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 chemoradiotherapy 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.
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. Risk factors for this kind of cancer include colon polyps, long-standing ulcerative colitis, diabetes II, and genetic history (HNPCC or Lynch syndrome). In order to get information about a diagnosis, therapy evaluation on colorectal cancer, analysis on radiological images can be performed through the application of dedicated algorithms.
In this scenario, the correct and fast identification of the cancer regions is a fundamental task. Up to now, this process is performed using manual or semi-automatic techniques, which are time-consuming and subjected to the operator's expertise.
The aim of this project is to develop and implement an automated pipeline to predict the response to neoadjuvant chemo-radiotherapy patients affected by colorectal cancer.

. . . 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].
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 cancer [2]. Preventive measures for colorectal cancer include physical activity, reducing the consumption of processed meat and alcohol, and avoiding smoking[3].

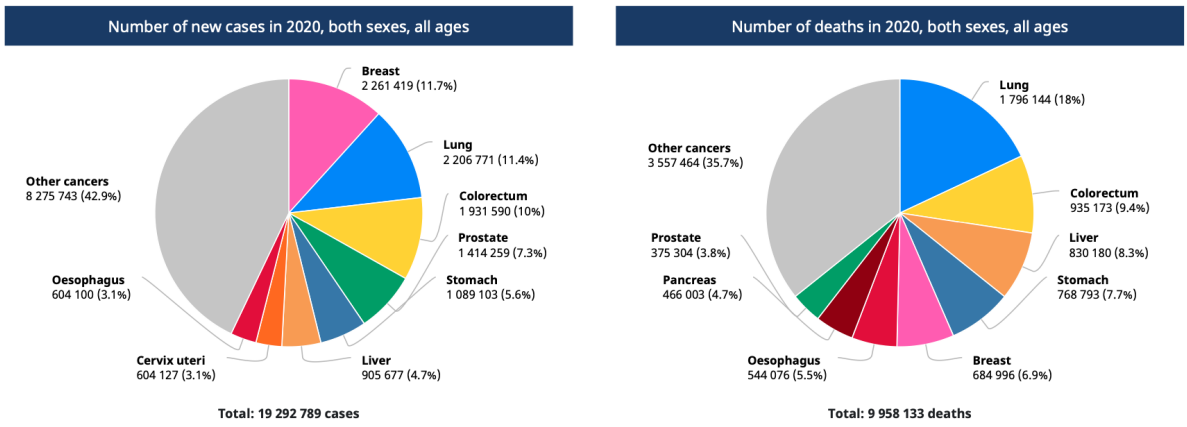

Figure 1: World's cancer cases and deaths in 2020. As you can see from the *Left* image, colorectal cancer is the third malignant tumor per number of new cases after breast and lung cancer while as shown on the *Right* image, it is the second one per number of deaths after lung cancer. From [1]

Screening and diagnosis methods for colorectal cancer involve different techniques. The

gold standard in medical routines is the colonoscopy which is an invasive technique[4]. Among medical imaging techniques, Magnetic Resonance Imaging (MRI) and Computed Tomography (CT) is the most used[2]. In particular, Magnetic Resonance Imaging (MRI) is used for pre-operative predictions and for the evaluation of the neo-adjuvant chemo-radiotherapy of patients affected by colorectal cancer[2].

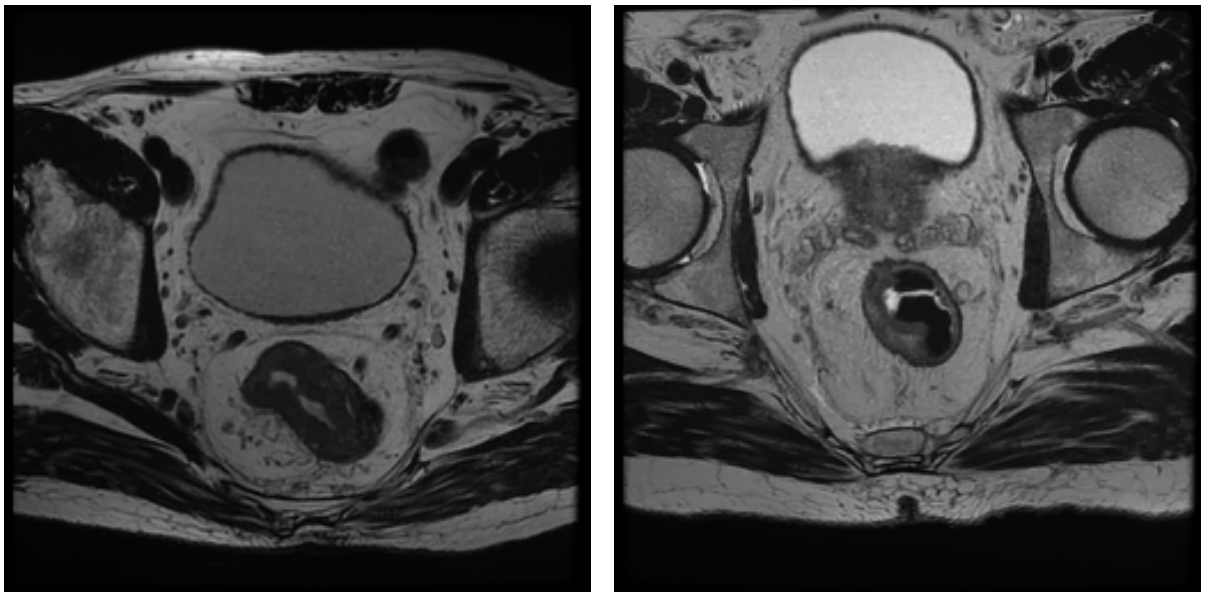

Figure 2: Magnetic Resonance images of patients affected by colorectal cancer. MRI scans can be used for pre-operative predictions and for the evaluation of the neo-adjuvant chemo-radiotherapy. From IRCCS Sant'Orsola-Malpighi Policlinic Dataset.

