Alma Mater Studiorum · University of Bologna

School of Science
Department of Physics and Astronomy
Master Degree in Physics

Implementation of an automated pipeline to predict the response to neoadjuvant chemo-radiotherapy of patients affected by colorectal cancer

Supervisor:
Prof. Gastone Castellani

Co-supervisor:
Dr. Nico Curti

Submitted by:
Giuseppe Filitto

Academic Year 2020/2021

Abstract

Colorectal cancer is a malignant neoplasm of the large intestine resulting from the uncontrolled proliferation of one of the cells making up the colorectal tract.
In order to get information about diagnosis, therapy evaluation on colorectal cancer, analysis on radiological images can be performed through the application of dedicated algorithms. Up to now, this process is performed using manual or semi-automatic techniques, which are time-consuming and highly operator dependent.
The aim of this project is to develop and apply an automated pipeline to predict the response to neoadjuvant chemo-radiotherapy of patients affected by colorectal cancer. Here, we propose an approach based on automatic segmentation and radiomic features extraction. The segmentation process exploits a Convolutional Neural Network like U-Net, trained with medical annotations to perform the segmentation of the tumor areas. Then, from the segmented regions, radiomic features are extracted and analyzed to obtain the prediction of response, based on the Tumor Regression Grade (TRG).
We tested and developed our pipeline on MRI scans provided by the IRCCS Sant'Orsola-Malpighi Polyclinic. The performance of the pipeline was measured for the segmentation purpose and for the prediction of response. The results of these preliminary tests show that the pipeline is able to achieve a segmentation consistent with the medical annotations and a Dice Similarity Coefficient (DSC) coherent with literature. Even for the prediction of response, the results show that the pipeline is able to correctly classify most of the cases.

. . . To my family and Nicole

Contents

Introduction

Colorectal cancer is a malignant neoplasm of the large intestine resulting from the uncontrolled proliferation of the cells making up the colorectal tract. Colorectal cancer is the second malignant tumor per number of deaths after lung cancer and the third per number of new cases after breast and lung cancer[1]. It is estimated that, in EU-27 countries in 2020, colorectal cancer accounted for 12.7% of all new cancer diagnoses and 12.4% of all deaths due to cancer[2].

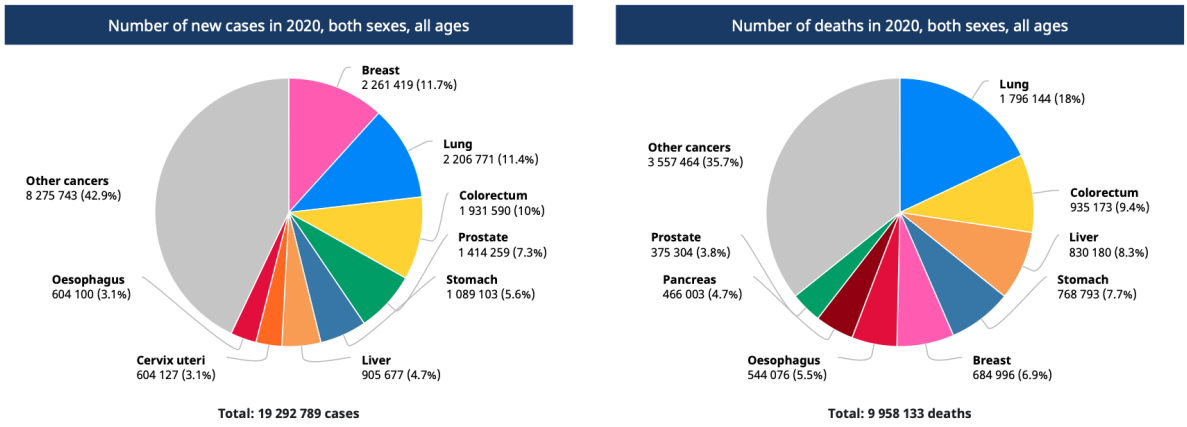

Figure 1: World's cancer cases and deaths in 2020. As you can see from the *Left* image, colorectal cancer is the third malignant tumor per number of new cases after breast and lung cancer while as shown on the *Right* image, it is the second one per number of deaths after lung cancer. From *International Agency for Research on Cancer*[1].

Among the risk factors for this kind of cancer, non-hereditary ones range from colon polyps to long-standing ulcerative colitis, from Crohn's disease to old age. Moreover, genetic history (HNPCC or Lynch syndrome) and nutritional factors as diabetes II can increase the probability of developing colorectal cancer [3].

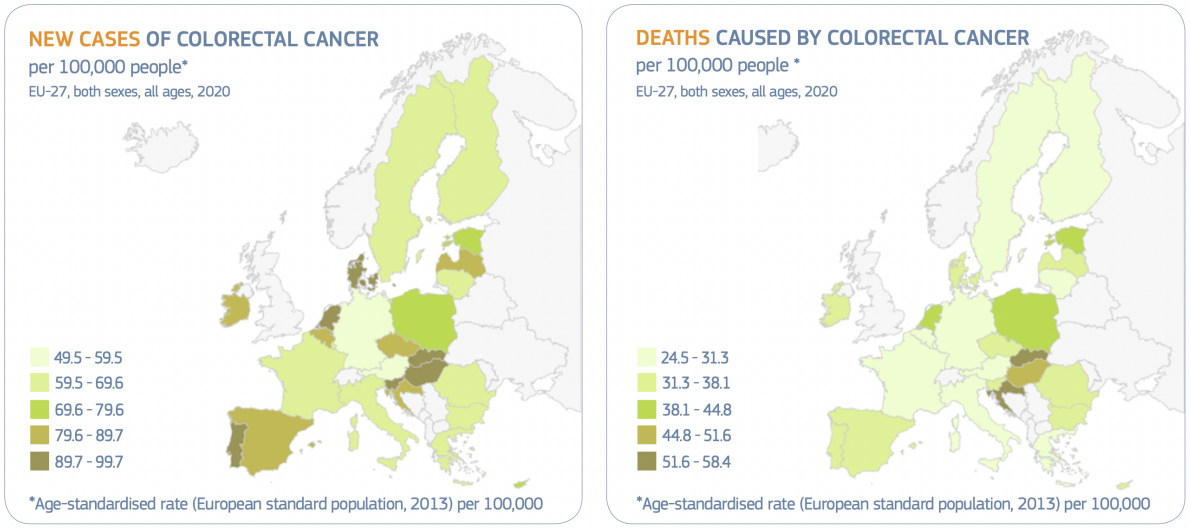

Figure 2: EU-27 countries' distribution map of colorectal cancer cases and deaths in 2020 per 100.000 people (both sex, all ages). *Left)*: EU-27 countries' new cases of colorectal cancer. *Right)*: EU-27 countries' deaths caused by colorectal cancer. From *ECIS – European Cancer Information System*[2].

Preventive measures for colorectal cancer include physical activity, reducing the consumption of processed meat and alcohol, and avoiding smoking[4].
Screening and diagnosis methods for colorectal cancer involve different techniques. The gold standard in medical routines is the colonoscopy which is an invasive technique[5]. Among medical imaging techniques, Magnetic Resonance Imaging (MRI) and Computed Tomography (CT) are the most used[3]. In particular, Magnetic Resonance Imaging (MRI) is used for pre-operative identification and for the evaluation of the neoadjuvant chemo-radiotherapy of patients affected by colorectal cancer[3].

In order to get information about diagnosis, therapy evaluation, stage of colorectal cancer, analysis on radiological images can be performed through dedicated algorithms. In this scenario, the correct and fast identification of the cancer regions is a fundamental task. This can be achieved by using segmentation techniques. The results of segmentation can be used to perform radiomic features extraction, which can provide fundamental information about organs or lesion volumes, to monitor the evolution of a particular disease, and/or to evaluate the effects of therapeutical treatment. Therefore, segmentation plays a crucial role for clinicians in identifying diseases such as tumors. Up to now, this task is performed by using manual or semi-automatic techniques which are time-consuming (requiring hours per day) since it requires interaction with trained specialists.[3, 5]. Moreover, the obtained results are highly sensitive to the specialist expertise. Therefore, the reproducibility of the same results is not always possible[6]. To

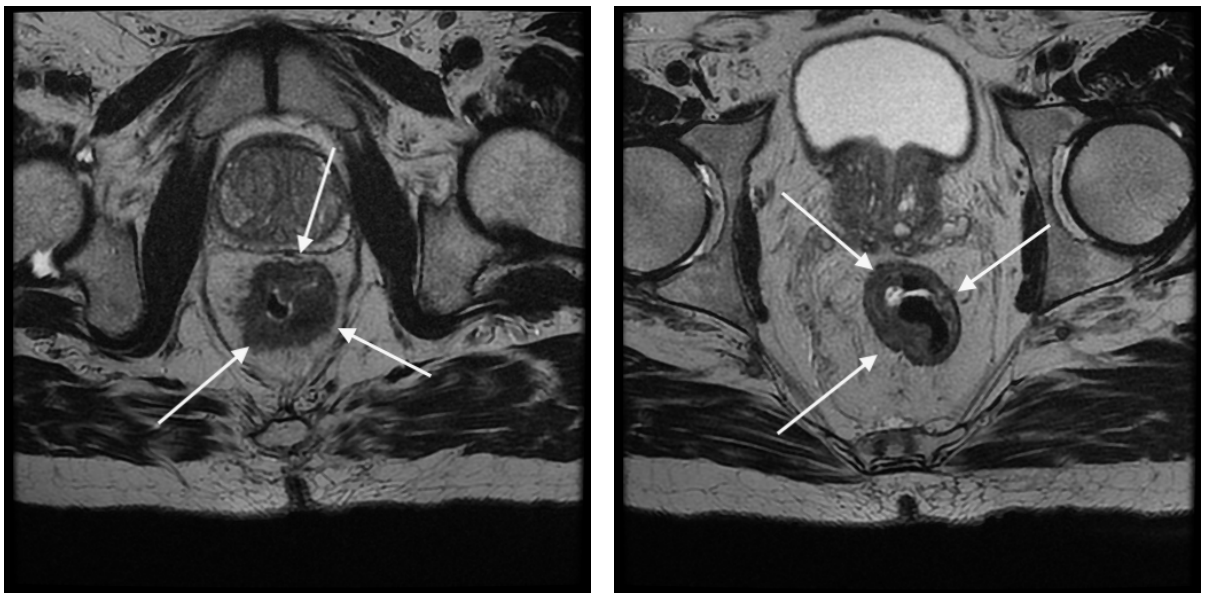

Figure 3: Axial T2-weighted Magnetic Resonance images of colorectal tract. The images provide a view from the transverse (or axial) plane of the colorectal tract, highlighting fat (low signal) and water (high signal) within the body. The tumor region is denoted by the arrows. From IRCCS Sant'Orsola-Malpighi Policlinic Dataset.

overcome these issues, an automated and fast approach is required.
The aim of this project is to develop and apply an automated pipeline for the segmentation of MRI scans of patients affected by colorectal cancer in order to predict the response to neoadjuvant chemo-radiotherapy by using radiomic features. Here, we propose an approach based on segmentation and radiomic features extraction. The segmentation process exploits a Convolutional Neural Network like U-Net to perform automatic segmentation of the tumor regions. Then, the results of segmentation are used to perform radiomic features extraction to obtain the prediction of response, based on the Tumor Regression Grade (TRG). The work was developed and validated on MRI scans provided by IRCCS Sant'Orsola-Malpighi Polyclinic.
The dissertation will start from the materials and methods. Firstly, we will focus on medical digital images, the general properties of medical digital images, and the DICOM format. After that, it will be the turn of the performed pre-processing techniques. They consist of a denoising technique and a gamma correction to remove possible noise sources and improve the image contrast, respectively. Moreover, we will cover the topic of segmentation, providing to the reader a description of Thresholding, Convolutional Neural Networks and U-Net. Then, we will talk about radiomics and its possible purposes. Even an overview of Principal Component Analysis and Support Vector Classifiers will be provided. To conclude the materials and methods chapter, we will see the main

methods used to evaluate the obtained results. Among them, we find the Dice Similarity Coefficient (DSC), the confusion matrix and the ROC curve.
The dissertation will continue describing the main pipeline characteristics and its structure. Firstly, we will cover the description of the pipeline workflow. In particular, about how each step of the pipeline workflow was achieved and performed, from the pre-processing of the images to the prediction of the response to neoadjuvant chemo-radiotherapy. This chapter will focus both on the pipeline description and implementation.
The obtained results will be presented in the aimed chapter. After a description of the dataset provided by the IRCCS Sant'Orsola-Malpighi Polyclinic, the performance of the pipeline will be checked respectively for the segmentation task and for the prediction of response. Moreover, the possible outputs of the implemented pipeline will be shown.
Finally, the results and some possible developments will be discussed in the conclusions. A comparison between two different segmentation models will be discussed in the Appendix of this text.

Chapter 1

Materials and Methods

1.1 Medical Digital Images

A medical digital image is the digital representation of the anatomical (or functional) structure of the patient. It is composed by a finite number of elements called *pixels*. A pixel is a discrete numeric representation of intensity or gray-level. It is an output coming from a two-dimensional function $f(x, y)$. The input of this function consists in the spatial coordinates, denoted with (x, y) on the x-axis and y-axis of the image plane[7].
A digital image can be processed by computers. This process is called *digital image processing* and it can be divided into two main categories: (*image processing*) and (*image analysis*). The former includes methods whose output and input data are images. The latter includes methods whose input data can be images and the output data are attributes extracted from the images.

1.1.1 General Properties

The physical meaning of the image data depends on the performed image modality. For example, Computed Tomography (CT) and Magnetic Resonance Imaging (MRI), give structural information about the anatomy of the patient. Other techniques, such as Positron Emission Tomography (PET) or Functional Magnetic Resonance Imaging (fMRI) give information about the functional properties of the patient's target organs. However, we can distinguish some general characteristics which involve both the acquisition modalities.

Pixel depth

It is the number of bits used to encode the values of each pixel and it is related to the memory space used to store the amount of the encoded information[8]. Higher the

number of bits, higher the information stored[8]. A group of 8 bits is called *byte* and represent the smallest quantity that can be stored in the memory of a computer. For example, if an image has a pixel depth of 16 or 12 bits the computer will always store two bytes per pixel with the drawback of an higher memory space required for the storage[8]. With a pixel depth of 8 bits it is possible to codify and store integer numbers between 0 and 255 ($2^8 - 1$). There are also two formats for the encoding in binary of floating-point numbers: single precision 32-bit and double precision 64-bit.

Pixel data

Pixel data represent numerical values of the pixels stored according to the data type. Pixel data can be complex values even if this data type is not common and can be bypassed by storing the real and imaginary parts as individual images. For example, complex data are provided in MRI acquired data before the reconstruction (the so called k-space)[8].

Metadata

Metadata are information that describe the image and the acquisition process used. It is usually stored at the beginning of the file as a header[8]. In the case of medical images, metadata have an important role due to the nature of the images. For example, a magnetic resonance image might have parameters related to the pulse sequence used, timing information, number of acquisitions while a PET image might have information about the radiopharmaceutical injected to the patient.

DICOM Format

Image file formats provide a standard way to store information of an image in a computer file[9]. DICOM is the acronym of Digital Imaging and COmmunications in medicine. It is not only a file format but also a network communication protocol[8]. However here, we will discuss DICOM only as a medical image format.
DICOM file format establishes that the pixel data cannot be separated from the metadata[8]. In other words, metadata and pixel data are merged in a unique file. The header contains the description of the entire procedure used to generate the image in terms of acquisition protocol and scanning parameters[8]. It also contains patient information such as name, gender, age. For these reasons, the DICOM header is modality-dependent and varies in size. In practice, the header allows the image to be *self-descriptive*.

1.2 Pre-processing techniques

Pre-processing is a common name for operations with images. Both input and output are images. The aim of pre-processing is an improvement of the image data that suppresses unwilling issues such as noise or enhances some image features important for further processing[10].
For the development of this project, the pre-processing involves a denoising filtering technique and a gamma correction to improve the contrast.

1.2.1 Spatial Domain Filtering

Filtering is a procedure used for modifying or enhancing an image. The value of any given pixel in the output image is determined by applying some operations to the neighborhood of the corresponding input pixel. A pixel's neighborhood is some set of pixels, defined by their locations relative to that pixel. The term *spatial domain* indicates that the procedures operate directly on pixels. Mathematically:

$$g(x, y) = T[f(x, y)] \tag{1.1}$$

where $f(x, y)$ is the input image, $g(x, y)$ the output image and T is an operator on f defined over some neighborhood of (x, y). The operation on the point located in (x, y) usually involves the application of a matrix called *mask* or *kernel*. The application of the above-mentioned mask (or kernel) on an image is called *spatial filtering*. Filters create a new pixel with the same coordinates of the center of the neighborhood, whose value is the result of the operation. For each (x, y) of the image, the filter $g(x, y)$ is a set of combination of the mask coefficient $w(s, t)$ and the pixels of the image affected by the mask itself. In general, we can write:

$$g(x, y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s, t) f(x + s, y + t) \tag{1.2}$$

Spatial filters include also smoothing filters. They are used for blurring and for noise reduction[11]. The general implementation for filtering an $M \times N$ image with a weighted averaging filter of size $m \times n$ is given by:

$$g(x, y) = \frac{\sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s, t) f(x + s, y + t)}{\sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s, t)} \tag{1.3}$$

where $m = 2a + 1$ and $n = 2b + 1$.
Smoothing spatial filters include *linear filters* and *nonlinear filters*[11].
The former are based on the mean filter which is a simple sliding spatial filter that replaces the center value in the mask region with the average of all the neighboring pixel

values including itself; the latter is based on the median filter which consist in evaluating the median of the pixel values in the mask region and replacing the center pixel of the mask region with the computed median value[12].
One drawbacks of smoothing filters is the blurring of the image that can cause the loss of important details. However, some filtering techniques and algorithms allow to remove noise without significantly blurring the structures of the image, preventing the loss of details[13].

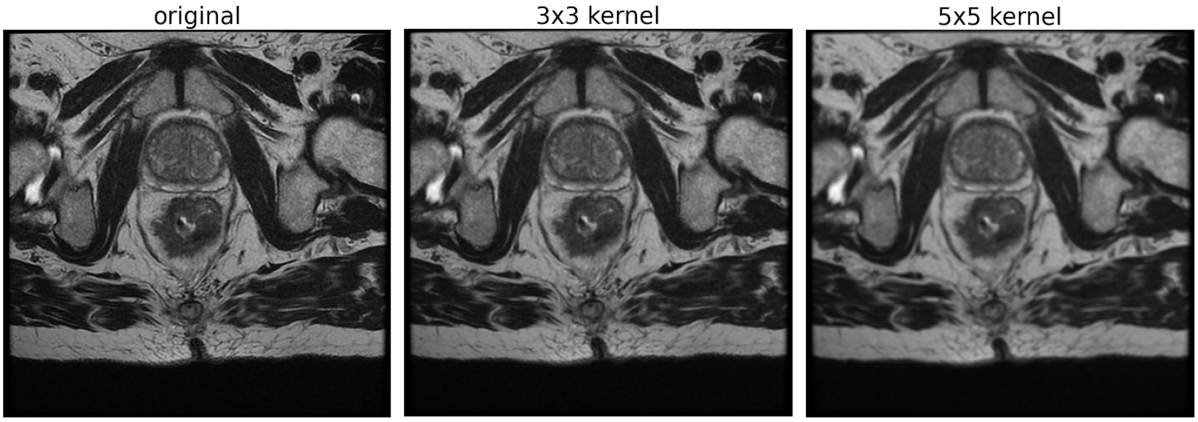

Figure 1.1: Application of mean filter on MR image varying the kernel size. As you can see, the larger the kernel, the higher the smoothness of the image but also the blurring with consequently detail loss. Images from IRCCS Sant'Orsola-Malpighi Policlinic Dataset.

Non-local means algorithm

The non-local means algorithm is used for image denoising. Unlike mean filters, which take the mean value of a group of pixels surrounding a target pixel to smooth the image, the non-local means algorithm replaces the value of a pixel by an average of a selection of other pixels values: small patches centered on the other pixels are compared to the patch centered on the pixel of interest, and the average is performed only for pixels that have patches close to the current patch. As a result, this algorithm can restore well textures, that would be blurred by other denoising algorithm[14].
Mathematically:

$$\hat{u}(p) = \frac{1}{C(p)} \sum_{q \in \Omega)} u(q) w(p,q) \qquad\qquad C(p) = \sum_{q \in \Omega} w(p,q) \tag{1.4}$$

where p and q are two points within the image Ω, $\hat{u}(p)$ is the filtered value of the image at point p, $u(p)$ is the unfiltered value of the image at point p and $w(p,q)$ is the

weighting function which purpose is to determine how closely related the image at the point p is to the image at the point q. It can take many forms. The gaussian weighting function can be written setting up a normal distribution with mean $\mu = B(p)$ and a standard deviation h [13]:

$$w(q,p) = e^{-\frac{\|B(q)-B(p)\|^2}{h^2}} \tag{1.5}$$

where $B(p)$ is the local mean value of the image point values surrounding p:

$$B(p) = \frac{1}{|R(p)|} \sum_{k \in R(p)} u(k) \tag{1.6}$$

where $|R(p)|$ is the number of pixels in the square region R centered in p.

1.2.2 Gamma Correction

Medical digital images can suffer from poor contrast[15]. In order to prevent information loss and to appreciate all the details of the image, it is necessary to enhance the contrast of such images. There are numerous existing techniques that can be performed for this purpose. In this work, I performed gamma correction.
It consists of a non-linear operation, defined in the simplest cases, by the following power-law expression:

$$I_{out} = C I_{in}^{\gamma} \tag{1.7}$$

where the output I_{out} is obtained multiplying by a constant C the input value I_{in} raised to the power γ. A value of $\gamma < 1$ is sometimes called an encoding gamma; conversely a gamma value $\gamma > 1$ is called a decoding gamma. In practice, powers of γ larger than 1 make the shadows darker, while powers smaller than 1 make dark regions lighter. In particular, for this work, the gamma correction was performed by the following expression:

$$I_{out} = (\frac{I_{in} - I_{min}}{I_{max} - I_{min}})^{\gamma} \tag{1.8}$$

where I_{min}, I_{max} are respectively the minimum and the maximum image value.

Figure 1.2: Example of gamma correction, applied to a MR image, varying γ. As you can see, a value of γ larger than 1 enhances darker regions while a value smaller than 1 makes dark regions lighter; a value of γ equal to 1 corresponds to the original image. Images from IRCCS Sant'Orsola-Malpighi Policlinic Dataset.

1.3 Segmentation

Image segmentation refers to the process of partitioning a digital image into multiple segments (i.e. set of pixels), regions or objects so as to change the representation of an image into something that is more meaningful and easier to analyze. Pixels in a region are similar according to some homogeneity criteria such as colour, intensity or texture, so as to locate and identify objects and boundaries in an image[16].
Image segmentation applications range from medical filed (i.e locate tumors and other pathologies, study of anatomical structure), object detection in satellite images (roads, forests, cars), face recognition, finger print recognition, etc...
In general, segmentation, depending on the technique, can be manual, semi-manual, or automatic:

Manual Segmentation

Manual techniques are still the most reliable and precise methods but they are time-consuming, highly operator-dependent, and require expert operators[3]. Examples of manual segmentation techniques include partitioning an image selecting manually the border of pixels belonging to a specific region or object (i.e tumors in medical images).

Semi-Manual Segmentation

Semi-Manual techniques are faster compared to manual ones. They are based on traditional image processing methods such as thresholding and clustering. Despite the time

savings they are still operator-dependent[3]. Moreover, techniques such as thresholding do not take into account the spatial characteristics of the image so it is not always possible to get the desired partitioning.

Automatic Segmentation

Automatic segmentation involve faster techniques compared to manual and semi-manual ones. Moreover, they are not operator-dependent. Among them we find Convolutional Neural Networks. However, the implementation of the networks ad algorithms is harder to perform[3].

1.3.1 Segmentation Methods Review

Many segmentation methods have been developed over the years. The choice of the best segmentation method depends by the particular type of image and characteristics of the problem being considered[16]. There are various ways to classify these methods. For example, depending if they require or not a training set of data, they can be classified into *supervised* or *unsupervised* methods. Moreover, they can be classified depending on the information type they use, like *Pixel classification* methods, which use only information about pixel intensity, or *Boundary following* methods that use edge information etc...[9]. At this point, I will provide a brief summary of one of the most common manual/semi-manual approach such as Thresholding. Then, I will focus on Convolutional Neural Networks and in particular on the U-Net architecture.

