tumor, and the general status of the patient. The patient, in partnership with family or caregivers, should participate in the decision of treatment, after the explanation of all the alternatives with their risks and benefits [6, 7].

Generally, the decisions for treatment depend mainly on the location of the tumor. Surgery is indicated in cases of small and easily removable lesions, like hypopharynx, paranasal sinuses or tongue tumors [7, 8]. Meanwhile, radiotherapy (usually with adjuvant chemotherapy) is mostly preferred for organ preservation, and for palliative treatments in patients with recurrent and/or metastatic diseases [6].

This thesis lies in the implementation of a tool for the radiotherapy treatment process, specifically to accelerate the planning stage. Hence, Radiation Therapy will be treated in the next point.

1.2 Radiotherapy

Radiation therapy consists of the use of ionizing radiation for the treatment of diseases, mainly of a malign nature. It comprises the administration of a dose of high energy x-rays at an anatomical volume, to eliminate the neoplastic cells, trying to produce the minimum possible damage to healthy tissue. It deals with the treatment of more than the 60% of the cancers and it is also indicated in many benign nature diseases, due to its antalgic and anti-inflammatory properties. Along with surgery and chemotherapy, radiotherapy constitutes the basis for the treatment of neoplastic diseases.

Technically, the modalities of radiation oncology can be classified according to the distance of the radiation source from the patient's anatomy [9]:

- External radiotherapy. The radiation is administered with a remote radiation generator. Now, linear accelerators are used for administration.
- Brachytherapy. The source is in direct contact with the tumor. Different types of radioisotopes are used, depending on the therapeutic objectives.

Radiotherapy can be indicated as an exclusive method with curative purpose, as a complementary method to surgery or chemotherapy to reduce the risk of local relapse, or with palliative intention, to control pain or improve symptoms.

In oncology, the choice of the best treatment is based on finding the one that offers the patient the best quality of life with the least number of side effects, with equal efficacy in survival. The advantage of radiation therapy is that it preserves the organ and, sometimes, the function, which means the possibility of keeping the voice in a larynx cancer or retaining the swallowing function in a tongue cancer. In addition, most patients do not require hospital admission during treatment. This only happens in 5-10% of cases, mostly due to a poor physical condition of the patient.

However, it is a long-lasting treatment, which is administered daily for 4-7 weeks in cases of curative purpose, which implies displacement and the stress of going to the hospital every day.

1.3 Radiotherapy Service

Once Radiation Therapy is selected as the first or next step for a cancer treatment, the starting point consists on basically determining where to aim radiation beams and how much radiation should the tumor get (treatment planning). This will shape a collimator for the treatment machine to direct the rays to the tumor and block the areas that do not need to be treated.

The Radiation Oncology Service offers personalized clinical and therapeutic care to the cancer patient, through the indication, planning, control, treatment and follow-up of the patient treated with radiation and associated therapies. The main services are:

Medical consultations

The service includes first visits, successive treatment visits and post-treatment reviews.

Simulation and immobilization

The first step of the treatment planning. It involves a computed tomography (CT) scan that will allow the doctor to locate the exact area to be treated and protect the healthy tissue.

Before taking the scan, the patient is positioned in the exact same position in which he will receive treatment. To do so, immobilization systems are used and reference marks are tattooed thanks to a laser system. This will assure that the patient is right where it is supposed to be when the treatment is delivered.

In head and neck cancers masks are used as immobilization systems. The patient lies down on the scanner bed and thanks to a thermoplastic mesh material, a mask of his face is made (Fig. 1.2). Reference marks are painted to assure that the position will be maintained during the treatment sessions.

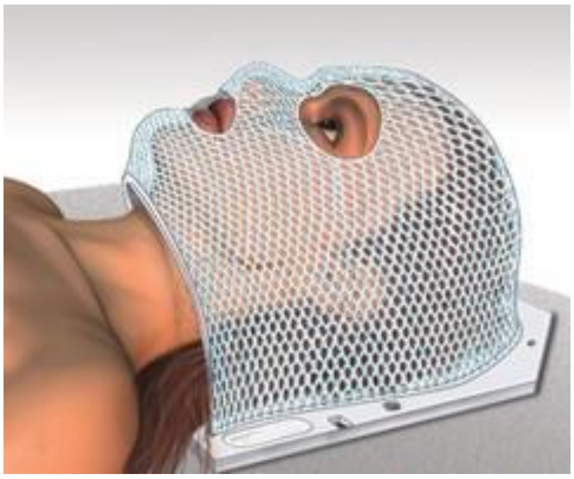


Fig. 1.2. Immobilizer for radiotherapy in head and neck cancers [10]

After the immobilization of the patient, a CT scan will be obtained to have a three-dimensional image of the treatment area.

Treatment planning

After the CT is obtained, technicians will manually delineate the organs at risk (OAR) in the tumor area, using a computer software. Doctors will then delineate the treatment area so that physicians can plan where to aim the radiation beams and where to avoid them.

This is a crucial step that will determine the success of the treatment and usually takes between 15 and 20 days.

Once the physicians have planned the dose, lead blocks are individually made so that the beam is shaped to irradiate the target volume and precisely deliver the dose on the volume.

Treatment delivery

After the planning step has finished and all the necessary tests have been carried out according to the dose delivery, the treatment process will begin, which is usually given once a day, five days a week, for five to seven weeks depending on the treatment.

2. BACKGROUND AND STATE OF THE ART

Segmentation of biomedical images involves the process of labeling the pixels of an image with the aim of highlighting regions of interest, like anatomical structures [11]. In Radiotherapy workflows, a medical image is manually delineated by a trained expert. However, this practice is labor-intensive and impractical, as it requires outlining the structures slice by slice. Besides, it can be inaccurate due to inter-operator's variability.

Research into automatic segmentation methodologies has been made on two main approaches [11, 12], reviewed in this chapter. Multi-atlas approach (reviewed in point 2.1 Registration-based segmentation methods) treat segmentation as a registration[1] problem, which is limited by the great variability between subjects. To overcome these challenges, registration-free methods (point 2.2 Registration-free methods) rely on looking for invariant features (manually imposed or learned) that characterize the anatomy in an unregistered training data set.

2.1 Registration-based segmentation methods

The first atlas-guided segmentations were based on a single atlas, a single image manually-labeled by a specialist [11, 12]. The mapping obtained in the registration of the two images was used to transfer the labels from the atlas to the novel image.

However, this technique does not consider the possible anatomical variations between subjects. Therefore, research on the multi-atlas registry was conducted from two approaches [11]. The first, based on a set of images, identified the most significant atlas and used it to segment the new image. However, the most popular approach was probabilistic segmentation based on atlas, which used a unique atlas built from the joint registration of the training set images. In addition, information on the likelihood of observing a label at a given location is derived from segmentation. The benefit of this method is that, once the atlas is generated, only one registration is needed to segment the desired image.

Later, the multi-atlas segmentation methods (MAS) appeared, which, instead of grouping all atlas into one model, register the new image with each of the atlas images. Then, the most repeated tag is selected in each voxel (known as "majority vote") [11].

2.1.1. Multi-atlas Segmentation methods

The main steps of a general MAS algorithm are represented as in the form of blocks in the figure below (Fig. 2.1). From there it can be seen that the first step is generating the atlases, which are a set of labeled images, typically obtained by an expert. As stated in

[1] Image registration is the process of aligning two or more images, by designating one of them as the reference image, and applying geometric transformations to the others so that they align with the reference [14].

[11], depending on the case of study, the atlases can be pre-processed to improve performance, or select only the high-quality images. Alternatively, one can exploit the variability of segmentations by selecting only images segmented by non-experts.

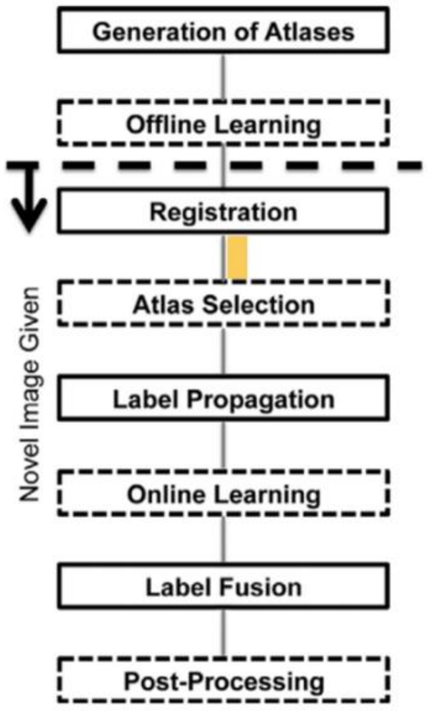


Fig. 2.1. Blocks for generating a MAS method. Dashed blocks are optional [11]

Optionally, before performing the registration, one can proceed with what is known as "Offline Learning" [11], a modern tool that consists of processing the atlases with no prior knowledge about the image to be segmented. Such pre-processing tools can be related to the weight assignment for the label fusion step. For example, co-registering the set of images and comparing the propagated labels can determine the atlases that better predict the rest of labels and assign them the higher weights [11].

The core step of the algorithm is the Registration, which has three main components: the deformation model, the objective function and the optimizer [11]. The deformation model involves type of transformations that will be made to one or more images to maximize an objective function. This function is typically related either to spatial distance between landmarks (manually or automatically detected) or to image intensities. The optimizer is usually an iterative method, such as gradient descent.

Normally, a registration is performed between the new image and each of the images in the atlas, which is a computationally expensive practice. To reduce run time, atlas selection may be implemented. Reducing the number of atlases improves both computational efficiency and segmentation accuracy, as irrelevant atlases may be removed.

Once the registration is performed (spatial correspondences between images are achieved), these methods proceed to transmit the labels from the atlases to the novel image to acquire segmentation. One of the most popular method is the nearest neighbor interpolation, where each label of the novel image will correspond to that of the closest voxel in the atlas space [11].

Label fusion is the step of combining propagated atlas labels, which can be achieved by best-atlas selection methods or majority voting, among many others [11]. In best atlas selection, a single atlas is chosen based on, for example, the match in intensities between the registered atlas and the new image. Meanwhile, in majority voting, the most frequent label at each location is chosen. This can be improved with weighted voting, where each atlas is associated to a weight depending on its similarity to the novel image.

Post processing can be applied to the label fusion outputs in a wide variety of ways, from starting an active contour method or fitting a contour into a bounding box, to comparing the intensities with the expected ones, among many others [13, 14, 15, 12].

MAS provides very precise segmentation algorithms, as they take into consideration all the images in the data set. Nevertheless, it is computationally expensive and time and memory consuming. This may be the main reason why it has not been widely adopted in clinical fields. In addition, the core of these methods is registration, which is highly conditioned by variations between subjects. That's why the most common application for MAS algorithms has been brain images, where good alignments are achieved even with the simplest algorithms [16, 11].

2.2 Registration-free methods

Recent advances in artificial intelligence, computational power and data availability, have enabled the training of more complex registration-free methods, which base their challenges on finding invariant features that optimally represent the anatomy in a training data set [12]. This concept lies at the basis of many Deep Learning algorithms: a set of processed algorithms based on Neural Networks that attempts to model high-level abstractions in data using architectures composed of multiple layers that transform input data into output.

Artificial Neural Networks (ANN) have been developed based on the structure and behavior of the nervous system [17]. They are information processing systems capable of generalizing and learning from experience, and in many biomedical applications they can reach better decisions than even doctors.

In a neural network, nodes are placed in the layers, where the leftmost layer is called the input layer, the layer in the right is the output layer, and the layers in the middle are known as hidden layers [17, 18]. Besides the input nodes, each node in the ANN has one or more inputs and only one output, as can be seen below in Fig. 2.2.

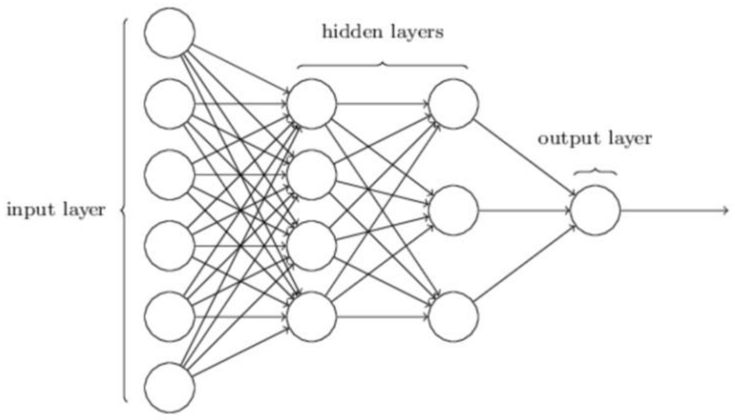


Fig. 2.2. Architecture of an Artificial Neural Network, here with two hidden layers [18]

Each node has specific parameters to compute its output as in Equation 2.1, where x_i are the inputs to the node, ω_i are the weights associated to the node, b is the bias of the node, and $f(x)$ is the activation function [17].

$$f(\omega_i * x_i + b) \tag{2.1}$$

The most extended activation function is the sigmoid function, whose output with inputs, weights and biases is:

$$\frac{1}{1+exp\,(-\sum_j \omega_j x_j - b)} \tag{2.2}$$

This activation function has a smooth shape when plotted, which means that small changes in weights and biases produce small changes in output. This is the basis of the learning of a network.

Given a set of inputs and the desired outputs, the goal of the training process is to find the weights and biases that minimize the difference between the desired outputs and outputs that the network gets. Such process is achieved with an error measurement.

The learning of a network is achieved by modifying the weights and biases of the nodes until the desired outputs are obtained. The network learns to perform certain functions through training with examples (modifying the weights in order to reduce the error). Gradient descent is the most widely used optimization algorithm, where the weights and biases are first initialized to a random point and they are adjusted iteratively to decrease the error with a backpropagation algorithm.

In backpropagation, for each iteration of the network, the output of each neuron is computed and the results are compared with the desired ones. Then, the error of each neuron is calculated backwards and the weights are adjusted so that it decreases. The training ends when the sum of the errors is below a threshold.

Several Deep Learning architectures, such as Convolutional Neural Networks (CNN) have been applied to biomedical fields proving to produce excellent results. They are characterized by applying techniques that allow avoiding the so-called "gradient vanishing" which makes it possible to effectively train very deep networks (more than 2 hidden layers).

2.2.1 Convolutional Neural Networks

A CNN is a type of neural network used in image recognition and classification where neurons are very similar to the ones in the primary visual cortex. In [19] it is explained how to use a CNN to classify an input into four categories: dog, cat, boat and bird (Fig. 2.3). As evident, given the input in the image below, the net should obtain the highest probability for boat among all four categories.

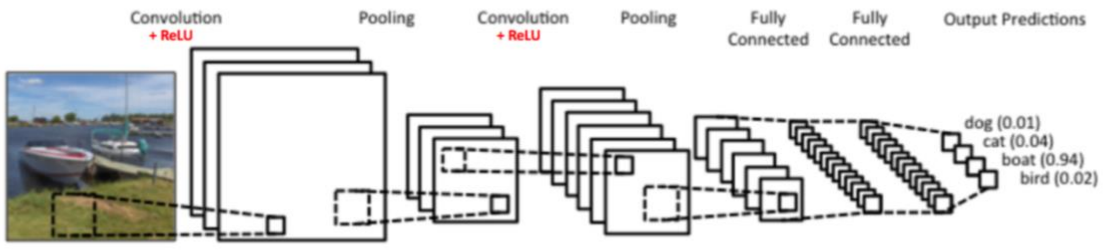


Fig. 2.3. Architecture of a convolutional neural network [19]

CNN usually have three types of layers: convolutional, pooling and fully connected layers.

Convolutional layers use kernels (filters) to extract information from inputs by performing convolutional operations. That is, given an input, the kernel will slide over the image computing the dot product to obtain what is called “Feature Map” or “Activation Map”. CNN usually use ReLU activation function to generate these feature maps. This activation function stands for “Rectified Linear Unit” and it performs a non-linear operation and replaces all negative values in the feature map by zero. For positive inputs, the ReLU function grows to infinite [19].

Next, the main purpose of the pooling layer is to reduce the number of parameters, which helps reduce overfitting and computational cost and, at the same time, extracts representative features from the input. Spatial pooling can be of different types: max, average, sum, etc. In case of max pooling, the max value for each kernel position is taken and put in the corresponding position in the output matrix. In case of average pooling, the average is taken for every position.

In the example in [19], we can appreciate that after the convolution and ReLU step, several feature maps are obtained. Pooling is then applied separately to each of them. Right after this step, we have another set of convolutions + pooling, which computes convolution with 6 types of filters, obtaining 6 different feature maps and performing max pooling to the 6 of them. After these steps, the net has managed to extract the most relevant features of the image and to reduce feature dimension.

Fully connected layers are the last few layers in the network, and their aim is to use the features obtained in the previous steps to classify the image into the classes, based on the training dataset. The sum of output probabilities in the output layer is 1.

Finally, the overall training process in a CNN is as follows: filters and weights are initialized randomly, the network takes a training image and follows all the steps forward to find the output probabilities for each class. Then, the error is obtained by computing the difference between the target probabilities and the output probabilities, and the backpropagation algorithm together with the gradient descent is used to adjust weights. When the error is lower than a threshold, the backpropagation algorithm ends, which means that the network learnt to classify that particular image [19].

All those steps are repeated with all the images in the training data set using the weights and parameters calculated in the previous step. If the training set is large enough, the network will generalize well to new images and classify them correctly [19].

Convolutional Neural Networks have proven to get optimal results concerning image analysis and classification, which make these methods the state-of-the-art of image segmentation and object recognition. They have been applied to many applications, such as digit or face recognition. Their great performance in these fields has placed CNN in the top of automatic methodologies for medical image segmentation.

3. PROJECT JUSTIFICATION AND OBJECTIVES

3.1 Project justification

Medical image segmentation refers to the process of labeling the anatomical structures of an image by a trained expert so that they are easily recognized by an observer. In a Radiotherapy Service, one of the main steps of treatment planning is the segmentation of the OARs, right after the CT scan has been obtained. The OARs are the structures closed to the tumor where radiation should be avoided so as not to damage healthy tissue, and they are different depending on the location of the tumor.

This task is performed manually by trained physicians, what makes this practice slow, prone to human error and difficult to reproduce. Once the delineation of the OARs has been performed, a doctor will review the segmentation and will proceed with the delineation of the tumor. This whole practice is time-consuming and, considering the large number of new cases that a hospital hosts in one day, impractical for Radiotherapy workflows, where time is a critical matter.

In the past few decades, research was carried out into automatic segmentation tools, motivated by the drawbacks mentioned above. From those, we can conclude that Multi-Atlas Segmentation methods offer high-quality segmentations, as they can accurately capture anatomical variations. However, this comes at a high computational and time cost, which has motivated to run investigations towards registration-free methods. In this way, deep learning models are the most extended tool for automatic segmentation, being the CNNs the state-of-the-art.

However, implementing deep learning models requires the installation of numerous libraries and frameworks which makes it very difficult to integrate them in the clinical workflow, causing a gap between the state-of-the-art and the real praxis.

The aim of this thesis is to integrate a deep-learning-based tool in the Radiotherapy Service of the HGUGM (Hospital General Universitario Gregorio Marañón) for the automatic segmentation of OARs in head and neck tumors. The student will be given an already-trained deep learning model with the goal of integrating it into the service providing an intuitive user interface. Specifically, the neural network is trained to perform the segmentation of six structures: left and right eye, brain stem, spinal cord and left and right parotid. For that purpose, this project is supported by the 3D Slicer environment, the Docker Container Platform and the already-built convolutional neural network that performs the segmentation.

In this way, this project constitutes an approach on how the process of planning a radiotherapy treatment could be accelerated, so that faster attention is given to patients. Besides, it is a demonstration of how to integrate a deep-learning based model for image segmentation into a real clinical workflow, which will bring the latest techniques in automatic segmentation to the real environment.