In order to get information about a diagnosis, therapy evaluation, stage of colorectal cancer, analysis on radiological images can be performed through the application of dedicated algorithms.
In this scenario, the correct and fast identification of the cancer regions is a fundamental task. Up to now, this segmentation task is performed using manual or semi-automatic techniques, which are time-consuming (requiring hours per day) and subjected to the operator's expertise since it requires interaction with trained specialists.[2, 4]. Moreover, due to the high sensitivity to the operator expertise, the obtained results cannot be reproduced[5]. To overcome these issues, an automatic and fast way is required.
The aim of this project is to develop and implement an automated pipeline to segment MRI scans of patients affected by colorectal cancer in order to predict the response to neo-adjuvant chemo-radiotherapy. The work is based and tested on MRI scans provided by IRCCS Sant'Orsola-Malpighi Polyclinic.
The discussion will start focusing on the materials and methods: firstly, medical digital images, to understand their properties and features. Then, an overview of the segmenta-

tion methods and the main architecture used for the identification of the cancer regions. Also, some words about radiomics and its possible purposes. Last but not least, an overview of Principal Component Analysis and Support Vector Classifiers will be given. The second chapter will regard the implemented pipeline: the main description and implementation will be provided. The obtained results will be shown and discussed in the third chapter. Finally, the Conclusions.

Chapter 1

Materials and Methods

This chapter will be focusing on the description of the main materials and methods used for the development of this project. The first topic to be treated will be medical images: what are medical images and what kind of properties we can distinguish from them. The second one will regard the filtering process, focusing on smoothing filters. The third section, instead, will regard the segmentation: an overview of the most common methods and the architecture used to achieve this task. Then, radiomics: features and possible purposes. Last but not least, an overview of Principal Components Analysis (PCA) and Support Vector Classifiers (SVCs).

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 picture elements called *pixels*. A pixel is a discrete numeric representation of intensity or gray-level, that is an output coming from a two-dimensional function $f(x, y)$ fed as input by its spatial coordinates denoted with (x, y) on the x-axis and y-axis[6].
A digital image can be processed by computers. This process is called *digital image processing*. It is useful to divide the mentioned process 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 of digital images:

Pixel depth 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[7]. Higher the number of bits, higher the information stored but more memory space is required[7]. 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[7]. 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 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 separate images. For example, complex data are provided in MRI acquired data before the reconstruction (the so called k-space)[7].

Metadata are information that describe the image. It is usually stored at the beginning of the file as a header[7]. 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 and the weight of the patient.

1.1.2 Medical Image Formats

Image file formats provide a standard way to store information of an image in a computer file[8]. Medical image file formats can be divided in two categories. The first intending to standardize the images generated by diagnostic modalities. The second is born with the aim to facilitate and strengthen post-processing analysis[7].

DICOM is the acronym of Digital Imaging and COmmunications in medicine. It is not only a file format but also a network communication protocol[7]. 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[7]. 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[7]. It also contains patient information such as name, gender, age, weight, and height. 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 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*. Filtering creates 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 transform $g(x, y)$ is the linear combination of the mask coefficient $w(s, t)$ and the pixels of the image affected by the mask itself. In general, we can write:

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

1.2.1 Smoothing Filters

Smoothing filters are used for blurring and for noise reduction[10]. This is used in removal of small details and bridging of small gaps in lines or curves. Smoothing spatial filters include *linear filters* and *nonlinear filters*[10].
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 $n = 2b + 1$.

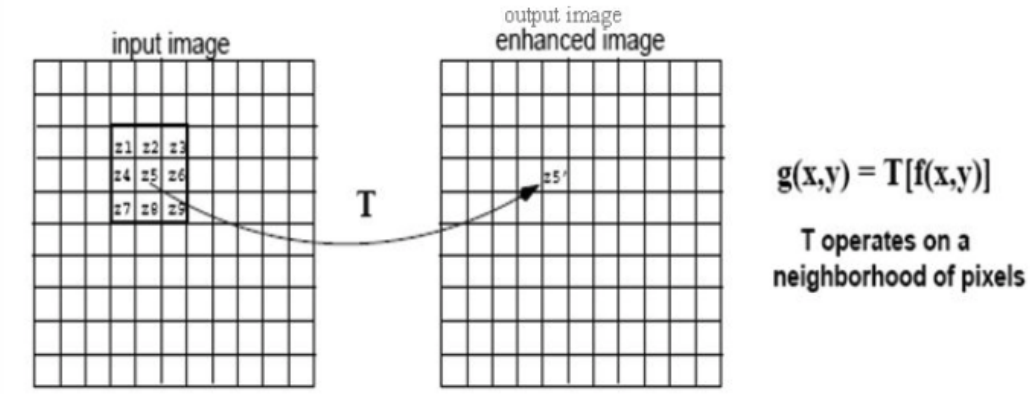

Figure 1.1: Example of spatial filtering. A filtered image is generated as the center of the mask or kernel, moves to every pixel in the input image. From [9]