Thresholding

Thresholding is a very simple and common approach to segmentation. This method is applied on the *histogram* of the image. The histogram of a digital image with intensity levels L in the range $[0,\ L-1]$, is a discrete function $h(l_k) = n_k$ where l_k is the k-th intensity value and n_k is the number of pixels with intensity l_k.
Thresholding consists in binarizing an image through an (if) clause on the intensity value of each pixel. This is done setting a threshold value $T \in [0,\ L-1]$. The threshold value T is usually chosen by visual assessment on the image histogram but it can be automatize by algorithms like the *Otsu algorithm*.
One drawback of this method is that some parts of the image can belong to the same class even if they belong to different objects. In fact, thresholding does not take into account the spatial characteristics of the image. Moreover, it is sensitive to noise and intensity inhomogeneity that can corrupt the image histogram and make difficult the classification of pixels[9].

Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are computational architectures commonly applied to analyze visual imagery. They belong to the class of Deep Neural Network, using a variation of the multilayer perceptrons. The word *Convolutional* indicates that the network employs convolution operations. A typical Convolutional Neural Network architectures is made up of three main structures: an *input layer* that is the set of data, *hidden layers* and an *output layer*.
The hidden layers include different units. Among them we find convolutional layers, that consist of convolution kernel sliding along the input matrix. The convolution operation generates a feature map, which in turn contributes to the input of the next layer[17]. This process can be followed by other kinds of layers such as *Activation layers*, *Pooling layers*, *Up-convolution layers*, each one with a different purpose. For example, the Pooling layer aims to reduce the dimensions of the feature maps; the Activation layer, consists of a pixel-wise non-linear function, usually the Rectified Linear Unit (ReLU); the Up-convolution layers aims to up-sample the feature maps. The choice of the layers depends on the characteristics of the problem and the kind of network architecture considered.
The last layer, the *output layer*, provides the result of the classification of the network resulting in a vector containing the belonging probability score of the object to the classes. For this purpose, different functions can be used [18]. In particular, for binary segmentation tasks, one of the most used is the Sigmoid function:

$$Sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{1.9}$$

Other components of the network structure are the optimizers and the loss function. Optimizer are methods responsible of changing the attributes of the neural network such as weights and learning rate to reduce the losses. The most used is the Adaptive moment estimation (Adam)[5].
The loss function affects the training phase by evaluating the error rate. This function must be minimized. Popular loss functions for segmentation purposes include the binary cross-entropy (BCE) and the Categorical Cross-Entropy (CCE):

$$BCE(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i)) \tag{1.10}$$

$$CCE(q) = \sum_{i=1}^{K} q(y_i) \cdot log(p(y_i)) \tag{1.11}$$

where for N data points and K classes, y_i is the truth-label and $p(y_i)$ is the probability for the i^{th} class.

However, we can choose between a wide range of different loss functions[18]. In particular, for this project the loss function consists of the combination between the Dice loss with the binary focal loss[19]. The former comes from the Dice similarity coefficient (DSC):

$$DL(y, \hat{p}) = 1 - \frac{2y\hat{p} + 1}{y + \hat{p} + 1} \tag{1.12}$$

where y is the the truth-label and $\hat{p}$ is the the predicted probability. Here, 1 is added in numerator and denominator to ensure that the function is not undefined in edge case scenarios such as when $y = \hat{p} = 0$.
The latter instead can be defined:

$$FL(y, \hat{p}) = -\alpha y(1 - \hat{p})^{\gamma} log(\hat{p}) - \alpha(1 - y)\hat{p}^{\gamma} log(1 - \hat{p}) \tag{1.13}$$

where y is the the truth-label, $\hat{p}$ is the predicted probability, α is weighting factor and γ a focusing parameter.
The performance of the network is measured by a function called metric. Metric functions are similar to loss function but they do not have to be minimized.
Several architectures have been developed over the years, for different tasks and fields of application. In bio-medical image processing, the so-called U-Net, is one of the most common and used architecture[20].

U-Net

The U-net is a convolutional network architecture for fast and precise segmentation of images especially in the biomedical field[20]. One of the main advantage of the U-net is the ability of dealing with small dataset. The name U-net refers to the U shape of the network architecture. The whole U-net structure, showed in Figure1.3, can be split into two main parts:

Encoder: or *contraction path* is a sequence of convolutional and max pooling layers with the aim of extracting features and reducing dimensionality.

Decoder: or *expansion path* is a sequence of transpose convolutional layers with the aim of reconstructing the feature map and consequently the segmentation mask.
The *Encoder* is a typical Convolutional Neural Network that consists in the repeated application of convolutions, followed by ReLu activation function and max pooling operations. During the contraction the input size is decreased and so the spatial information, while the information about features is increased. The *Decoder* combines the feature maps and the spatial information by a sequence of transpose convolutions (or up-convolutions) and concatenations with the features from the contracting path (grey arrows in Figure1.3).

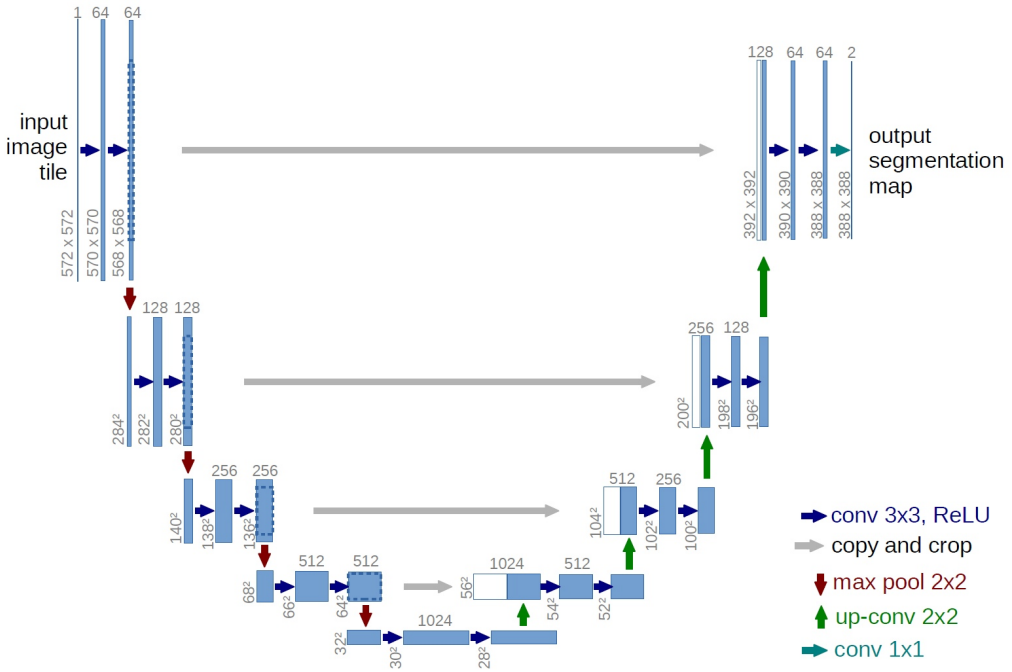

Figure 1.3: Original U-Net architecture. We can notice the U-shape made by the expansion and contraction path. Each blue box corresponds to a series of feature maps. White boxes represent copied feature maps. The arrows denote the different operations. From *U-Net: Convolutional Networks for Biomedical Image Segmentation*[20].

1.4 Radiomics

Radiomics consists in methods that extract from medical images a large number of features, which have the potential to uncover disease characteristics that fail to be appreciated by the naked eye[21]. The main objective of radiomics is to assist the subjective interpretation of the clinicians with an objective prediction. In the new era of precision medicine, radiomics is an emerging translational research field that aims to find associations between qualitative and quantitative information extracted from clinical images and clinical data to support the decision making process[3]. Radiomic features can be divided into different classes:

- First Order Statistics
- Shape based features 2D and 3D
- Gray Level Co-occurrence Matrix (GLCM)
- Gray Level Size Zone (GLSZM)

- Gray Level Run Length Matrix (GLRLM)
- Neighbouring Gray Tone Difference Matrix (NGTDM)
- Gray Level Dependence Matrix (GLDM)

1.4.1 Possible Purposes Of Radiomics

The possible applications of radiomics are based on a very wide range, from the prediction of clinical outcomes to the oncological diagnosis. In this subsection, a brief overview of some general possible purposes will be given.

Prediction of clinical outcomes

Radiomic features may be useful for predicting patient survival and describing intratumoral heterogeneity as demonstrated in a study by Aerts et al. [22]. More, the usefulness of radiomics for predicting the immunotherapy response of patients with non-small cell lung cancer (NSCLC) using pretreatment CT and PET-CT images has been demonstrated by other studies[3].

Prediction of metastases

Radiomic features can also predict the metastatic stage of tumors. For example, many radiomic features were identified as predictors of distant metastasis of lung adenocarcinoma in a study by Coroller et al.[23]. They concluded that radiomic features may be useful in identifying patients at high risk of developing distant metastases, guiding clinicians in choosing the most effective treatment for individual patients.

Prediction of physiological events

Another possible application of radiomics analysis is the prediction of physiological events. Indeed, radiomics can be applied for the characterization and investigation of complex physiological events such as brain activity, which is usually studied with specific imaging techniques such as functional magnetic resonance "fMRI"[3].

1.5 Principal Component Analysis

Principal component analysis (PCA) is a technique to reduce data dimensionality. It replaces the n original variables by a smaller number, q, of linear combinations, called principal components, of the original variables. The application areas include data compression, image analysis, pattern recognition, regression and classification prediction[24]. The most common definition of PCA, due to Hotelling, states that for a set of observed data vectors $\{\mathbf{t}_n\}$, $n \in \{1, \ldots, N\}$, the q principal axes $\mathbf{w}_j$, $j \in \{1, \ldots, q\}$ are those orthonormal axes onto which the retained variance under projection is maximal. The vectors $\mathbf{w}_j$ are given by the q dominant eigenvectors (i.e. those with the largest associated eigenvalues λ) of the sample covariance matrix $\mathbf{S} = \sum_n (\mathbf{t_n} - \bar{\mathbf{t}})(\mathbf{t_n} - \bar{\mathbf{t}})^T / N$ such that $\mathbf{S}\mathbf{w}_j = \lambda_j \mathbf{w_j}$ and where $\bar{\mathbf{t}}$ is the sample mean. The vector $\mathbf{x}_n = \mathbf{W^T}(\mathbf{t_n} - \bar{\mathbf{t}})$, where $\mathbf{W} = (\mathbf{w}_1, \mathbf{w}_2 \ldots \mathbf{w}_j)$ is thus a q-dimensional reduced representation of the observed vector $\mathbf{t}_n$ [24]. As we have mentioned, λ_j is just the variance of each new feature dimension. How to choose an appropriate q depends on the Variance Contribution Rate $\alpha_j = \lambda_j / \sum_j \lambda_j$. This can be determined by looking at the cumulative explained variance ratio as a function of the number of components as shown in figure 1.5.

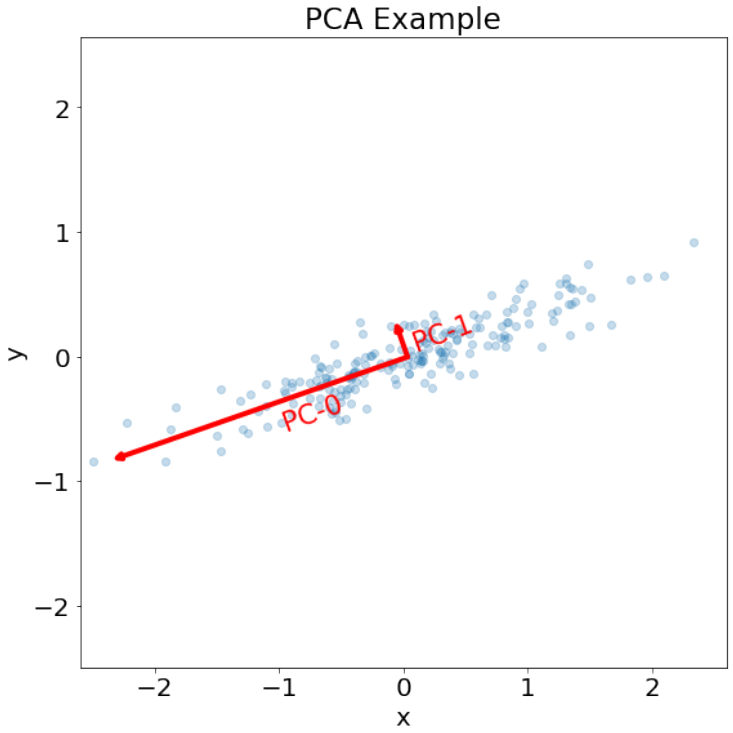

Figure 1.4: Example of Principal Component Analysis. The red vectors represent the principal axes of the data, and the length of the vector is an indication of the variance of the data when projected onto that axis.

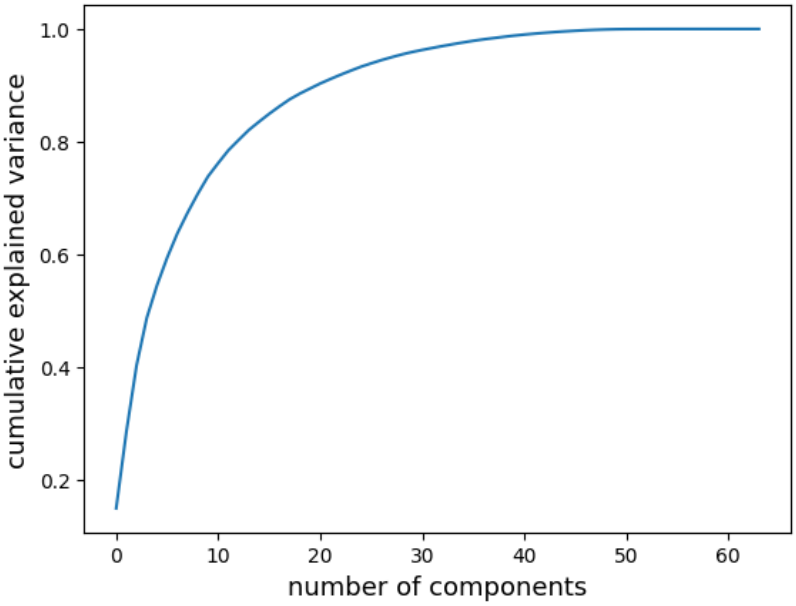

Figure 1.5: Example of cumulative explained variance ratio as a function of the number of components. This curve quantifies how much of the total variance is contained within the first N components. We can see that the first 10 components contain approximately 75% of the total variance, while to the reach the 100% you need around 50 components.

1.6 Support Vector Classifiers

Support Vector Classifiers (SVCs) are a subclass of Support Vector Machines (SVMs) that are a set of supervised learning methods (i.e. requires training data) used for purposes such as classification and regression. A Support Vector Machine constructs a hyper-plane or set of hyper-planes in a high or infinite dimensional space. Intuitively, a good separation is achieved by the hyper-plane that has the largest distance to the nearest training data points of any class (functional margin), since in general the larger the margin the lower the generalization error of the classifier[25, 26].
For a SVC, mathematically, given training vectors $\mathbf{x}_i \in \mathbb{R}^p$, $i \in \{1, \ldots, n\}$, and a vector $\mathbf{y} \in \{-1, 1\}^n$ (or $\{0, 1\}^n$), the goal is to find $\mathbf{w} \in \mathbb{R}^p$ and $b \in \mathbb{R}$ such that the prediction given by $sign(\mathbf{w^T}\mathbf{\Phi}(\mathbf{x}) + b)$ is correct for most samples [25].
The function $\Phi(x)$ provides a convenient way of extending the analysis from the input space to a non-linear feature space by using a high-dimensional mapping. Finding a linear separating hyperplane in this feature space is equivalent to find a non linear decision boundary in the input space[27].

A SVC solves a primal problem and a dual problem. The primal:

$$\min_{w,b,\zeta} \frac{1}{2}(\mathbf{w}^T\mathbf{w}) + C\sum_{i=1}^{n}\zeta_i \tag{1.14}$$

$$\text{under the following conditions: } \begin{cases} y_i \cdot (\mathbf{w}^T\mathbf{\Phi}(\mathbf{x_i}) + b) \geq 1 - \zeta_i, \\ \zeta_i \geq 0 \quad i = 1, \ldots, n \end{cases}$$

From an intuitive point of view, the minimization $(\mathbf{w}^T\mathbf{w})$ corresponds to maximize the functional margin while incurring a penalty when a sample is misclassified or within the margin boundary. The penalty term C controls the strength of this penalty, and acts as an inverse regularization parameter[25]. Ideally, the term ζ_i should be 0 for a perfect prediction but real data are usually not always perfectly separable with a hyperplane, so some samples will be at a distance ζ_i from their correct margin boundary.
The dual problem instead:

$$\min_{a} \frac{1}{2}(\mathbf{a}^T\mathbf{Q}\mathbf{a}) - \mathbf{e}^T\mathbf{a} \tag{1.15}$$

$$\text{under the following conditions: } \begin{cases} \mathbf{y}^T\mathbf{a} = 0, \\ 0 \leq a_i \leq C \quad i = 1, \ldots, n \end{cases}$$

where $\mathbf{e}$ is the vector of all ones, $\mathbf{Q}$ is a $n \times n$ matrix: $Q_{ij} \equiv y_i y_j K(\mathbf{x_i}, \mathbf{x_j})$ where $K(\mathbf{x_i}, \mathbf{x_j}) = \mathbf{\Phi}(\mathbf{x_i})^T\mathbf{\Phi}(\mathbf{x_j})$ is the so called *kernel*. The a_i terms are called *dual coefficients*, and they are upper-bounded by C.

Once these problems are solved, for a sample $\mathbf{z}$, the decision function is given by:

$$\sum_{i \in SV} y_i a_i \mathbf{K}(\mathbf{x_i}, \mathbf{z}) + b \tag{1.16}$$

where the sum is over the supported vectors (SV) that are the samples that lie within the margin because the dual coefficients a_i are zero for the other samples[25].

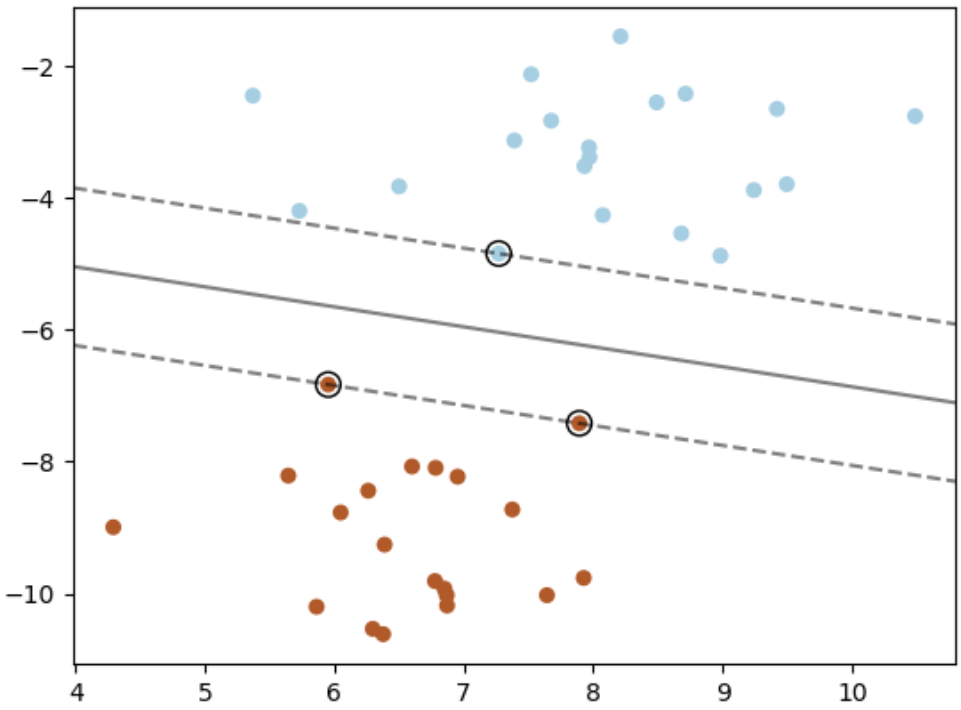

Figure 1.6: Classification made by a Support Vector Classifier (SVC) for a linearly separable problem. The gray line represents the line of the separation hyper-plane. The three samples on the margin boundaries (dashed lines) are called *support vectors*. From Skikit-learn[25]

1.7 Performance Evaluation Metrics

Performance evaluation metrics refer to a series of methods used to measure the performance of the developed pipeline. This section is aimed at their definition.

Precision

The precision is the ratio between true positives tp and the sum between true positives tp with false positives fp. The precision is intuitively the ability of the classifier to label not as positive a sample that is negative. The best value is 1 and the worst value is 0.

$$Precision = \frac{tp}{tp + fp}$$

Recall

The recall is the ratio between true positives tp and the sum between true positives tp with false negatives fn. The recall is intuitively the ability of the classifier to find all the positive samples. The best value is 1 and the worst value is 0.

$$Recall = \frac{tp}{tp + fn}$$

Dice Similarity Coefficient

The Dice Similarity Coefficient or Dice-Sørensen coefficient (DSC) is a widely used metric in computer vision community to calculate the similarity between two images[28]. Given two sets, X and Y, it is defined as:

$$DSC = 2 \cdot \frac{\mid X \cap Y \mid}{\mid X \mid + \mid Y \mid}$$

where $\mid X \mid$ and $\mid Y \mid$ are the cardinalities of the two sets (i.e. the number of elements in each set).
Using the definition of true positive tp, false positive fp, and false negative fn, it can be written as:

$$DSC = \frac{2tp}{2tp + fp + fn}$$

The best value is 1, meaning a perfect overlap between the two sets (i.e. ground-truth and predicted), while the worst value is 0 meaning no overlap.
The Dice Similarity Coefficient is also known as F1-Score, and using the definition of *precision* and *recall* can be written as:

$$F1 = 2 \cdot \frac{precision \times recall}{precision + recall}$$

Matthews correlation coefficient

The Matthews correlation coefficient (MCC) is a measure of the quality of binary (two-class) classifications. The MCC is a correlation coefficient ranging between -1 and +1. A coefficient of +1 represents a perfect prediction, 0 is an average random prediction, and -1 is an inverse prediction. The MCC, using the definition of true positive tp, false positive fp, true negatives tn, and false negative fn,can be calculated by:

$$MCC = \frac{tp \times tn - fp \times fn}{\sqrt{(tp + fp)(tp + fn)(tn + fp)(tn + fn)}}$$

Confusion Matrix

The Confusion Matrix is a specific table that allows visualization of the performance of an algorithm. Each row of the matrix represents the instances in an actual class while each column represents the instances in a predicted class. The diagonal elements represent the number of points for which the predicted label is equal to the true label, while off-diagonal elements are those that are mislabeled by the classifier.

Receiver Operating Characteristic curve

The Receiver Operating Characteristic curve, or ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR). By analyzing the ROC curves, the ability of the classifier to discern, for example, between a set A and B of population, is assessed, calculating the area under the ROC curve (AUC). The AUC value, between 0 and 1, is equivalent to the probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one. The ROC curves pass through the points (0,0) and (1,1), (0,1) and (1,1) represent two limit curves:

- The first cuts the graph at 45 degrees, representing the case of the random classifier ("no benefit" line), with the AUC equal to 0.5.
- The second curve is represented by the segment that from the origin rises to the point (0,1) and by the one that connects the point (0,1) to (1,1), having the AUC equal to 1, meaning a perfect classifier.

Chapter 2

Pipeline

The aim of this work of thesis is the implementation of an automated pipeline to predict the response to neoadjuvant chemo-radiotherapy of patients affected by colorectal cancer by using radiomic features.
The work was developed and validated on MRI scans provided by IRCCS Sant'Orsola-Malpighi Polyclinic.