3.2 Objectives

The overall objective of the project is to accelerate the planning process of the Radiotherapy treatment by integrating a tool for automatic segmentation of the OARs in head and neck cancers. Specific objectives of the project are:

1. Create a Docker repository to host the deep learning model
2. Develop a module in Slicer that runs the model by calling the Docker container
3. Validate the module with a set of training images
4. Design the new workflow that images will follow from its obtaining in the CT scan to its segmentation in the planning room
5. Implement the new workflow in the service

4. TIMELINE AND WORKING METHODOLOGY

In this section, the activities to perform by the student will be presented (Fig. 4.1), as well as the working methodology and the way the results will be evaluated.

In stages 1 to 4, the student underwent a series of tutorials and courses to learn programming in Python, to deeply understand neural networks and to study how Docker works and learn to containerize an application.

Once accomplished, the student started the programming stages to achieve the objectives of this project: creating an application for the radiotherapy service of the HGUGM to automatically segment the OARs of a given image. It should be highlighted that along the first stages of the process, the student performed a professional internship in the radiotherapy service of to get to know the way it worked, the workflow of the images, the whole process of treatment, etc. This way, the student could ascertain how to integrate such a tool into the service, considering the distribution of the working stations and the point of view of the physicians working there. These actions were performed in the last weeks of June. And from that stage onwards, the student underwent an improvements stage, were corrections were made to the Slicer module, remotely from Valencia.

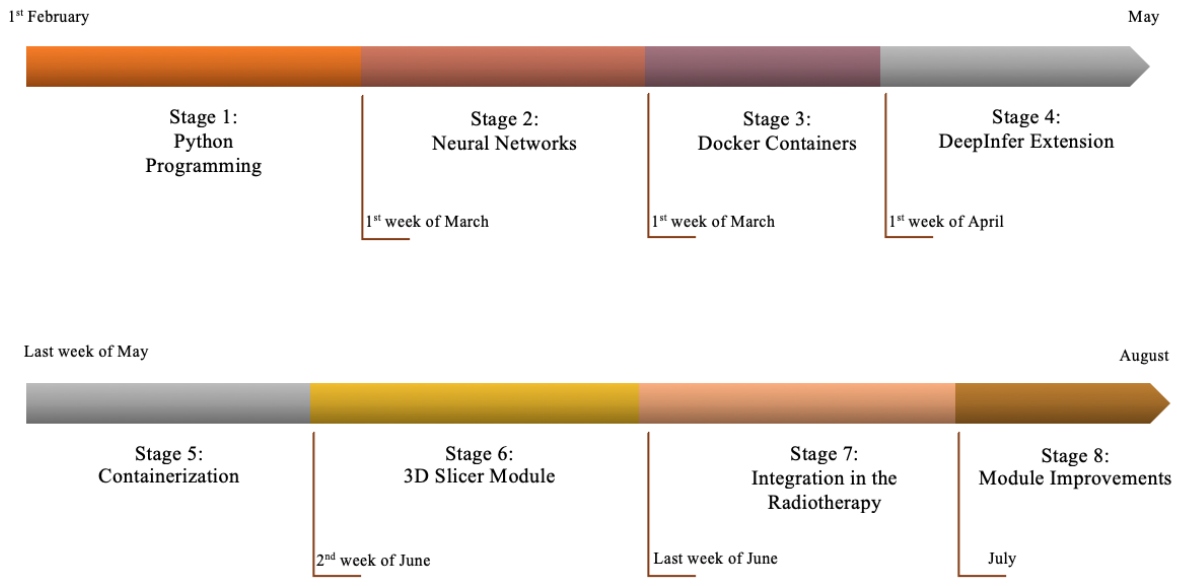

1st February
May
Stage 1: Python Programming
Stage 2: Neural Networks
Stage 3: Docker Containers
Stage 4: DeepInfer Extension
1st week of March
1st week of March
1st week of April
Last week of May
August
Stage 5: Containerization
Stage 6: 3D Slicer Module
Stage 7: Integration in the Radiotherapy
Stage 8: Module Improvements
2nd week of June
Last week of June
July

Fig. 4.1. Timeline of the project

The student underwent through all the stages with the help of her supervisor Manuel Concepción Brito. The overall working methodology basically consisted on the following steps:

1. Problem understanding. Along the student's internship, she was able to understand the needs of the service and the importance of automatizing the treatment planning process.

2. Solution research. Once the problem was understood the student started a research on how to give a solution to the problem, taking into consideration the materials available for the project (described in the following point, MATERIALS).
3. App programming. The programming stage consisted on performing the solution established, with the guidance and feedback of the supervisor.
4. App evaluation. The final steps consisted on performing the evaluation methods described in 6.2 Evaluation methods point.

Along the last steps of the study, the student performed several tests on the software to check the adequacy of the results obtained and execute the changes required.

5. MATERIALS

5.1 3D Slicer

3D Slicer is a free open source software platform to analyze and visualize medical images, supporting multi-modal imaging (CT, MRI, PET). It is available on numerous operating systems and includes several tools like segmentation and registration [20, 21]. From the perspective of a biomedical engineer or computer scientist, 3D Slicer is a platform for development and delivery of algorithmic programs.

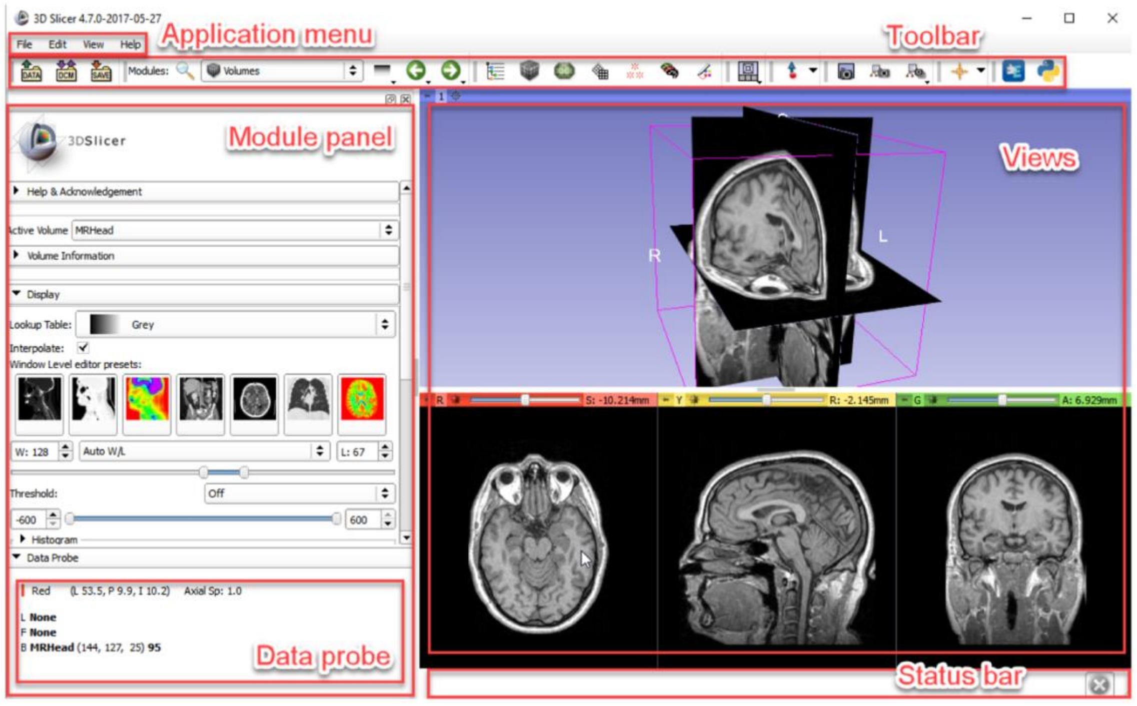
5.1.1 Hardware requirements

3D Slicer package that can be run on Mac, Linux and Windows with graphics capabilities and memory enough to hold the original image data and process results. Besides, a 64-bit system is required [20].

There are two available versions to download for each operating system. The nightly version has the newest releases, as it is updated every night with changes that group of developers make. The stable version has older releases, but they are more carefully tested.

5.1.2 Application overview

Slicer offers an intuitive user interface that allows the implementation of several tools and modules for specific tasks delivery in clinical research applications. The user interface is divided into six components: the Application menu, the Toolbar, the Module panel, the Data probe panel, the 3D Viewer and the Status bar (Fig. 5.1).



```
Red    (L 53.5, P 9.9, I 10.2)    Axial Sp: 1.0
L None
F None
B MRHead (144, 127, 25) 95
```

Fig. 5.1. 3D Slicer application main view [20]

Toolbar offers quick access to commonly used tasks, such as modules list and loading/saving data.

Module panel provides all the options and features that the selected module offers to the user.

Slicer saves all the loaded images in the Slicer Scene and represents the data as a node in the scene. When moving the mouse over the image the information at each point is shown.

5.1.3 Data loading and saving

There are two main types of data that can be loaded into Slicer, accessible from the leftmost side of the Toolbar:

- DICOM[2], which can be loaded from the DICOM browser once it's been imported to the data base.
- Non-DICOM, covering both conventional image formats(JPG, TIFF…) and medical field specific formats (NIfTI…), which can be loaded with drag and drop on the Slicer window, or by using the button Data in the Toolbar.

The data loaded in the Scene is available through the Data module, which can be found in the Toolbar or the Modules list. There, the node hierarchy can be modified to create subjects, studies and so on.

Saving can be accessed through the button Save in the Toolbar. The node can be exported in all image formats. Hence, with Slicer we can create a DICOM image from a nifty image (or nrrd), and vice versa.

5.1.4 Slicer architecture

Slicer is composed mainly of three structural levels (Fig. 5.2). The lowest one includes fundamental libraries that the operating system provides to allow the use of windowing and graphics resources of the host system. In the intermediate level, there are the languages (such as python or C++) and libraries like Qt (for GUI framework), DICOM Toolkit (used to interact with DICOM data and services) and jqPlot (provides charting capabilities) [21]

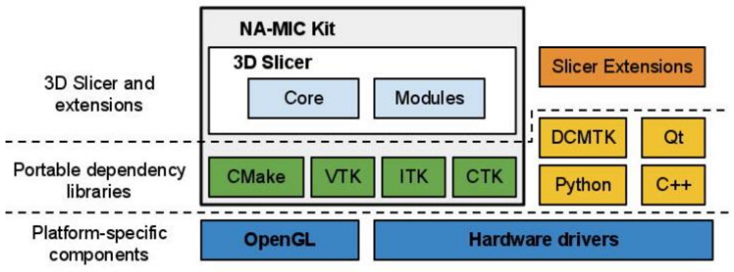


Fig. 5.2. Slicer architecture [21]

The highest level, the application itself, consists of the application core, Slicer modules and Slicer extensions. The core deals with data visualization and developer interfaces that support extension of the application with new plugins. Slicer modules are those internal

[2] DICOM is a widely-used set of standards for digital radiology [15]

plugins dependent on the core to implement new functionalities. Meanwhile, Slicer extensions are modules that are not packaged as part of Slicer distribution, they are freely installed by the user through the Extensions Manager [21].

Slicer modules allow the implementation of different tasks for clinical research applications, such as segmentation, registration, volume rendering, etc. A wide variety of modules are available to the user through the interface. Furthermore, the developer can create its own extensions and modules by implementing a python program based on the templates that Extension Wizard delivers.

The Extension Wizard is a module that allows the user to select an extension (directory) and add a module that provides template functionalities. After that, changes can be made in the program using Python language to add the buttons and functionalities desired and obtain a module with a user interface that performs whatever function wanted.

In this thesis, a module will be created to deliver the segmentation of the OARs in a head and neck CT image.

5.2 Docker

Docker is a tool for deploying and running applications in an isolated environment by using containers. This device makes sure that different resources are isolated so that there is no clash in any environment by using a build-once deploy-everywhere concept: basically, write the code once and deploy it everywhere, without going to the hassle of version collisions [22, 23].

It is similar to running an application inside a Virtual Machine (VM), but without the difficulty of setting up and managing a VM. Instead, in Docker the code and environment are wrapped inside a container that runs on the host operating system.

5.2.1 Docker containers vs Virtual Machines

Virtual Machines import a guest Operating System on top of a host Operating System (OS). In Fig. 5.3 there are three guest OS (which are the three VMs) over a host OS. This architecture will lead to poor performance and will take a big amount of system resources, because each guest will have its own libraries [22, 24].

Containerization brings the virtualization to the operating system level. Instead of bringing abstraction to the hardware as VMs do, containers bring abstraction to the software. It is more efficient because libraries and resources are shared in the host, which makes the process faster. Containers are lighter and faster than VMs [22, 24].

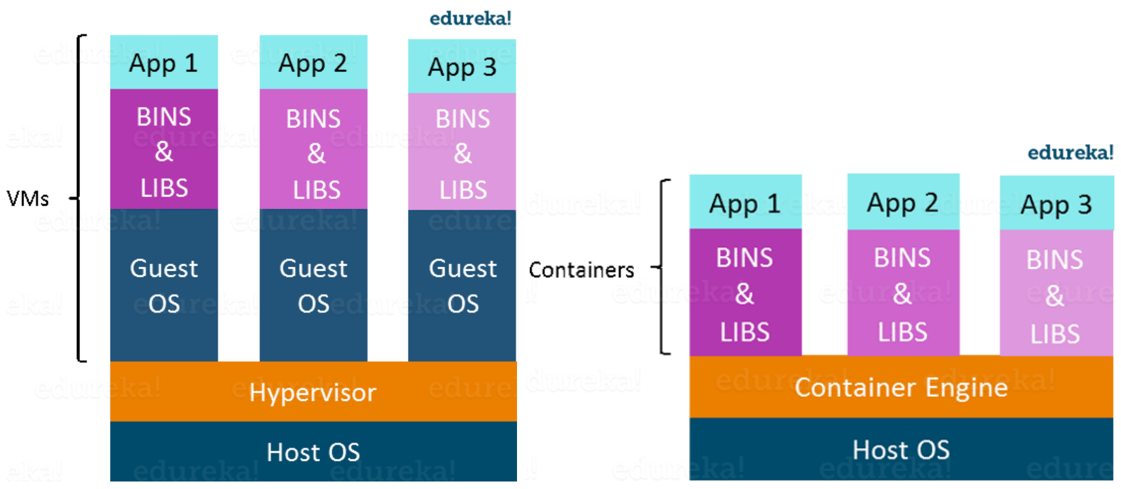


Fig. 5.3. Comparison between Virtual Machines(left) and Docker Containers (right) [24]

The Docker containers use a Host Operating System. On the top of it there's the Docker Engine, where Docker containers are formed. These containers have applications running on them as well as all the requirements they need (binaries and libraries). Each application will run on a separate container.

5.2.2 Docker Overview

Let us now understand the Docker system in detail, which is a client-server application with the following major components [24]:

- A client, which is the command line interface
- A server, which is a running program called Daemon
- A rest API, used for communicating between the client and server

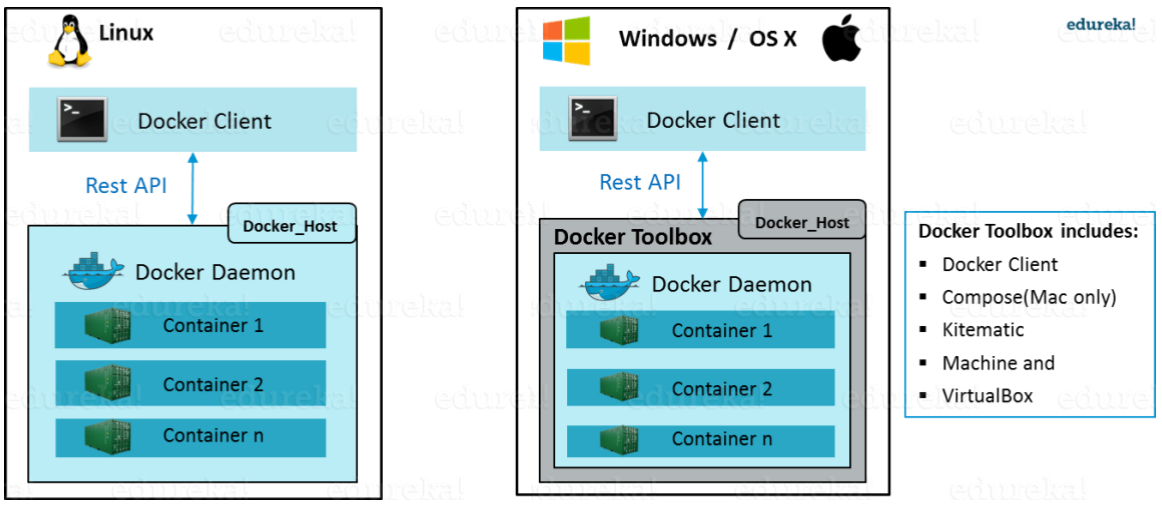


Fig.5.4. Docker Engine in Linux OS (left) and Windows/MAC OS (right) [24]

The difference between a Linux OS and a Windows/OS X is that the first one accesses the Daemon directly from the client (the terminal), while the other two use an additional component, the Docker Toolbox, to communicate with the client (Fig.5.4).

5.2.3 Docker Architecture

Before going through the architecture and the way the communication is performed between the components, some concepts will be explained.

Docker Images are the blocks that build containers providing the desired environment to create the application. They are defined using a Docker File, which is just a file with a list of steps to create that image.

Docker Containers are running instances of Docker Images. They hold the entire set to deploy an application.

The Docker Registry is the cloud where developers and communities upload images so that they can be publicly accessed by other users.

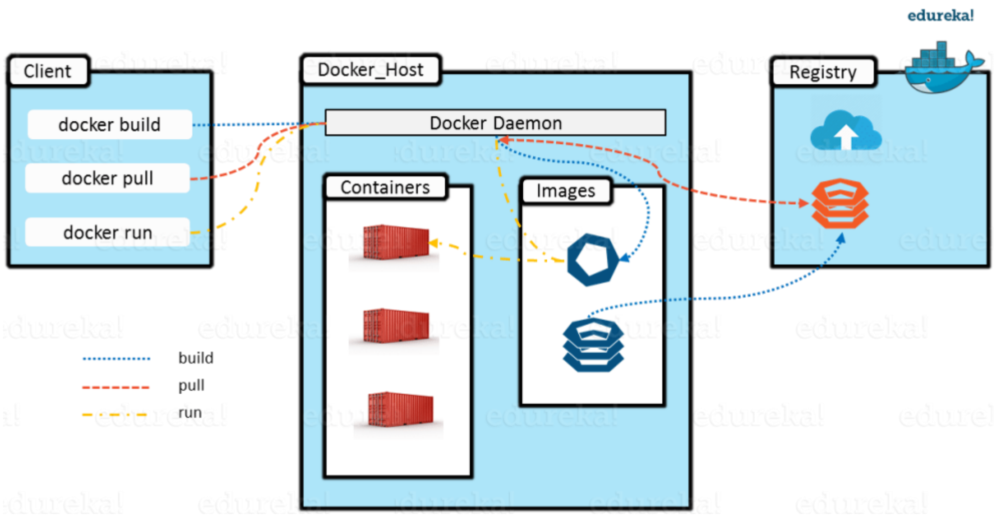


Fig. 5.5. Docker architecture and flow of the instructions [24]

The Docker Daemon is responsible for images and containers and for processing the commands of the client.

To build a Docker Image, the Docker client will call the Daemon by using the *docker build* command. The Daemon will build an Image based on the inputs that a Dockerfile offers and will save it in the Registry, which can be either the Docker Hub[3] or a local repository (Fig. 5.5).