Linear filtering is based on the *mean filter* [11]. The mean filter 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. These filters are also called *low pass filters* since the process of averaging drastically lowers high frequencies. The mask or kernel is a square. Larger kernels (5×5 or 7×7) produce more denoising that smaller ones (3×3) but make the image more blurred[11]. A common mean filter can be described by a 3×3 matrix with all elements equal to 1, so that the output pixel corresponds to a value of:

$$R = \frac{1}{9}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \mathbf{z} = \frac{1}{9}\sum_{i=1}^{9} z_i \tag{1.4}$$

or using a weighted mean filter:

$$R' = \frac{1}{16}\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix} \mathbf{z} \tag{1.5}$$

Non-Linear filtering is based on the *median filter*[11]. The median filter principle is similar to the mean filter. The mask or kernel is scanned over the pixels of the entire image. The median of the pixel values in the mask region is calculated, and the center pixel of the mask region is replaced with the calculated median value[11]. This filter is particularly effective in the presence of *impulse noise* (or *salt-and-pepper noise*)[10]. Mathematically:

$$g(p) = median\{f(p), where\, p \in N_8(p)\} \tag{1.6}$$

where $g(p)$ is the median pixel value, $f(p)$ all pixel values under mask, and $N_8(p)$ 8-neighborhood of pixel p.

Notes: Adaptive filters are commonly used in image processing to enhance or restore data by removing noise without significantly blurring the structures in the image[12]. This means not smoothing the areas of the image in which there is a large jump in intensity values (i.e. when there is an *edge*) and at the same time applying the filter to lower the noise. In this case, the local variance will be evaluated concerning the variance of the noise that occurs.
Mathematically:

$$\hat{f}(x,y) = f(x,y) - \frac{\sigma^2_{noise}}{\sigma^2_{local}}[f(x,y) - m_{local}] \tag{1.7}$$

1.3 Segmentation

Image segmentation is the partitioning of an image into non-overlapping consistent regions that are homogeneous to some characteristics, such as intensity or texture[8]. The results of segmentation can be used to perform image feature extraction, which provides 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. Segmentation, depending on the technique, can be manual, semi-manual, or automatic:

Manual techniques are still the most reliable and precise methods but they are time-consuming, highly operator-dependent, and subject to operator expertise[2].

Semi-Manual techniques are faster compared to manual ones. They are based on traditional image processing methods such as thresholding and clustering. However, despite the time savings they are operator-dependent[2].

Automatic techniques are the faster methods compared to manual and semi-manual ones. Moreover, they are not operator-dependent. However, the implementation of the algorithms is harder to perform[2].

1.3.1 Segmentation Methods Review

During the years several segmentation methods have been developed[8]. 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...[8].
Among the most common ones we found:

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[8].

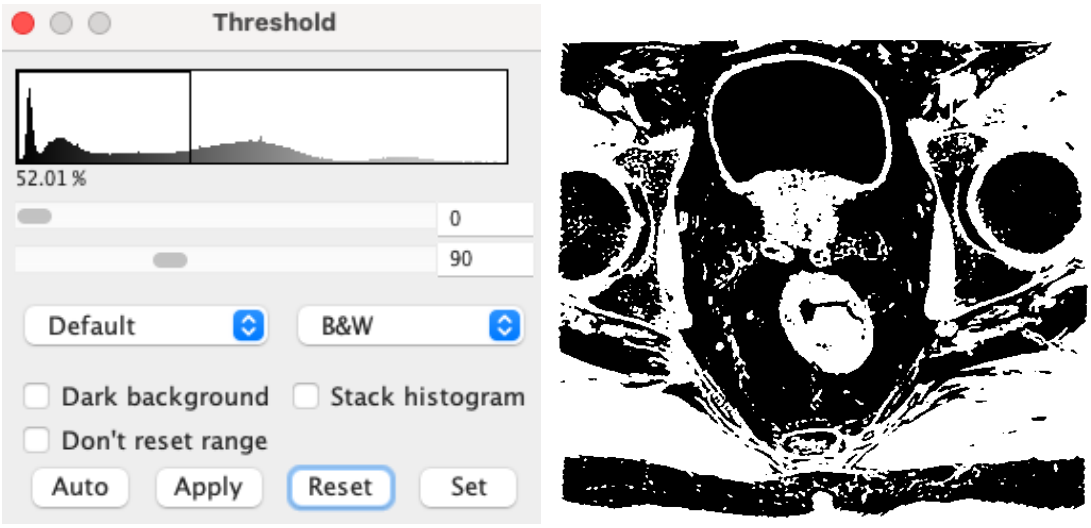

Figure 1.2: Example of thresholding segmentation on a Magnetic Resonance image of a patient affected by colorectal cancer using Fiji software[13].
Left): Image Histogram and thresholding settings. *Right):* Result of thresholding.