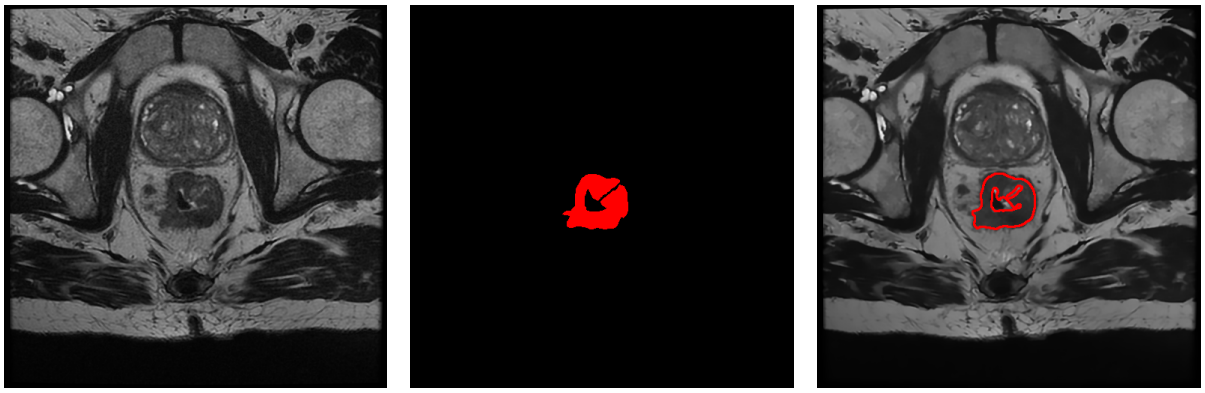

Figure 2.1: *From left to right:* the original MR image of a patient affected by colorectal cancer; the segmentation mask of the tumor region obtained from manual annotations made by expert radiologists; the same image with identified tumor area.

The starting point is the MRI scans. Firstly, I started with the visualization and the pre-processing of the scans.
The work was then split into two main frameworks: *segmentation* and *radiomics*. The basic idea was to train a Convolutional Neural Network like U-Net, for the segmentation of the images. The training process was supervised exploiting segmentation masks of the tumor regions. The masks come from manual annotations made by expert radiologists. Once trained, the Convolutional Neural Network model was applied to perform the seg-

mentation of the images to get the tumor regions for each patient.
The next step is the extraction of radiomic features. They are extracted from the obtained segmented regions. The features are then analyzed to implement a model for the prediction of response. The prediction is based on the Tumor Regression Grade (TRG), which gives an evaluation of how much the chemo-radiotherapy was effective.
The workflow of the developed pipeline can be seen in Figure 2.2

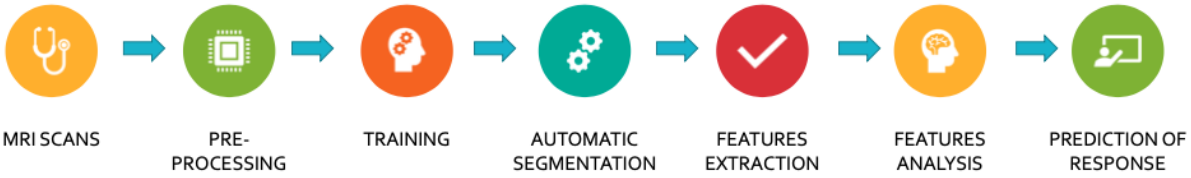

Figure 2.2: Workflow of the developed pipeline. The starting point is the MRI scans. Then, the images are pre-processed for the training of a Convolutional Neural Network model. After the segmentation of the images, radiomic features are extracted from the obtained segmented regions. They are analyzed to implement a model for the prediction of response based on the Tumor Regression Grade (TRG).

Obviously, the final pipeline structure does not involve a learning process and a feature analysis step since the models are already implemented. As consequence, the final structure of the pipeline looks like in the following Figure 2.3

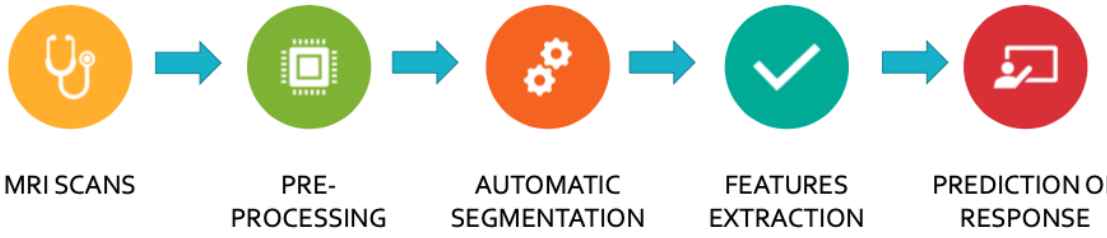

Figure 2.3: Final pipeline structure. The MRI scans are pre-processed with the non-local means algorithm and gamma correction to remove possible noise sources and improve the image contrast, respectively. Then, the segmentation is achieved by the trained CNN model. After the segmentation, radiomic features are extracted. Finally, the prediction of response is obtained by the implemented model.

2.1 Pipeline Workflow

As we have seen, the pipeline workflow is divided into many steps, each one performing a different task. In this section, I will describe in details how each step of the pipeline is achieved. This section involves only a description. The implementation will be discussed in the next one.

2.1.1 Pre-processing

This preliminary step is performed before the training and segmentation process. It involves the application of the non-local means algorithm to remove possible noise sources and a gamma correction to improve the image contrast.
First of all, the MRI scans are acquired from the dataset, slice by slice, in DICOM format and converted into pixel arrays with the original pixel depth in order to preserve all the original information. Then, to have pixel data in the range [0, 1], images are normalized and rescaled to binary floating point 32-bit. This is required to work with TensorFlow[29] and Keras' [30] functions that deal with the segmentation process, as you will see in the implementation section.
The images are then filtered by using the non-local means algorithm. In Figure 2.4, you can see the an example of application of the non-local means algorithm on the same MR image of a patient affected by colorectal cancer. The original image on the *left*, is affected by noise while the filtered one, on the *right*, results smoothed without significant detail loss. In fact, one drawback of smoothing filters is the blurring of small details but in this case, they are preserved. This behavior is reflected in the image histogram. As you can see in Figure 2.5, the original image histogram (red) is affected by great fluctuations of pixel intensity, that highlight the presence of noise. Instead, the denoised image histogram (blue) results in a smoothed histogram, preserving the original shape.
After performing the non-local means algorithm, the image contrast is enhanced by a gamma correction. In this case, the γ parameter has been set to a value of $1/1.5$ to make darker regions lighter. An example of the performed gamma correction can be seen in Figure 2.6. Moreover, the rise of contrast can be noticed on the gamma-corrected image histogram, shown in Figure 2.7.
In summary, the preprocessing steps are:

- normalization and rescaling
- denoising
- gamma correction

A comparison of the image before and after the pre-processing can be seen in Figure 2.8.

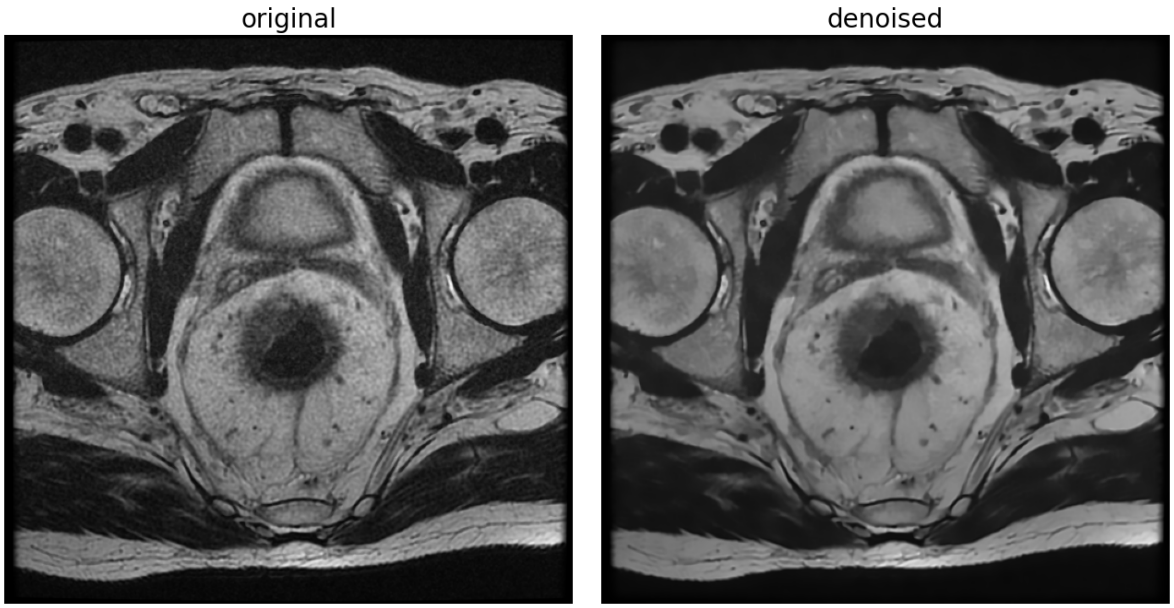

Figure 2.4: *Left)* Original MR image of a patient affected by colorectal cancer. *Right)* The same image after non-local mean algorithm.

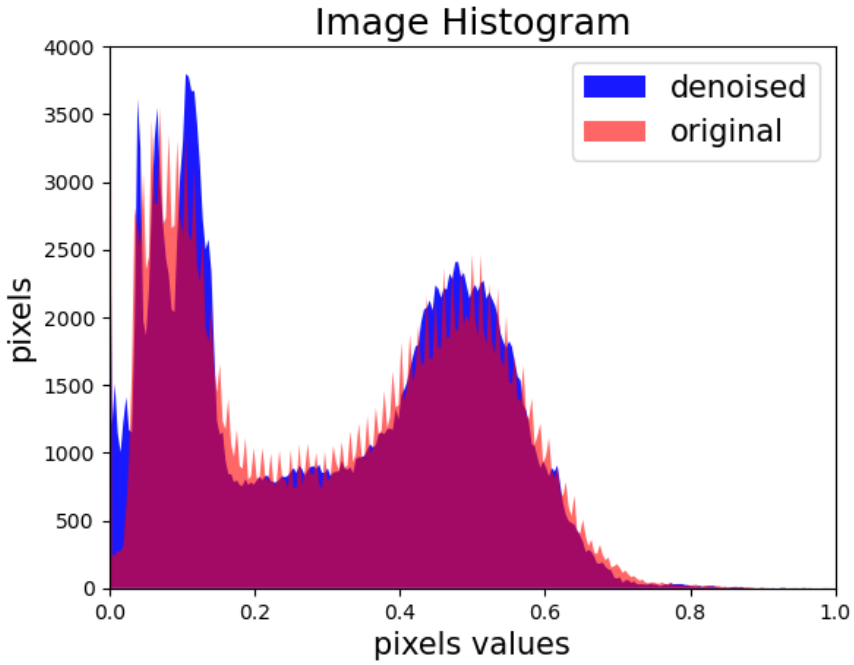

Figure 2.5: Original vs denoised image histogram. The original image histogram (red) is affected by great fluctuations of pixel intensity, highlighting the presence of noise. The denoised one (blue) results in a smoothed histogram, preserving the original shape.

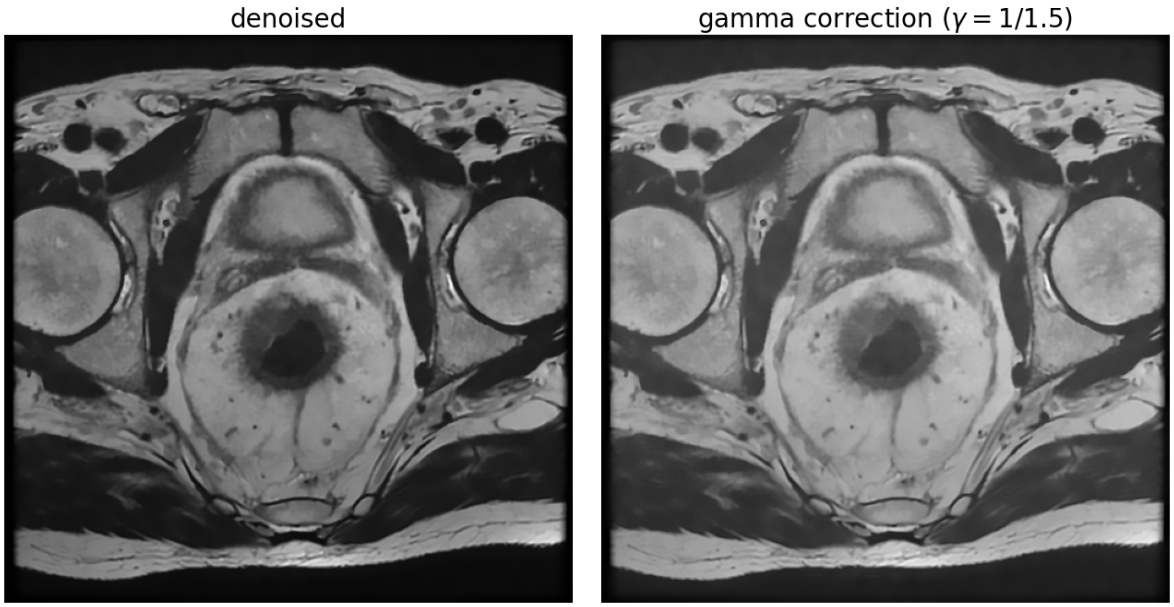

Figure 2.6: *Left)* denoised MR image of a patient affected by colorectal cancer. *Right)* The same image after gamma correction.

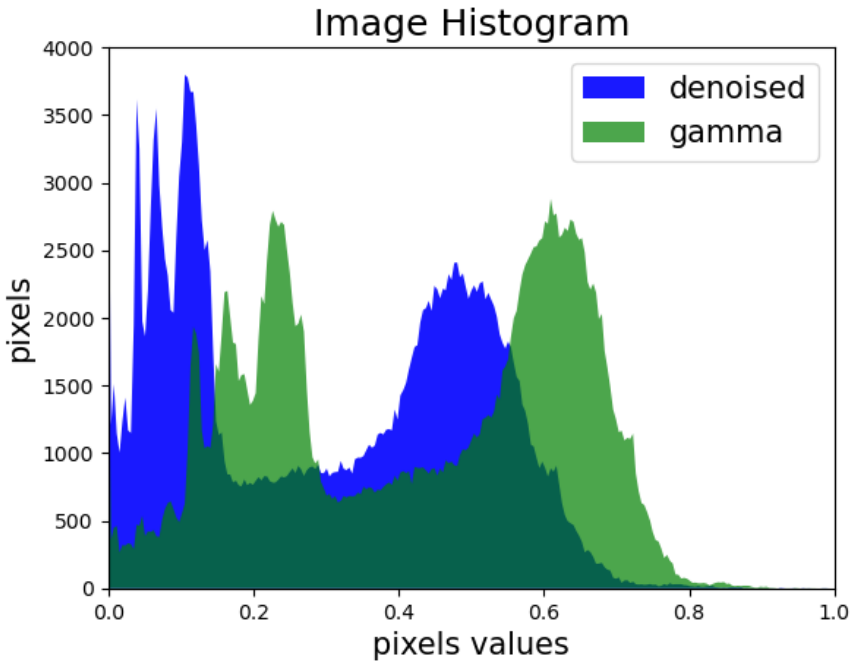

Figure 2.7: Denoised vs gamma-corrected image histogram. The gamma-corrected image histogram (green) results in a right-shifted histogram.

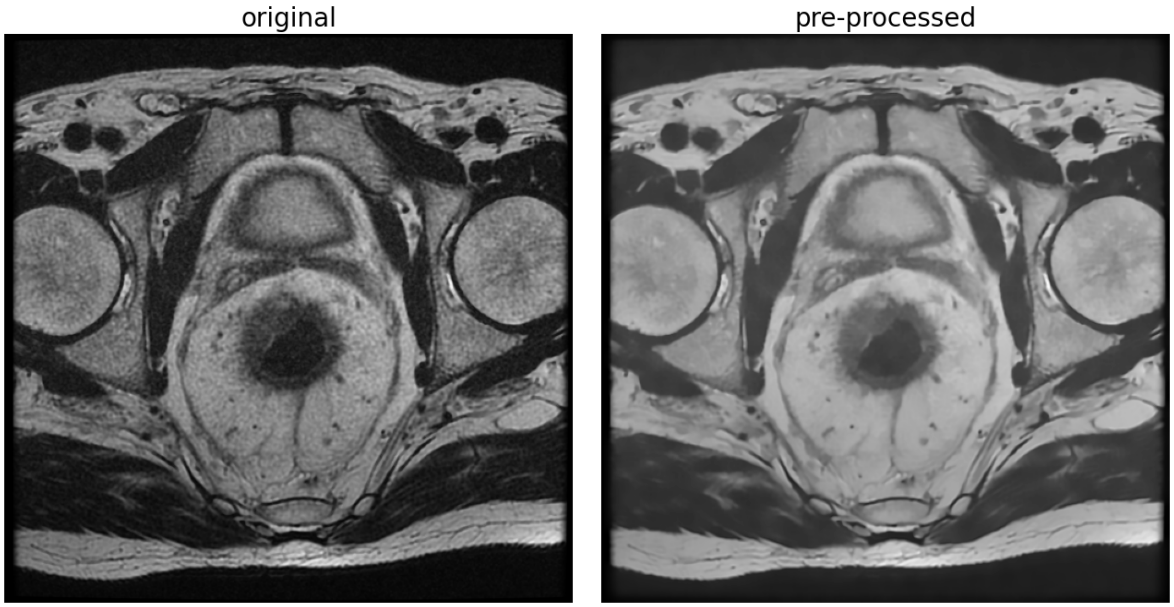

Figure 2.8: *Left)* Original MR image of a patient affected by colorectal cancer. *Right)* The same image after the pre-processing steps.

2.1.2 Training

This step is one of the most important since the Convolutional Neural Network (CNN) model coming from the training process will be used for the segmentation of the images. Before proceeding with the training, the MRI scans and the medical annotation of each patient were manually checked since for some of them there was misregistration between the image and the medical annotation (i.e. the medical annotation did not correspond to the correct slice). The medical annotations consist of a set of (x, y) points that border the tumor area on the image. Pixels inside the bordered area were labeled with a value of 255 while the other ones with 0.
The images were then selected and stored in the proper directories, one for the images in DICOM format and one for the ground-truth images stored as 8-bit unsigned integers images.
The training was performed by using a custom data generator which was responsible for providing the right input and ground-truth images, pre-processing them and splitting the images into training and validation sets.
Some data augmentation was also performed on the training set of data, to overcome the lack of data and to reduce overfitting. It consists in adding slightly modified copies of already existing data, using geometrical transformations such as flipping and rotations of the images. In this particular case, the images were randomly horizontal or vertical flipped.

In summary, the custom data generator provides the following steps:

- read the input images and ground-truth images from the relative directories
- shuffle and split data into training and validation sets
- pre-process the input images following the steps described in the previous subsection
- normalize and rescale the images
- zip the input images with the correct ground-truth images
- perform data-augmentation on training set

An example of input and ground-truth images from the training set is shown in Figure 2.9. Instaed, An example of input and ground-truth images from the validation set is shown in Figure 2.10. In this case no data augmentation is performed.
The network architecture used for this project is a U-Net-like structure made by a contraction and expansion path. The difference between the original architecture and the one used in this project consists of the use of a so-called *backbone* which refers to the feature extracting network. Just to clarify this notion, in Figure 1.3, the feature maps (blue boxes) are extracted by a series of Convolutional, ReLu, and Max-pooling layers (denoted by the arrows). However, one can use various combinations of different layers to extract feature maps. So, the combination of layers can create new networks and architectures. In this case the *backbone* consists of an architecture called *EfficientNetb0*[31].
The loss function is the combination between Dice loss with the binary focal loss.
The algorithm used to minimize the loss function consists of the default *Adam* (Adaptive Moment Estimation) optimizer provided by TensorFlow.
The metric function used to judge the performance during the training of the model was the Dice Similarity Coefficient (DSC).

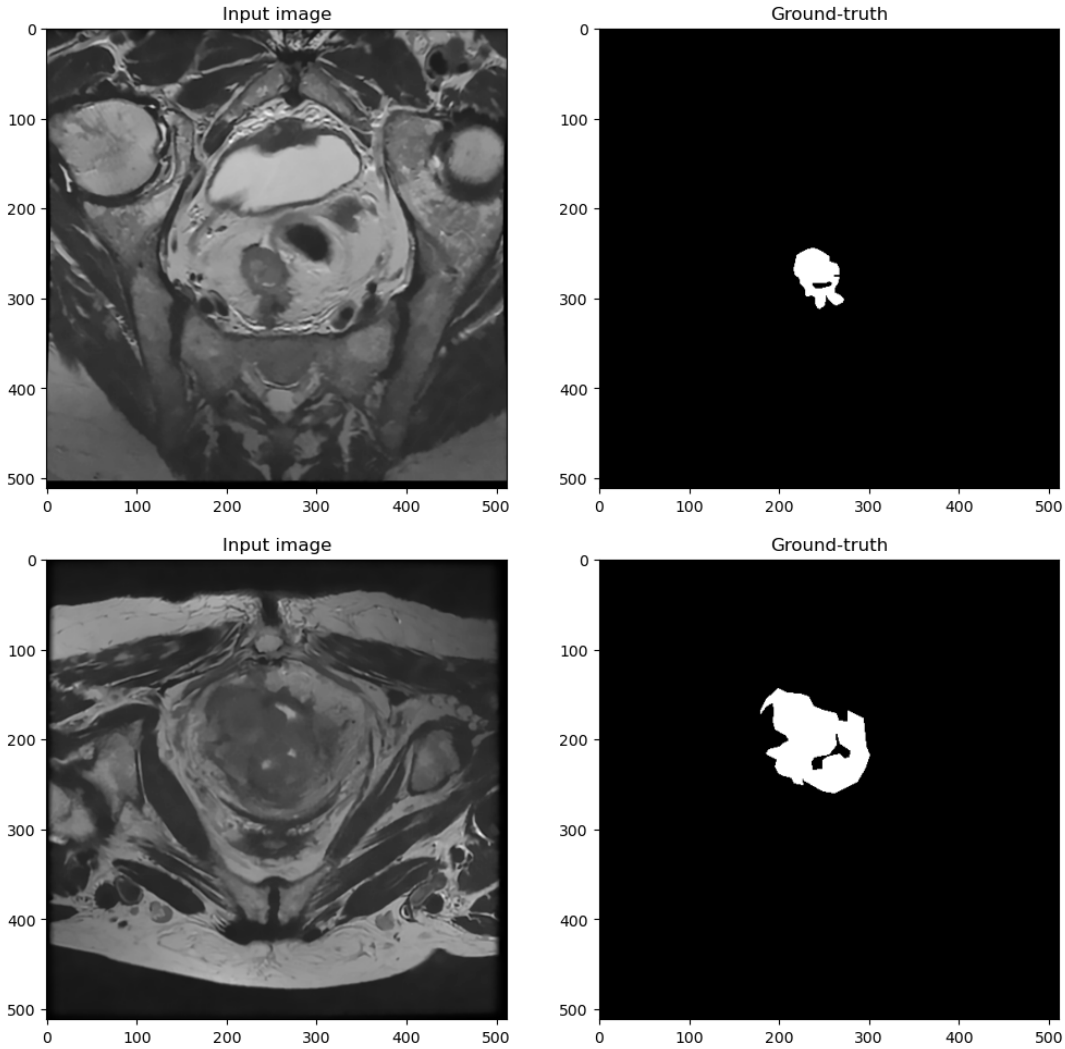

Figure 2.9: Example of input and ground-truth images from the training set. The white area on the ground-truth images represents the tumor area. As you can see, the images are randomly horizontally or vertically flipped.