If preferred, an already-built Image can be pulled from the Docker hub with the *docker pull* command.

Finally, to build a container, the Docker client will call the Daemon with the *docker run* command.

[3] Docker Hub is a cloud registry where developers and communities upload Docker Images

5.2.4 Docker Workflow

To fully understand how to build a containerized application, let us go through the flow of instructions (Fig. 5.6).

The first step is building a Docker File, which is a file that contains a bunch of instructions that define the libraries and required dependencies for a particular application. To build a Docker Image out of this file, the *docker build* command will be employed.

The image can be run to create as many Docker Containers as desired. Every time the application needs to run, a Docker Container is created.

Finally, the Image can be uploaded to Docker Hub, where anyone can pull it to run the application.

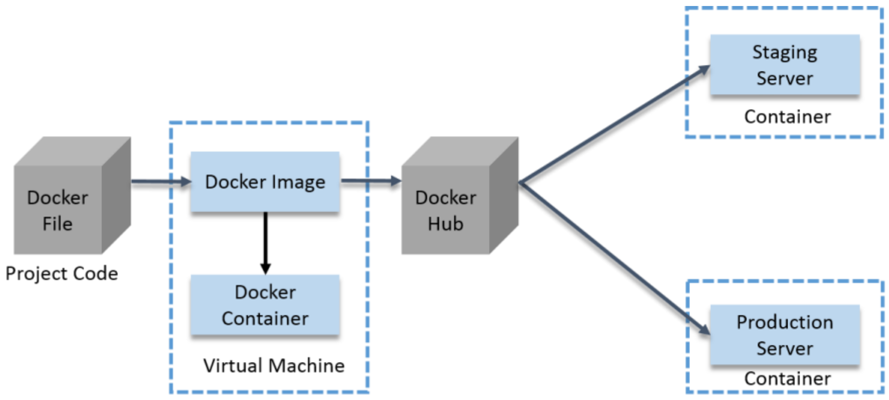


Fig. 5.6. Docker workflow for developing an application [24]

5.3 Convolutional Neural Network

The student was given an already-trained Neural Network for performing segmentation of 6 OARs in head and neck cancers to a given CT image.

The Neural Network is based on an example that demonstrates the use of a NiftyNet [25] to train a Dense V-Net to perform multi segmentation of organs on abdominal CTs [12]. NiftyNet is an open-source platform intended to share CNN and pre-trained models with researchers to quickly give deep-learning-based solutions to image analysis problems.

The network, provided by Manuel Concepcion, uses a V-shaped architecture showed in Fig. 5.7, which is mainly described in terms of 3 key features: downsamplings, upsamplings and convolutions. As it can be observed, after a convolution, a cascade of dense feature stacks and convolutions generate activation maps at three resolutions. The dense feature stacks constitute the downsampling stages and are basically a sequence of convolution blocks where the inputs are concatenated features from the preceding blocks. A convolution is applied to each resolution to reduce the number of features, and all outputs are upsampled to the initial downsampled size. If initial prior is given, it is added to the prediction made after the final concatenation.

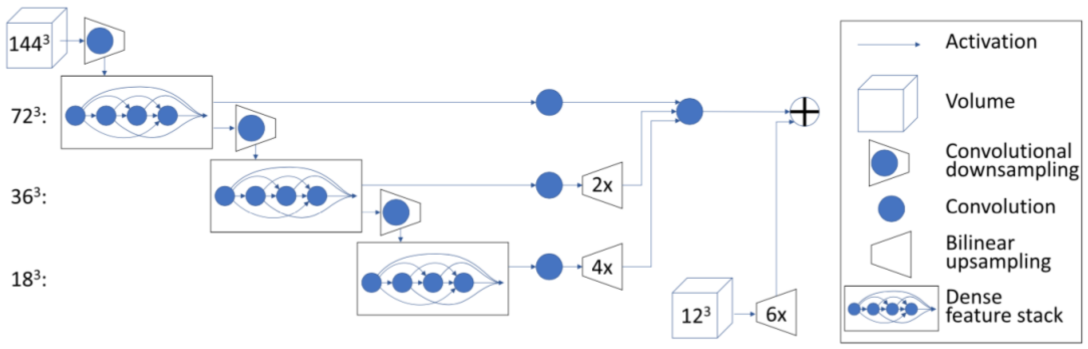


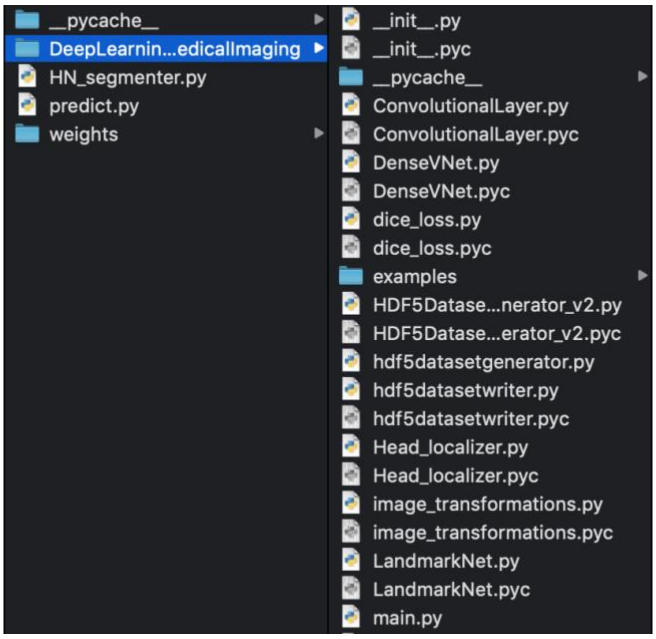
Fig. 5.7. DenseVNet network architecture [12]

The data used to train the network comprises a set of 59 head and neck CT images, manually segmented by experts, under the same criteria for the results to be the best possible.

Apart from the Neural Network itself, the student was also given two files that execute the network. The main one, 'predict.py', defines the weights path and loads the models. The segmentations are performed individually loading one model for every organ at risk. Once loaded, the segmentation process is initiated by calling the other file, 'HN_Segmenter.py', which preprocesses the image and obtains segmentations. Finally, results are stored in a default directory. The network is trained to segment images in nifty format, and results are also obtained in the same format.

From this it can be concluded that to obtain segmentations of a given image, the predict.py must be executed. This will be the basis of the project.

In Fig. 5.8 below, an overview through all the files that compose the Neural Network is provided.



```
__pycache__                          __init__.py
DeepLearnin...edicalImaging          __init__.pyc
HN_segmenter.py                      __pycache__
predict.py                           ConvolutionalLayer.py
weights                              ConvolutionalLayer.pyc
                                     DenseVNet.py
                                     DenseVNet.pyc
                                     dice_loss.py
                                     dice_loss.pyc
                                     examples
                                     HDF5Datase...nerator_v2.py
                                     HDF5Datase...erator_v2.pyc
                                     hdf5datasetgenerator.py
                                     hdf5datasetwriter.py
                                     hdf5datasetwriter.pyc
                                     Head_localizer.py
                                     Head_localizer.pyc
                                     image_transformations.py
                                     image_transformations.pyc
                                     LandmarkNet.py
                                     LandmarkNet.pyc
                                     main.py
```

Fig. 5.8. Folder organization of the Neural Network provided

6. METHODS

As already mentioned, this thesis is an approach on how to give an answer to the problems and needs of the Radiotherapy Service in the HGUGM in relation to improving and accelerating the process of the treatment planning. The solution proposed by the student to provide automatic segmentation of the OARs is: to create a module in 3D Slicer that automates the process of calling a neural network trained to segment 6 OARs for head and neck CT images and stored in a Docker container. This chapter demonstrates the methods used to implement and evaluate such solution.

6.1 Implementation

The three main steps for implementing this project are: creating a Docker container to store the provided Neural Network, creating the Slicer module to provide a User Interface to all the process, and integrating the designed application into the Radiotherapy Service.

6.1.1 Containerization of the Neural Network

6.1.1.1 Dockerfile guide

First step to store an application inside a Docker container is creating a Dockerfile, which is a plain text file that contains all the commands a user should enter in the command line to create a Docker Image [26].

The format of the Dockerfile is as follows:

```
# Comment

INSTRUCTION arguments
```

Docker treats lines that begin with # as comments. Instructions are run in order and, although they are not case-sensitive, they are usually uppercase to make the document more readable.

Hereby the main instructions that can be used to build a Dockerfile are explained, and the Dockerfile used for the project is shown in Fig. 6.2. This Dockerfile was based on the official Dockerfile for Keras [27] and on the one used in [28] to containerize a Keras model.

FROM

```
FROM <image>

FROM <image>[:<tag>]

FROM <image>[@<digest>]
```

This is the first command of a Dockerfile. It sets the base Image for subsequent instructions. Every Docker image is based on a base image, which is a way to specify the exact configuration to start with and then one by one we specify the libraries to install on the top of it. Tags and digest are two ways of specifying the image versions [26].

There is a wide range of available images to download in the Docker Hub [29].

ENV

```
ENV <key> <value>

ENV <key>=<value> ...
```

The ENV instruction sets the environment variable <key> to the value <value>. The difference between the two forms is basically that with the second one more that one value can be set to the variable [26].

RUN

```
RUN <command>
```

The RUN instruction executes any command in a new layer on top of the current image and commits the results to be used for the next step in the Dockerfile. A backslash \ can be used to continue a single RUN instruction on to the next line [26].

WORKDIR

```
WORKDIR /path/to/workdir
```

The WORKDIR instruction sets the working directory for any RUN, CMD, ENTRYPOINT, COPY and AND instructions that follow it in the Dockerfile.

ADD

```
ADD <src>... <dest>
```

This instruction copies the files in <src> into the <dest> directory inside the filesystem of the image. The <dest> is an absolute path where the source will be copied inside the container. The <src> is a relative path, being interpreted as relative to the place of the Dockerfile.

ENTRYPOINT

```
ENTRYPOINT ["executable", "param1", "param2"] (exec form, preferred)

ENTRYPOINT command param1 param2 (shell form)
```

This instruction is used when an executable is desired to be run when calling the container. This is, the program that the container will execute to achieve the function. It is the last instruction in the Dockerfile.

CMD

```
CMD ["executable","param1","param2"] (exec form, this is the preferred form)

CMD ["param1","param2"] (as default parameters to ENTRYPOINT)

CMD command param1 param2 (shell form)
```

This instruction can be used to set some default parameters to the ENTRYPOINT executables. In case there is no ENTRYPOINT, the instruction will include an executable.

6.1.1.2 Creating the Dockerfile

As per the Docker Images provided in [27] and [28], the Dockerfile will have the commands shown below in Fig. 6.2. It first starts from the Ubuntu base Image and creates a conda environment.

In line 11, it is created a working directory inside the image where to store the program. Specifically, it stores the contents of the HN_Segmenter folder, (previously mentioned) as per the command in line 12.

Next, it creates a directory, which is where all the segmentation results will go, as ordered in predict.py file.

Then, it runs the installation of all the packages stored inside the requierements.txt file.

Finally, it executes the file predict.py, which provides the segmentations. This is the file that is desired to be executed every time the Image is run.

The files hierarchy must be as follows in order to succeed with the creation of the file are shown in Fig. 6.1 below.

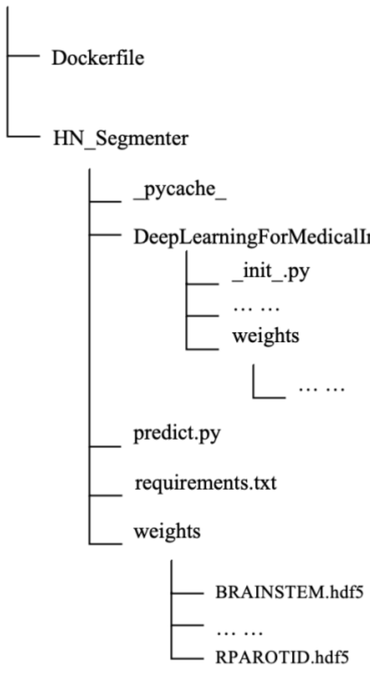
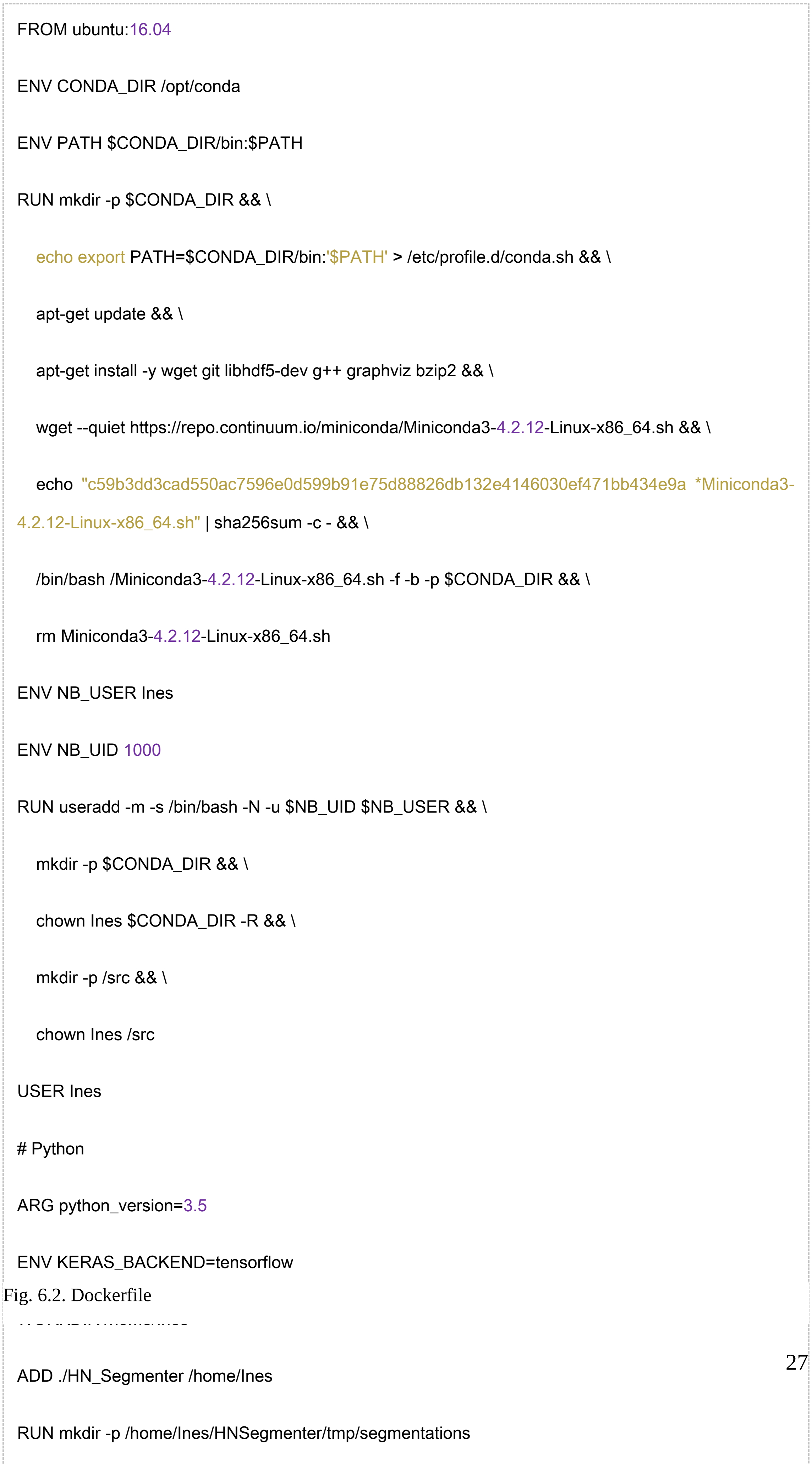


Fig. 6.1. Files hierarchy. Dockerfile must be outside the Neural Network files



```
FROM ubuntu:16.04

ENV CONDA_DIR /opt/conda

ENV PATH $CONDA_DIR/bin:$PATH

RUN mkdir -p $CONDA_DIR && \

    echo export PATH=$CONDA_DIR/bin:'$PATH' > /etc/profile.d/conda.sh && \

    apt-get update && \

    apt-get install -y wget git libhdf5-dev g++ graphviz bzip2 && \

    wget --quiet https://repo.continuum.io/miniconda/Miniconda3-4.2.12-Linux-x86_64.sh && \

    echo  "c59b3dd3cad550ac7596e0d599b91e75d88826db132e4146030ef471bb434e9a  *Miniconda3-
4.2.12-Linux-x86_64.sh" | sha256sum -c - && \

    /bin/bash /Miniconda3-4.2.12-Linux-x86_64.sh -f -b -p $CONDA_DIR && \

    rm Miniconda3-4.2.12-Linux-x86_64.sh

ENV NB_USER Ines

ENV NB_UID 1000

RUN useradd -m -s /bin/bash -N -u $NB_UID $NB_USER && \

    mkdir -p $CONDA_DIR && \

    chown Ines $CONDA_DIR -R && \

    mkdir -p /src && \

    chown Ines /src

USER Ines

# Python

ARG python_version=3.5

ENV KERAS_BACKEND=tensorflow

WORKDIR /home/Ines

ADD ./HN_Segmenter /home/Ines

RUN mkdir -p /home/Ines/HNSegmenter/tmp/segmentations
```

Fig. 6.2. Dockerfile

6.1.1.3 Creating the Docker Image

Once the Dockerfile is done, we must build the Docker Image. The line of code to be written in the Terminal is the following:

```
docker build -t [Image name] .
```

Here, the dot in the end determines that the Docker Image will be created in the current directory. Hence, to build the Image for this project, the directory in the Terminal should be the folder where the Dockerfile is (by “cd” to the desired folder).

Once inside the right folder, the exact commands are:

```
docker build –t hn-segmenter .
```

Terminal starts executing all instructions in order and after several minutes it finishes and prints a success message. It can be verified that the image was properly created by checking out the images list with the command:

```
docker images
```

6.1.1.4 Creating the Docker Container

To create the container, a running instance of the Image must be initiated. To do so, the following commands must be scripted in the Terminal:

```
docker run [Image name] [args]
```

In the case at hand, as per the last instruction in the Dockerfile, the Image will always run the commands “python predict.py”. Hence, the arguments that have to be stated in the command line are: “—path path/to/file”. The call results as follows:

```
docker run hn-segmenter --path path/to/file
```

In any case, this call will be made automatically by the Slicer module created.

The current containers can also be verified by typing:

```
docker ps –a
```

6.1.2 Creation of the Slicer module

6.1.2.1 Slicer Developer Guide

Slicer modules usually consist of a group of files, such as CMake files and source files that together constitute an application. An extension compresses one or more modules [30].

To create and develop a module, there is a tool in Slicer that simplifies the process by creating template files for setting up a module: Extension Wizard [30].

Creating extensions

In the module seeker in Slicer Toolbar, there is a module called ‘Developer Tools’, inside which it can be found the ‘Extension Wizard’. It can also be found by simply typing ‘Extension Wizard’ in the seeker.

By clicking on ‘Create an Extension’ a name and an empty directory can be specified. Then, a module can be added to that extension by clicking ‘Add Module to Extension’. By default, the module type is Scripted, which are the modules written in Python, and the ones used to build this project.

Once it’s been created, the code can be found in the selected extension, which can be modified and used immediately.

Scripted Modules

With the skeleton provided by the ‘Extension Wizard’, the main structure of the scripted modules can be appreciated: Module Class, Widget Class and Logic Class (Fig. 6.3). The module creates the widget and logic, but the logic has to be instantiated in the widget class.

The widget class is needed if the module has a user interface. It’s the responsible for creating the buttons that compose the interface and for launching processing methods implemented in the logic class.