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. Convolutional Neural Networks architectures consist of an *input layer*, *hidden layers* and an *output layer*. The hidden layers include convolutional layers. As the convolution kernel slides along the input matrix for the layer, the convolution operation generates a feature map, which in turn contributes to the input of the next layer[14]. This process can be followed by other kind of layers such as *Pooling layers*, *Activation layers*, *Fully Connected layers* as shown in Figure1.3. The Pooling layer aims to reduce the dimensions of the feature maps. The Activation layer consists in a pixel-wise non-linear function, usually the Rectified Linear Unit (ReLU). The Fully Connected layer connects every neuron in the previous layer to the neuron in the next layer. This aims to collect all the relevant deep features for the classification. 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. The most commonly used functions for this purpose are Sigmoid for binary predictions and Softmax for multi-labels ones:

$$Sigmoid(x) = \frac{1}{1+e^{-x}} \qquad Softmax(\mathbf{z})_i = \frac{e^{\mathbf{z}_i}}{\sum_{j=1}^{K} e^{\mathbf{z}_j}} \tag{1.8}$$

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). The loss function affects the training phase, by evaluating the error rate. This function must be minimized. Among the most common functions used there are the binary cross-entropy (BCE) used for binary classification and the categorical cross-entropy (CCE):

$$BCE(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1-y_i) \cdot log(1-p(y_i)) \tag{1.9}$$

$$CCE(q) = \sum_{i=1}^{K} q(y_i) \cdot log(p(y_i)) \tag{1.10}$$

where for N data points and K classes, y_i is the truth label and $p(y_i)$ is the Softmax or Sigmoid probability for the i^{th} class.
The performance of the network is achieved by a function called metric. Metric functions are similar to loss function but they do not have to be minimized. Among the metrics,

one of the most used for segmentation purposes is the Dice Similarity Coefficient (DSC):

$$DSC = \frac{2 \mid X \cap Y \mid}{\mid X \mid + \mid Y \mid} \tag{1.11}$$

where $\mid X \mid$ and $\mid Y \mid$ are the cardinalities of the given sets.
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[15], is one of the most common and used architecture.

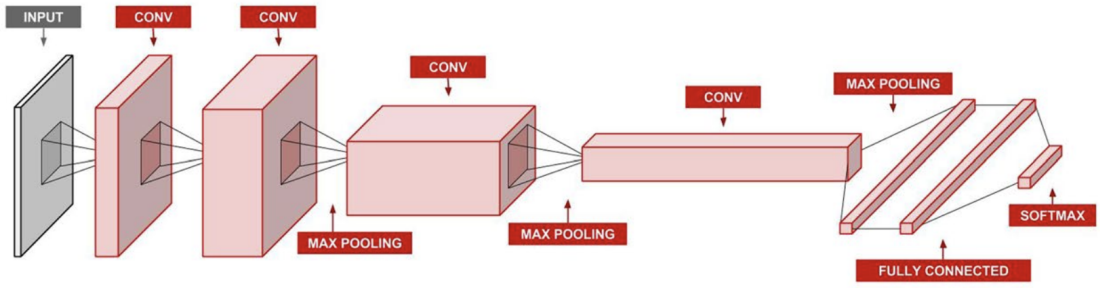

Figure 1.3: Example of Convolutional Neural Network architecture. As you can see the architecture is made by a series of layers. The input layer is followed by convolutional and max-pooling layers. Then, the fully connected layer and softmax one complete the structure. From [5].

U-Net

The U-net is a convolutional network architecture for fast and precise segmentation of images especially in the biomedical field[15]. 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 Figure1.4, 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 to with the aim of reconstruct 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 features extracted in the contraction path with tha spatial information by a sequence of transpose convolutions (or up-convolutions) and concatenations (grey arrows in Figure1.4).

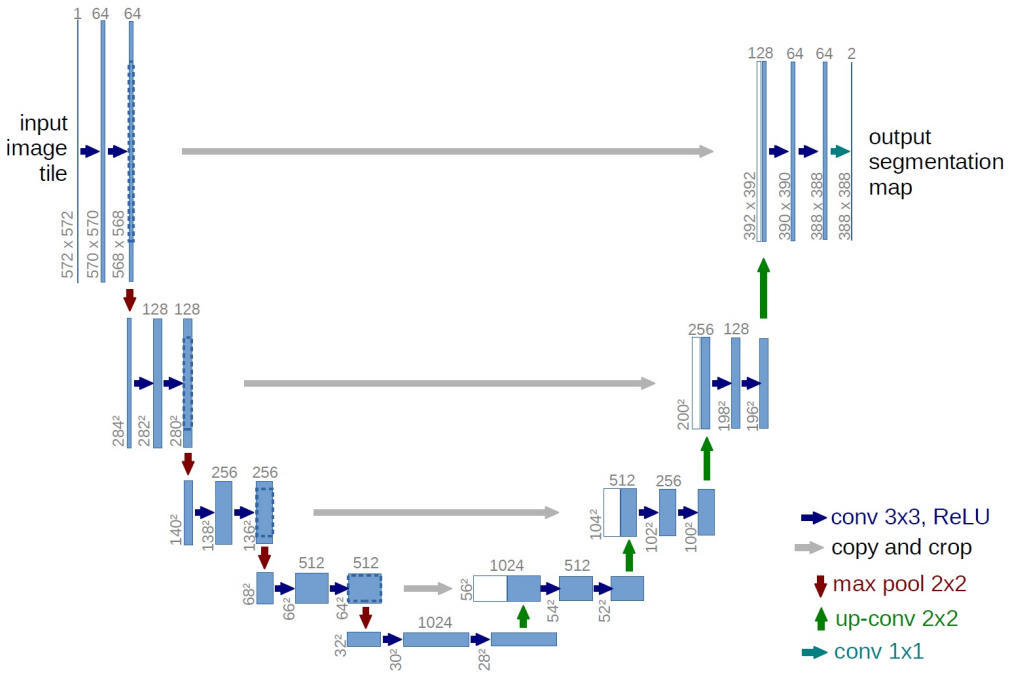

Figure 1.4: Original U-Net architecture. We can notice the U-shape made by the expansion and contraction path. Each blue box corresponds to a series of feature maps. White boxes represent copied feature maps. The arrows denote the different operations. From[15].