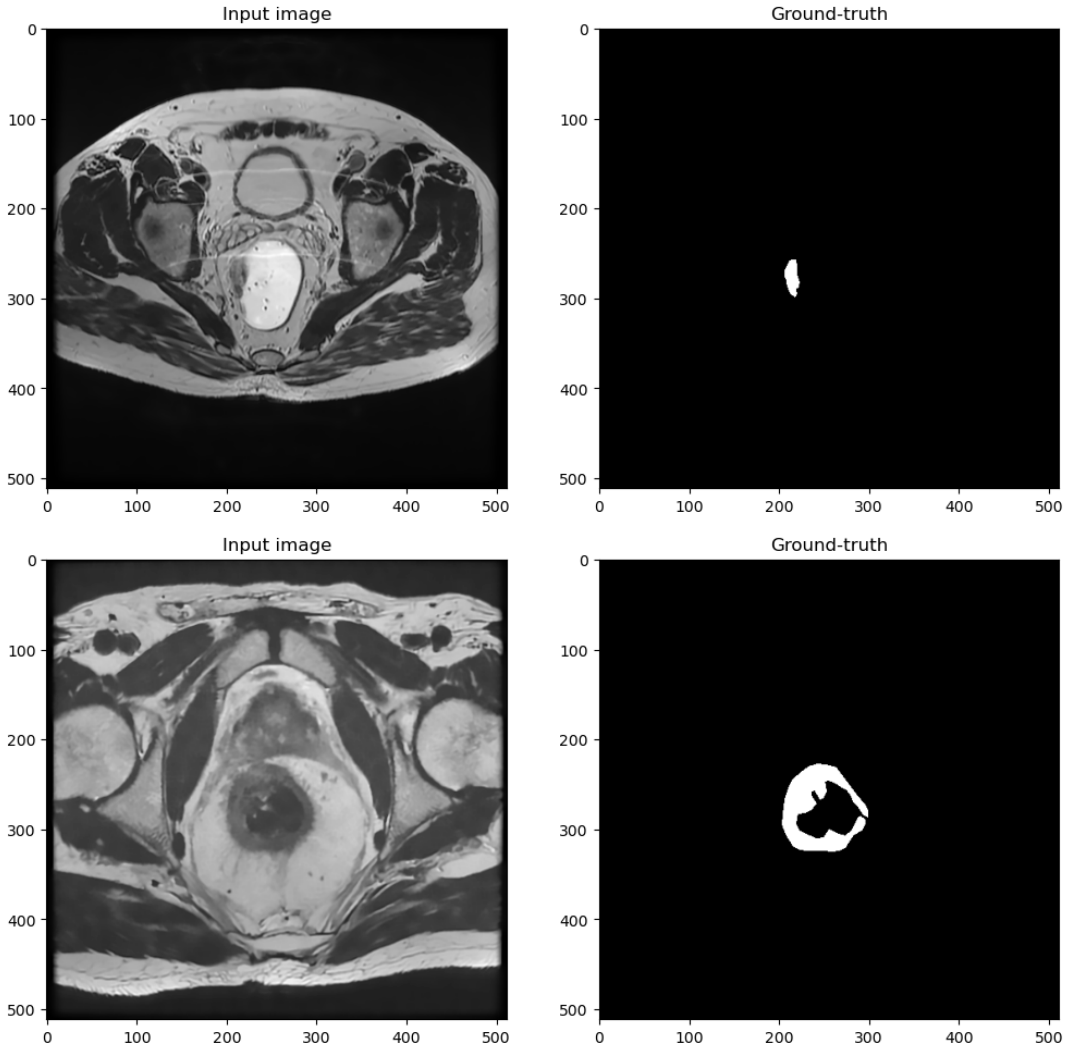

Figure 2.10: Example of input and ground-truth images from the validation set.The white area on the ground-truth images represents the tumor area. As you can see, no data augmentation was performed on this set

2.1.3 Segmentation

Once trained, the CNN model was used for the segmentation of the MRI scans of each patient. Before the segmentation, the scans are pre-processed as previously described. The segmentation is done slice by slice, for each patient, by using the trained CNN model to obtain the predicted tumor area. The prediction values range between 0 and 1. Then, using OpenCV[32] functions it is possible to obtain a segmented area like the one in Figure 2.11, where the red contour represents the border of the predicted tumor area.

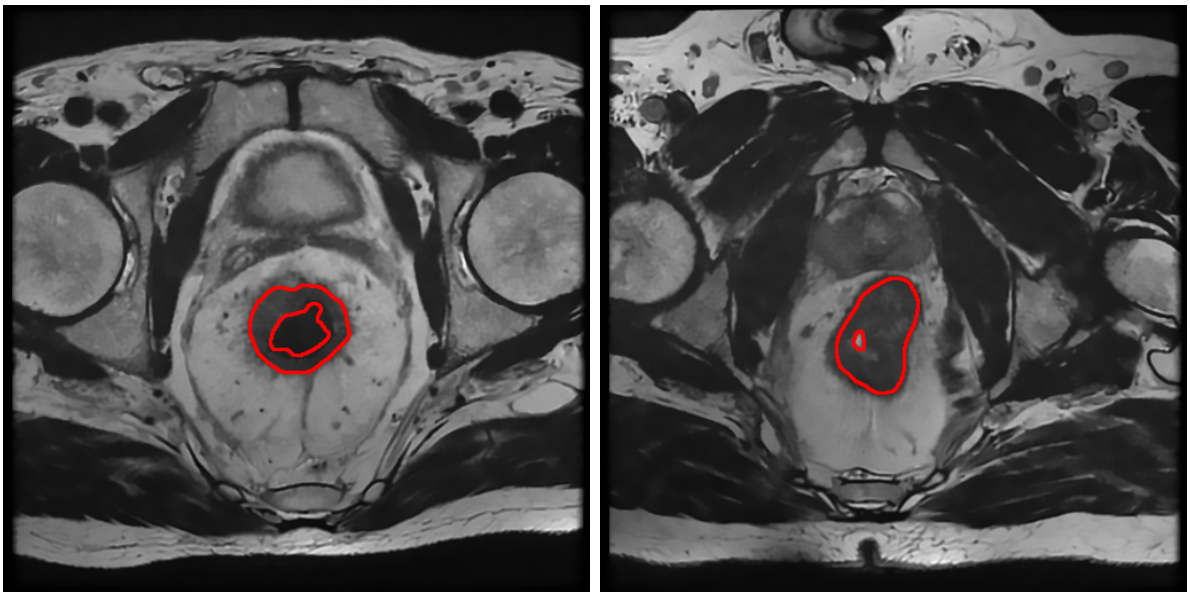

Figure 2.11: Images of colorectal cancer with identified tumor areas from two different patients. The red contour represents the border of the predicted tumor area.

2.1.4 Features Extraction

This step consists of the extraction of the radiomic features from the images. For this purpose, the original MRI scans of each patient and the segmented ones are saved, as 3D images in a particular format (.nrrd) required by PYRADIOMICS library [33] to extract features from the segmented areas of every single slice of the patient.
From each patient, a total of 100 radiomic features were extracted. The extraction settings were stored in a `Params.yaml` file required by PYRADIOMICS. The data were then stored in a data frame containing 100 columns (one for each feature) and 48 rows (one for each patient) ready for the analysis step.

2.1.5 Features Analysis

This step came immediately after the features extraction. Firstly, I have been looking for some correlation between features to reduce as much as possible the number of irrelevant ones. As you can see from Figure2.13, showing the features correlation heatmap, there is the presence of highly correlated (and anti-correlated) features. The names of the features are set by PYRADIOMICS. In particular, the name before the underscore (_) highlights the feature class. For example `glcm` stands for *Gray Level Co-occurrence Matrix (GLCM)* or `firstorder` means that the feature belongs to that class.
Principal Component Analysis (PCA) was performed to reduce the number of features on the standardized data frame containing the extracted 100 features for each patient. Standardization was performed to have a comparable scale between features. PCA resulted in 6 Principal Components (PCs) of the extracted features, corresponding to the 90% of the total variance. The importance of each feature is reflected by the magnitude of the corresponding values in the eigenvectors (higher magnitude — higher importance). To sum up, we look at the absolute values of the eigenvectors' components corresponding to the k largest eigenvalues. The larger they are these absolute values, the more a specific feature contributes to that principal component. In figure 2.12, you can see for each principal component (on the right), which is the original feature (on the left) that contributes more to the relative principal component.
In this case:

- For PC-0: glcm_jointEntropy
- For PC-1: gldm_DependenceNonUniformity
- For PC-2: gldm_LargeDependenceLowGrayLevelEmphasis
- For PC-3: firstorder_Kurtosis
- For PC-4: glcm_ldn
- For PC-5: shape_Flatness

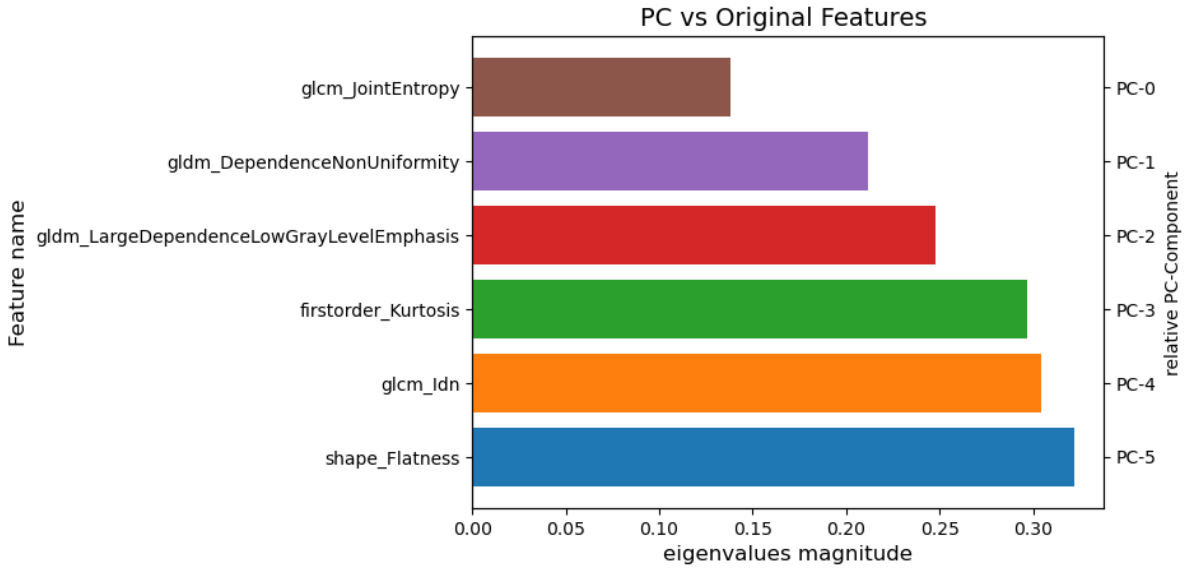

Figure 2.12: For each principal component (on the *right*), you can see which is the original feature (on the *left*) that contributes more to the relative principal component.

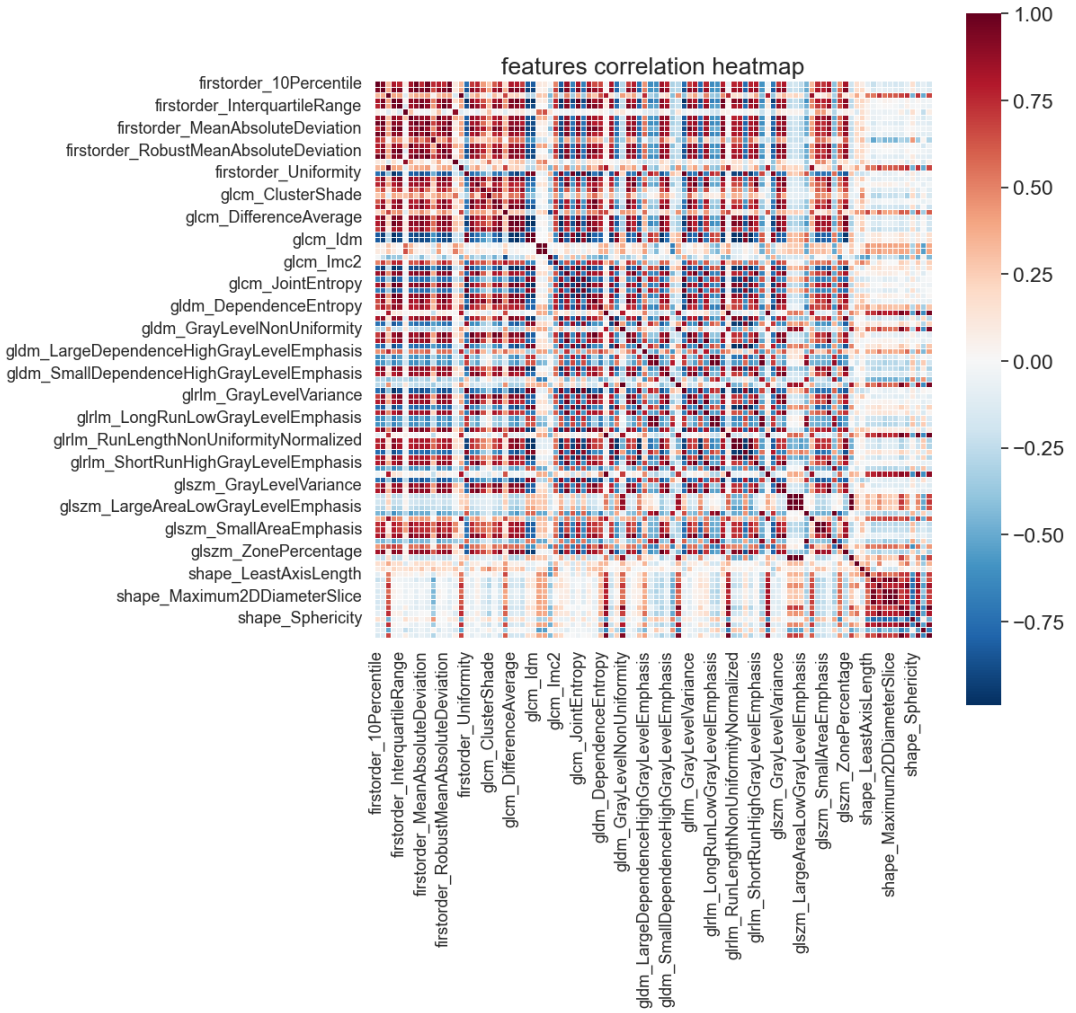

Figure 2.13: Features correlation heatmap. The red color indicates a correlation, the white color indicates no correlation, and the blue color indicates anti-correlation between features. *Note:* not every feature name has been plotted since 100 features caused the font size to be extremely small.

2.1.6 Prediction of Response

The prediction of response is based on the Tumor Regression Grade (TRG) which indicates the degree of response to neoadjuvant therapy[3]. TRG ranges from 0 to 5, resulting in a different response. Lower is the TRG higher the response.
To obtain the prediction of response, a custom Support Vector Classifier (SVC) model has been implemented. The classification is made after the standardization and PCA performed as described before.
The performed steps are:

- Standardization of the data
- PCA
- Classification

Unfortunately, not for every patient, the TRG was registered in the clinical database provided by the IRCCS Sant'Orsola-Malpighi Policlinic, so patients without it were excluded from the analysis. Moreover, since the lack of data TRG values were binarized into two main classes: 0 and 1. Class 0 means a complete response to the neoadjuvant chemo-radiotherapy (TRG values $\in$ [0, 1]) while class 1 means a moderate response (TRG values $\in$ [2, 3]) . In Figure 2.14 you can see the distribution of the two classes.

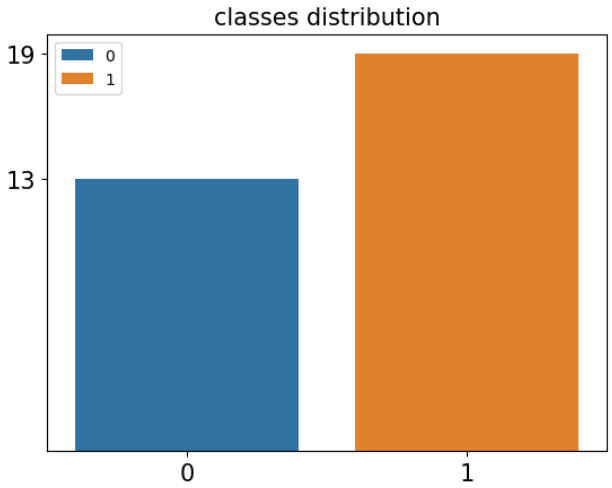

Figure 2.14: Distribution of the classes. Class 0 means a complete response to the neoadjuvant chemo-radiotherapy (TRG values $\in$ [0, 1]) while class 1 means a moderate response (TRG values $\in$ [2, 3])

2.2 Implementation

I have implemented the pipeline by using python, which is an high level object oriented programming language. The supported python versions are: 3.6|3.7|3.8|3.9. To perform operations on images (image filtering, input-output operations, etc...), I've used different libraries depending on the specific purpose, as in some cases anticipated in the previous section.
The whole code is open-source and available on GitHub [34] and the relative documentation, made by using Sphinx [35], is available at: `https://img-segm.readthedocs.io/en/latest/?badge=latest`. The installation is managed by setup.py, which also provides the full list of dependencies. The pipeline installation is tested on MacOS (base environment) and on Linux by using the TravisCI host.
The pipeline implementation provides also modules that allow one to load, visualize, processing the DICOM series and to train a U-Net model and provides scripts to handle DICOM series from command line.
The detailed description of each module and script is available on GitHub. Once you have installed it, you can start to segment the images directly from your shell, passing as input the path of the directory containing the DICOM series, to obtain the prediction of response.
There are different output options that will be described in the *Results* chapter.
This section will be aimed at the implementation of the main steps of the pipeline.

2.2.1 Pre-processing

As we said in the previous section, the main steps of the pre-processing consist of the application of the non local-means algorithm and gamma correction.
Before the processing operations, for each patient, the images needed to be read from the DICOM files as an array of pixels in order to apply the pre-processing functions. This was done by the function `get_slices`:

```
import numpy as np
import pyradiomics

def read_slices(filename):

    _, ext = filename.split(".")

    if ext != "dcm":
        raise ValueError("Input filename must be a DICOM file")

    pix_arr = pydicom.dcmread(filename).pixel_array

    return pix_arr

```

```
15 def get_slices(dir_path)
16
17     files = glob.glob(dir_path + "/*.dcm")
18
19     # ordering as istance number
20     z = [float(pydicom.read_file(f, force=True).get(
21         "InstanceNumber", "0") - 1) for f in files]
22     order = np.argsort(z)
23     files = np.asarray(files)[order]
24
25     slices = [read_slices(f) for f in files]
26     slices = np.asarray(slices)
27
28     return slices
```

Listing 2.1: `get_slices` implementation

For this purpose, I used the NUMPY [36] and PYDICOM [37] libraries. In particular, PYDICOM provided functions to read DICOM files as pixel arrays and to access the *InstanceNumber* (the current slice number stored in the header), while NUMPY provided functions to sort the slices by the *InstanceNumber* and get the images as an array of shape: (depth, height, width) where depth is the number of slices and (height, width) the image size.
Once the images have been obtained, I could perform the following steps :

- normalization and rescaling
- denoising
- gamma correction

For this purpose, I used the OPENCV [32] and SCIKIT-IMAGE [14] libraries:

```
1 import cv2
2 from skimage.restoration import denoise_nl_means, estimate_sigma
3
4 def rescale(img):
5     rescaled = cv2.normalize(img, dst=None, alpha=0, beta=1, norm_type=
    cv2.NORM_MINMAX, dtype=cv2.CV_32F)
6     return rescaled
7
8
9 def denoise(img, alpha=10):
10
11     patch_kw = dict(patch_size=5, patch_distance=6,)
12     sigma_est = np.mean(estimate_sigma(img))
13     denoised = denoise_nl_means(img, h=alpha * sigma_est, sigma=
    sigma_est, fast_mode=True, **patch_kw)
14     return denoised
```

```
15
16
17 def gamma_correction(img, gamma=1.0):
18     igamma = 1.0 / gamma
19     imin, imax = img.min(), img.max()
20
21     img_c = img.copy()
22     img_c = ((img_c - imin) / (imax - imin)) ** igamma
23     img_c = img_c * (imax - imin) + imin
24     return img_c
25
26
27
28 def pre_processing_data(slices, alpha=10):
29
30     imgs = []
31     for layer in range(slices.shape[0]):
32         img = slices[layer, :, :]
33         if slices.shape[1:3] != 512:
34             resized = cv2.resize(img, (512, 512))
35         else:
36             resized = img
37         rescaled = rescale(resized)
38         denoised = denoise(rescaled, alpha)
39         gamma = gamma_correction(denoised)
40         imgs.append(gamma)
41
42     images = [np.expand_dims(im, axis=-1) for im in imgs]
43     images = np.array(images)
44
45     return images
```

Listing 2.2: pre-processing function implementation

In particular: the `rescale` function is used for the normalization and the rescaling of the images to binary floating-point 32-bit; the `denoise` function is used for the denoising process exploiting the Scikit-Image library; the `gamma_correction` function is used for the gamma correction. All these three functions have been put together into `pre_processing_data` with a silent check on the image size, to get a single-shot pre-processing function. The final output is an array, like `slices`, containing the relative pre-processed images.

2.2.2 Training and Segmentation

Training was performed using TensorFlow[29] and Segmentation-models API[38]. The core of the training process is the custom `DataGenerator`, which provides data, split them into training and validation sets, pre-processes the images, and performs data

augmentation on the training set. Data augmentation consists of the random vertical or horizontal flip of the input and label images. The peculiarity consists of the capability of working directly with DICOM input files. The input and label images are taken directly by the `DataGenerator`, providing the relative `source_path` and `label_path`. They are the paths of the directories containing the relative DICOM input and label images.

```
import tensorflow as tf
import pyradiomics

class DataGenerator:

    def __init__(self, batch_size, source_path, label_path, aug=False, seed=123, validation_split=0., subset='training'):


        np.random.seed(seed)
        source_files = sorted(glob.glob(source_path + '/*.dcm'))
        source_files = np.asarray(source_files)


        labels_files = sorted(glob.glob(label_path + '/*.png'))
        labels_files = np.asarray(labels_files)

        assert source_files.size == labels_files.size


        source_files, labels_files = self.randomize(source_files, labels_files)

        idx = np.arange(0, source_files.size)
        np.random.shuffle(idx)

        self._source_trainfiles = source_files[idx[int(source_files.size * validation_split):]]
        self._labels_trainfiles = labels_files[idx[int(labels_files.size * validation_split):]]

        self._source_valfiles = source_files[idx[:int(source_files.size * validation_split)]]
        self._labels_valfiles = labels_files[idx[:int(labels_files.size * validation_split)]]

        self.subset = subset

        if self.subset == 'training':
            self._num_data = self._source_trainfiles.size
        elif self.subset == 'validation':
            self._num_data = self._source_valfiles.size

```

```
38          self.aug = aug
39          self._batch = batch_size
40          self._cbatch = 0
41          self._data, self._label = (None, None)
42
43      @property
44      def num_data(self):
45          return self._num_data
46
47      def randomize(self, source, label):
48
49          random_index = np.arange(0, source.size)
50          np.random.shuffle(random_index)
51          source = source[random_index]
52          label = label[random_index]
53
54          return (source, label)
55
56
57      def resize(self, img, lbl):
58
59          height, width = img.shape[0], img.shape[1]
60
61          if height != 512:
62              img = cv2.resize(img, (512, 512))
63              lbl = cv2.resize(lbl, (512, 512))
64          else:
65              img = img
66              lbl = lbl
67
68          return (img, lbl)
69
70      def random_vflip(self, img, lbl):
71          idx = np.random.uniform(low=0., high=1.)
72          if idx > 0.5:
73              return (cv2.flip(img, 0), cv2.flip(lbl, 0))
74          else:
75              return (img, lbl)
76
77      def random_hflip(self, img, lbl):
78          idx = np.random.uniform(low=0., high=1.)
79          if idx > 0.5:
80              return (cv2.flip(img, 1), cv2.flip(lbl, 1))
81          else:
82              return (img, lbl)
83
84      def rescale(self, img):
85          rescaled = cv2.normalize(img, dst=None, alpha=0, beta=1,
    norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_32F)
```

```
86          return rescaled
87
88      def denoise(self, img):
89
90          patch_kw = dict(patch_size=5, patch_distance=6)
91          sigma_est = np.mean(estimate_sigma(img))
92          denoised = denoise_nl_means(img, h=10 * sigma_est, sigma=
    sigma_est, fast_mode=True, **patch_kw)
93          return denoised
94
95      def gamma_correction(self, img, gamma=1.0):
96          igamma = 1.0 / gamma
97          imin, imax = img.min(), img.max()
98
99          img_c = img.copy()
100         img_c = ((img_c - imin) / (imax - imin)) ** igamma
101         img_c = img_c * (imax - imin) + imin
102         return img_c
103
104     def __iter__(self):
105         self._cbatch = 0
106         return self
107
108     def __next__(self):
109         if self._cbatch + self._batch >= self._num_data:
110             self._cbatch = 0
111             self._source_trainfiles, self._labels_trainfiles = self.
    randomize(self._source_trainfiles, self._labels_trainfiles)
112             self._source_valfiles, self._labels_valfiles = self.
    randomize(self._source_valfiles, self._labels_valfiles)
113
114         if self.subset == 'training':
115             c_sources = self._source_trainfiles[self._cbatch:self.
    _cbatch + self._batch]
116             c_labels = self._labels_trainfiles[self._cbatch:self.
    _cbatch + self._batch]
117         elif self.subset == 'validation':
118             c_sources = self._source_valfiles[self._cbatch:self._cbatch
     + self._batch]
119             c_labels = self._labels_valfiles[self._cbatch:self._cbatch
    + self._batch]
120
121         # load the data
122
123         images = [pydicom.dcmread(f).pixel_array for f in c_sources]
124         labels = [cv2.imread(f, 0) for f in c_labels]
125
126
127         # check size
```

```
128
129         images, labels = zip(*[self.resize(im, lbl) for im, lbl in zip(
    images, labels)])
130
131         # cast
132
133         images = [self.rescale(im) for im in images]
134         labels = [self.rescale(lbl) for lbl in labels]
135
136         # denoise
137         images = [self.denoise(im) for im in images]
138
139         # gamma correction
140         images = [self.gamma_correction(im, gamma=1.5) for im in images
    ]
141
142         if self.aug:
143
144             # random horizontal flip
145             images, labels = zip(*[self.random_hflip(im, lbl) for im,
    lbl in zip(images, labels)])
146
147             # random vertical flip
148             images, labels = zip(*[self.random_vflip(im, lbl) for im,
    lbl in zip(images, labels)])
149
150         images = [im[..., np.newaxis] for im in images]
151         labels = [lbl[..., np.newaxis] for lbl in labels]
152
153         # to numpy
154
155         images = np.array(images)
156         labels = np.array(labels)
157
158         self._cbatch += self._batch
159
160         return (images, labels)
```