The logic class is the responsible for the processing, and it must not rely on the widget class, it should be independent and usable from any other module.

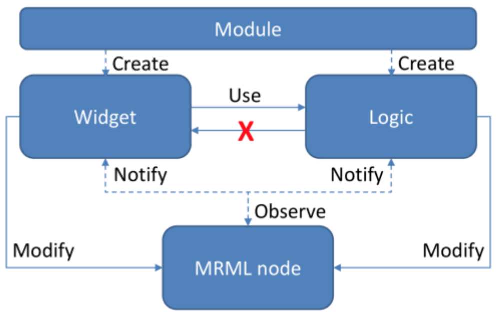


Fig. 6.3. Scripted modules implementation [31]

6.1.2.2 Module overview

This automatic-segmentation tool is designed to be integrated into the Radiotherapy Service to replace manual delineation of OARs and accelerate the treatment planning process. Hence, it is intended to be used by either doctors or technicians. For this reason, it is required to design an intuitive application that automates all the tasks. As mentioned, this application will be created in Slicer by programming a Slicer module. The requirements of this module are:

- Loading a head and neck CT image from a directory in DICOM format
- Calling the Neural Network and running it with the new image to obtain a segmented image
- Saving the image again so that it can be stored in the system
- Segmentations obtained must be compatible with the workstations in the service, so they can be modified by doctors
- Automatie these tasks as much as possible

As per the requirements above, the module will have three main tasks, which will be accomplished with three main buttons: loading button, segmentation button and saving button. The main window of Slicer after loading the module HN_Segmenter, with the module panels and options, is shown below in Fig. 6.4.

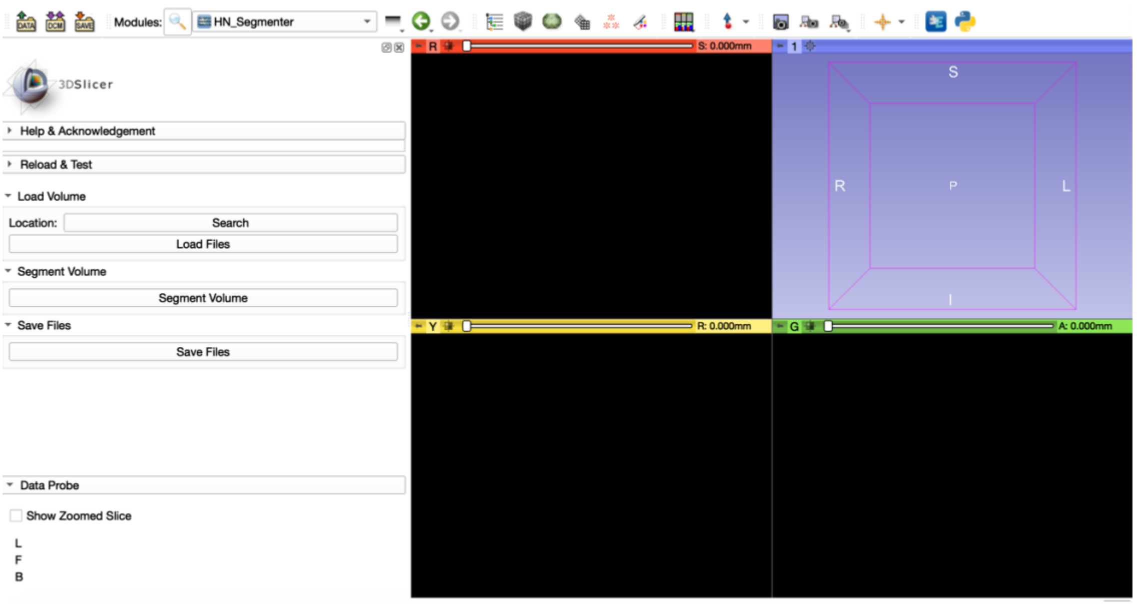


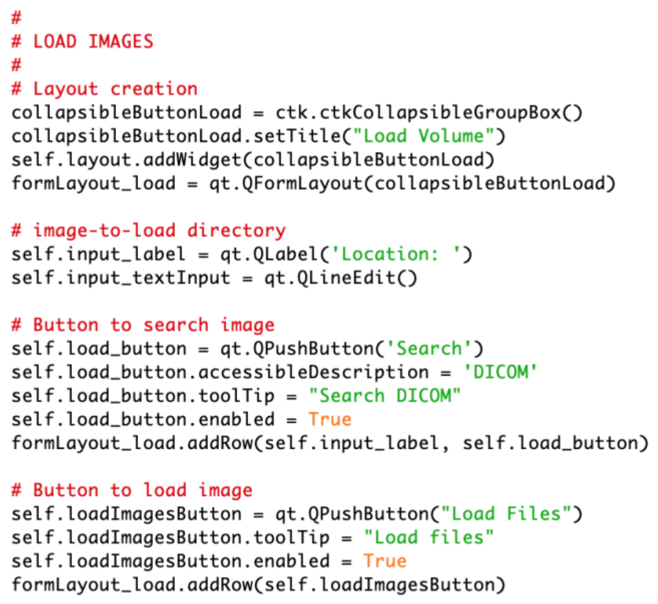
Fig. 6.4. Start window of Slicer after loading the module for segmentation. On the left-hand side, the module panels with the buttons for accessing principal functions

In the next points, we will go through the three main sections of the module, related to its three main functions: loading, segmenting and saving.

6.1.2.3 Load Volume

The Load Volume section has two buttons: one for choosing the directory of the file and another one for loading the volume to the scene.

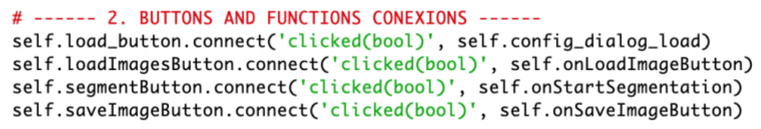
The buttons are created in the widget as in Fig. 6.5.



```
#
# LOAD IMAGES
#
# Layout creation
collapsibleButtonLoad = ctk.ctkCollapsibleGroupBox()
collapsibleButtonLoad.setTitle("Load Volume")
self.layout.addWidget(collapsibleButtonLoad)
formLayout_load = qt.QFormLayout(collapsibleButtonLoad)

# image-to-load directory
self.input_label = qt.QLabel('Location: ')
self.input_textInput = qt.QLineEdit()

# Button to search image
self.load_button = qt.QPushButton('Search')
self.load_button.accessibleDescription = 'DICOM'
self.load_button.toolTip = "Search DICOM"
self.load_button.enabled = True
formLayout_load.addRow(self.input_label, self.load_button)

# Button to load image
self.loadImagesButton = qt.QPushButton("Load Files")
self.loadImagesButton.toolTip = "Load files"
self.loadImagesButton.enabled = True
formLayout_load.addRow(self.loadImagesButton)
```

Fig. 6.5 Piece of code for creating the buttons for Load Volume section. Two buttons are created and added to the layout

Once the buttons are created, they must be linked to their functions, through the following piece of code (Fig. 6.6), which connects them when buttons are clicked:



```
# ------ 2. BUTTONS AND FUNCTIONS CONEXIONS ------
self.load_button.connect('clicked(bool)', self.config_dialog_load)
self.loadImagesButton.connect('clicked(bool)', self.onLoadImageButton)
self.segmentButton.connect('clicked(bool)', self.onStartSegmentation)
self.saveImageButton.connect('clicked(bool)', self.onSaveImageButton)
```

Fig. 6.6. Piece of code for connecting buttons to functions

The button intended to get the directory of the file is the first one, load_button. This is connected to the function config_dialog_load, which is directly connected to the open_dialog function.

This function gets the directory of the file and modifies the button to show the path. As the description of the button is 'DICOM', the dialog will look for a whole folder, and not for a single file. If the button was intended to load a nifty image, its description would be 'Volume' and the dialog would look for a single file.

The code is shown in Fig. 6.7 below. As it can be seen, the functions are connected to the logic (Fig. 6.8) through a function that stores the volume location for using it later to load it in the scene.

```
def config_dialog_load(self):
     self.open_dialog(self.load_button)

def open_dialog(self, button):

    if button.accessibleDescription in ['DICOM']:
        path = str(qt.QFileDialog().getExistingDirectory())
    elif button.accessibleDescription in ['Volume']:
        path = str(qt.QFileDialog().getOpenFileName())
    else:
        path = None

    if path:
        path = path.replace('\\', '/')
        # modify text of button to show path
        self.modify_button(button, path)
        # store file location
        self.logic.update_locations({button.accessibleDescription: path})

def modify_button(self, button, new_text):
  button.setText(new_text)
```

Fig. 6.7. Piece of code to add a function to the buttons for searching a volume

```
def update_locations(self, new_element):
  self.locations_dict.update(new_element)
```

Fig. 6.8. Piece of code of the logic function to store the location of the DICOM volume

The button intended to load the volume to the scene is linked to the DICOM browser module. It is connected to the function onLoadImageButton as already shown in Fig. 6.9. That function gets the directory of the volume available through the button's information and passes it to the logic.

```
def onLoadImageButton(self):
  try:
      path_to_file = self.load_button.text
      self.logic.loadVolumes(path_to_file)

  except KeyError, IndexError:
    pass
```

Fig. 6.9. Widget function connected to the second button that calls the logic to connect with DICOM browser

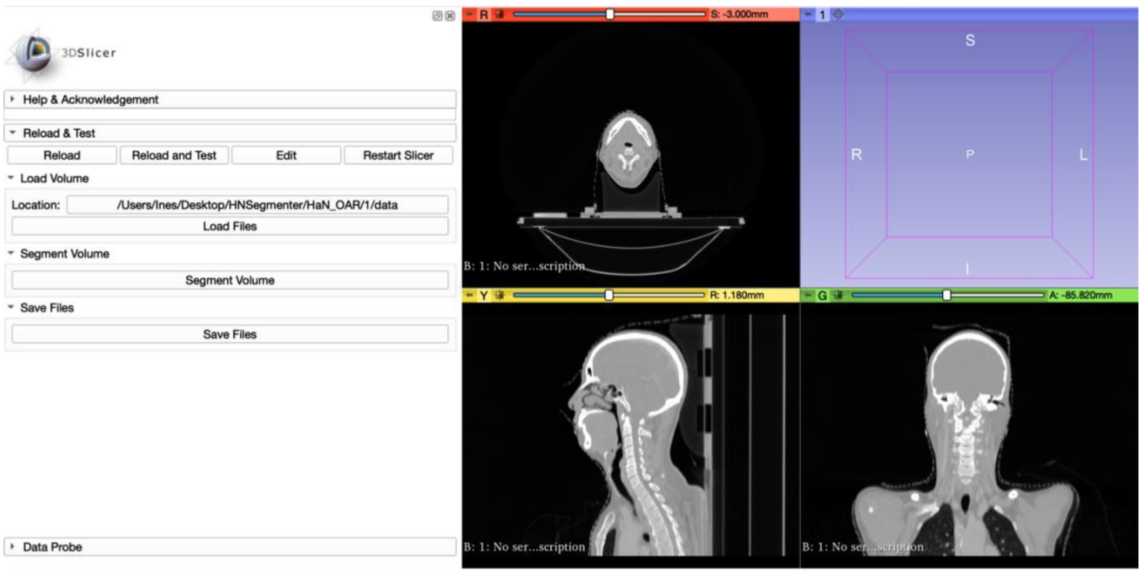
The loadVolumes function performs two tasks: clears the scene in case there is some volume previously loaded and calls the function load_DICOM. This last function interacts with the DICOM module through the DICOMUtils function. Imports the directory previously stored in locations_dict and loads the first patient in the data base, which is the last patient loaded.

Finally, it automatically opens the module again. The code is shown in below in Fig. 6.10

```
def loadVolumes(self,path_to_file):
    # clear scene at start
    self.clear_scene()
    #slicer.util.loadVolume(path_to_file)
    self.load_DICOM()

def load_DICOM(self):
  # create temporary database. Save original database path to restore
  # it when finished

  self.original_db_path = DICOMUtils.openTemporaryDatabase()
  #assert temporary database is open

  # import DICOM to database
  DICOMUtils.importDicom(self.locations_dict['DICOM'], slicer.dicomDatabase)

  # get patient name
  patient_name = slicer.dicomDatabase.nameForPatient(
    slicer.dicomDatabase.patients()[0]
    )

  # open DICOM by name
  DICOMUtils.loadPatientByName(patient_name)
  slicer.util.selectModule('dicomsegment')

def clear_scene(self):
  slicer.mrmlScene.Clear(0)
```

Fig. 6.10. Loading the DICOM volume through the DICOM browser piece of code

After loading the volume, the Slicer window results as in Fig. 6.11.



Location: /Users/Ines/Desktop/HNSegmenter/HaN_OAR/1/data

Fig. 6.11. Main window after loading the volume. The value of the button for location updates with the directory of the volume and the volume is shown in the scene. The volume loaded is the first patient of the data set.

5.1.2.2 Segment Volume

The Segment Volume section is the core of the module. There's only a button, intended for performing the delineation of the organs in the loaded image. This button has several hidden sub-functions:

- Transform the DICOM image to nifty format, so that it is readable by the Neural Network
- Create a Docker container to run the Docker Image that obtains the label maps
- Transfer the label maps from the Docker container to a local host that is deleted every time the module is reloaded
- Deletes the container created to save memory
- Import the label maps to the scene and transform them to segmentations
- Give a different color for each segment to make it easier for doctors to check segmentations
- Show segmented image in the scene.

Let us go through all the functions that compose this button.

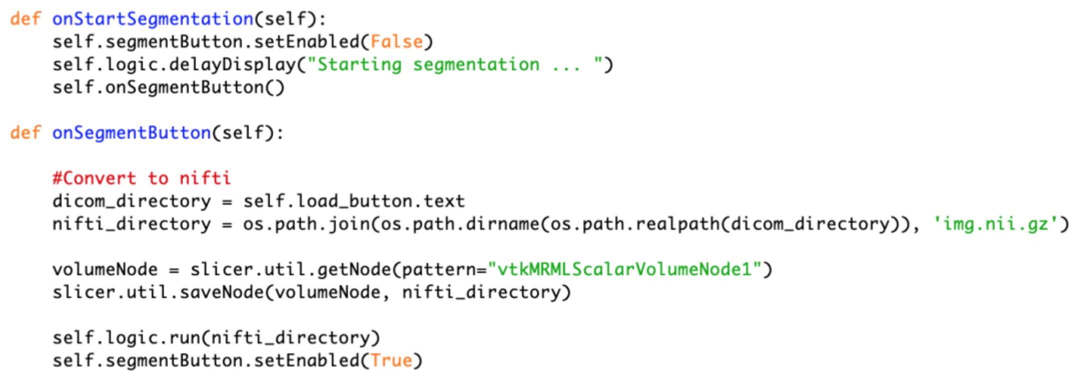
Widget

The button is created in the same way as the buttons for loading the volume, and the connection of this button to its function is shown in Fig. 6.6.

This function, onStartSegmentation, first blocks the button to avoid pressing it again and shows a message that states that segmentation started (Fig. 6.12). This is accomplished with the function delayDesplay already provided by the template that Slicer creates.

Second, it calls onSegmentButton function, which converts the loaded image into nifty format to make it runnable in the Neural Network before calling the logic. To do so, it first creates the directory (gets the one where the volume was saved and adds the new image there) and uses the saveNode function to save the scene into that directory, which results in the conversion from DICOM to nifty.

Secondly, it calls the logic to proceed with segmentation.

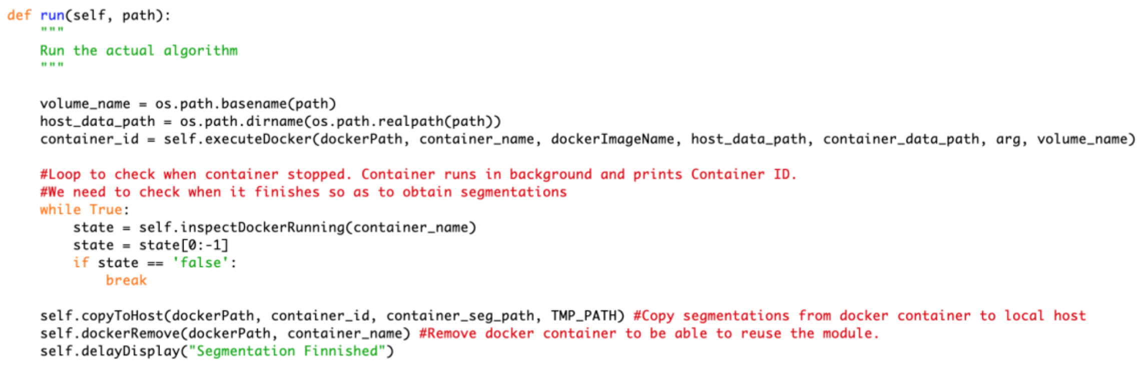


```
def onStartSegmentation(self):
    self.segmentButton.setEnabled(False)
    self.logic.delayDisplay("Starting segmentation ... ")
    self.onSegmentButton()

def onSegmentButton(self):

    #Convert to nifti
    dicom_directory = self.load_button.text
    nifti_directory = os.path.join(os.path.dirname(os.path.realpath(dicom_directory)), 'img.nii.gz')

    volumeNode = slicer.util.getNode(pattern="vtkMRMLScalarVolumeNode1")
    slicer.util.saveNode(volumeNode, nifti_directory)

    self.logic.run(nifti_directory)
    self.segmentButton.setEnabled(True)
```

Fig. 6.12. Widget functions for starting the segmentation.

Logic

The run function is the core of the Segment button, and it leads with the call of all the sub-functions: calling Docker to execute the Neural Network, importing segmentations obtained, removing the container for further segmentations and showing a message when finished (Fig. 6.13). Let us go through all those.



```
def run(self, path):
    """
    Run the actual algorithm
    """

    volume_name = os.path.basename(path)
    host_data_path = os.path.dirname(os.path.realpath(path))
    container_id = self.executeDocker(dockerPath, container_name, dockerImageName, host_data_path, container_data_path, arg, volume_name)

    #Loop to check when container stopped. Container runs in background and prints Container ID.
    #We need to check when it finishes so as to obtain segmentations
    while True:
        state = self.inspectDockerRunning(container_name)
        state = state[0:-1]
        if state == 'false':
            break

    self.copyToHost(dockerPath, container_id, container_seg_path, TMP_PATH) #Copy segmentations from docker container to local host
    self.dockerRemove(dockerPath, container_name) #Remove docker container to be able to reuse the module.
    self.delayDisplay("Segmentation Finnished")
```

Fig. 6.13. Run function piece of code, from where all the sub-functions are called

Execute Docker

The first task this button must accomplish is calling Docker to execute the Docker Image and perform the segmentation of the volume. To do so, the executeDocker function is called, which builds the call by joining the commands needed (Fig. 6.14). Its parameters are:

- dockerPath. Global parameter, automatically obtained at the beginning of the program depending on the operating system of the computer
- container_name. Global parameter, already predefined at the beginning of the program
- dockerImageName. Global parameter predefined at the beginning, the name of the image
- host_path. Calculated in the run function. It is the directory where the image is, the one in nifty format.
- container_data_path. Global parameter predefined at the beginning of the program. It is the working path inside the container, according to what was stated in the Dockerfile
- arg. Global parameter predefined at the beginning. Its value is 'path', because the parameter that the call must have is the path where the image to segment is.
- volume_name. Name of the nifty image, obtained before the call to the function (Fig. 5.13), through the information saved in the search volume button.

```
def executeDocker(self, dockerPath, container_name, dockerImageName, host_path, container_data_path, arg, volume_name):

    #Command line to run the image in a container named hn-segmenter. The container id is shown while the segmentation is executed
    cmd = list()
    cmd.append(dockerPath)
    cmd.extend(('run','--name'))
    cmd.append(container_name)
    cmd.extend(('-d','-v'))
    cmd.append(host_path + ':' + container_data_path)
    cmd.append(dockerImageName)
    cmd.append('--' + arg)
    cmd.append(os.path.join(container_data_path, volume_name))
    cmd = " ".join(cmd) #delete commas from the list
    print(cmd)

    #Obtain the container id
    container_id = subprocess.check_output(cmd, shell=True)
    container_id = container_id[0:12]
    return container_id
```

Fig. 6.14. Function to execute the Docker call and perform the segmentation

This function basically builds the call that the user should write on the command window to run the container. The call is as follows:

```
/usr/local/bin/docker run --name hn-segmenter -d -v /Users/Ines/Desktop/HNSegmenter/HaN_OAR/1:/home/Ines/data inesmunoz/hn-segmenter --path
/home/Ines/data/img.nii.gz
```

This call sets a container name, hn-segmenter, which will be useful for further actions related to the container, and connects the two directories, the one where the file actually is and the working directory in the container, specified in the Dockerfile. This way, all the information in the host directory is transferred to the container. Thanks to this, the application can find the file to segment inside the container and execute the application to obtain the segmentations.