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[16]. 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[2]. 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. [17]. 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[2].

Prediction of metastases: Radiomic features can also predict the metastatic potential of tumors. For example, many radiomic features were identified as predictors of distant metastasis of lung adenocarcinoma in a study by Coroller et al.[18]. 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.

Genetic evaluation of cancer: The biological mechanisms of colorectal cancer were studied for the construction of different imaging models. In particular, It has been showed that radiomic features can be associated with some biological genes[2].

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"[2].

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. Its many application areas include data compression, image analysis, pattern recognition, regression and classification prediction[19].
The most common definition of PCA, due to Hotelling, is that, for a set of observed d-dimensional 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. It can be demonstrated that 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 $\mathbf{t}_n$ [19]. 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.

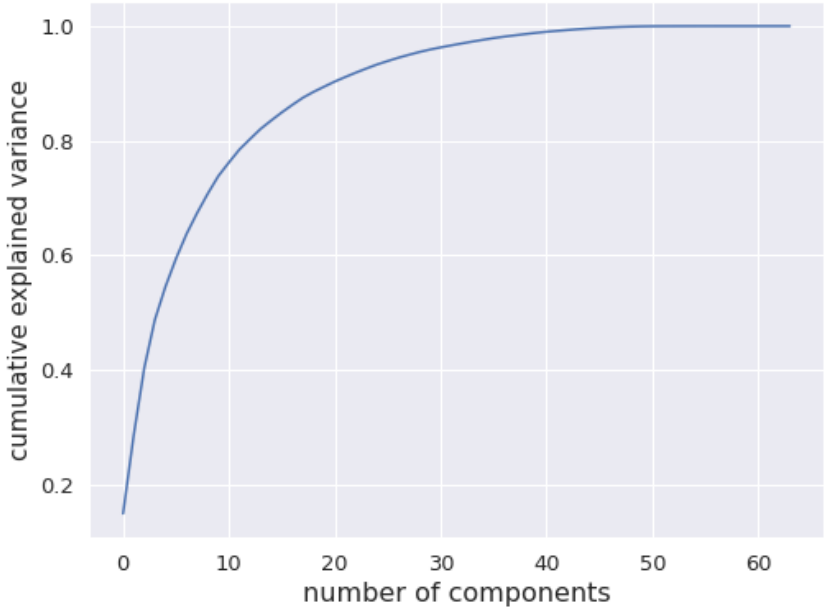

Figure 1.5: Example of cumulative explained variance ratio as a function of the number of components. This curve quantifies how much of the total variance is contained within the first N components. We can see that the first 10 components contain approximately 75 % of the total variance, while to the 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[20, 21].
For a SVC, mathematically, given training vectors $\mathbf{x}_i \in \mathbb{R}^p$, $i \in \{1, \ldots, n\}$, in two classes, 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 [20].
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 finding a non linear decision boundary in the input space[22].
A SVC solves a primal problem and a dual problem. The primal:

$$\min_{w,\zeta} \frac{1}{2}(\mathbf{w}^T\mathbf{w}) + C\sum_{i=1}^{n}\zeta_i \tag{1.12}$$

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

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[20]. 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:

$$\min_{\alpha} \frac{1}{2}(\mathbf{a}^T\mathbf{Q}\mathbf{a}) - \mathbf{e}^T\mathbf{a} \tag{1.13}$$

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

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:

$$\sum_{i \in SV} y_i a_i \mathbf{K}(\mathbf{x_i}, \mathbf{z}) + b \tag{1.14}$$

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[20].