Listing 2.3: Custom `DataGenerator` implementation

The model and the losses used for the training, come from SEGMENTATION-MODELS API. In particular, the model consists of a U-net architecture using *efficientnetb0* as backbone encoder. The metric instead, `dice_coeff`, has been implemented by using TENSORFLOW functions.

```
1 import tensorflow as tf
2 import segmentation_models as sm
3
4 BACKBONE = 'efficientnetb0'
```

```
model = sm.Unet(BACKBONE, input_shape=(IMAGE_HEIGHT, IMAGE_WIDTH, 1),
    encoder_weights=None, activation='sigmoid')

optimizer =  'adam'

dice_loss = sm.losses.DiceLoss()
focal_loss = sm.losses.BinaryFocalLoss()
loss = dice_loss + (1 * focal_loss)

def dice_coef(y_true, y_pred):
    intersection = tf.reduce_sum(y_true * y_pred, axis=[1, 2, 3])
    total = tf.reduce_sum(y_true, axis=[1, 2, 3]) + tf.reduce_sum(
    y_pred, axis=[1, 2, 3])
    dice = tf.reduce_mean((2. * intersection + smooth) / (total + 1.))
    return dice

metrics = [ dice_coef ]

model.compile(optimizer=optimizer, loss=loss, metrics=metrics)
```

Listing 2.4: model implementation

The prediction of the images is obtained using the function `predict_images`, providing the `slices` and the trained model. The result is an array, like `slices`, containing the relative prediction for each slice.

```

def predict_images(slices, model, pre_processing=False, t=0.1):

    if pre_processing:
        prep_slices = pre_processing_data(slices)
    else:
        prep_slices = slices

    predicted_slices = np.zeros_like(prep_slices)

    for layer in range(slices.shape[0]):
        img = prep_slices[layer, ...]
        predicted_slices[layer, ...] = model.predict(img[np.newaxis,
    ...])[...]
        predicted_slices[layer, ...] = np.where(predicted_slices[layer,
     ...] <= 0.1, 0, predicted_slices[layer, ...])


    return predicted_slices
```

Listing 2.5: prediction function implementation

2.2.3 Features Extraction and Analysis

For this purpose, I used the PYRADIOMICS library[33] to extract features from each single slice of the patient after storing for each patient input and segmented images as 3D images (.nrrd format) in a dedicated directory, by using SIMPLEITK library [39], required by PYRADIOMICS. From each patient a total of 100 radiomic features were extracted. The extraction settings were stored in a `Params.yaml` file.

```
from radiomics import featureextractor
import pandas as pd
import numpy as np
import SimpleITK as sitk

params  = '../extras/Params.yaml'
extractor = featureextractor.RadiomicsFeatureExtractor(params)

features = {}

for patient in good_patients:

    dirs = glob.glob(src + '/' + patient + '/*_NRRD')


    for directory in dirs:

        original = sitk.ReadImage(directory + "/original.nrrd")
        segmented = sitk.ReadImage(directory + "/segmented.nrrd")

        folder_name = os.path.split(directory)[1]
        fold_prefix = folder_name.split('_')[0]

        features[patient, fold_prefix] = extractor.execute(original,
   segmented)
```

Listing 2.6: Features extraction implementation

Features are stored in a python dictionary. The key is a tuple made by the caseID of the patient and the the name of the examination directory containing the DICOM series. I started to visualize and sort the features for each patient for storing them into a pandas Dataframe `df` made by 100 columns (one for each extracted feature) and the number of rows made by the number of patients.

```

dict_list = list(features)
feature_names = list(sorted(filter ( lambda k: k.startswith("original_"
   ), features[dict_list[0]] )))

print('NUMEBR OF CASE_ID: ', len(dict_list))
print('NUMEBR OF FEATURES: ', len(feature_names))
```

```
print(dict_list)

sorted_list = sorted(dict_list, key=lambda x: int(x[0].replace('BO', ''
    )))


sorted_ID = list(map(lambda x: x[0], sorted_list))

samples = np.zeros((len(sorted_list), len(feature_names)))

for k, case_id in enumerate(sorted_list):
    a = np.array([])
    for feature_name in feature_names:
        a = np.append(a, features[case_id][feature_name])
    samples[k, ...] = a

#for possible NaNs
samples = np.nan_to_num(samples)

samples.shape

df = pd.DataFrame(data=samples, columns=feature_names, index=sorted_ID)
```

Listing 2.7: Features dataframe implementation

Then, I uploaded the database containing clinical data to access the Tumor Regression Grade (TRG) values. Unfortunately, not for every patient data were available, so the patients rows without TRG values were dropped, thus excluded from the analysis. To overcome the lack of data, TRG values were binarized. In order to reduce features, PCA was performed setting `n_components` = .9 (meaning the number of components that gives the 90% of the toal variance) by using SCIKIT-LEARN library[40], exploiting `make_pipeline` to standardize data before the PCA. Then, thanks to the attribute `components_` , that outputs an array of shape [`n_components`, `n_features`], it is possible to get how components are related to the original features.

```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.pipeline import make_pipeline

clinical_df = pd.read_excel('../data/clinical_db.xlsx', sheet_name='
    data', index_col='PatientID')

TRG = clinical_df['TRG']

df = pd.concat([df, TRG], axis=1)
df = df[df['TRG'].notna()]


X = df.drop('TRG', axis=1)
```

```
y = df['TRG'].values
y = np.where(y <=1, 0, 1)

pca = make_pipeline(StandardScaler(), PCA(n_components=.9, svd_solver='
    full'))
pca.fit(X, y)


print(pd.DataFrame(pca.components_,columns=X.columns, index = ['PC-0','
    PC-1', 'PC-2', 'PC-3', 'PC-4', 'PC-5']))
```

Listing 2.8: PCA implementation

2.2.4 Prediction of Response

This step was implemented exploiting `make_pipeline` to get the prediction of response after scaling and performing PCA. The prediction is given by a Support Vector Classifier `SVC`, setting the parameter $C = 100$. Cross validation was performed exploiting `cross_val_predict` and `StratifiedKFold` of the SCIKIT-LEARN library[40]. In particular, to get the `random_state` of the `StratifiedKfold` was set to the median value of the Matthews Correlation coefficient distribution, obtained after 500 simulations. Finally the classification report was provided by using the `classification_report` of SCIKIT-LEARN.

```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import cross_val_predict, StratifiedKFold
from sklearn.svm import SVC
from sklearn.metrics import matthews_corrcoef
from sklearn.metrics import classification_report

X = df.drop('TRG', axis=1)
y = df['TRG'].values
y = np.where(y <=1, 0, 1)

pipeline = make_pipeline(StandardScaler(), PCA(n_components=.9,
    svd_solver='full'), SVC(C=100, probability=True, random_state=0))

n_splits = 10

M_coeffs = []
for i in range(500):
    skf = StratifiedKFold(n_splits=n_splits, shuffle=True, random_state
    =i + 1)
    y_pred = cross_val_predict(pipeline, X, y, cv=skf)
    MCC = matthews_corrcoef(y, y_pred)
```

```
22      #M_coeffs[i] = MCC
23      M_coeffs.append(MCC)
24
25  data = M_coeffs
26  median = np.argsort(data)[len(data)//2]
27  print(median)
28  data[median]
29
30  skf = StratifiedKFold(n_splits=n_splits, shuffle=True, random_state=
        median + 1)
31  y_pred = cross_val_predict(pipeline, X, y, cv=skf)
32
33  target_names = ['class 0', 'class 1']
34  print(classification_report(y, y_pred, target_names=target_names))
```

Listing 2.9: Prediction of response implementation

Chapter 3

Results

The pipeline was developed and applied on the dataset provided by the IRCCS Sant'Orsola-Malpighi Polyclinic, described in the first section.
The performance of the pipeline was evaluated for both the main frameworks: *segmentation* and *prediction of response*.
The possible outputs of the implementation will be also shown in this chapter.

3.1 Dataset Description

The main dataset used for this project was provided by IRCCS Sant'Orsola-Malpighi Polyclinic. It consists of MRI scans from 48 patients affected by colorectal cancer undergoing neoadjuvant radio-chemotherapy, from January 2018 to the end of December 2019. The scans are provided slice by slice in DICOM format. The scans are T2-weighted images resulting in images that highlight fat (low signal) and water (high signal) within the body.
Within the scans, also manual annotations, made by expert clinicians, were provided. The manual annotations consist of sets of (x, y) points that border the tumor area on the relative image.
Clinical data were also provided through a dedicated database. Among the information stored, the clinical database contains the Tumor Regression Grade (TRG) which indicates the degree of response to neoadjuvant therapy.

Property	Value
Number of patients	48
Distribution by sex (M/F)	29/19
Distribution by age (min/median/max)	48/70/89

Table 3.1: Dataset properties.

3.2 Performance

In this section, I will show the results of the accuracy achieved by the implemented pipeline. The segmentation performance has been evaluated since the training process and the obtained results are compared with the literature. A comparison between the results of segmentation and the medical annotation (ground-truth) will be also provided. The accuracy for the prediction of response has been measured by using dedicated metrics.

3.2.1 Segmentation

The metric used for checking the performance of the segmentation results is given by the Dice Similarity Coefficient (DSC) evaluated on the validation set. This is because it was the most used in literature for this purpose and so for make it possible results comparison. The total number of involved patients lowered from 48 to 37 because of misregistration between images and medical annotations for some of them. The training process was performed for 150 epochs[1] on 391 images (training set) and validated on 97 images (validation set). The Training process took almost 7 hours, on the new Apple Silicon M1 Macbook Pro equipped with 8 GB of RAM. The results of the training process can be seen in Figure 3.1. The plots show the curves for the model dice coefficient and the model loss as a function of the epochs. In particular, the blue curve represents the results obtained for the training set of data while the green one represents the results obtained for the validation set.
In Table 3.2, you can see the comparison between the state of the art and the implemented pipeline about colorectal cancer segmentation. It must be said that the literature about MRI colorectal cancer segmentation performed by using CNNs, unlike other topics, is not very wide. However, it comes out that automatic segmentation is quite hard to perform on Magnetic Resonance colorectal cancer images due to different issues: data; medical annotations; loss function. In fact, as showed by Jovana Panic at al.[41] mucinous cases can considerably affect the performances. *Mucinous* consists of a tumor subtype

[1]The term epoch indicates the number of passes of the entire training dataset the machine learning algorithm has completed.

characterized by bright tumoral areas on MRI scans, different from the *adenocarcinomas* characterized by dark tumoral areas. Moreover, the performance can be affected by how medical annotations are made. Trebeschi et al. [6] showed that the DSC of the same model trained using medical annotations made by different experts can give different DSC scores. Another sensitive factor is the loss function. As shown by Yi. Jie Huang et al.[42], depending on the network's architectures and the loss functions you can have different performances.

Trebeschi et al.	**Panic et al.**	**Yi-Jie Huang et al.**	**Xiaoling Pang et al.**	**Implemented pipeline**
DSC= 0.68 DSC= 0.70	DSC = 0.58	DSC = $[0.66 - 0.72]$	DSC = 0.66	**DSC = 0.71**
140 patients	33 patients (5 mucinous cases)	64 cancerous cases	275 patients excluding mucinous cases	**37 patients including some mucinous cases**
Custom Network architecture	Custom Network architecture	Custom network architectures	U-net	**U-net, backbone EfficientNetb0**

Table 3.2: Results comparison between state-of-art and the implemented pipeline about MRI colorectal cancer scans segmentation by using CNNs.

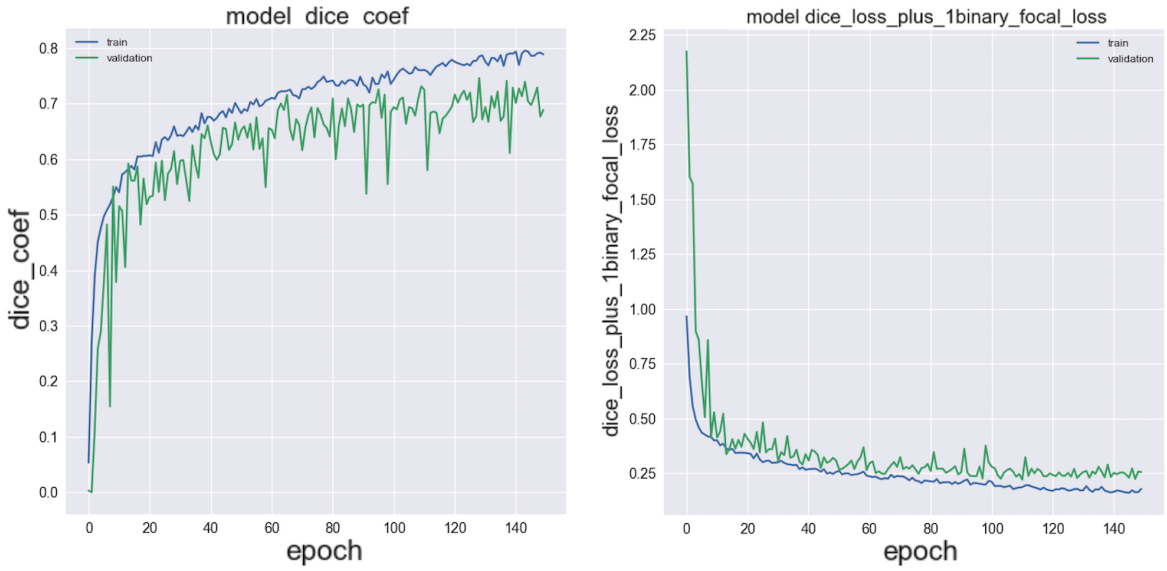

Figure 3.1: Model history plots on training (blue) and validation (green) set. *Left*): DSC as a function of the epochs. *Right*): model loss as a function of the epochs.

Comparison with Manual Annotations

To check the pipeline performances, I have also compared the obtained segmentation with the manual annotation (Ground-truth) made by expert radiologists. In Figure 3.2, you can see the comparison for images belonging to the training set while in Figure 3.3 the comparison for the ones belonging to the validation set. The prediction is plotted as a probability density map between 0. and 1. as shown by the sequential colormap. Both in Figure 3.2 and 3.3, I also included a case of *mucinous* (first row), showing that the model is able to distinguish also this type of tumor, even if the contour is not as precise as the ground-truth one.
Unfortunately, for a few cases, the model failed to segment correctly the right Region of Interest (ROI). Some of this cases are shown in Figure 3.4.
The goodness of the comparison can also be appreciated from the comparison between the ground truth and the prediction over the original image. In Figure 3.5 and 3.6 the images belong to the training set while in Figure 3.7 and 3.8 the images belong to the validation one. Also for this case, the prediction is plotted as a probability density map between 0. and 1. as shown by the sequential colormap.

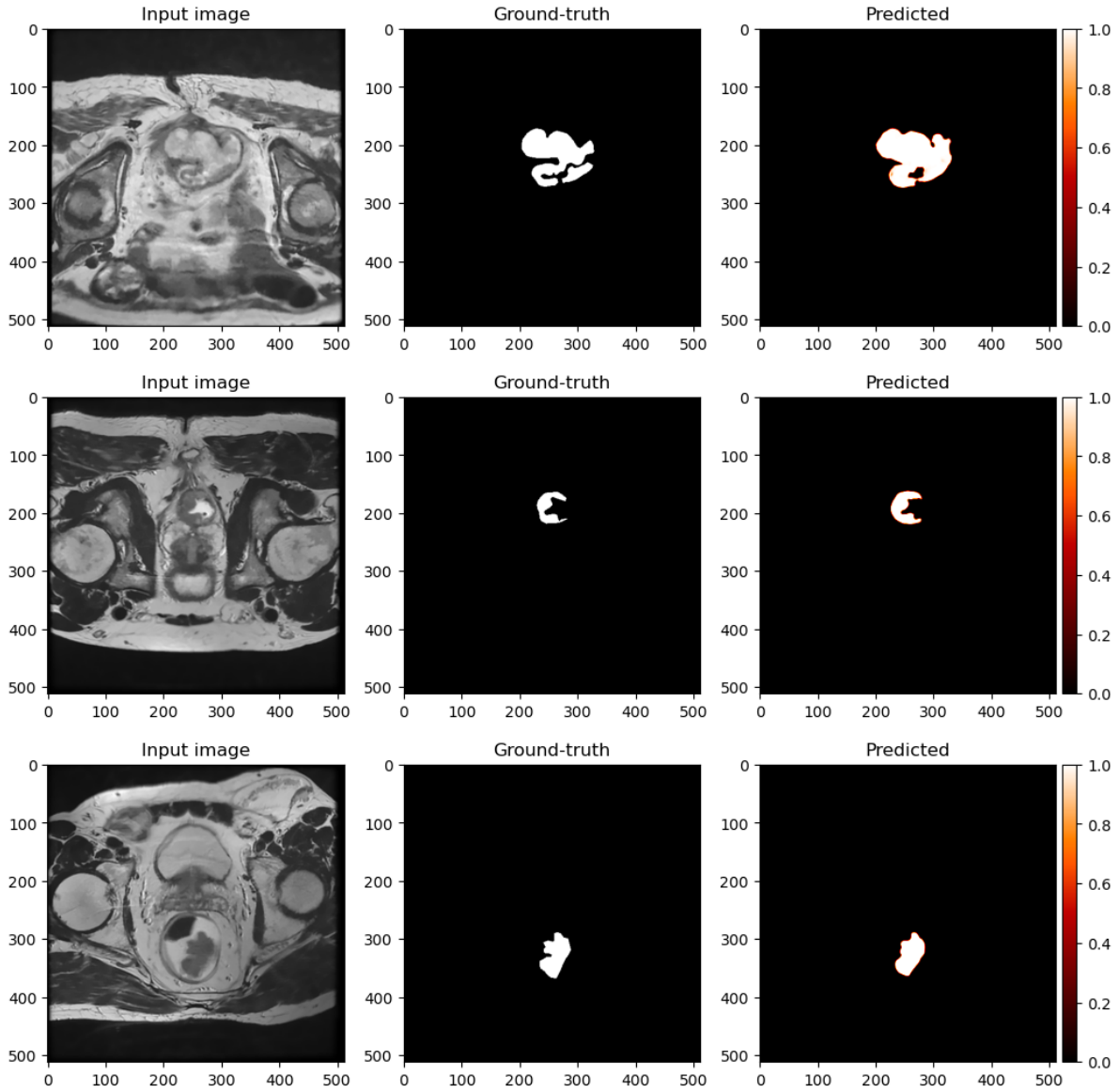

Figure 3.2: Comparison between the ground-truth image and the predicted one by the CNN model. The first column represents the input image. The second one represents the ground-truth image. Finally, the third one represents the predicted tumor area. The prediction is plotted as a probability density map between 0. and 1. as shown by the sequential colormap. From training set.

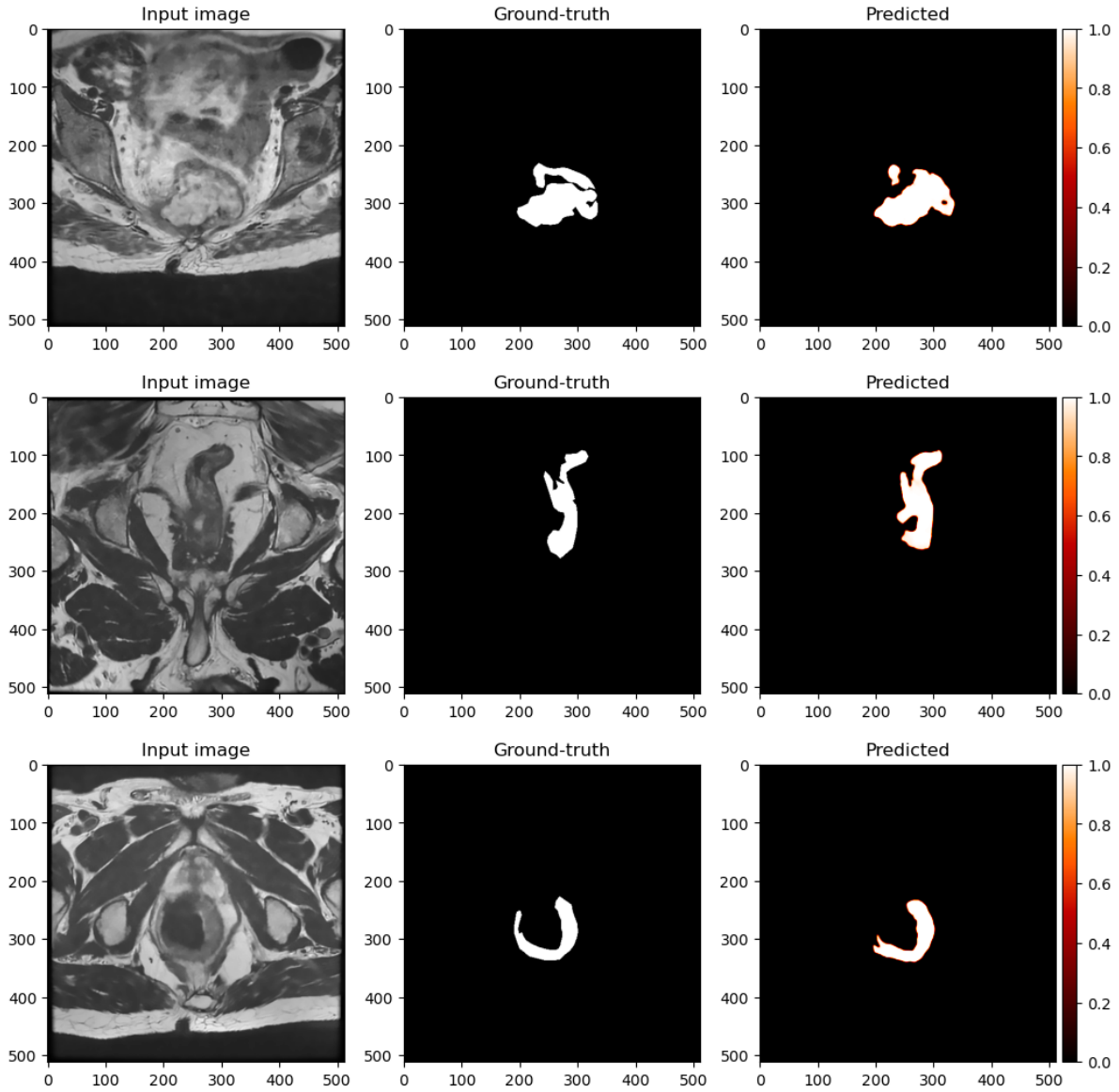

Figure 3.3: Comparison between the ground-truth image and the predicted one by the CNN model. The first column represents the input image. The second one represents the ground-truth image. Finally, the third one represents the predicted tumor area. The prediction is plotted as a probability density map between 0. and 1. as shown by the sequential colormap. From validation set.