This way of running a Docker Image returns the container ID immediately while running at the back, so a loop that checks if the container is still running is needed. This is done in Fig. 6.15. by calling the function inspectDockerRunning:

```
def inspectDockerRunning(self, container_name):
    docker_running = list()
    docker_running.append(dockerPath)
    docker_running.extend(('inspect', '-f', "'{{.State.Running}}'"))
    docker_running.append(container_name)
    docker_running = " ".join(docker_running)


    state = subprocess.check_output(docker_running, shell=True)
    return state
```

Fig. 6.15. Function that checks if the container is running. Returns 'true' if it's running and 'false' if opposite

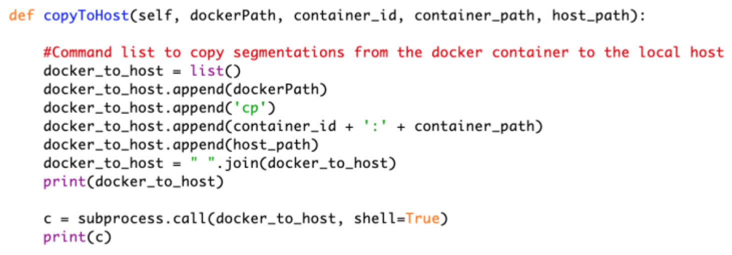
This function makes a call to the command window that inspects, by providing the container ID, if the container is running or not. The loop created inside the run function stops when the value returned is 'false'.

```
/usr/local/bin/docker inspect -f '{{.State.Running}}' hn-segmenter
```

Finally, the executeDocker function returns the container ID, on its short form, which is used in further steps.

Copy to Host

The next task that the run function performs is copying the segmentations to the host so that they can are available to be imported into Slicer. This is achieved with a call to Docker (Fig. 6.16).



```
def copyToHost(self, dockerPath, container_id, container_path, host_path):

    #Command list to copy segmentations from the docker container to the local host
    docker_to_host = list()
    docker_to_host.append(dockerPath)
    docker_to_host.append('cp')
    docker_to_host.append(container_id + ':' + container_path)
    docker_to_host.append(host_path)
    docker_to_host = " ".join(docker_to_host)
    print(docker_to_host)

    c = subprocess.call(docker_to_host, shell=True)
    print(c)
```

Fig. 6.16. Piece of code to transfer the obtained segmentations to a local directory

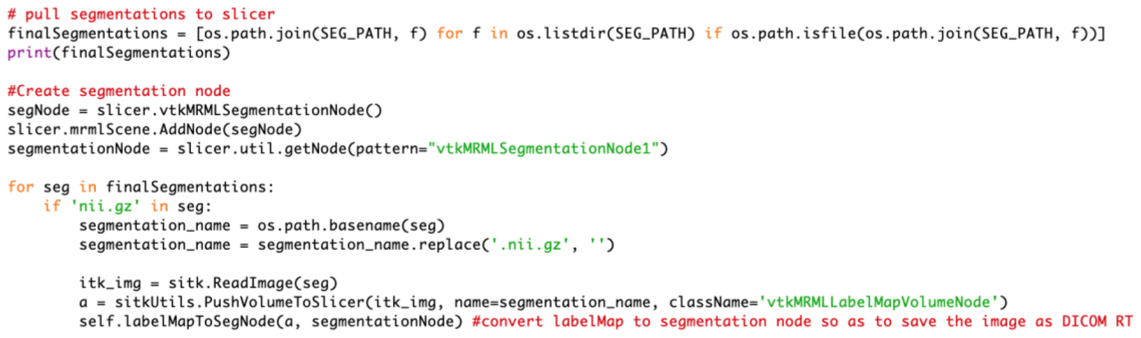
The call results as follows, which basically is read as "copy the file inside this directory of this container to this other directory in the host".

```
/usr/local/bin/docker cp fce11ab4b8c9:/home/Ines/HNSegmenter/tmp/segmentations /Users/Ines/HNSegmenter/tmp
```

The directory where the segmentations are stored is removed and created again every time the module is reloaded to ensure that no information remains from one patient to another.

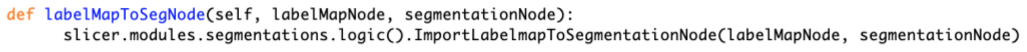
The next step consists of importing the segmentations to Slicer. Note that the Neural Network obtains the segmentations as labelmaps, so they'll have to be imported and transformed into segmentations.

The code (Fig. 6.17) looks for the labelmaps in the directory where they've been stored and imports them to slicer through the sitkUtils module. After that, it converts all the labelmaps to segmentations and copy them under the previously created segmentation node. This conversion is achieved with the labelMapToSegNode function (Fig. 6.18).



```
# pull segmentations to slicer
finalSegmentations = [os.path.join(SEG_PATH, f) for f in os.listdir(SEG_PATH) if os.path.isfile(os.path.join(SEG_PATH, f))]
print(finalSegmentations)

#Create segmentation node
segNode = slicer.vtkMRMLSegmentationNode()
slicer.mrmlScene.AddNode(segNode)
segmentationNode = slicer.util.getNode(pattern="vtkMRMLSegmentationNode1")

for seg in finalSegmentations:
    if 'nii.gz' in seg:
        segmentation_name = os.path.basename(seg)
        segmentation_name = segmentation_name.replace('.nii.gz', '')

        itk_img = sitk.ReadImage(seg)
        a = sitkUtils.PushVolumeToSlicer(itk_img, name=segmentation_name, className='vtkMRMLLabelMapVolumeNode')
        self.labelMapToSegNode(a, segmentationNode) #convert labelMap to segmentation node so as to save the image as DICOM RT
```

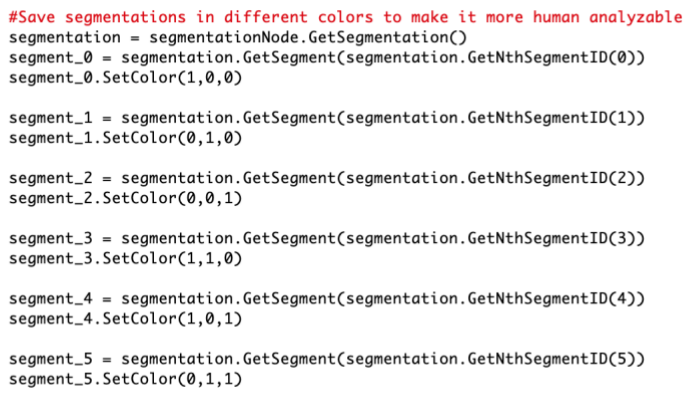
Fig. 6.17. Importing segmentations to Slicer piece of code



```
def labelMapToSegNode(self, labelMapNode, segmentationNode):
      slicer.modules.segmentations.logic().ImportLabelmapToSegmentationNode(labelMapNode, segmentationNode)
```

Fig. 6.18. Function to convert a labelmap into a segmentation

Finally, the following piece of code is executed to give every segment a different color (Fig. 6.19).

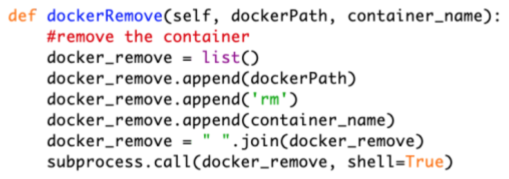


```
#Save segmentations in different colors to make it more human analyzable
segmentation = segmentationNode.GetSegmentation()
segment_0 = segmentation.GetSegment(segmentation.GetNthSegmentID(0))
segment_0.SetColor(1,0,0)

segment_1 = segmentation.GetSegment(segmentation.GetNthSegmentID(1))
segment_1.SetColor(0,1,0)

segment_2 = segmentation.GetSegment(segmentation.GetNthSegmentID(2))
segment_2.SetColor(0,0,1)

segment_3 = segmentation.GetSegment(segmentation.GetNthSegmentID(3))
segment_3.SetColor(1,1,0)

segment_4 = segmentation.GetSegment(segmentation.GetNthSegmentID(4))
segment_4.SetColor(1,0,1)

segment_5 = segmentation.GetSegment(segmentation.GetNthSegmentID(5))
segment_5.SetColor(0,1,1)
```

Fig. 6.19. Piece of code to set a different color to every segment

Docker remove

Finally, the button executes a short piece of code to remove the container (Fig. 6.20), so that it can be run again with the same name in further segmentations:



```
def dockerRemove(self, dockerPath, container_name):
    #remove the container
    docker_remove = list()
    docker_remove.append(dockerPath)
    docker_remove.append('rm')
    docker_remove.append(container_name)
    docker_remove = " ".join(docker_remove)
    subprocess.call(docker_remove, shell=True)
```

Fig. 6.20. Function to remove the container once the actions have been finished

The call results as follows and the Slicer window after segmentation is achieved is shown below in Fig. 6.21.

```
/usr/local/bin/docker rm hn-segmenter
```

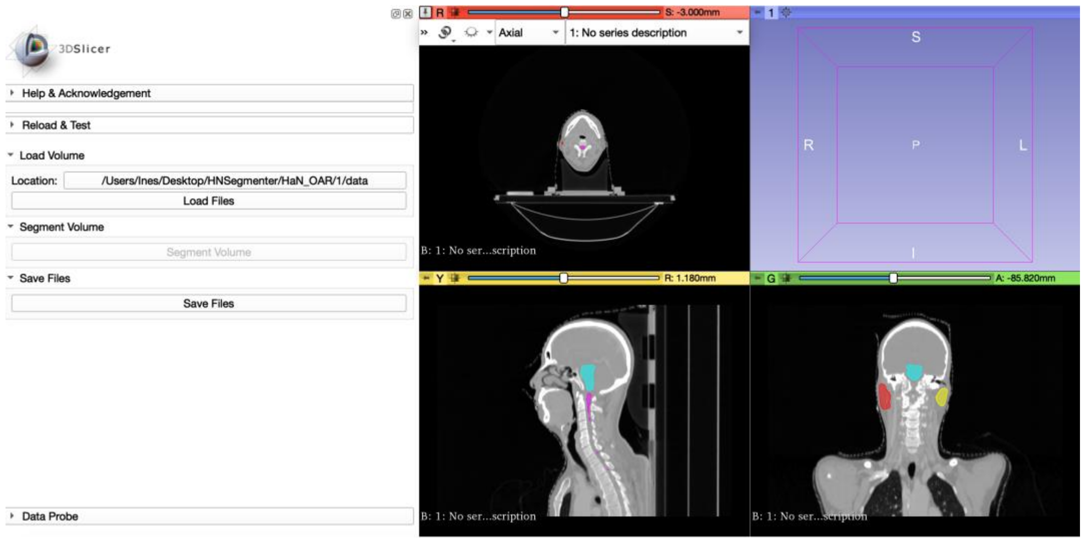


Fig. 6.21. Slicer window after segmentation is performed. The segment button is blocked and can only be able when reloading the module

6.1.2.2 Save volume

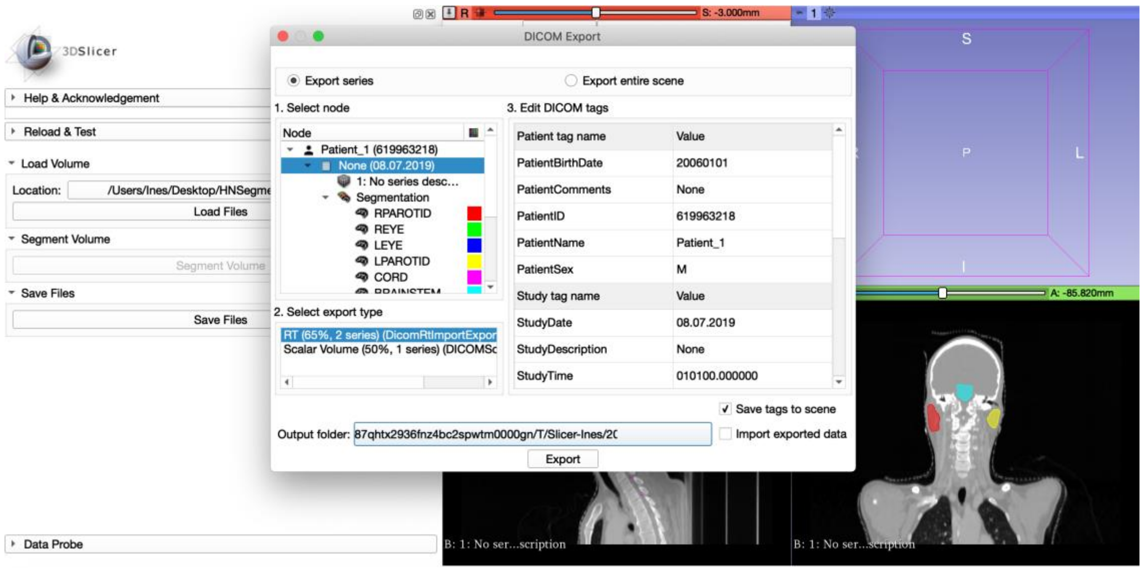
This last section is intended to save de segmented volume in DICOM RT format, so that it can be opened and read by the workstations in the service.

The button is created as the rest of the buttons, and it is connected to onSaveImageButton as Fig. 6.6 shows. This function first creates a folder where to store the final volume inside the folder where the original data was stored, though it is optional to choose it. Immediately after, it calls the saveImage function with volumeNode and segmentationNode as parameters (Fig. 6.22).

```
def onSaveImageButton(self):
    #Create a folder to store the segmented volume
    dicom_directory = self.load_button.text
    segment_directory = os.path.join(os.path.dirname(os.path.realpath(dicom_directory)), 'DCMData')
    if os.path.isdir(segment_directory):
        shutil.rmtree(segment_directory)
    os.mkdir(segment_directory)

    volumeNode = slicer.util.getNode(pattern="vtkMRMLScalarVolumeNode1")
    segmentationNode = slicer.util.getNode(pattern="vtkMRMLSegmentationNode1")

    try:
        self.logic.saveImage(volumeNode, segmentationNode)
    except KeyError, IndexError:
        pass
```

Fig. 6.22. Function to store the segmented volume. It calls the logic through the saveImage function

This saveImage function communicates with DICOM Export module and opens the dialog where the nodes to export can be selected, and the metadata can be modified (Fig. 6.23). To do so, the function gets the Study and Segment ID, through the Subject Hierarchy Node and pass them to the DICOM Export Dialog.



DICOM Export

(●) Export series ( ) Export entire scene

1. Select node

Node	
Patient_1 (619963218)	
None (08.07.2019)	
1: No series desc...	
Segmentation	
RPAROTID	🟥
REYE	🟩
LEYE	🟦
LPAROTID	🟨
CORD	🟪

2. Select export type

RT (65%, 2 series) (DicomRtImportExpor
Scalar Volume (50%, 1 series) (DICOMSc

3. Edit DICOM tags

Patient tag name	Value
PatientBirthDate	20060101
PatientComments	None
PatientID	619963218
PatientName	Patient_1
PatientSex	M
Study tag name	**Value**
StudyDate	08.07.2019
StudyDescription	None
StudyTime	010100.000000

[✓] Save tags to scene

Output folder: 87qhtx2936fnz4bc2spwtm0000gn/T/Slicer-Ines/2C

[ ] Import exported data

Export

Fig. 6.23. DICOM Export dialog to save the volume as DICOM RT format. By default, the study is selected as the node to export and RT is selected as the export type. Hence, the user must select an output folder and modify the tags if desired

When this dialog appears, the folder to store the data is already created, so it can be chosen by the user. This is because an empty folder must be chosen to store the volume.

6.1.3 Integrating the application into the Radiotherapy Service

Once the application is created, the last step is to integrate it inside the workflow of the service for which it was designed.

The tool implemented in this thesis is intended to basically receive incoming images from the CT, segment them and send them to the server so they are stored and can be processed by radiophysicists. Let us first understand the current workflow that the images follow in the service (Fig. 6.24), and afterwards, discuss the way of integrating the application.

The Radiotherapy Service in the HGUGM is managed by Elekta's Information System MOSAIQ, a complete patient management information system that provides connectivity between any linear accelerator and treatment planning system from any vendor, to freely choose the best treatment solution [32].

When a new CT image is obtained, it is sent from the CT to the DICOM server. The new patient can be imported from any of the workstations in the planning room, by using the "Import Patient" tool inside the Monaco software, which accesses the server information. Once imported, the new patient appears inside the local folder which contains all the patients and it is ready to be segmented. Every patient inside that folder can be opened from any of the workstations and any change is automatically saved in the server. Hence, when a new segmentation is performed, it is automatically saved and it can be imported from radiophysics department to proceed with the dose delivery planning.

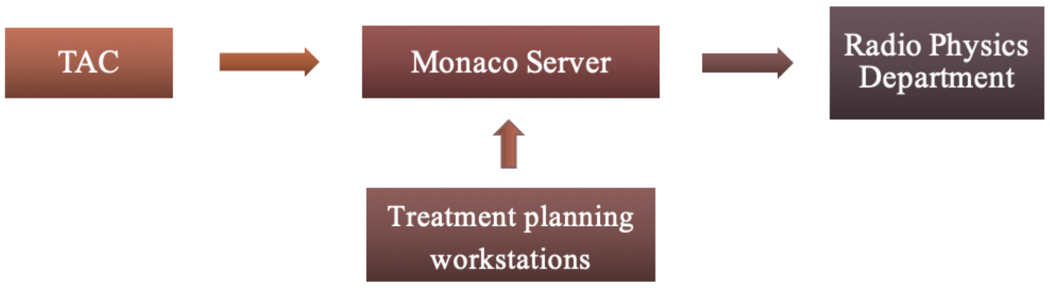


Fig. 6.24. Current workflow in the Radiation Therapy Service of the HGUGM

Therefore, the ideal way to proceed with the integration of the tool is to create a DICOM server in the computer that hosts Slicer and the application designed, and get the CT to send images to that new server.

DICOM Listener, DICOM Query/Retrieve interface, and a DICOM Send options are supported by Slicer. In order to exchange information between Slicer and CT, IP address, port and AE Title must be set up correctly on both sides.

This way, the CT would have a node to send the images to that new server, so that Slicer can access new patients and import them. Once the segmentation is performed, slicer would send the segmented volume to Monaco Server so that it can be accessed from radio radiophysics department. Also, once in the server, it could be imported from any of the workstations through Monaco software to review the segmentations and perform the delineation of the tumor, as the procedure dictates.

The new workflow proposed is shown below in Fig. 6.25.