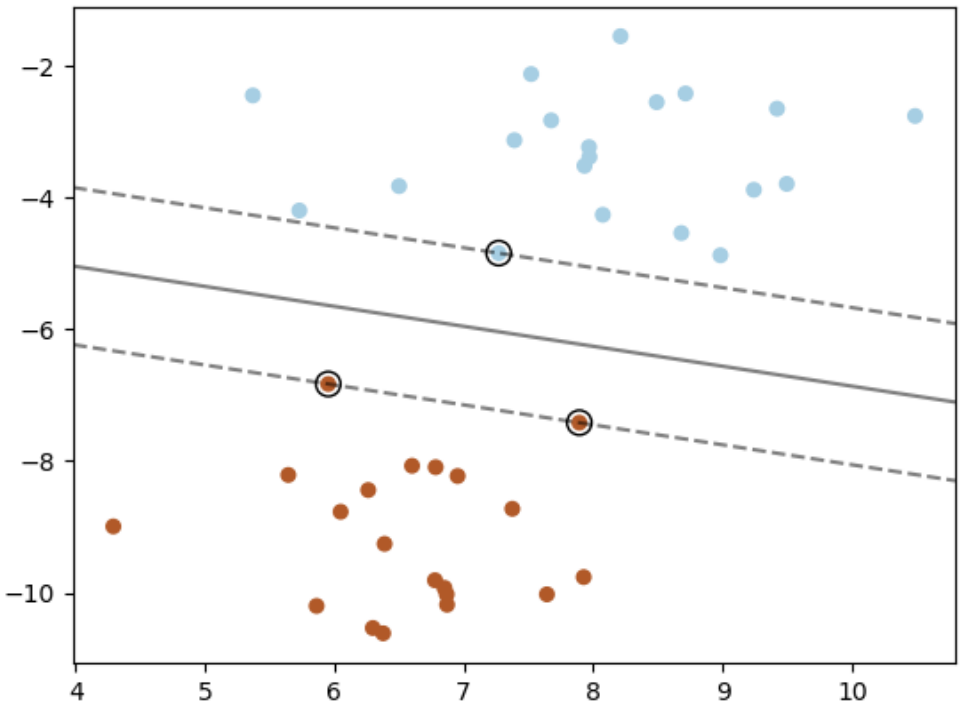

Figure 1.6: Classification made by a Support Vector Classifier (SVC) for a linearly separable problem. The gray line represents the line of the separation hyper-plane. The three samples on the margin boundaries (dashed lines) are called *support vectors*. From [20]

Chapter 2

Pipeline

The aim of this project is to implement an automated pipeline based on automatic segmentation of T2 weighted Magnetic Resonance (MR) images exploiting Convolutional Neural Networks in order to predict the response to neoadjuvant chemo-radiotherapy of colorectal cancer by using radiomic features.

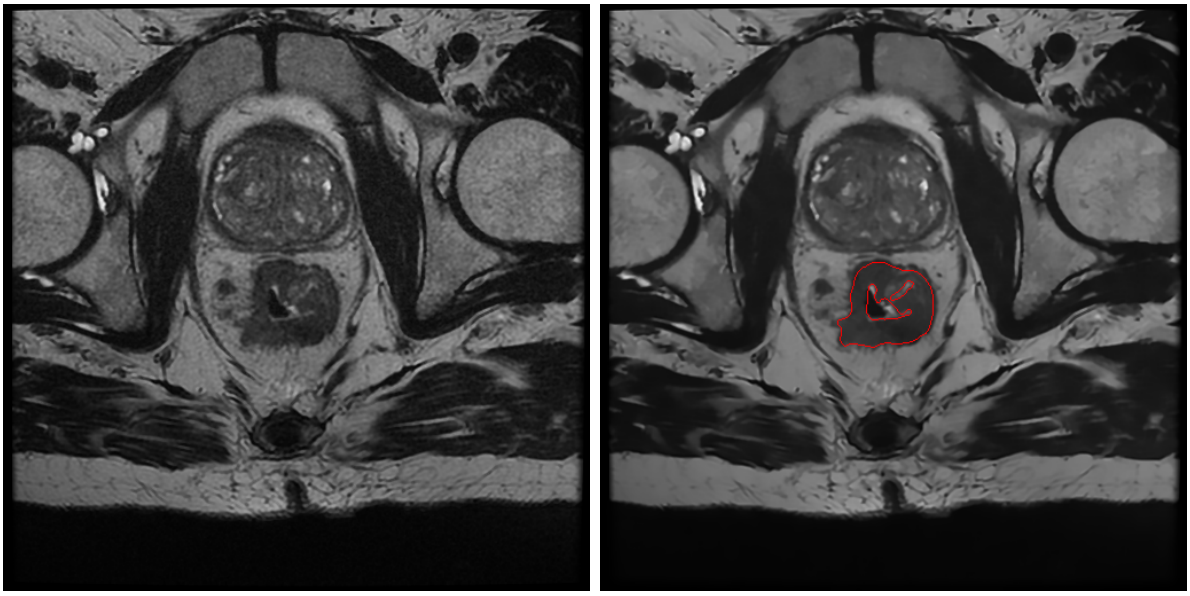

Figure 2.1: *Left)* Original MR image of a patient affected by colorectal cancer. *Right)* The same image with identified tumor area.

The starting point is the MRI scans. Firstly, I started with the visualization and the pre-processing of the scans. The pre-processing techniques consists of a smoothing filter to remove noise which could be a potential source of false positives and gamma correction to increase the brightness of the images. 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. The input images consisted of the MRI scans and the medical annotation as ground-truth images. Once trained the CNN model and segmented the images to obtain the colorectal cancer region, the next step was the extraction of the radiomic features. For each patient's examination in the dataset, I extracted 100 radiomic features. The features were then analyzed to implement a predictive model to obtain the prediction of response. The prediction is based on the Tumor Regression Grading (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 were pre-processed for the training of a Convolutional Neural Network model. After the segmentation of the images, radiomic features were extracted from the images. They were analyzed to implement a predictive model to obtain 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 obtained. As consequence, the final structure of the pipeline looks like in the following Figure 2.3

Figure 2.3: Final pipeline structure. From left to right. The MRI scans are pre-processed with a smoothing filter to remove noise and with a gamma correction to increase the brightness. Then, the segmentation is achieved by the trained CNN model. After the segmentation, radiomic features are extracted. Finally, the prediction is obtained by the implemented predictive model.