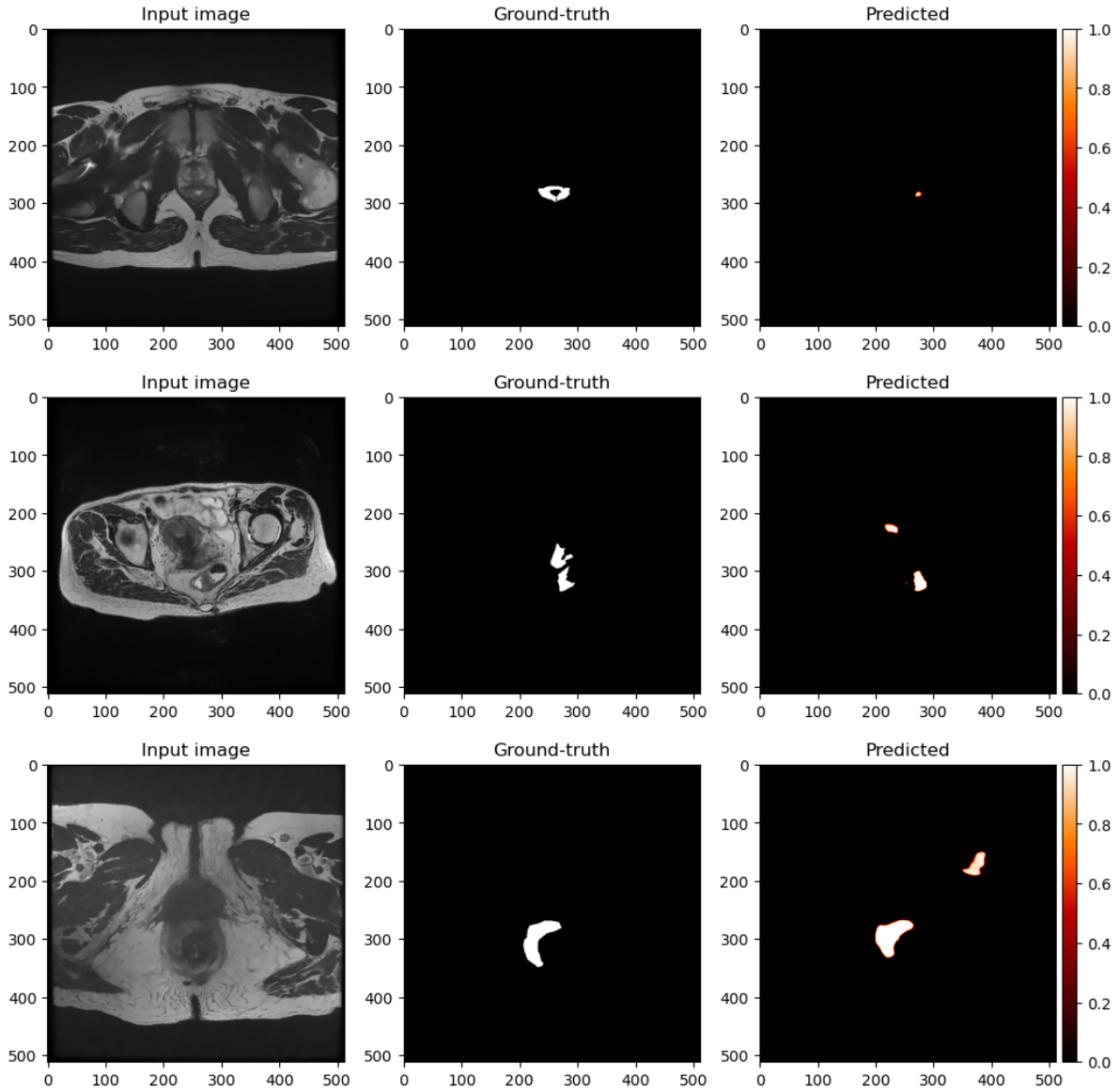

Figure 3.4: Comparison between the ground-truth image and the predicted one by the CNN model. The first column represents the input image. The second one represents the ground-truth image. Finally, the third one represents the predicted tumor area. The prediction is plotted as a probability density map between 0. and 1. as shown by the sequential colormap. Bad segmentation cases.

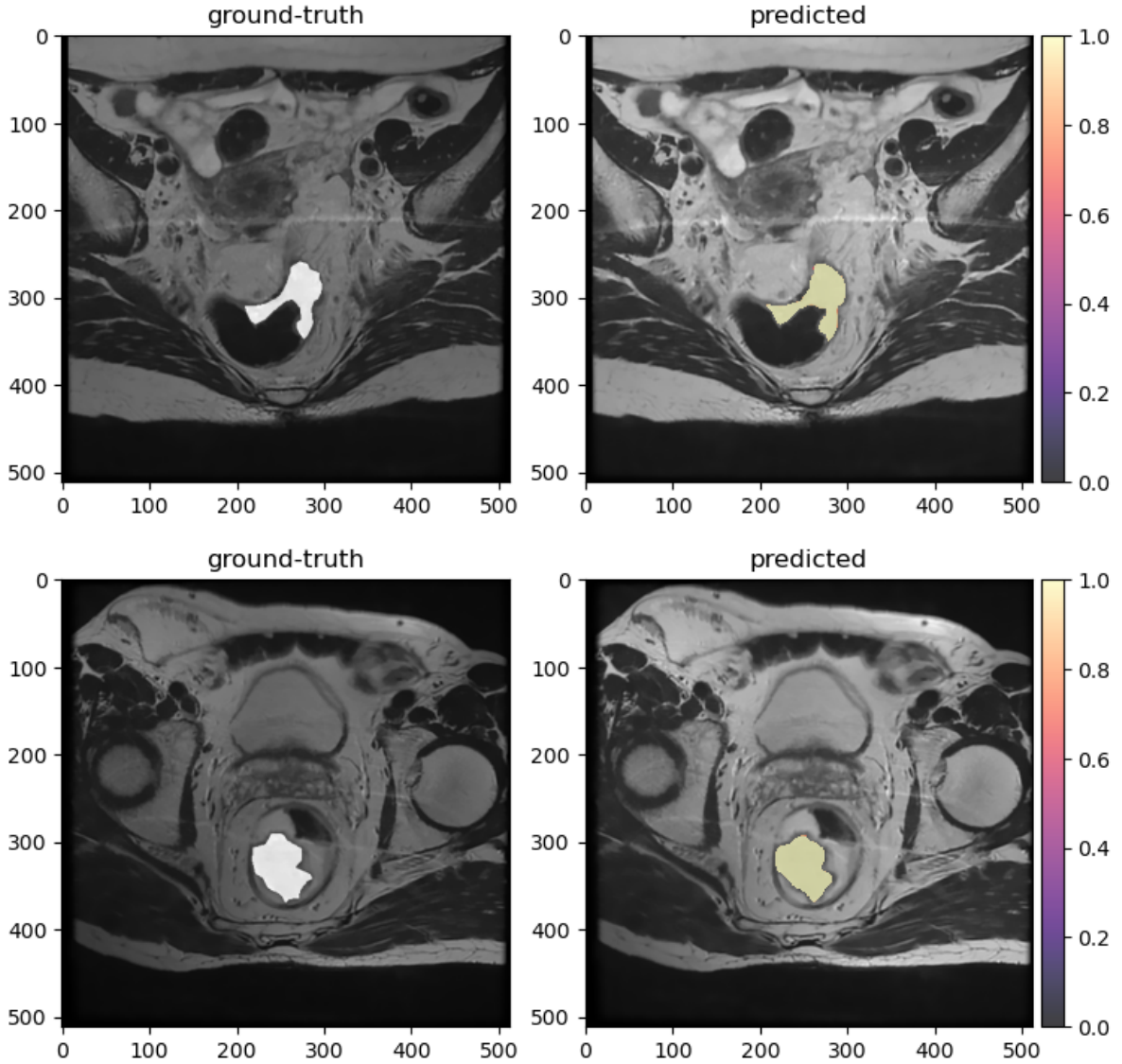

Figure 3.5: Comparison between the ground-truth image and the prediction over the original image. The prediction is plotted as a probability density map between 0. and 1. as shown by the sequential colormap. From training set.

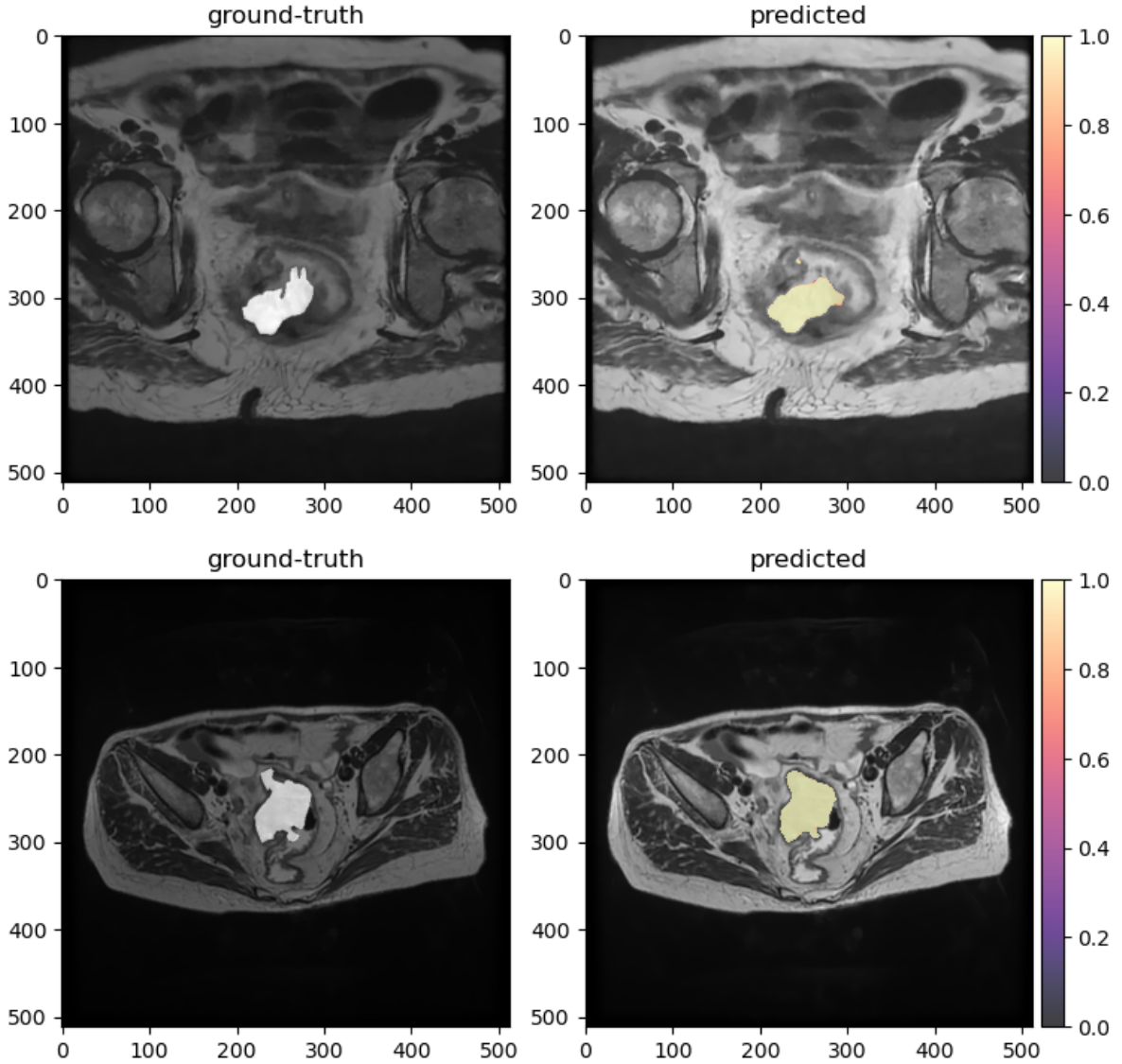

Figure 3.6: Comparison between the ground-truth image and the prediction over the original image. The prediction is plotted as a probability density map between 0. and 1. as shown by the sequential colormap. From training set.

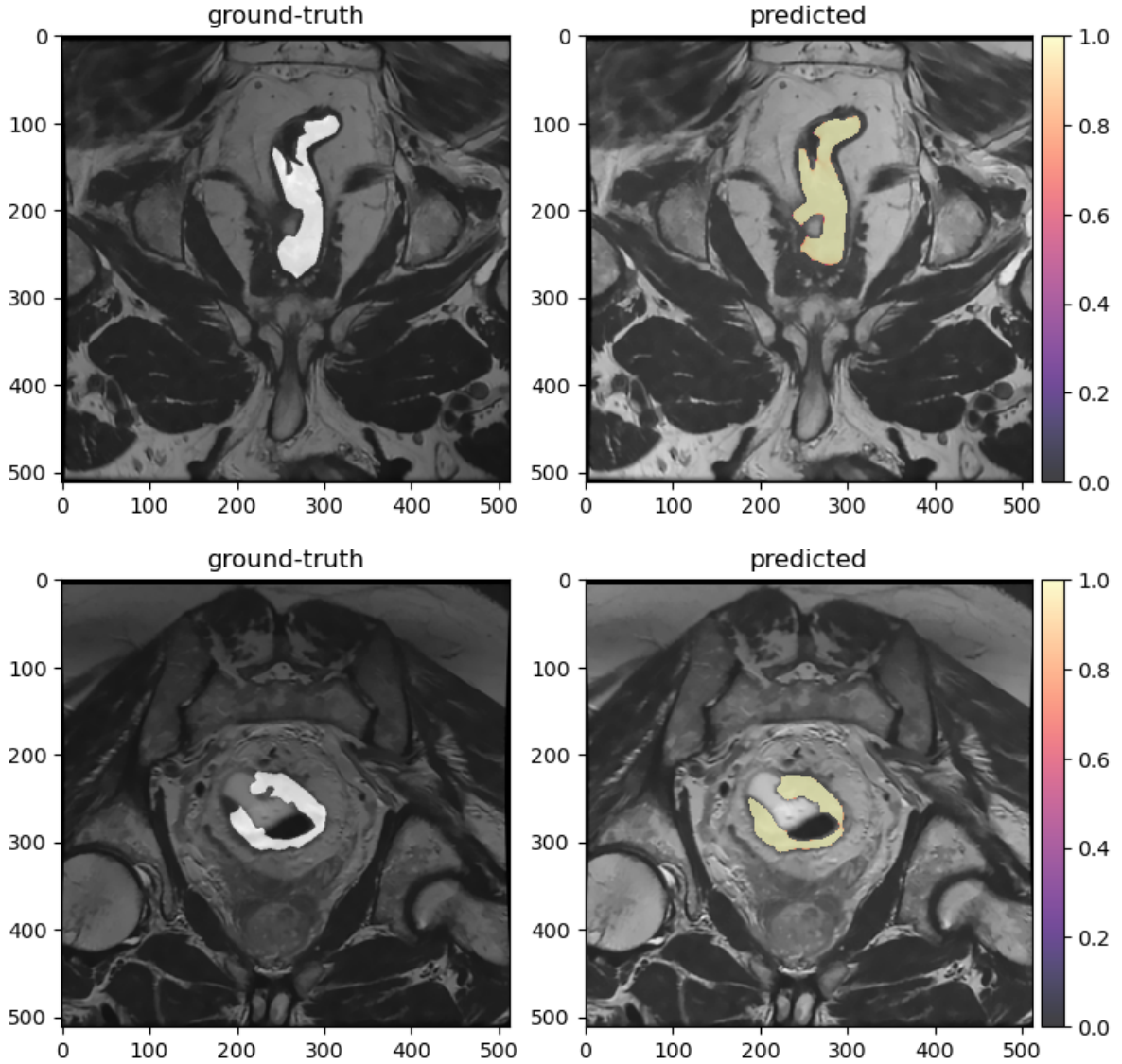

Figure 3.7: Comparison between the ground-truth image and the prediction over the original image. The prediction is plotted as a probability density map between 0. and 1. as shown by the sequential colormap. From validation set.

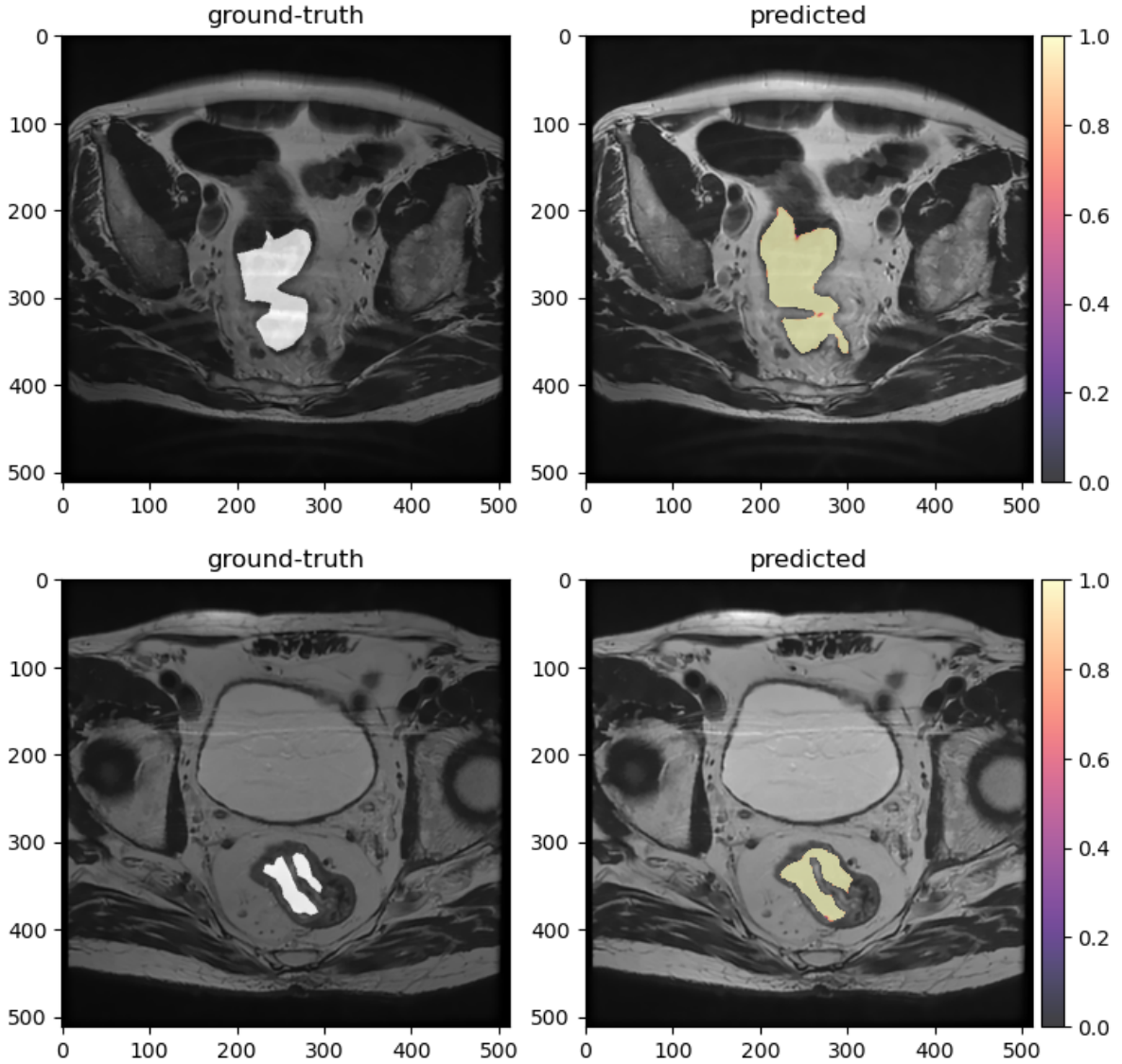

Figure 3.8: Comparison between the ground-truth image and the prediction over the original image. The prediction is plotted as a probability density map between 0. and 1. as shown by the sequential colormap. From validation set.

3.2.2 Prediction of Response

The accuracy for the prediction of response was measured by using different metrics. In Table 3.3 you can see the classification report made by the Precision, Recall, and F1 score for each class (Class 0 for complete response and Class 1 for moderate response). Unfortunately, the TRG value was missing for some patients, therefore some of them were excluded from the analysis. In the end, the total number of patients was 32, that is the sum of the *support* column values for Class 0 and Class 1.
The classifier model was cross-validated to avoid the presence of *bias* during the split into training and test set of the data on 500 simulations, evaluating the Matthews Correlation coefficient (MCC). The result is a distribution of the MCC, that you can see in Figure 3.9, with a median $MCC = 0.55$.

	Precision	**Recall**	**F1-Score**	**Support**
Class 0	0.71	0.77	0.74	13
Class 1	0.83	0.79	0.81	19

Table 3.3: Classification report

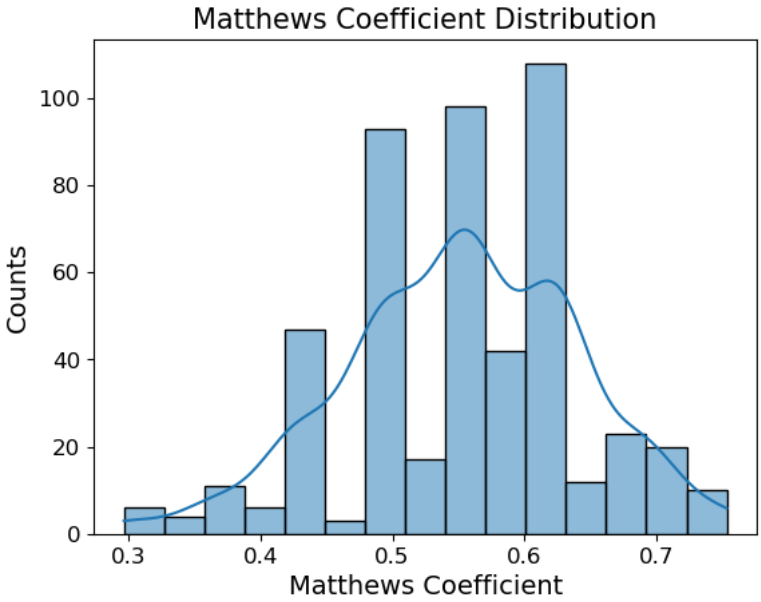

Figure 3.9: Matthews Correlation coefficient distribution for the pipeline obtained after 500 simulations.

I also computed the confusion matrix , in Figure 3.10, to evaluate the accuracy of the classification. For Class 0, so for a complete response, the well-classified instances are 10 over a total of 13, thus the wrong-classified ones are 3. For Class 1, so for a moderate response, the well-classified instances are 15 over a total of 19, thus the wrong-classified ones are 4.

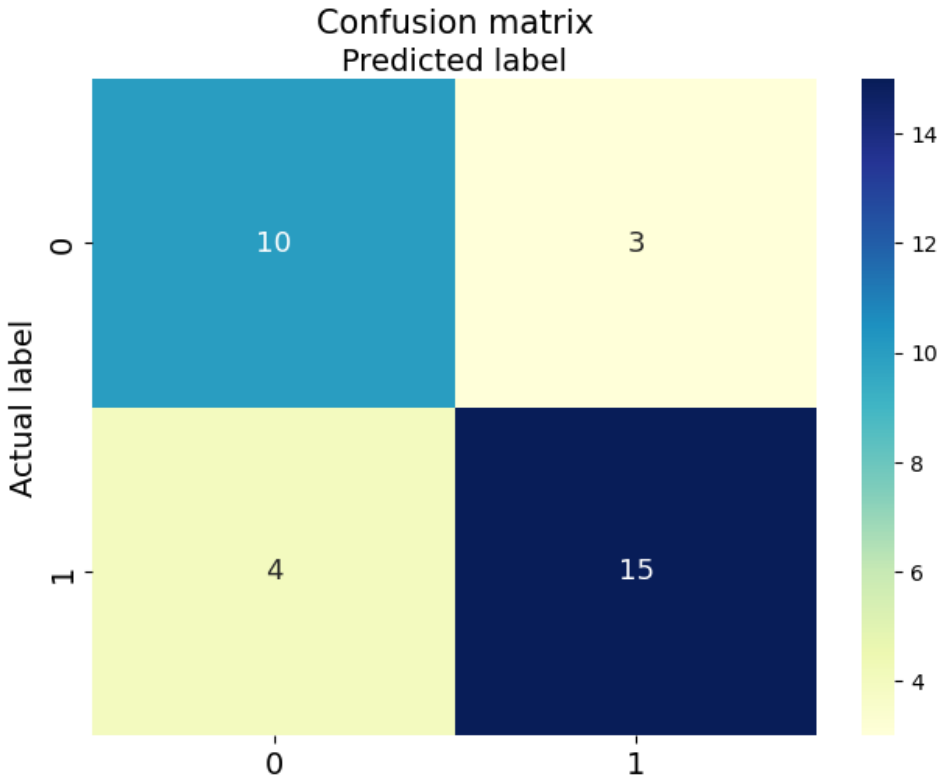

Figure 3.10: Confusion Matrix. The rows of the matrix represents the instances in an actual class while each column represents the instances in a predicted class. Class 0 means a complete response while Class 1 a moderate one.