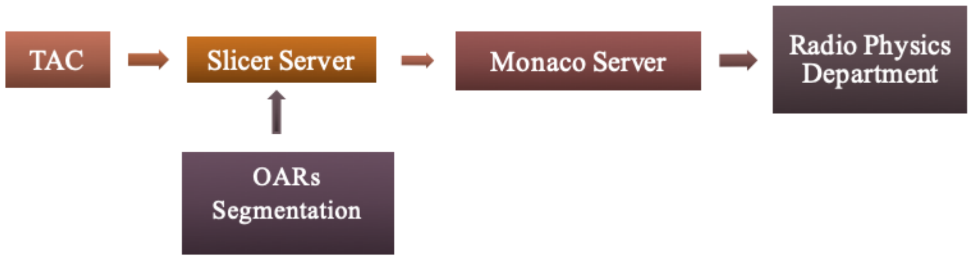


Fig. 6.25. Alternative workflow to integrate the designed application

There is a simpler way to proceed with the integration of the module into the clinical workflow, as the folder with all the patients can be accessed through the browser. The thing is that it has a numerical route so as not to be intuitive and accessible from the computers by anyone. In any case, the folder could be accessed from the Slicer module and the new images could be imported. This option is feasible when testing the module, but not to implement it in the real workflow, since access to that folder would allow modifications to the hospital database, which is not allowed.

According to the installation of the application in the workstations, the computers should install 3D Slicer and Docker Desktop, pull the Image that stores the deep learning model and install the Slicer module inside 3D Slicer. Once finished, the setup would be ready to start the automatic segmentation process.

To sum up, the main steps to integrate the application into the service are the following:

1. Install Slicer and Docker Desktop in one of the workstations
2. Pull the docker Image available in a public repository
3. Install the Slicer module inside the application
4. Create a DICOM server in the computer
5. Start sending the images from the CT to the new server
6. Import patients from Slicer
7. Execute the module and obtain segmentations
8. Send the segmented volumes to the Monaco Server to close the workflow

6.2 Evaluation methods

6.2.1 Module evaluation

The dataset used to test and evaluate the performance of the module is composed by a set of CT scans obtained from the Structure Segmentation for Radiotherapy Planning Challenge 2019, a competition of MICCAI 2019 Challenge [33]. This dataset consists of 50 CT scans of nasopharynx cancer patients, segmented by experts and released to the public as training data.

To access the download of the images, the user must register and submit an agreement so as not to redistribute the database. Once downloaded, the dataset is organized in 50 folders, each one containing both the images and the labels in nifty format.

To proceed with the evaluation, the images will be first transformed into DICOM format through Slicer's Save module to:

1. Prove metadata conservation after segmentation
2. Test module behavior and performance
3. Find strengths and weaknesses

Apart from that, the whole application will be tested in other computers to:

1. Check installation times, portability, ease of installation
2. Test run time
3. Check segmentations obtained for the same patients

6.2.2 Integration approach evaluation

Student will make few visits to the service to evaluate the compatibility between Slicer and the workstations by:

1. Checking that segmentations obtained with Slicer are readable and editable in the Monaco Software
2. Checking behavior when an existing patient is imported
3. Learning if new patients can be imported from outside the hospital database and how

The whole workflow setup will be tested in the service to ensure that communication through DICOM nodes is successfully achieved.

6.3 Methods for sharing results

6.3.1 Publishing the Docker Image

Docker uses Docker Hub to store public and private repositories. Once an image is uploaded in the Docker Hub, there's access to it from the Docker Cloud. In this way, if a user uploads an image to the Docker Cloud, any Docker user can pull it to load it and execute the program inside it, without the need of installing all the packages and requirements.

The instructions to do so are:

```
docker push $DOCKER_ID_USER/image_name
```

With the aim of making the application easy to deploy, the Docker Image will be published in a public repository in Docker Hub [34], named inesmunoz. To do so the following commands were written in the command line:

```
docker push inesmunoz/hn-segmenter
```

Once the Docker Image is available in the public repository, any Docker user can pull it by typing the following instructions:

```
docker pull inesmunoz/hn-segmenter
```

6.3.2 Publishing the Slicer module

Slicer extensions can be published by making them available in a public repository, such as GitHub [35]. Once ready, they can be added to the extensions manager by making a pull request in the public extension index. If approved, they appear as extensions in the extensions manager inside the nightly version.

Being the application still in test stages, the code will be published in a private repository and access to it will be share with the lab members for further investigations in these lines.

This way, the application will be available for any changes or improvements that may be needed, before sharing it with the whole community of developers. A home web page will be created to show how set up the application and execute the module.

There is this module, "Developer Tools for Extensions", which can be installed from the Extensions Manager. This module is intended for loading extensions and modules from files. This first option allows the user to test their extensions before publishing them, distribute them on their own websites by distributing the archive, and also, load a module saved in the computer by selecting the python file. This tool will be used to distribute the module designed.

Moreover, Slicer can be launched directly from a USB drive, and all the extensions installed are kept inside it and remain in the new location [36]. This property will be used to make the installation easier: by drag and dropping the application into the desired directory, the installation of Slicer can be achieved, automatically installing all the extensions desired.

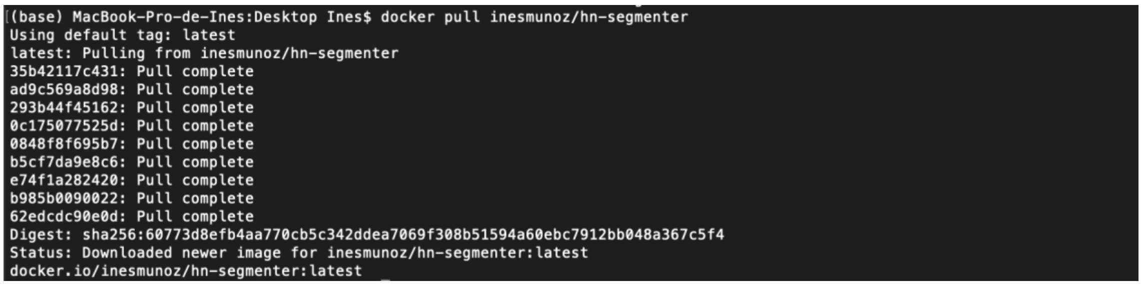
7. RESULTS AND DISCUSSION

7.1 Containerization of the Neural Network

Docker Hub allows developers to publish their Images so that anyone can pull them. This procedure was followed in the thesis development, and a public repository was created to store the image.

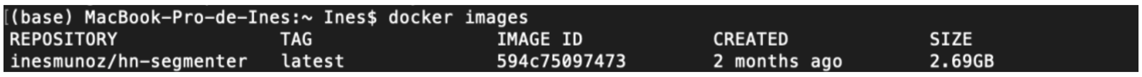
The Docker Image is pulled from the repository by typing the instruction below, and it is downloaded within few minutes using 2.69 GB. Results shown in Fig. 7.1 and Fig. 7.2.

```
docker pull inesmunoz/hn-segmenter
```



```
(base) MacBook-Pro-de-Ines:Desktop Ines$ docker pull inesmunoz/hn-segmenter
Using default tag: latest
latest: Pulling from inesmunoz/hn-segmenter
35b42117c431: Pull complete
ad9c569a8d98: Pull complete
293b44f45162: Pull complete
0c175077525d: Pull complete
0848f8f695b7: Pull complete
b5cf7da9e8c6: Pull complete
e74f1a282420: Pull complete
b985b0090022: Pull complete
62edcdc90e0d: Pull complete
Digest: sha256:60773d8efb4aa770cb5c342ddea7069f308b51594a60ebc7912bb048a367c5f4
Status: Downloaded newer image for inesmunoz/hn-segmenter:latest
docker.io/inesmunoz/hn-segmenter:latest
```

Fig. 7.1. Terminal screenshot after downloading the Docker Image from Docker Hub

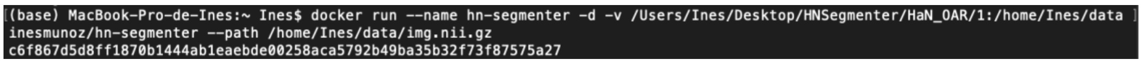


```
(base) MacBook-Pro-de-Ines:~ Ines$ docker images
REPOSITORY               TAG                 IMAGE ID            CREATED             SIZE
inesmunoz/hn-segmenter   latest              594c75097473        2 months ago        2.69GB
```

Fig. 7.2. Terminal screenshot with instruction to check docker images downloaded

Once downloaded, a running instance of this Image can be made to perform segmentations to an image, for which a container will be created. As stated in the Dockerfile, this call will execute the 'predict.py' file and obtain segmentations for the image specified in the call after the argument '--path /path/to/file'.

The command is shown below in Fig. 7.3.



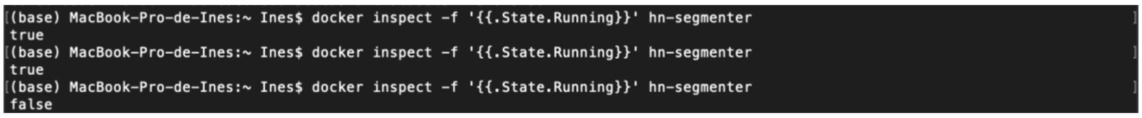
```
(base) MacBook-Pro-de-Ines:~ Ines$ docker run --name hn-segmenter -d -v /Users/Ines/Desktop/HNSegmenter/HaN_OAR/1:/home/Ines/data
inesmunoz/hn-segmenter --path /home/Ines/data/img.nii.gz
c6f867d5d8ff1870b1444ab1eaebde00258aca5792b49ba35b32f73f87575a27
```

Fig. 7.3. Terminal screenshot with instruction to run the image

Note that this call connects the folder where the to-be-segmented image is stored with the working directory of the container, already specified in the Dockerfile, so it is the same in any computer. This connection can be seen as copying the information of the local folder inside the working directory. Hence, to specify which image to segment, the path is no longer the local path, but the container path, which will then contain the image desired. Therefore, this last argument of the call should always be the same, with the concrete name of the image in each case.

Apart from that, the call displays the container ID and exits the loop to be able to continue working while executing the segmentations.

Creating the container with a specified container name allows to carry out several functions addressing to that specific container. For example, to check whether the container is running or not, before importing the segmentations (Fig. 7.4).



```
(base) MacBook-Pro-de-Ines:~ Ines$ docker inspect -f '{{.State.Running}}' hn-segmenter
true
(base) MacBook-Pro-de-Ines:~ Ines$ docker inspect -f '{{.State.Running}}' hn-segmenter
true
(base) MacBook-Pro-de-Ines:~ Ines$ docker inspect -f '{{.State.Running}}' hn-segmenter
false
```

Fig. 7.4. Terminal screenshot with instruction to check when the container stops running

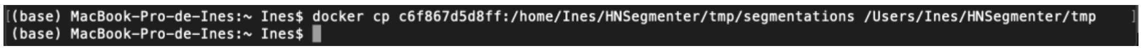
Here the student made the calls separated in time, but in the module, this instruction is executed with a loop that runs while the answer is true.

Once the container stops running, a similar call to the first one can be executed, but in the other way around. That is, to connect to the container path where segmentations are stored and transfer them to a desired local path.

Again, the path where the segmentations are stored is predefined by the Dockerfile, so it cannot be modified. Hence, the path is always the same, and the call remains as shown in Fig. 7.5.

Note that the container path is specified by giving the container ID on its short format, only the first 12 digits.

The call is the following and the segmentation results can be observed in Fig. 7.6.



```
(base) MacBook-Pro-de-Ines:~ Ines$ docker cp c6f867d5d8ff:/home/Ines/HNSegmenter/tmp/segmentations /Users/Ines/HNSegmenter/tmp
(base) MacBook-Pro-de-Ines:~ Ines$
```

Fig. 7.5. Terminal screenshot with the instruction to transfer segmentations to a desired folder

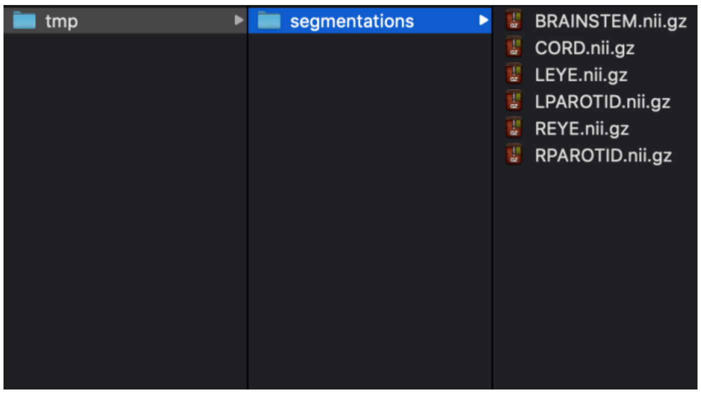


Fig. 7.6. Screenshot to the local directory where the segmentations are stored.

All these calls and interactions with the Docker Image are executed from the Slicer Module to achieve segmentations to a given image. What this module additionally does is to delete both the container and the directory where the results are stored. This way, the same code, container name and path can be reused for all the incoming images.

7.2 Creation of the Slicer Module

The Slicer module was successfully designed to automate the task of communicating with the Docker Image to execute the neural network by providing an intuitive user interface to make the process easy to follow by anyone.

The user interface is composed by only 3 sections, according to the three main functions of the module: loading an image, obtaining segmentations and saving the new volume. This is possible thanks to the segmentation button, which performs few hidden functions. This button managed to automate the conversion of DICOM to nifty format, the importation of labelmaps into segmentations and all Terminal calls to execute Docker.

Besides, with the aim of giving feedback to the user and avoiding the program to collapse, the module shows a message while performing segmentations and blocks the button. This is shown in Fig. 7.7.

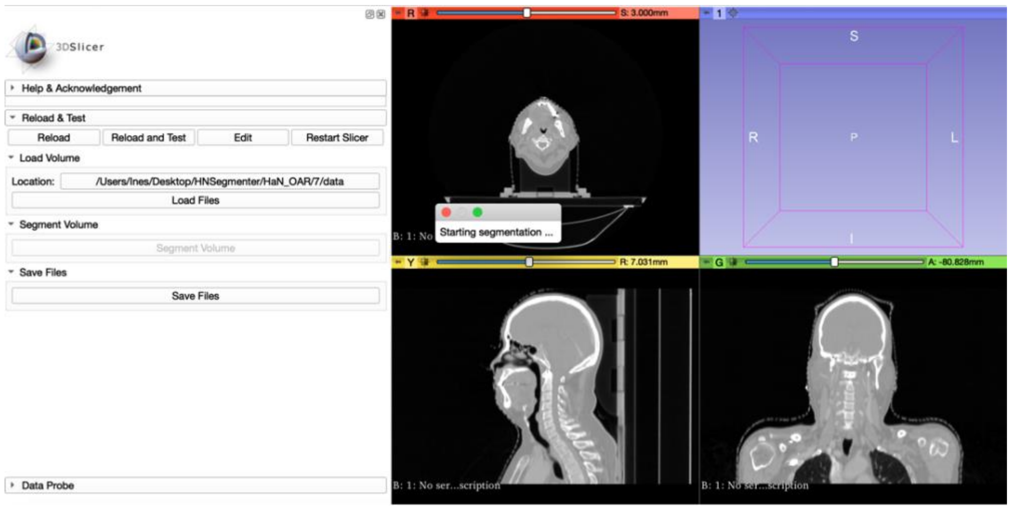


Fig. 7.7. Screenshot of Slicer window while executing the module. Segmentation button is blocked and a message is shown to indicate that the process is ongoing.

When it finishes, the button recovers its original status and the segmentations appear on scene (Fig. 7.8). It is of standing of that metadata of the DICOM image is preserved, as the only action performed is importing the segmentations obtained in the Docker Image into Slicer.

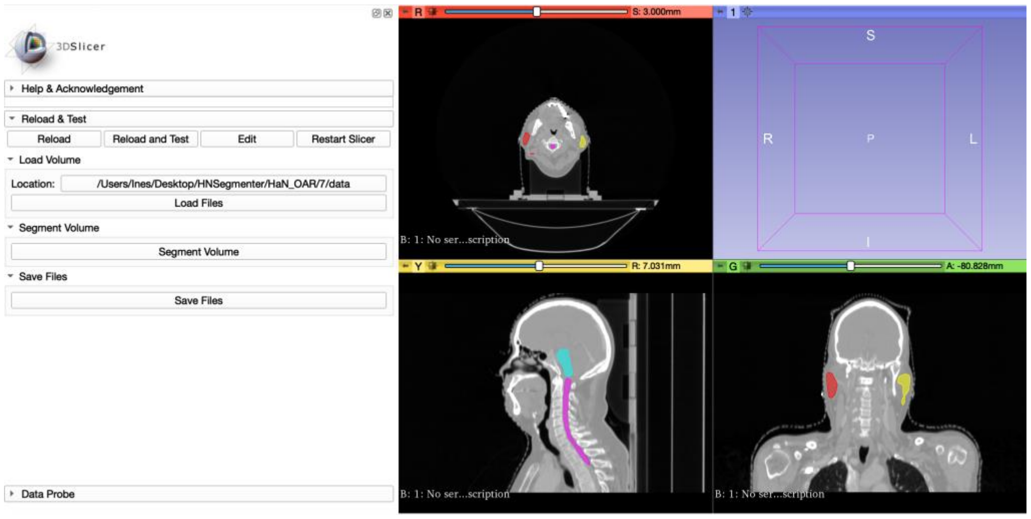


Fig. 7.8. Screenshot of Slicer once the segmentation has been performed.

Referring to the evaluation points in 6.2.1 Module evaluation, the dataset used to evaluate the performance of the module was obtained from the Structure Segmentation for Radiotherapy Planning Challenge 2019 [33]. The images in this dataset are in nifty format, as they must be anonymized so as not to share patient's information.

Therefore, prior to evaluation, the images were transformed into DICOM format to stick to reality and check if the metadata is maintained after the segmentation. The module was tested in 10 different patients and segmentations were compared to the ones offered by the challenge, which were manually segmented by experts. Results obtained are summarized in the list below:

- Metadata is maintained after segmentations are obtained, which is reasonable as the initial loaded image is not changed, segmentations resulting from the Docker Image are simply imported into the initial volume
- The module failed only with one patient, as the image was cut and the entire head could not be seen (see Fig. 7.9 below)
- Segmentations were similar to those of the challenge in most cases, proving that neural network is robust
- In some cases, segmentations were defective (see Fig. 7.10), but failures are easy detect and to correct by an expert
- Generally, segmentations obtained with this module have a smoother shape that the ones segmented by experts in the European challenge (see Fig. 7.10)
- The drawback could be said to be that the module cannot be interrupted while running, as there are no buttons to do so

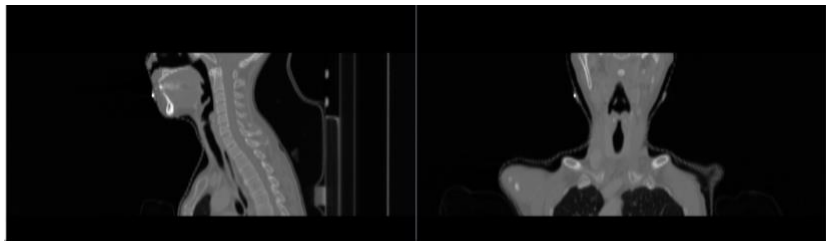


Fig. 7.9. Patient #8, for which the module failed to obtain segmentations, as image was cut

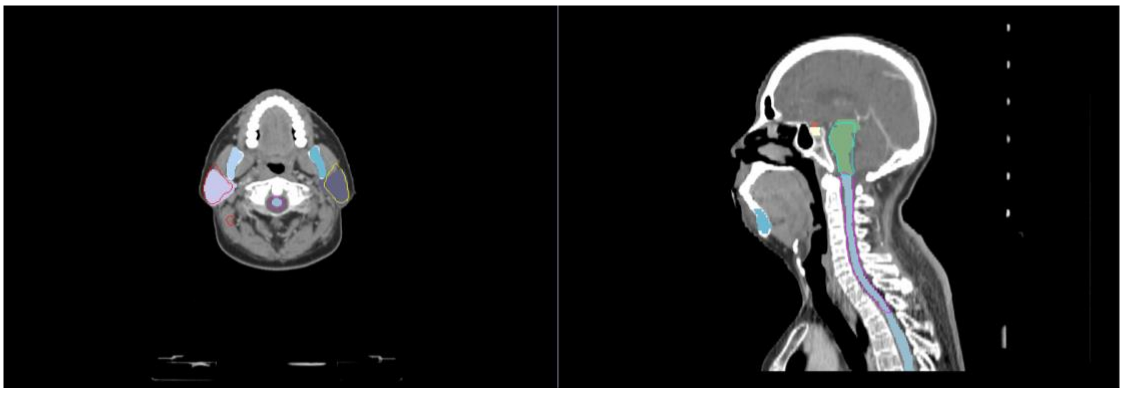


Fig. 7.10. Segmentation results. Segmentations obtained with the module are outlined only, over the other segmentations, which are color-filled. It can be observed how our segmentations are smoother in shape. Also, in the left-hand image, a fake part of the right parotid is wrongly segmented (red).