2.1 Description

We have seen that the pipeline development consists of various steps. In this section I will describe how each step of the pipeline was achieved. The first one is the pre-processing. Then, the training of the Convolutional Neural Network and the segmentation will be described following. After that, the description of radiomic features extraction and analysis. Finally the prediction of response.
This section is aimed only to the description since the implementation will be treated 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 a smoothing filter to remove noise which can be a source of false positive in the classification of pixels and a gamma correction filtering to increase the brightness of the images.
First of all, for each patient, the MRI scans are read with the original pixel depth, that is 16-bit integers, 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, required to work with TensorFlow[23] and Keras API[24]. Moreover, if the image size does not match the most common one which is 512×512 pixels, the image is resized to the latter in order to have the same image size for all the images.
The images are filtered to remove noise, using the non-local means algorithm from Scikit-Image[25] library. 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[25]. In Figure 2.4, you can see the result 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 in less noise and is 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) presents 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 denoising, the image brightness is enhanced using a gamma correction. It consists of a non-linear operation, defined in the simplest cases, by the following power-law expression:

$$I_{out} = CI_{in}^{\gamma} \tag{2.1}$$

where the output I_{out} is obtained multiplying by a constant C the input value I_{in} raised to the power γ. In this case, the gamma correction was achieved by the following expression:

$$I_{out} = (\frac{I_{in} - I_{min}}{I_{max} - I_{min}})^{\gamma} \tag{2.2}$$

where I_{min}, I_{max} are respectively the minimum and the maximum image value. The reason for the gamma correction is, as above mentioned, to increase the brightness of the image, especially for the center of the image which is our region of interest (i.e. the tumor region) and it is often quite dark. The result of the gamma correction can be seen in Figure 2.6. The increase of brightness is also visible on the gamma-corrected image histogram, shown in Figure 2.7.
In summary, the preprocessing steps are:

- normalization and rescaling
- denoising
- gamma correction

The comparison of the image before and after the pre-processing can be seen in Figure2.8.

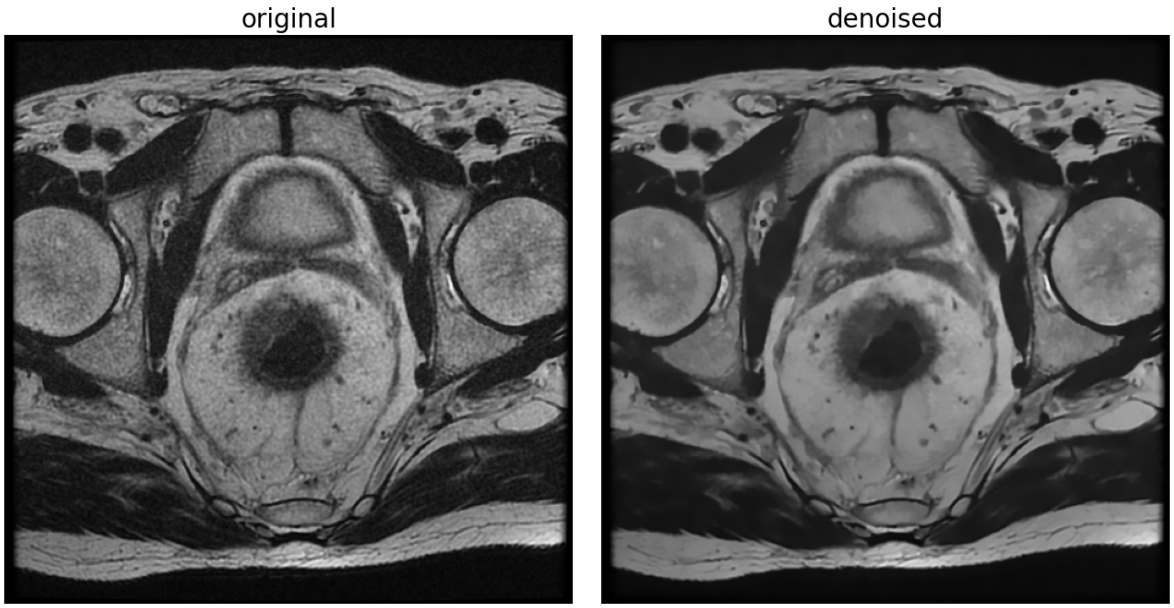

Figure 2.4: *Left)* Original MR image of a patient affected by colorectal cancer. *Right)* The same image after non-local mean algorithm.