The diagnostic ability of the classifier was also measured by computing the Receiver Operating Characteristic curve, or ROC curve, in Figure 3.11. As you can see, the AUC for both the classes is 0.82. I computed the average curve for Class 0 and Class 1, both by taking into account class imbalance (micro-average) and by giving the same weight to the classes (macro-average).

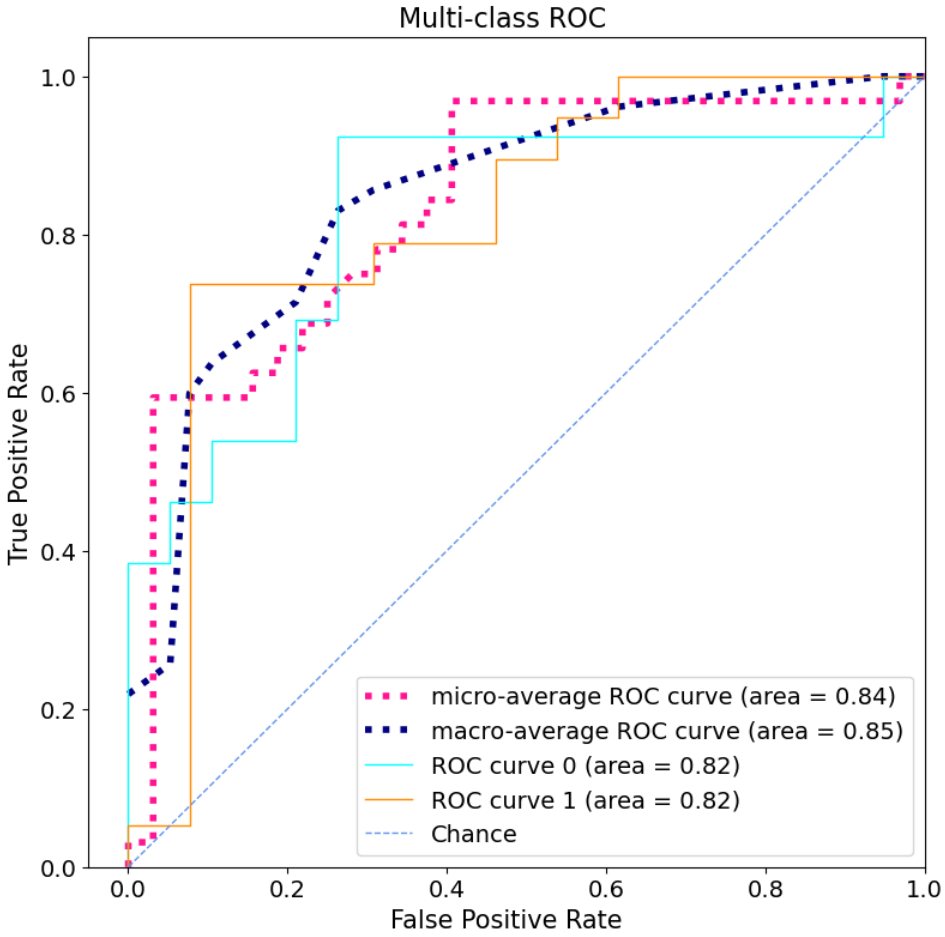

Figure 3.11: Receiver Operating Characteristic curve, or ROC curve. Class 0 indicates a complete response to chemo-radiotherapy while Class 1 a moderate one. The average curve for Class 0 and Class 1, is computed both by taking into account class imbalance (micro-average) and also by giving the same weight to the classes (macro-average).

3.3 Outputs

Once you have installed the implemented pipeline package, available on GitHub[34], you can get different outputs. This section is aimed to show the output results and how to get it from your shell.

Red Contours Segmentation

The output is the stack of segmented input slices. The input `dir` is the path of the dir containing the DICOM series. To get it:

```
python -m MRIsegm --dir='/path/to/input/series/'
```

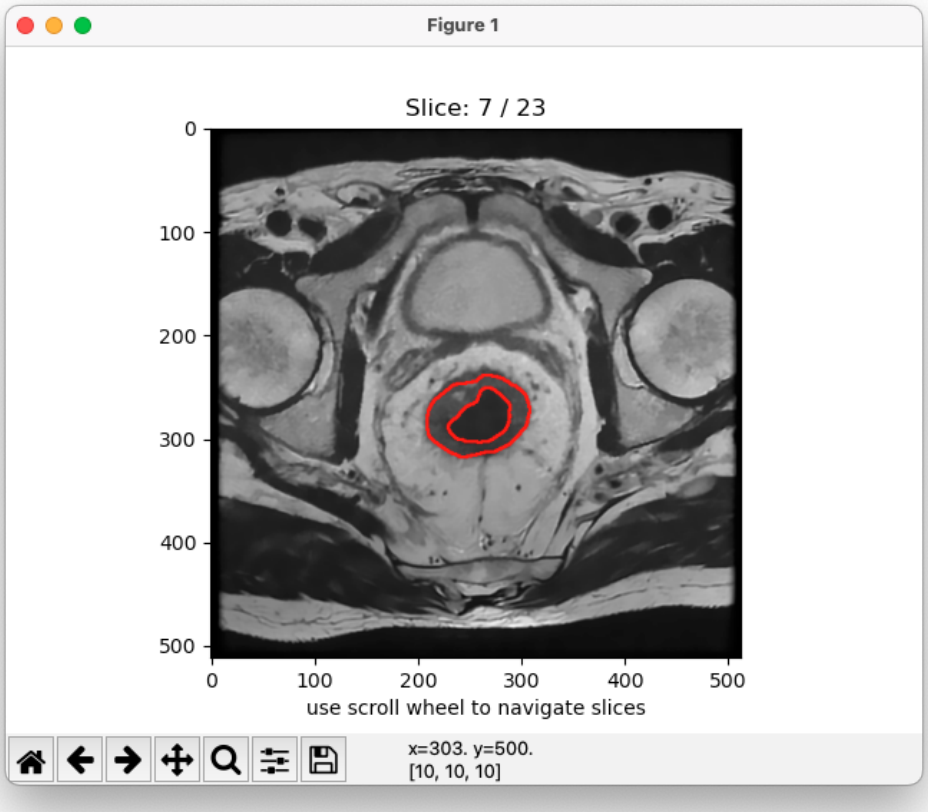

Figure 3.12: Identified tumor area inside the red contours.

mask

The output is the stack of predicted binary [0, 1] mask of input each slice. To get it:

```
python -m MRIsegm --dir='/path/to/input/series/'  --mask
```

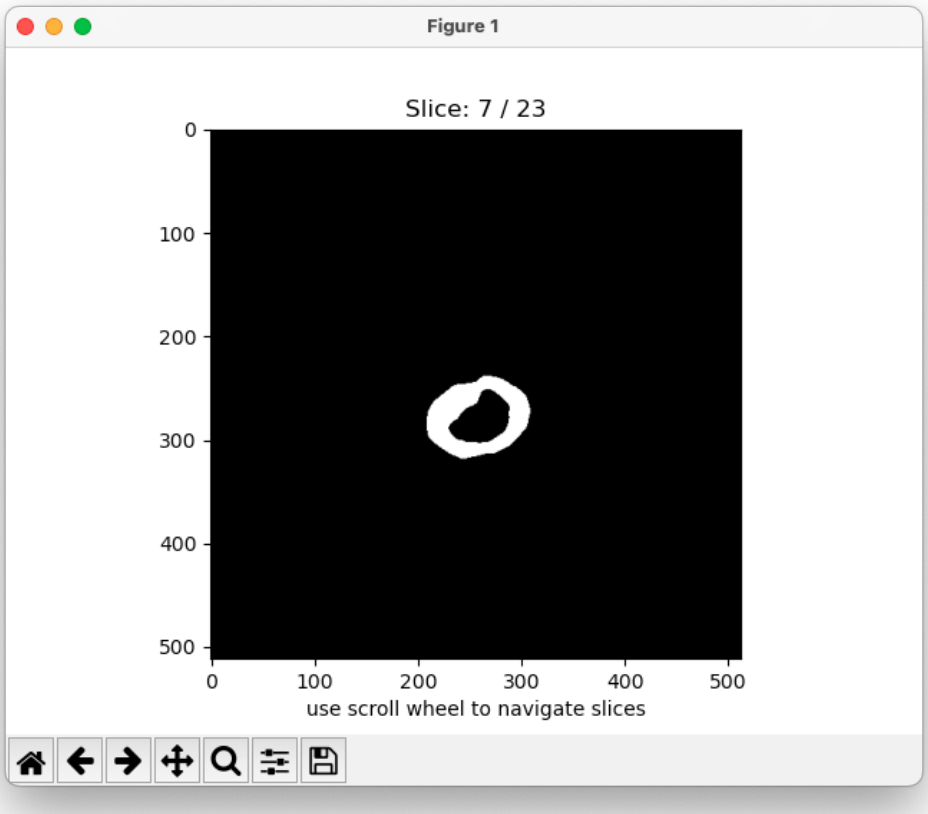

Figure 3.13: Binary mask of the segmented tumor area.

density

The output is the stack of probability map between 0. and 1. of each input slice over the original image. To get it:

```
python -m MRIsegm --dir='/path/to/input/series/'  --density
```

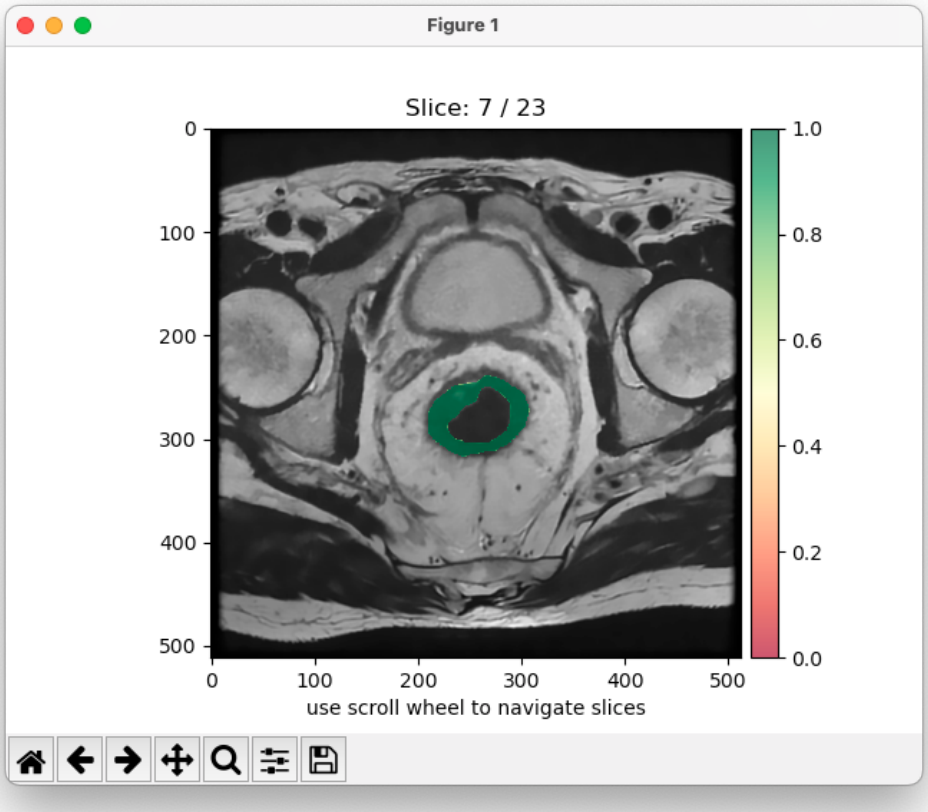

Figure 3.14: Probability map of the segmented tumor area.

3D

The output is a 3D mesh plot of the segmented areas. To get it:

```
python -m MRIsegm --dir='/path/to/input/series/'  --3D
```

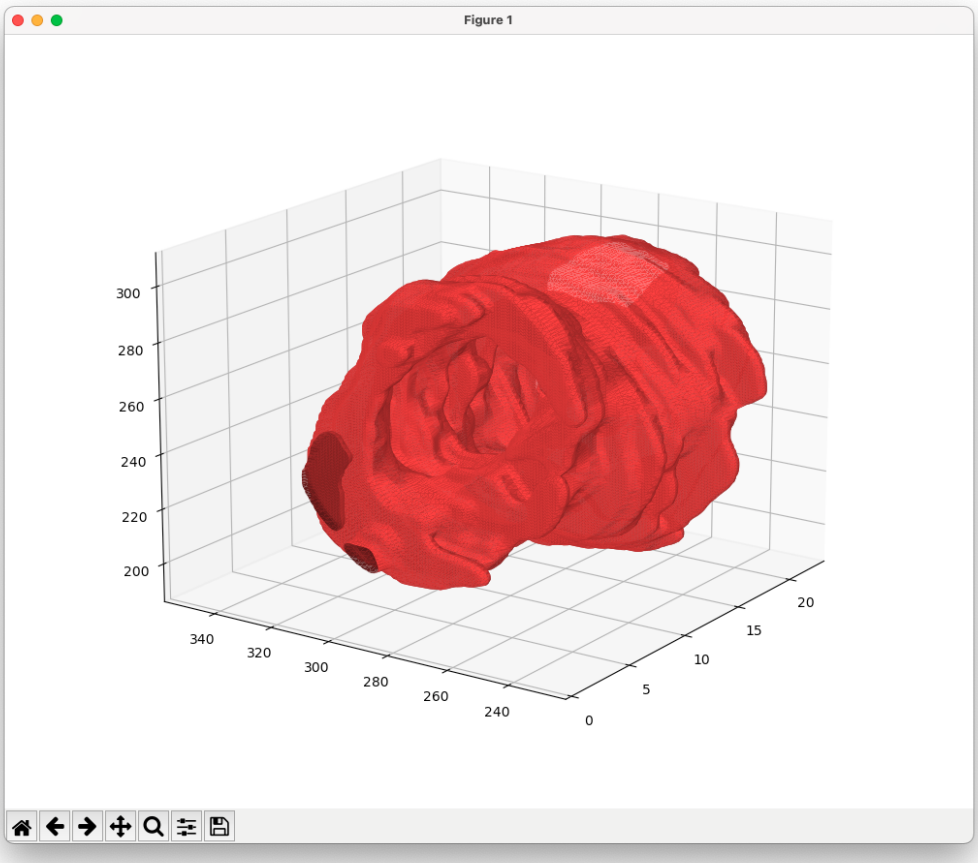

Figure 3.15: 3D mesh of the segmented areas.

Conclusions

In this work of thesis, I have developed and applied an automated pipeline to predict the response to neoadjuvant chemo-radiotherapy of patients affected by colorectal cancer. Here, we proposed an approach based on automatic segmentation and radiomic features extraction.
The pipeline was developed and tested on MRI scans provided by the IRCCS Sant'Orsola-Malpighi Policlinic. The results of these preliminary tests show that the pipeline is able to segment the correct tumor regions, with the correct shape and contours, without interaction with trained personnel. This was checked by comparing the segmentation results of the implemented pipeline with the manual annotations made by expert radiologists. The implemented model was able to segment correctly even *mucinous* cases, which are characterized by bright tumoral areas on MRI scans, different from the *adenocarcinomas* characterized by dark tumoral areas. Unfortunately, for some restricted cases it failed to segment correctly the images. Among the possible issues, there might be the lack of data caused by the exclusion of some patients, due to the misregistration between the input images and the manual annotations. This could have affected the training phase and the ability of the model to generalize.
The performance evaluation about the segmentation obtained by the pipeline on data coming from the validation set resulted in a DSC=0.71, that outperforms most of the ones in literature. This is a significant result since from literature it comes out that the segmentation achieved by CNNs is quite hard to perform on colorectal cancer MRI scans. The issues include the quality of data, medical annotations, and loss functions.
From the segmented regions, radiomic features were extracted and analyzed to obtain the the prediction of response. It is based on the Tumor Regression Grade (TRG) binarized into two main classes: Class 0 for complete response and Class 1 for moderate response. The results for the prediction of response show that the pipeline is able to classify correctly most of the cases. The prediction presents high scores for classification metrics such as Precision, Recall, and F1-score. In particular, the prediction is more accurate for Class 1 than Class 0 due to data imbalance towards Class 1. The implemented pipeline

was able to classify correctly 10 cases over 13 belonging to Class 0 and 15 cases over 19 for Class 1.
The performance of the classifier was also tested by computing the ROC curve, to evaluate the area under the curve (AUC). The AUC resulted in 82% for both classes. Also after computing the average ROC curve, the AUC resulted in 84% and 85% by taking into account class imbalance (micro-average) and by giving the same weight to the classes (macro-average), respectively. The AUC resulted in all the cases higher than the 80%, greater than 50% which would correspond to a random classifier.
The pipeline was implemented and developed by using the python language on the GitHub platform as an open-source project with the relative documentation. The pipeline implementation is able to provide different outputs: the identified tumor area with red contours, the predicted binary mask of the segmented slice, the predicted probability map of each slice and the 3D mesh of the segmented areas.
Further development of this project is possible, like embedding more information about the clinical status of the patient, genetic information, case history, etc... Moreover, increasing the number of patients in the analysis can give more general and robust results.
In the end, this project provided a suitable approach with satisfactory results to segment MRI colorectal cancer scans, in order to predict the response to neoadjuvant chemoradiotherapy by using radiomics features.

Appendix - Segmentation Models Comparison

MRI colorectal cancer segmentation is a particularly hard task to perform. Among the issues, there is the quality of data (i.e. the presence of different tumor subtypes such as *mucinous*), the medical annotations (subjected to operator expertise), the loss functions, and even model parameters that can affect the results of segmentation.
The aim of this Appendix is to give a comparison of two different CNN models, trained on the same data and medical annotations, changing the loss function and the number of epochs. In particular, one consists of the model used in the implemented pipeline, Model 1, and the other one, Model 2, of the same architecture, that is U-net with EfficientNetb0 as a backbone, and the Dice binary cross-entropy (DCE) as Loss function. The results and the specifications are reported in Table 3.4.
As you can see, by changing the Loss function and the number of epochs you can get a different performance as shown by the Dice Similarity Coefficient (DSC) score and the Loss one. Model 1 got a DSC score that is higher than the one obtained by Model 2, meaning a better performance. This is reflected even looking at the Loss score (lower the Loss score higher the performance).
In Figure 3.16 and 3.17, the curves of the training (blue) and validation (green) process for both the models are shown. As you can see, Model 1 results more stable than Model 2 since the fluctuations of the DSC and the loss are more restrained.
The difference in performances can also be appreciated by comparing the segmentation results with the ground-truth images. This is shown in Figure 3.18 and 3.19. The prediction is plotted as a probability density map between 0. and 1. as shown by the sequential colormap.
You can notice that the results obtained from Model 1 are better than the ones obtained from Model 2. In particular, Model 1 allows one to get better contours so as to be closer to the ground-truth image. However, it must be said that Model 1 was trained for 150 epochs versus 100 epochs for Model 2. Therefore, Model 1 has the benefit of more

learning processes.
The difference in performances can be better perceived in Figure 3.20 and 3.21 where the ground-truth image and the predicted ones overlap the input image so embedding spatial information about patient's anatomy. The prediction is plotted as a probability density map between 0. and 1. as shown by the sequential colormap.
Even in this case, you can see how Model 1 segmentation results are more accurate than the ones obtained by Model 2. As a matter of fact, Model 2 is able to correctly discriminate the tumor region but the contours are not very close to the ground truth ones like instead are for the predictions of Model 1. In Figure 3.21, on second row, I also included a case of *mucinous* tumor subtype. This kind of tumor is characterized by bright tumoral areas on MRI scans, different from the *adenocarcinomas* characterized by dark tumoral areas so the model can have more difficult in segmentation tasks. Indeed, Model 2 correctly discriminate the tumor region but the contours are not very accurate like the ones obtained by Model 1 predictions.
In the end, Model 1 has been proved to get better results compared to Model 2. This was achieved only by using a different Loss function and rising the number of epochs while keeping the same model architecture, U-Net with EfficientNetb0 as a backbone.

	Architecture	**Loss Function**	**Epochs**	**DSC**	**Loss**
Model 1	U-Net, backbone EfficientNetb0	Dice Loss + Focal Loss	150	0.71	0.25
Model 2	U-Net, backbone EfficientNetb0	Dice binary cross-entropy Loss	100	0.68	0.35

Table 3.4: Table of comparison between the two models. As you can see, by changing the loss function and the number of epochs you can get different performances.

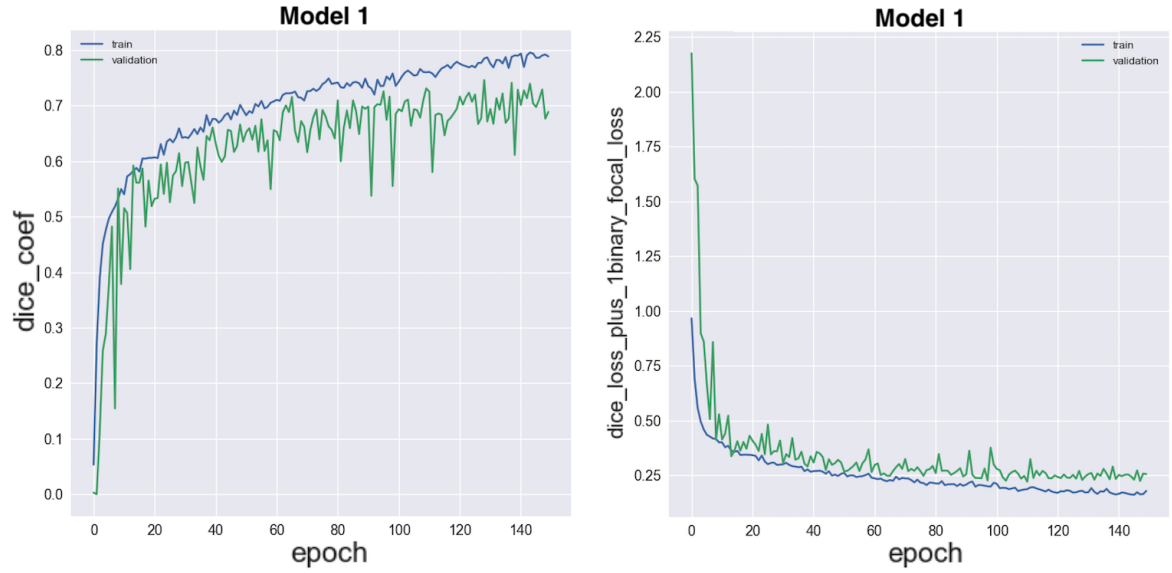

Figure 3.16: Training (blue) and validation (green) model history plots for Model 1. *Left*): model dice coefficient as a function of the epochs. *Right*): model loss as a function of the epochs.

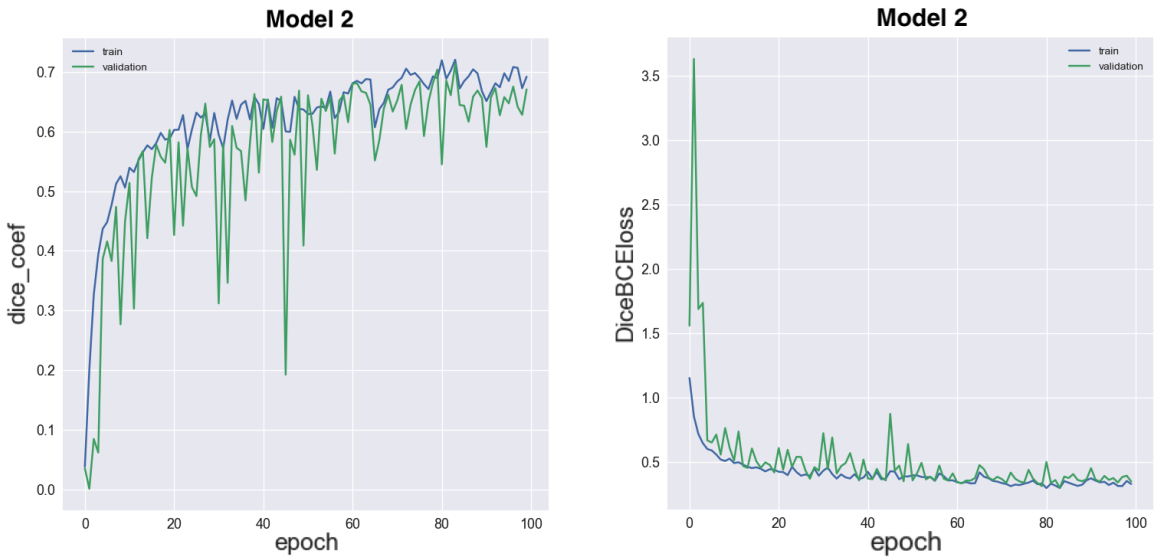

Figure 3.17: Training (blue) and validation (green) model history plots for Model 2. *Left*): model dice coefficient as a function of the epochs. *Right*): model loss as a function of the epochs.