The whole setup docker + slicer was installed in another computer (MacBook Pro 13-inch, 2017) to check the portability of the whole application and test how easy it is to install it. Slicer was copied from a USB, with all the extensions needed for the performance inside it. Execution time was measured in both computers to check if it is environment dependent.

The results are listed below:

- Docker took 3’20’’ to pull the Docker Image from the public repository
- It took 58 seconds to copy Slicer from the USB to the applications folder
- Segmentation module was already inside Slicer, so no time to install it was required
- It takes less that 3 minutes to obtain segmentations, depending on the computer (see the comparison in Table 7.1)
- Segmentations obtained for the same patients are always the same

Table 7.1. Execution time comparison on two computers

Patient #	Execution Time MacBook Pro (13-inch, 2015)	Execution Time MacBook Pro (13-inch, 2017)
Patient 1	1’57”	2’53”
Patient 2	1’52”	2’43”
Patient 3	1’55”	2’44”
Patient 4	1’53”	2’43”
Patient 5	1’53”	2’41”

With all these results, it can be concluded that the whole installation is fast and easy to perform, and segmentations are obtained in few minutes. Although the application was no tested in an operating system different from macos, it is designed to run in any environment.

7.3 Integration into the service

During the compatibility tests between the results of the Slicer module and the Monaco software, it was verified that the labelmaps were readable by the software, but could not be modified, which is a fundamental requirement considering that a doctor will have to review them and made the changes considered. Instead, the segmentations allow to save the volume as DICOM RT, which is readable by the software and can also be modified. To be able to save the volume as DICOM RT, extension Slicer RT must be installed.

In Fig. 7.11, it is shown how the segmented volume is opened from Monaco software. Segmentations can me modified, changing the colors, names and shapes, and new segmentations can be created.

The relevant information here is that once the volume is imported and opened in Monaco, it is available in the server and it can be accessed from any of the workstations, including the radiophysics department, from where dose planning is performed.

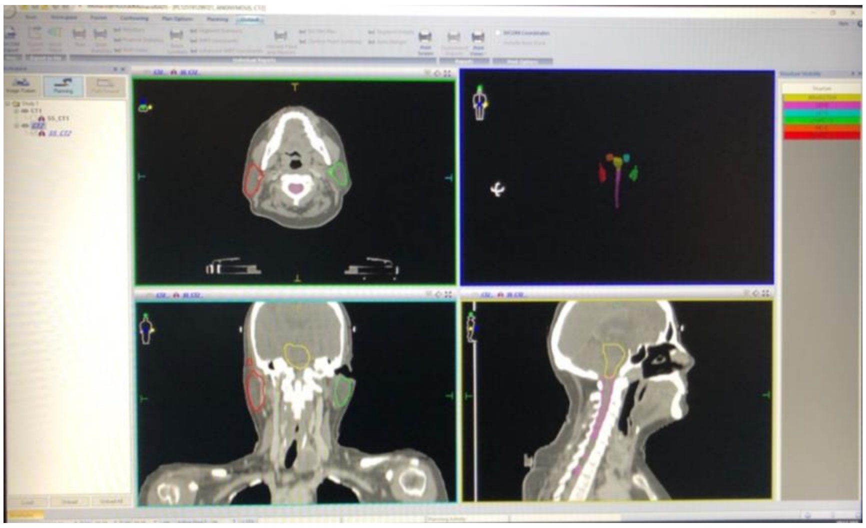
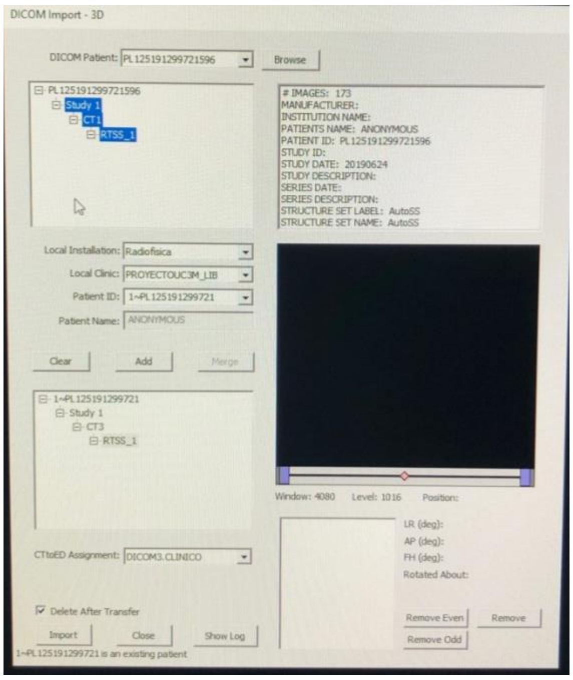


Fig. 7.11. Picture of Monaco main view after loading a new patient. In the left-hand side, the hierarchy of the images is shown

In the clinical workflow, new patients are imported directly from a list where all new patients appear, which is the DICOM node where the CT sends the images. Patients can also be imported from a local folder with the Browse button in Fig. 7.12.This is the option that was followed to test the compatibility of results.



DICOM Import - 3D

DICOM Patient: PL125191299721596 Browse

```
⊟ PL125191299721596
  ⊟ Study 1
    ⊟ CT1
      ⊟ RTSS_1
```

```
# IMAGES: 173
MANUFACTURER:
INSTITUTION NAME:
PATIENTS NAME: ANONYMOUS
PATIENT ID: PL125191299721596
STUDY ID:
STUDY DATE: 20190624
STUDY DESCRIPTION:
SERIES DATE:
SERIES DESCRIPTION:
STRUCTURE SET LABEL: AutoSS
STRUCTURE SET NAME: AutoSS
```

Local Installation: Radiofisica
Local Clinic: PROYECTOUC3M_LIB
Patient ID: 1~PL125191299721
Patient Name: ANONYMOUS

```
⊟ 1~PL125191299721
  ⊟ Study 1
    ⊟ CT3
      ⊟ RTSS_1
```

Window: 4080 Level: 1016 Position:

LR (deg):
AP (deg):
FH (deg):
Rotated About:

CTtoED Assignment: DICOM3.CLINICO

☑ Delete After Transfer

1~PL125191299721 is an existing patient

Fig. 7.12. Menu to import a new patient from Monaco server

The picture above shows the window that opens when importing a new patient. The browse button allows to look for a volume inside the computer, and once it is loaded it appears in the list with all new patients with the patient ID. It can be chosen what to load (images, series or study), and the folder where to store the patient (inside the server).

In this concrete case, the same patient had been already imported to check the behavior with already existing patients. And what happens in these cases is that the new volume is stored under the same study, but in a different series, which is a good implementation considering that the entire patient's history should be kept (Fig. 7.13).

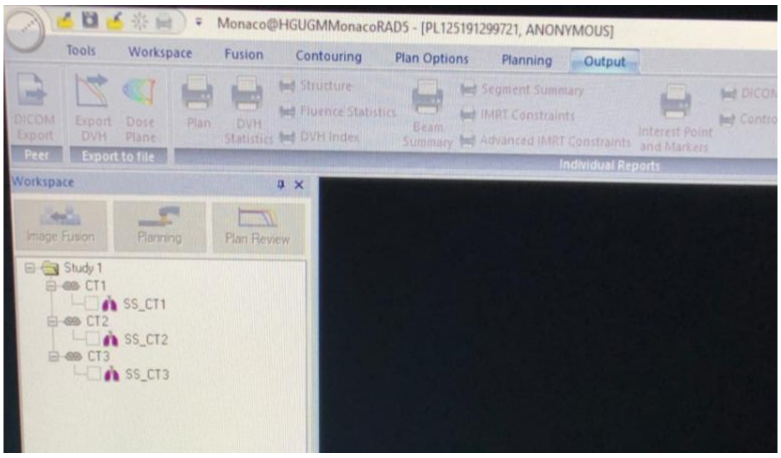


Fig. 7.13. Screenshot showing how the series are stored under the same study for the same patient

It is to be said that the whole implementation of the application could not be installed in the service due to lack of time. Hence, the integration was not checked and DICOM communication between the CT and Slicer remains an unknown.

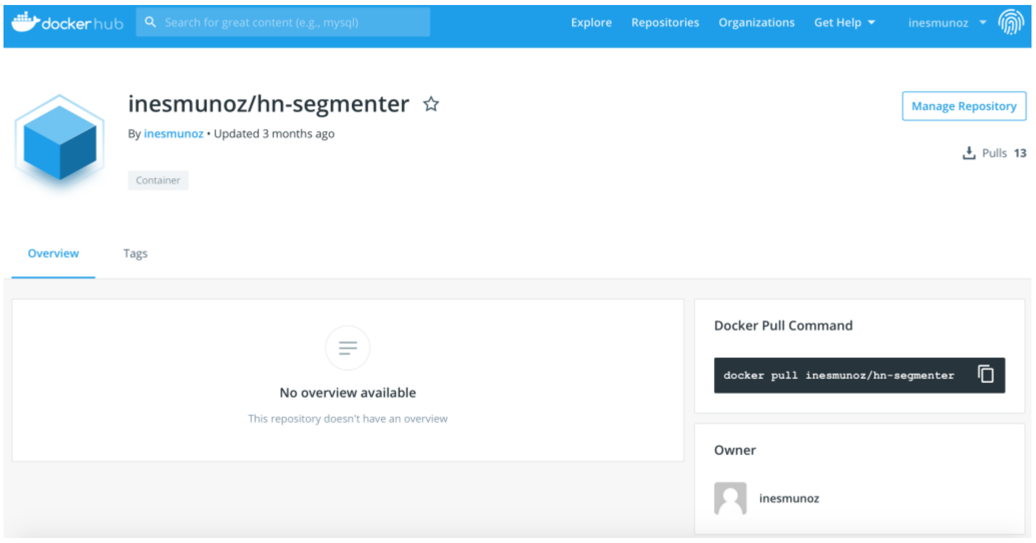
In any case, the approach for integrating the tool and the steps to accomplish it were exposed in point 6.1.3 Integrating the application into the Radiotherapy Service.

Therefore, it was decided to share the results obtained in relation to code, and leave this topic open for further investigations. Find all the information related to the shared results in the following point.

7.4 Sharing results

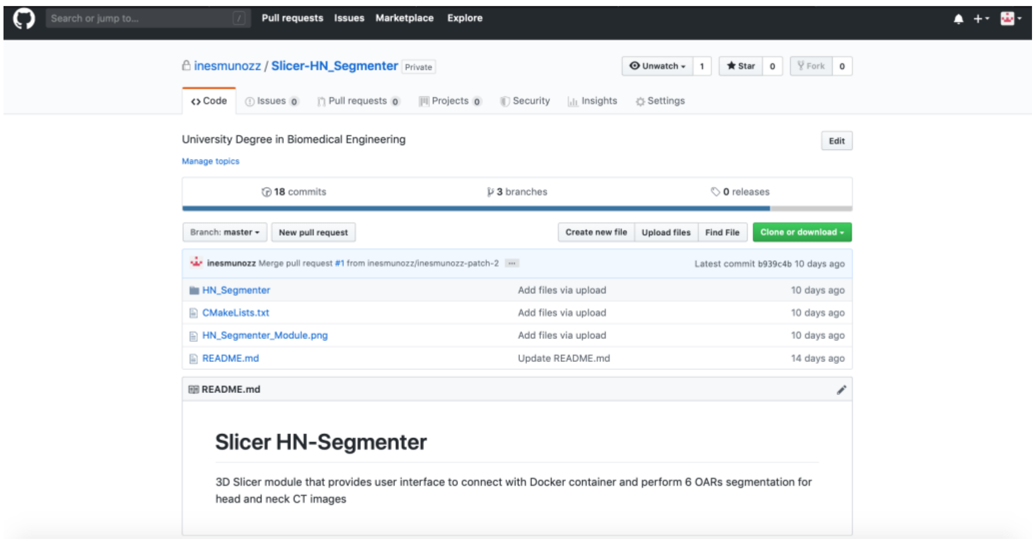
Docker Image was shared in a public repository [37], which is easy and fast to pull by any Docker user. Find the webpage view below in Fig. 7.14.

The code of the Slicer Module, which hosts the extension of the module, was shared in a private repository (Fig. 7.15), which will be shared with the lab members, and which can turn into public at any time.



```
docker pull inesmunoz/hn-segmenter
```

Fig. 7.14. DockerHub webpage view with the hn-segmenter repository



University Degree in Biomedical Engineering

inesmunozz Merge pull request #1 from inesmunozz/inesmunozz-patch-2 — Latest commit b939c4b 10 days ago

HN_Segmenter	Add files via upload	10 days ago
CMakeLists.txt	Add files via upload	10 days ago
HN_Segmenter_Module.png	Add files via upload	10 days ago
README.md	Update README.md	14 days ago

Slicer HN-Segmenter

3D Slicer module that provides user interface to connect with Docker container and perform 6 OARs segmentation for head and neck CT images

Fig. 7.15. GitHub main window where the source code of the module can be seen

Once the extension folder is inside the computer, the module can be loaded through the "Developer Tools for Extensions" module. Then, the module will be found under "Examples" category (Fig. 7.16).

There is also a homepage of the module [38], where the main information about the module is found: description, setup guide, panel description, etc. Find the page overview in Annex 11.1 Slicer H&N Segmenter webpage view.

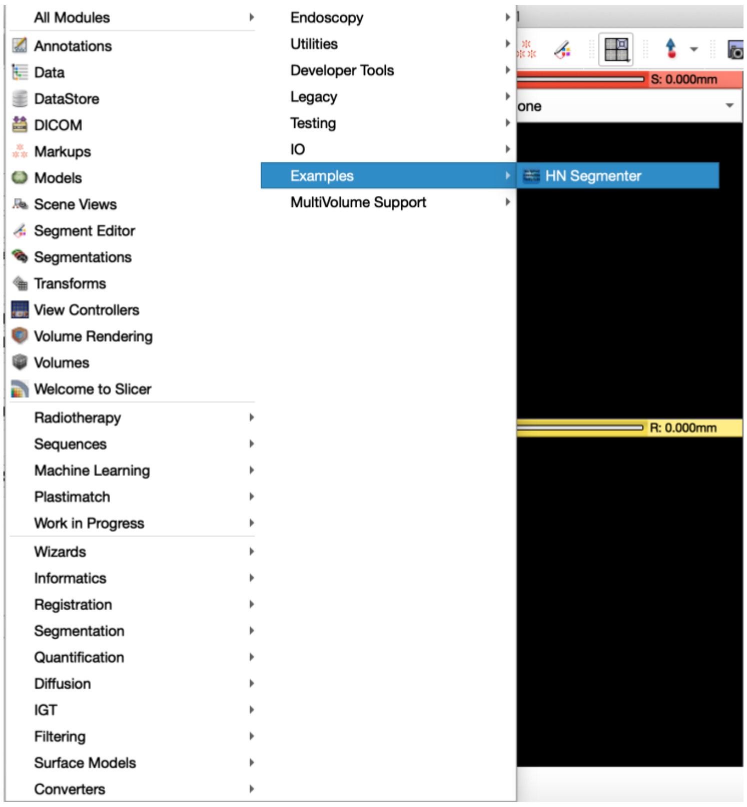


Fig. 7.16. Modules list where the HN Segmenter module can be found

To conclude, the main steps to perform when installing the application in another environment are listed:

1. Copy the extension file in a folder
2. Through "Developer tools for extensions" module, select and load the module
3. Find it under "Examples" category
4. Assure that "Slicer RT" extension is installed
5. Save Slicer application inside a USB
6. Download Docker and install Docker Desktop
7. Pull hn-segmenter Image from the public repository
8. Drag and drop Slicer into the applications folder of the new computer
9. Open Slicer and select the "HN Segmenter" module
10. Run the module

The whole setup doesn't take longer than 10 minutes.

This availability of results will allow the investigation to continue along the same lines in order to adopt the automatic segmentation tool in clinical practice in a future.

8. SOCIO-ECONOMIC IMPACT AND REGULATORY FRAMEWORK

8.1 Socio-economic impact

8.1.1 Social impact

The Radiotherapy Service of the HGUGM hosts around 3.300 new cases of cancer every year [9], where head and neck cancers represent 2,6% of the total, which is a total of 88 new cases every year.

According to this, the project designed in this thesis would automatically segment the OARs in these incoming cases, making the treatment planning a much faster, and shortening waiting time for those patients.

In addition, once this is implemented in the service, research will continue along the same lines to implement automatic segmentation models for other locations to get to accelerate treatment process for all cancers.

Regarding this final objective, which is to implement automatic segmentation as a routine in the radiotherapy service, this thesis can be considered part of a project of great social impact, considering that the reduction of treatment planning times will give greater coverage to cancer patients.

8.1.2 Economic impact

As both Slicer and Docker are free to use, the implementation of this application in the service would cause no hospital expenses.

Since this tool allows to dispense with the services of some technicians, the hospital would greatly contribute to reduce the costs of the hospital.

8.1.3 Budget estimation

The costs of this thesis can be divided into: Human resources (Table 8.1) and equipment costs (Table 8.2).

Human Resources

Table 8.1. Human Resources costs

SERVICE	Time estimation (hours)	Cost (€/hour)	Total (€)
Biomedical Engineer	400	12	4800
Senior Engineer (Supervisor)	100	33	3300

Technician	12	25	300
Physician	3	33	99
		Total cost	**8499**

Equipment

Test cases have no cost, as they are imported from the European challenge database.

Table 8.2. Equipment costs

MATERIAL	**Cost/unit (€)**	**Quantity**	**Total Cost (€)**
Test cases	0	50	0

Therefore, the total budget for the project would be:

Table 8.3. Total costs

SERVICE	**Cost per service (€)**
Human Resources	8499
Equipment cost	0
Total	8499

Computer costs belong to amortization costs of the project.

Table 8.4. Amortization costs

Amortization Costs	**Cost/unit (€)**	**Quantity**	**Total Cost (€)**
Computer	1000	1	1000

8.2 Regulatory framework

Since this thesis lies at the basis of designing a tool for improving a clinical workflow, extreme caution should be exercised when testing the whole tool or the sub-parts of it. Tests should be well identified to be considered as experimental and not to be confused with real cases. This means that, for example, when testing how segmented volumes are read by the software in the workstations, we must tread very carefully on the names of the files that are imported, so as not to interfere with clinical operations, as they will be saved in the server with the rest of patients.

As a research project related with medical images, compliance with the General Data Protection Regulation (GDPR) must be ensured. According to it, to guarantee personal data protection, before facilitating images for research purposes, any personally identifiable information should be removed (anonymization) [39].