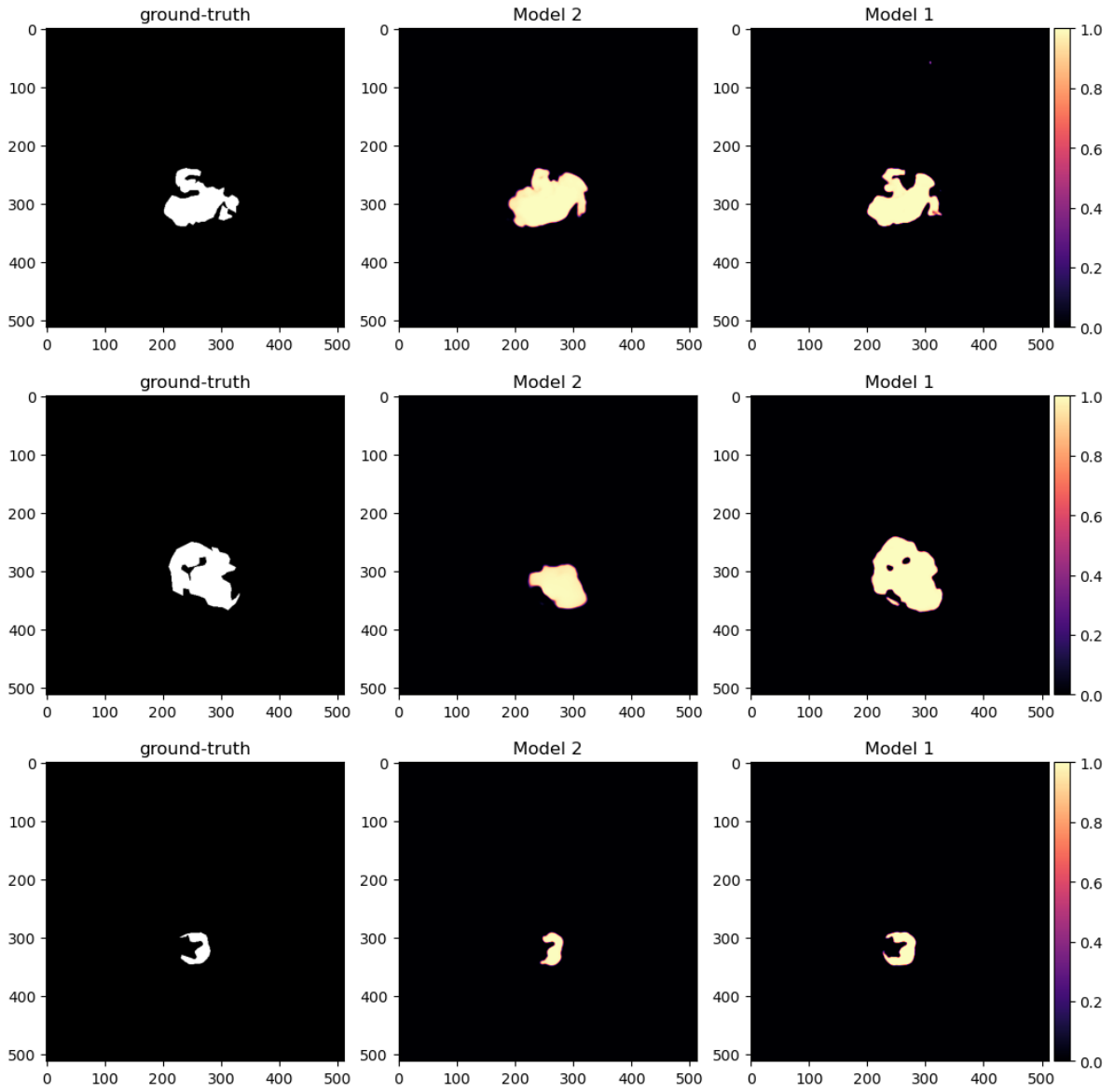

Figure 3.18: Comparison between the ground-truth image and the prediction of both models. The first column represents the input image. The second one represents the prediction of Model 2. The third one represents the prediction of Model 1. The prediction is plotted as a probability density map between 0. and 1. as shown by the sequential colormap.

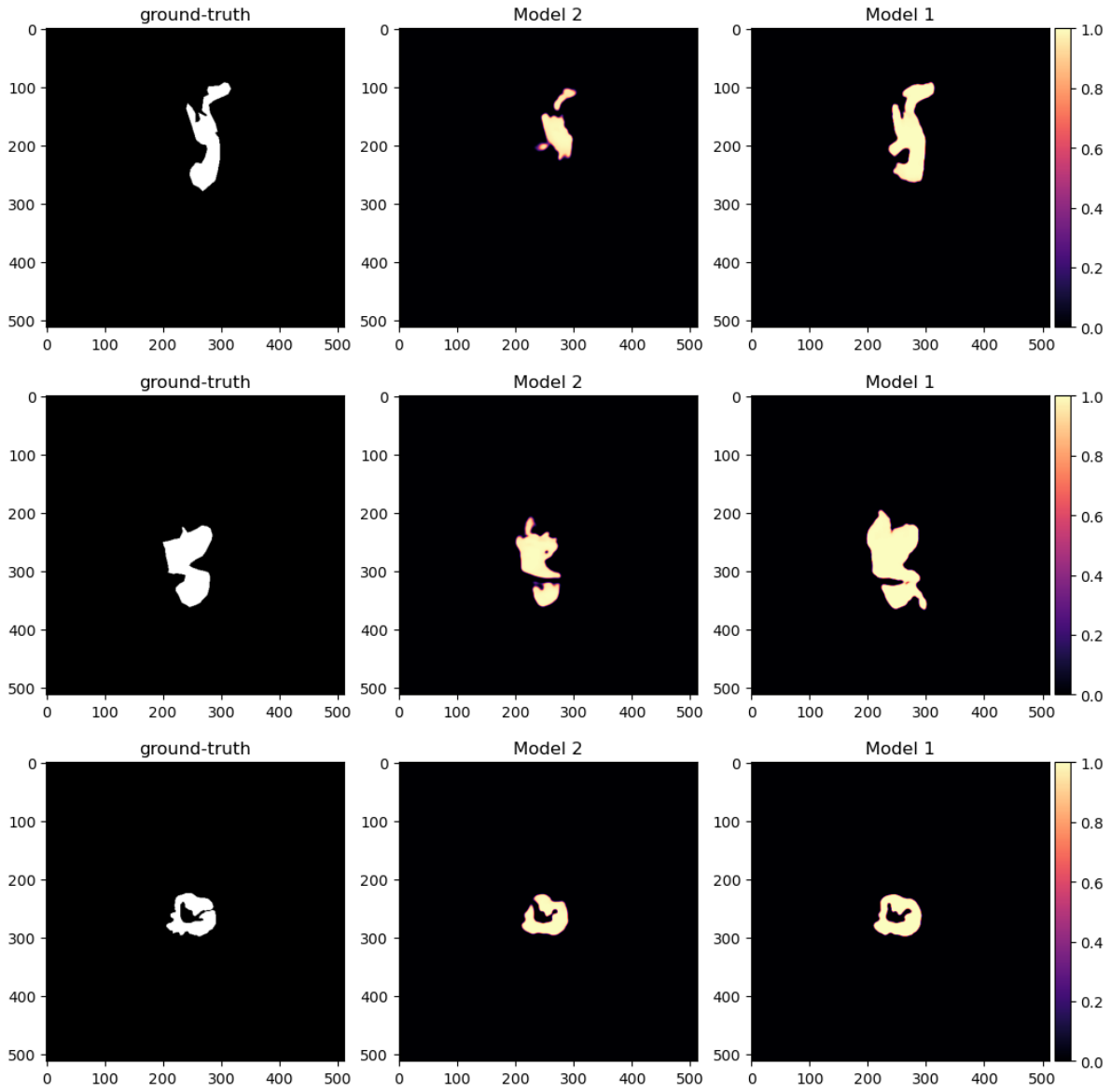

Figure 3.19: Comparison between the ground-truth image and the prediction of both models. The first column represents the ground-truth image. The second one represents the prediction of Model 2. The third one represents the prediction of Model 1. The prediction is plotted as a probability density map between 0. and 1. as shown by the sequential colormap.

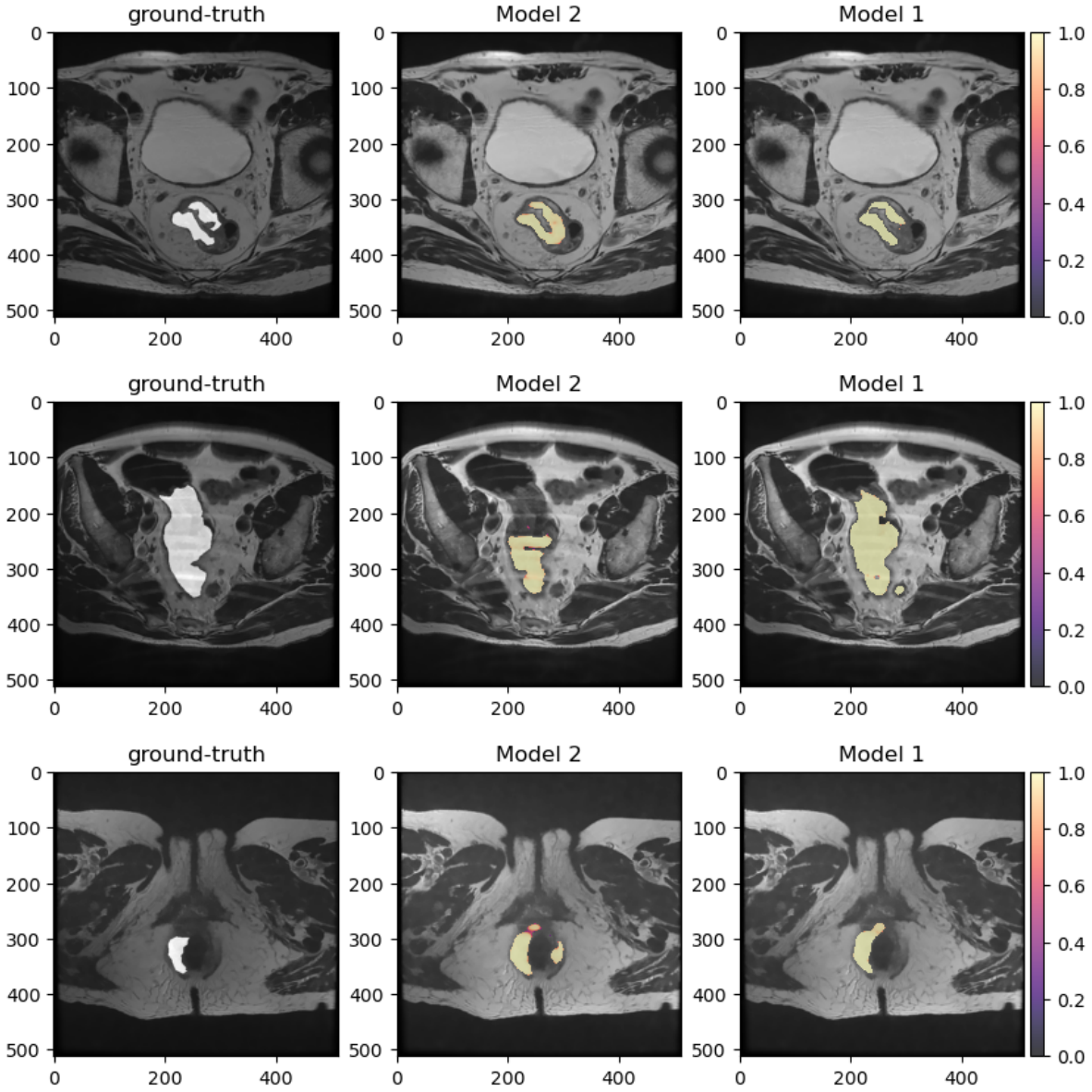

Figure 3.20: Comparison between the ground-truth image and the prediction of both models over the input image. The first column represents the ground-truth image. The second one represents the prediction of Model 2. The third one represents the prediction of Model 1. The prediction is plotted as a probability density map between 0. and 1. as shown by the sequential colormap.

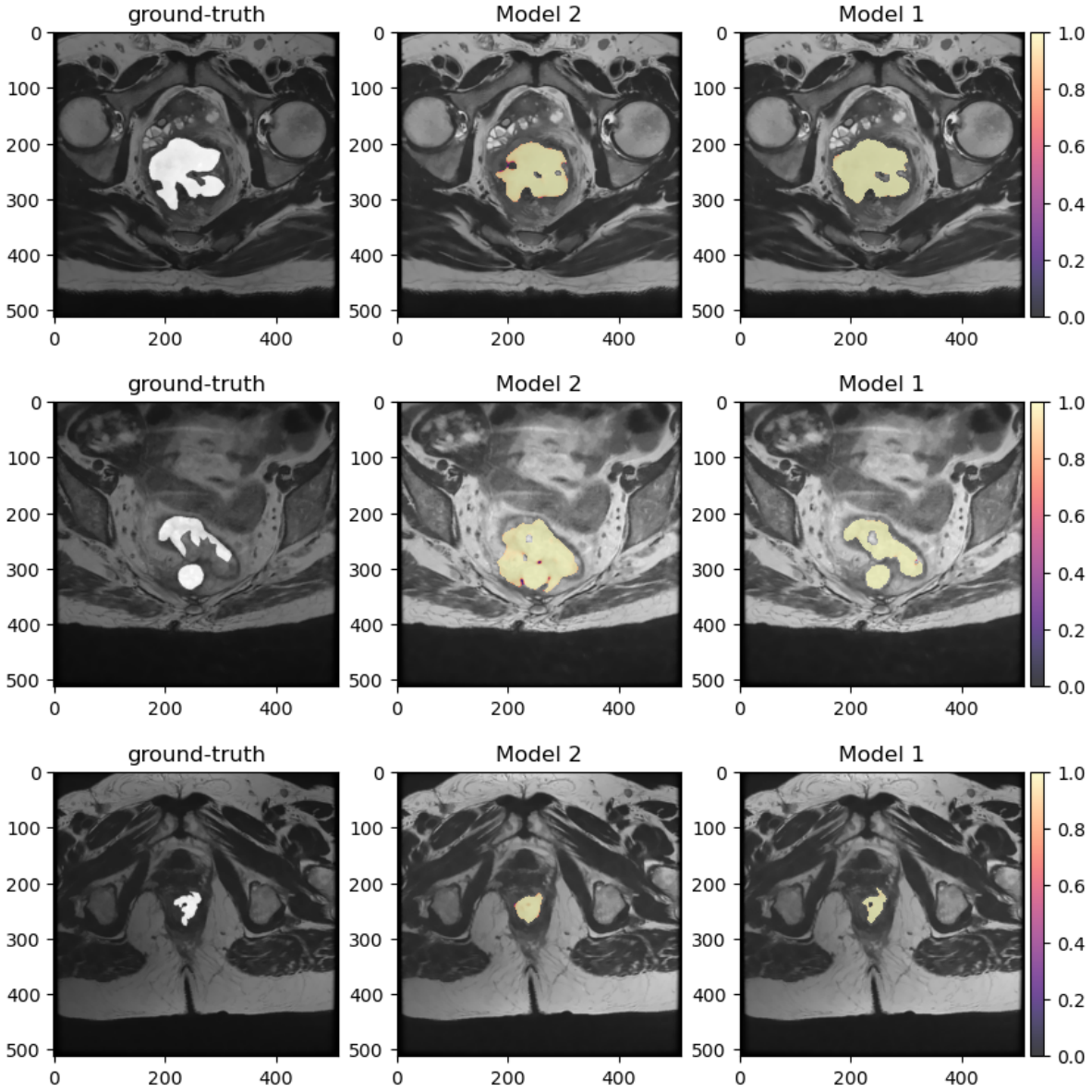

Figure 3.21: Comparison between the ground-truth image and the prediction of both models over the input image. The first column represents the ground-truth image. The second one represents the prediction of Model 2. The third one represents the prediction of Model 1. The prediction is plotted as a probability density map between 0. and 1. as shown by the sequential colormap.

ACKNOWLEDGEMENTS

I would like to thank everyone who has contributed to this work.
First of all, my parents and all my family, that with their tireless support, both moral and economic, allowed me to get here today.
I particularly thank my uncles who have become a second family for me, and Mariella who has ever been to my side during these years, since the first day.
A special thank to Nicole, which has ever believed and supported me.
I would like to thank Prof. Gastone Castellani for the project and the availability.
Last but not least I would really like to thank Dr. Nico Curti, for the availability and precision shown to me throughout the working period. Without him, I would not have learned and improved as much as I did so far.

Bibliography

[1] International Agency for Research on Cancer. *Rectum-fact-sheet*. 2020. URL: https://gco.iarc.fr/today/data/factsheets/cancers/9-Rectum-fact-sheet.pdf.

[2] ECIS – European Cancer Information System. *Colorectal cancer burden in EU-27*. 2021. URL: https://ecis.jrc.ec.europa.eu/pdf/Colorectal_cancer_factsheet-Mar_2021.pdf.

[3] Jia Cheng Yuan. "Analisi radiomica per predire la risposta alla chemio-radioterapia neoadiuvante nel carcinoma del colon retto". Tesi di laurea. Alma Mater Studiorum Università di Bologna, 2019/2020.

[4] Jemal A Siegel RL Miller KD. "Cancer statistics, 2019". In: *CA Cancer J Clin* 69(1).7-34 (2019).

[5] Jovana Panic. "Colorectal cancer segmentation on MRI images using Convolutional Neural Networks". Tesi di Laurea Magistrale. Politecnico di torino, 2018/2019.

[6] Trebeschi et al. "Deep Learning for Fully-Automated Localization and Segmentation of Rectal Cancer on Multiparametric MR". In: *Scientific Reports* 7.1 (2017), p. 5301. DOI: 10.1038/s41598-017-05728-9. URL: https://doi.org/10.1038/s41598-017-05728-9.

[7] Rafael C. Gonzalez and Richard E. Woods. *Digital image processing*. 2018.

[8] Michele Larobina and Loredana Murino. "Medical image file formats". In: *Journal of digital imaging* 27.2 (Apr. 2014), pp. 200–206. DOI: 10.1007/s10278-013-9657-9. URL: https://pubmed.ncbi.nlm.nih.gov/24338090.

[9] Riccardo Biondi. "Implementation of an Authomated Pipeline for the Identification of Ground Glass Opacities in Chest CT Scans of Patient Affected by COVID-19". MA thesis. riccardo.biondi7@unibo.it: Alma Mater Studiorum - Università di Bologna, School of Science, Dec. 2020.

[10] Milan Sonka, Vaclav Hlavac, and Roger Boyle. "Image pre-processing". In: Boston, MA: Springer US, 1993, pp. 56–111.

[11] *Processing in Spatial Domain – Spatial Filtering*. URL: https://ceng.eskisehir.edu.tr/serkangunal/BIM472/odev/BIM472%20-%2004%20-%20Processing%20in%20Spatial%20Domain%20(Part%202).pdf.

[12] Bhumika Gupta and Shailendra Singh Negi. "Image Denoising with Linear and Non-Linear Filters: A REVIEW". In: *International Journal of Computer Science Issues (IJCSI)* 10.6 (Nov. 2013), pp. 149–154. URL: https://www.proquest.com/scholarly-journals/image-denoising-with-linear-non-filters-review/docview/1498210097/se-2?accountid=9652.

[13] Antoni Buades, Bartumeu Coll, and Jean Micheal Morel. *On image denoising methods*. https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.100.81&rep=rep1&type=pdf.

[14] Stefan Van der Walt et al. "scikit-image: image processing in Python". In: *PeerJ* 2 (2014), e453.

[15] K Somasundaram and P Kalavathi. *Medical Image Contrast Enhancement based on Gamma Correction*.

[16] Rajeshwar Dass and Swapna Devi. *Image Segmentation Techniques 1*.

[17] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. *Understanding of a convolutional neural network*. 2017.

[18] Caterina Camborata. "Capsule Networks: a new approach for brain imaging". MA thesis. Alma Mater Studiorum - Università di Bologna, School of Science, 2018.

[19] Tsung-Yi Lin et al. *Focal Loss for Dense Object Detection*. 2018. arXiv: 1708.02002 [cs.CV].

[20] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *CoRR* abs/1505.04597 (2015). arXiv: 1505.04597. URL: http://arxiv.org/abs/1505.04597.

[21] *Radiomics*. URL: https://en.wikipedia.org/wiki/Radiomics.

[22] Michal R. Tomaszewski and Robert J. Gillies. "The Biological Meaning of Radiomic Features". In: *Radiology* 298.3 (2021). PMID: 33399513, pp. 505–516. DOI: 10.1148/radiol.2021202553. eprint: https://doi.org/10.1148/radiol.2021202553. URL: https://doi.org/10.1148/radiol.2021202553.

[23] Thibaud P Coroller et al. "CT-based radiomic signature predicts distant metastasis in lung adenocarcinoma." In: *Radiother Oncol* 114.3 (Mar. 2015), pp. 345–350.

[24] Michael E. Tipping and Christopher M. Bishop. "Probabilistic Principal Component Analysis". In: *Journal of the Royal Statistical Society. Series B (Statistical Methodology)* 61.3 (1999), pp. 611–622. ISSN: 13697412, 14679868. URL: http://www.jstor.org/stable/2680726.

[25] URL: https://scikit-learn.org/stable/modules/svm.html#svm-mathematical-formulation.

[26] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[27] S VIJAYAKUMAR. "Sequential Support Vector Classifiers and Regression". In: *Proc. SOCO'99* (1999), pp. 610–619. URL: https://ci.nii.ac.jp/naid/10010222296/en/.

[28] Shruti Jadon. "A survey of loss functions for semantic segmentation". In: *2020 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB)* (Oct. 2020). DOI: 10.1109/cibcb48159.2020.9277638. URL: http://dx.doi.org/10.1109/CIBCB48159.2020.9277638.

[29] *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[30] François Chollet et al. *Keras*. https://keras.io. 2015.

[31] Mingxing Tan and Quoc V. Le. "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks". In: *CoRR* abs/1905.11946 (2019). arXiv: 1905.11946. URL: http://arxiv.org/abs/1905.11946.

[32] G. Bradski. "The OpenCV Library". In: *Dr. Dobb's Journal of Software Tools* (2000).

[33] Joost J. M. van Griethuysen et al. "Computational Radiomics System to Decode the Radiographic Phenotype". In: *Cancer Research* 77.21 (Nov. 2017), e104.

[34] Giuseppe Filitto. *MRI colorectal cancer segmentation*. https://github.com/giuseppefilitto/img-segm. 2021.

[35] Georg Brandl. "Sphinx documentation". In: *URL http://sphinx-doc. org/sphinx. pdf* (2021).

[36] Charles R. Harris et al. "Array programming with NumPy". In: *Nature* 585.7825 (2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[37] et al Mason D. L. *pydicom: An open source DICOM library*. https://github.com/pydicom/pydicom.

[38] Pavel Yakubovskiy. *Segmentation Models*. https://github.com/qubvel/segmentation_models. 2019.

[39] Ziv Yaniv et al. "SimpleITK Image-Analysis Notebooks: a Collaborative Environment for Education and Reproducible Research". In: *Journal of Digital Imaging* 31.3 (2018), pp. 290–303.

[40] Fabian Pedregosa et al. "Scikit-learn: Machine learning in Python". In: *Journal of machine learning research* 12.Oct (2011), pp. 2825–2830.

[41] Panic J at al. *A Convolutional Neural Network based system for Colorectal cancer segmentation on MRI images*. 2020. DOI: 10.1109/EMBC44109.2020.9175804.

[42] Yi-Jie Huang et al. "3-D RoI-Aware U-Net for Accurate and Efficient Colorectal Tumor Segmentation". In: *IEEE Transactions on Cybernetics* (2020), pp. 1–12. ISSN: 2168-2275. DOI: 10.1109/tcyb.2020.2980145. URL: http://dx.doi.org/10.1109/TCYB.2020.2980145.