Therefore, it is not allowed to import DICOM files from the server, as they contain all patient and study information. DICOM files should be anonymized before using them. That is, removing the name of the patient and institution, the date and other text data on image metadata. This is the reason why in this thesis the database used to test the application was obtained from a European challenge, where all the patients where in nifty format. That is, no metadata at all.

It is worth mentioning that, when the time to integrate the application into de service comes, it must be agreed with the operators of the other DICOM nodes how to manage configuration-related issues, such as network ports and application entity titles (AE Titles), so that exchange of information between CT and workstations is accomplished successfully.

9. CONCLUSION

In recent decades, image analysis and classification has quickly evolved reaching the medical field and the medical image analysis, with truly promising results. This thesis constitutes an approach on how to bring the state of the art in medical image segmentation to a real clinical workflow.

To do so, research was conducted on how to develop a simple and portable tool that runs an already-trained Neural Network provided by another intern student in the lab inside the Radiotherapy Service.

Results showed that storing the Neural Network in a Docker Container and developing an intuitive Slicer module to interact with it, was a quite good approach in terms of execution time and simplicity to install. Hence, it can be concluded that the overall objectives of the project were positively achieved. Unfortunately, the whole application could not be tested or integrated in the service due to lack of time. To overcome this problem, results where shared with the lab members to allow simplify research in this field.

Therefore, this project laid the foundations for future related research. As the student shared the results with the lab members, improvements and changes may be implemented to get the best results possible. In addition, the tool may be extended by developing new models for automatic-segmentation of OARs at other locations.

This thesis is no more than a drop in the ocean in biomedical research, but it constitutes a great step towards improving the lifestyle of cancer patients, by providing a tool that shall shorten waiting times before receiving treatment, which must be a long and difficult struggle.

10. BIBLIOGRAPHY

[1] Sociedad Española de Oncología Médica, «Las cifras del cáncer en España 2018,» 2018.

[2] J. A. Ridge, «Modern Medicine Network,» 2 Junio 2016. [En línea]. Available: https://www.cancernetwork.com/cancer-management/head-and-neck-tumors. [Último acceso: Agosto 2019].

[3] H. M. e. al, «Interaction between tobacco and alcohol use and the risk of head and neck cancer: pooled analysis in the International Head and Neck Cancer Epidemiology Consortium.,» vol. 18, nº 2, pp. 541-50, 3 Febrero 2009.

[4] D. G. Rettig EM1, «Epidemiology of head and neck cancer,» *Surg Oncol Clin N Am,* vol. 24, nº 3, pp. 379-96, July 2015.

[5] Global Cancer Observatory, Agosto 2019. [En línea]. Available: http://gco.iarc.fr.

[6] S. Paoli, «A review of scientific papers about head and neck cancers,» *Brazilian archives of biology and technology,* vol. 51, pp. 63-69, December 2008.

[7] American Speech-Language-Hearing Association, [En línea]. Available: https://www.asha.org/Practice-Portal/Clinical-Topics/Head-and-Neck-Cancer/. [Último acceso: Agosto 2019].

[8] M. Shyh-An Yeh, «Radiotherapy for Head and Neck Cancer,» *Seminars in Plasric Surgery,* vol. 24, nº 2, pp. 127-136, May 2010.

[9] Hospital General Universitario Gregorio Marañón, «Oncología Radioterápica ORT,» [En línea]. Available: http://www.madrid.org/cs/Satellite?blobcol=urldata&blobheader=application%2Fpdf&blobheadername1=Content-disposition&blobheadername2=cadena&blobheadervalue1=filename%3DOncolog%C3%ADa+RT_2016.pdf&blobheadervalue2=language%3Des%26site%3DHospitalGregorioMaranon&blobkey=id&blobtable=MungoBlobs&blobwhere=1352927905360&ssbinary=true. [Último acceso: September 2019].

[10] National Cancer Institute, [En línea]. Available: https://visualsonline.cancer.gov/details.cfm?imageid=9414. [Último acceso: September 2019].

[11] M. R. S. Juan Eugenio Iglesias, «Multi-atlas segmentation of biomedical images: A survey,» *Medical Image Analysis,* vol. 24, pp. 205-219, July 2015.

[12] Eli Gibson, Francesco Giganti, Yipeng Hu, Ester Bonmati, Steve Bandula, Kurinchi Gurusamy, Brian Davidson, Stephen P. Pereira, Matthew J. Clarkson, Dean C. Barratt, «Automatic Multi-Organ Segmentation on Abdominal CT With Dense V-Networks,» *IEEE TRANSACTIONS ON MEDICAL IMAGING,* vol. 37, nº 8, August 2018.

[13] van Rikxoort E, Arzhaeva Y, van Ginneken B., «Automatic segmentation of the liver in computed tomography scans with voxel classification and atlas matching.,» *Proceeding MIC-CAI Workshop, Citeseer,* p. 101–108, 2007.

[14] Fritscher, K. D., Peroni, M., Zaffino, P., Spadea, M. F., Schubert, R., & Sharp, G., «Automatic segmentation of head and neck CT images for radiotherapy treatment planning using multiple atlases, statistical appearance models, and geodesic active contours,» *Medical physics,* vol. 41, nº 5, 2014.

[15] Ledig C, Heckemann RA, Hammers A, Lopez JC, Newcombe VF, Makropoulos A, Lötjönen J, Menon DK, Rueckert D., «Robust whole-brain segmentation: application to traumatic brain injury.,» *Medical Image Analysis,* 2014.

[16] P. Aljabar, R.A. Heckemann, A. Hammers, J.V. Hajnal, D. Rueckert, «Multi-atlas based segmentation of brain images: Atlas selection and its effecton accuracy,» *NeuroImage,* February 2009.

[17] M. Lai, «Deep Learning for Medical Image Segmentation,» April 2015. [En línea]. Available: https://arxiv.org/pdf/1505.02000.pdf?.

[18] M. A. Nielsen, Neural Networks and Deep Learning, Determination Press, 2015.

[19] «An Intuitive Explanation of Convolutional Neural Networks,» August 2016. [En línea]. Available: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/. [Último acceso: August 2019].

[20] Slicer Community, «Slicer Documentation,» June 2019. [En línea]. Available: https://buildmedia.readthedocs.org/media/pdf/slicer/latest/slicer.pdf. [Último acceso: August 2019].

[21] Fedorov, A., Beichel, R., Kalpathy-Cramer, J., Finet, J., Fillion-Robin, J. C., Pujol, S., … Kikinis, R., «3D Slicer as an image computing platform for the Quantitative Imaging Network. Magnetic resonance imaging,» *Magnetic Resonance Imaging,* vol. 30, nº 9, pp. 1323-1341.

[22] Docker, «Enterprise Container Platform,» [En línea]. Available: https://www.docker.com.

[23] [En línea]. Available: https://docker-curriculum.com. [Último acceso: August 2019].

[24] V. Chaturvedi, «Docker Tutorial – Introduction To Docker & Containerization,» May 2019. [En línea]. Available: https://www.edureka.co/blog/docker-tutorial. [Último acceso: August 2019].

[25] E. G. e. al., «NiftyNet: a deep-learning platform for medical imaging,» *Computer methods and programs in biomedicine,* vol. 158, pp. 113-122, 2018.

[26] docker-docs, «Dockerfile reference,» [En línea]. Available: https://docs.docker.com/engine/reference/builder/. [Último acceso: September 2019].

[27] keras-team, «GitHub,» [En línea]. Available: https://github.com/keras-team/keras/blob/master/docker/Dockerfile. [Último acceso: September 2019].

[28] J. Skałecki, «How to reuse Keras Deep Neural Network using Docker,» *Rock-IT,* 20 May 2017.

[29] [En línea]. Available: https://hub.docker.com.

[30] slicer.org, «Documentation/Nightly/Developers/ExtensionWizard,» [En línea]. Available: https://www.slicer.org/wiki/Documentation/Nightly/Developers/ExtensionWizard. [Último acceso: September 2019].

[31] slicer.org, «Documentation/Nightly/Developers/Modules,» [En línea]. Available: https://www.slicer.org/wiki/Documentation/Nightly/Developers/Modules. [Último acceso: September 2019].

[32] «Elekta,» [En línea]. Available: https://www.elekta.com/radiotherapy. [Último acceso: September 2019].

[33] Automatic Structure Segmentation for Radiotherapy Planning Challenge 2019, «Grand Challenge,» [En línea]. Available: https://structseg2019.grand-challenge.org/Home/. [Último acceso: September 2019].

[34] «dockerhub,» June 2019. [En línea]. Available: https://cloud.docker.com/u/inesmunoz/repository/docker/inesmunoz/hn-segmenter.

[35] [En línea]. Available: https://github.com. [Último acceso: September 2019].

[36] slicer.org, «Slicer Wiki,» [En línea]. Available: https://www.slicer.org/wiki/Documentation/4.8/SlicerApplication/ExtensionsManager.

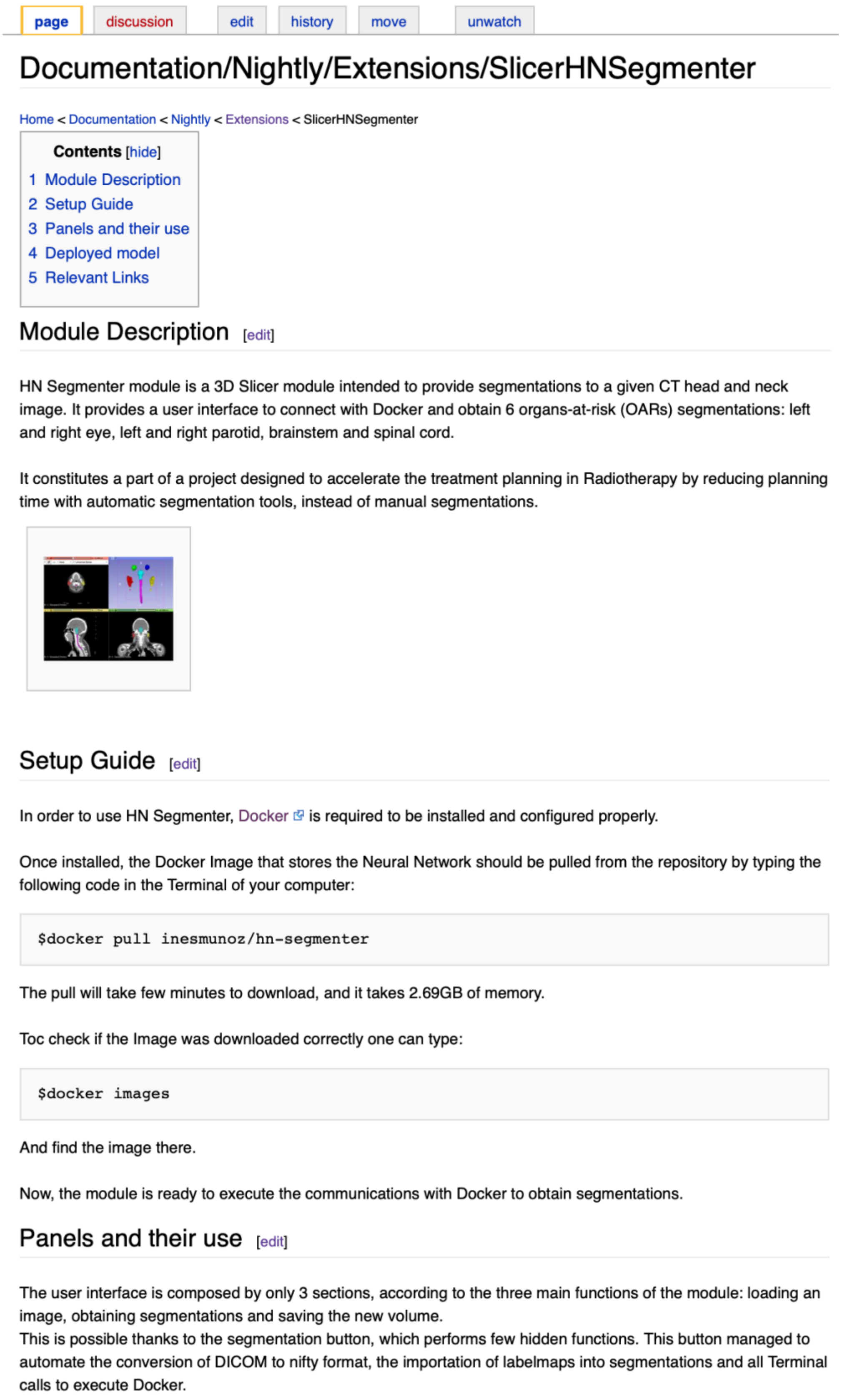
[37] I. Muñoz, «DockerHub,» [En línea]. Available: https://hub.docker.com/r/inesmunoz/hn-segmenter.

[38] I. Muñoz, «Documentation/Nightly/Extensions/SlicerHNSegmenter,» 2019. [En línea]. Available: https://www.slicer.org/wiki/Documentation/Nightly/Extensions/SlicerHNSegmenter.

[39] European Society of Radiology (ESR), «The new EU General Data Protection Regulation: what the radiologist should know,» *Insights Imaging*, vol. 8, nº 3, p. 295–299, April 2017.

11. ANNEX

11.1 Slicer H&N Segmenter webpage view



page discussion edit history move unwatch

Documentation/Nightly/Extensions/SlicerHNSegmenter

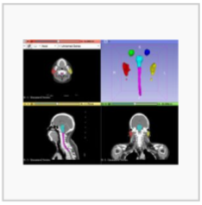
Home < Documentation < Nightly < Extensions < SlicerHNSegmenter

Contents [hide]

Module Description [edit]

HN Segmenter module is a 3D Slicer module intended to provide segmentations to a given CT head and neck image. It provides a user interface to connect with Docker and obtain 6 organs-at-risk (OARs) segmentations: left and right eye, left and right parotid, brainstem and spinal cord.

It constitutes a part of a project designed to accelerate the treatment planning in Radiotherapy by reducing planning time with automatic segmentation tools, instead of manual segmentations.



Setup Guide [edit]

In order to use HN Segmenter, Docker is required to be installed and configured properly.

Once installed, the Docker Image that stores the Neural Network should be pulled from the repository by typing the following code in the Terminal of your computer:

```
$docker pull inesmunoz/hn-segmenter
```

The pull will take few minutes to download, and it takes 2.69GB of memory.

Toc check if the Image was downloaded correctly one can type:

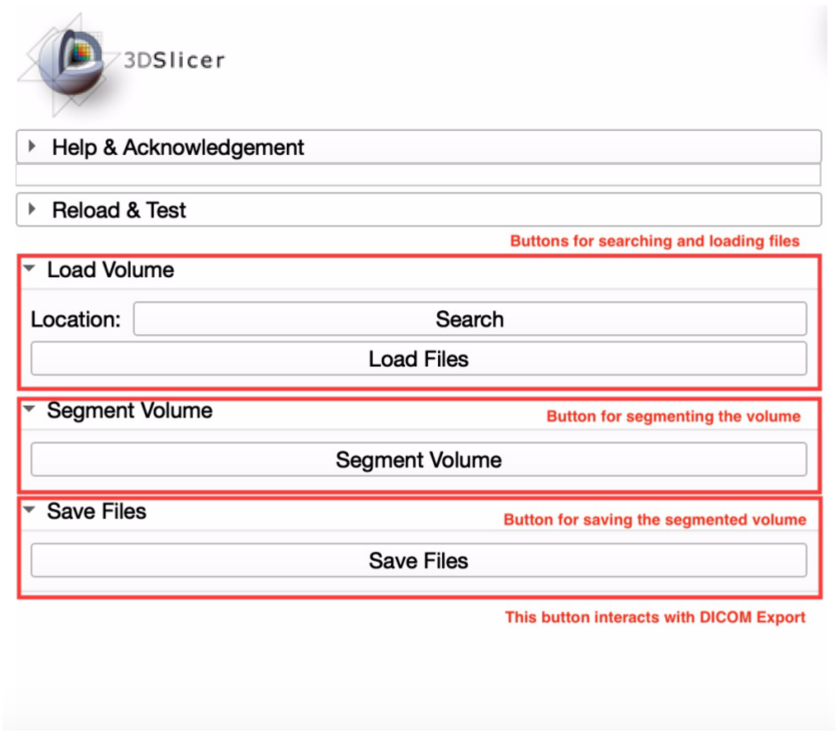
```
$docker images
```

And find the image there.

Now, the module is ready to execute the communications with Docker to obtain segmentations.

Panels and their use [edit]

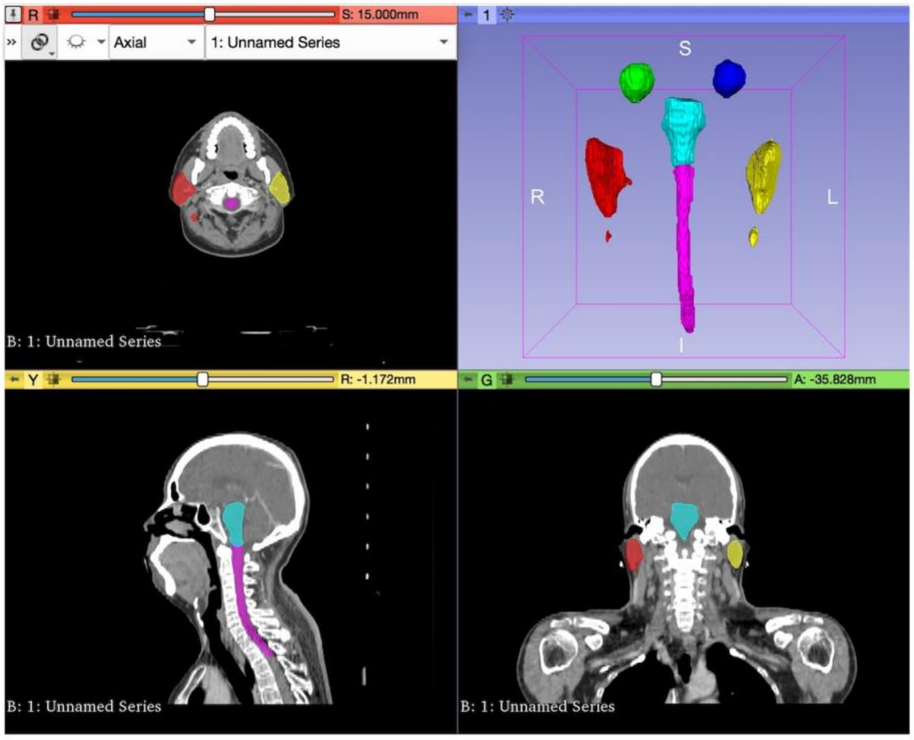
The user interface is composed by only 3 sections, according to the three main functions of the module: loading an image, obtaining segmentations and saving the new volume.
This is possible thanks to the segmentation button, which performs few hidden functions. This button managed to automate the conversion of DICOM to nifty format, the importation of labelmaps into segmentations and all Terminal calls to execute Docker.


3DSlicer
Help & Acknowledgement
Reload & Test
Buttons for searching and loading files
Load Volume
Location:
Search
Load Files
Segment Volume
Button for segmenting the volume
Segment Volume
Save Files
Button for saving the segmented volume
Save Files
This button interacts with DICOM Export

Deployed model [edit]

The dataset used to test and evaluate the performance of the module is composed by a set of CT scans obtained from the Structure Segmentation for Radiotherapy Planning Challenge 2019, a competition of MICCAI 2019 Challenge. Find here the link.

The module proved to be consistent and robust obtaining segmentations for all cases except for those images that were cut. Segmentations are usually smooth in shape and the failures are easy to detect and correct.



Relevant Links [edit]

Slicer HN-Segmenter Module Source Code