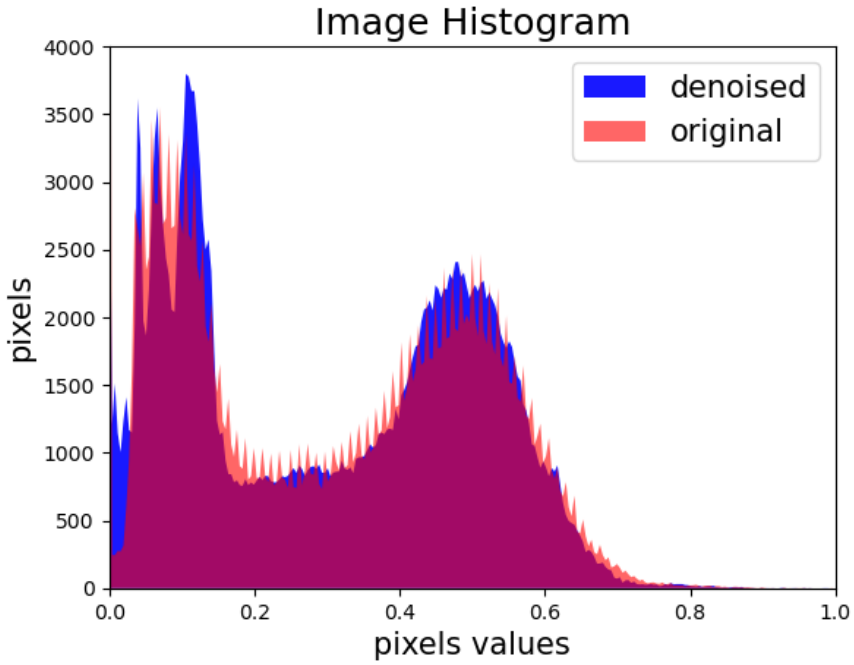

Figure 2.5: Original vs denoised image histogram. The original image histogram (red) presents great fluctuations of pixel intensity, highlighting the presence of noise. The denoised image one (blue) results in a smoothed histogram preserving the original shape.

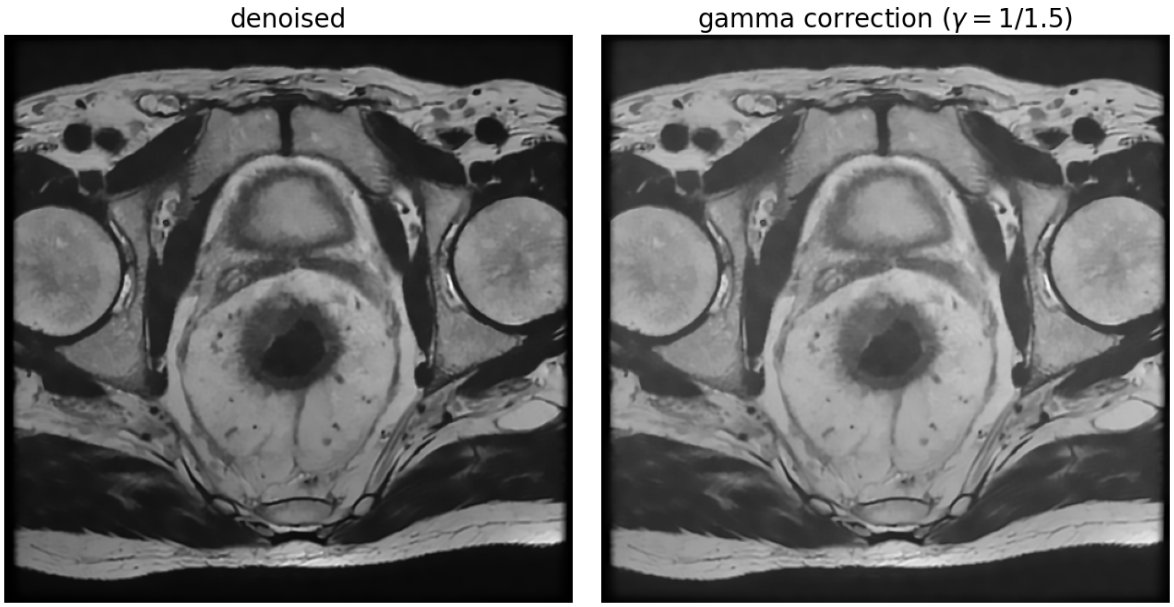

Figure 2.6: *Left)* denoised MR image of a patient affected by colorectal cancer. *Right)* The same image after gamma correction.

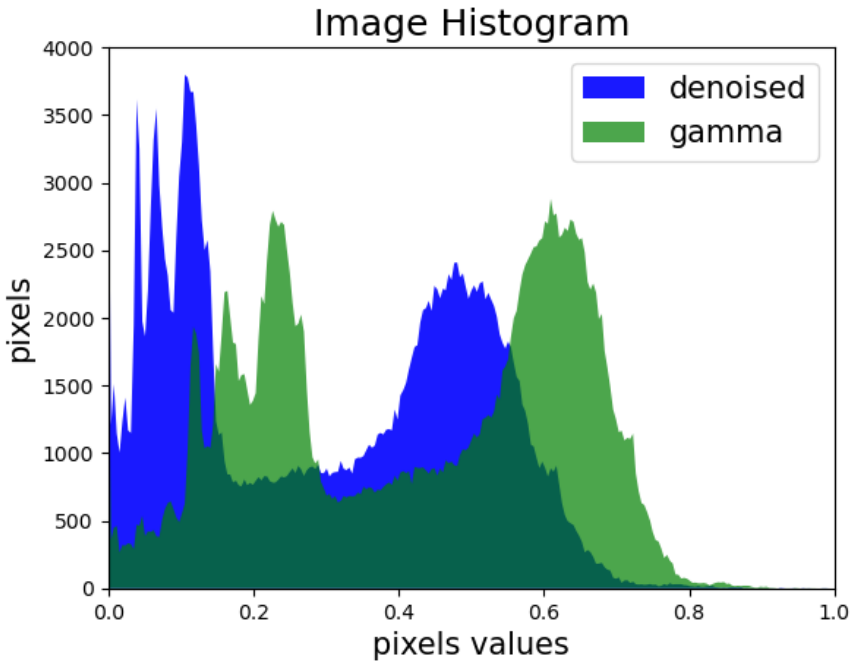

Figure 2.7: Denoised vs gamma-corrected image histogram. The gamma-corrected image histogram (green) results in a right-shifted histogram, thus in increased brightness.

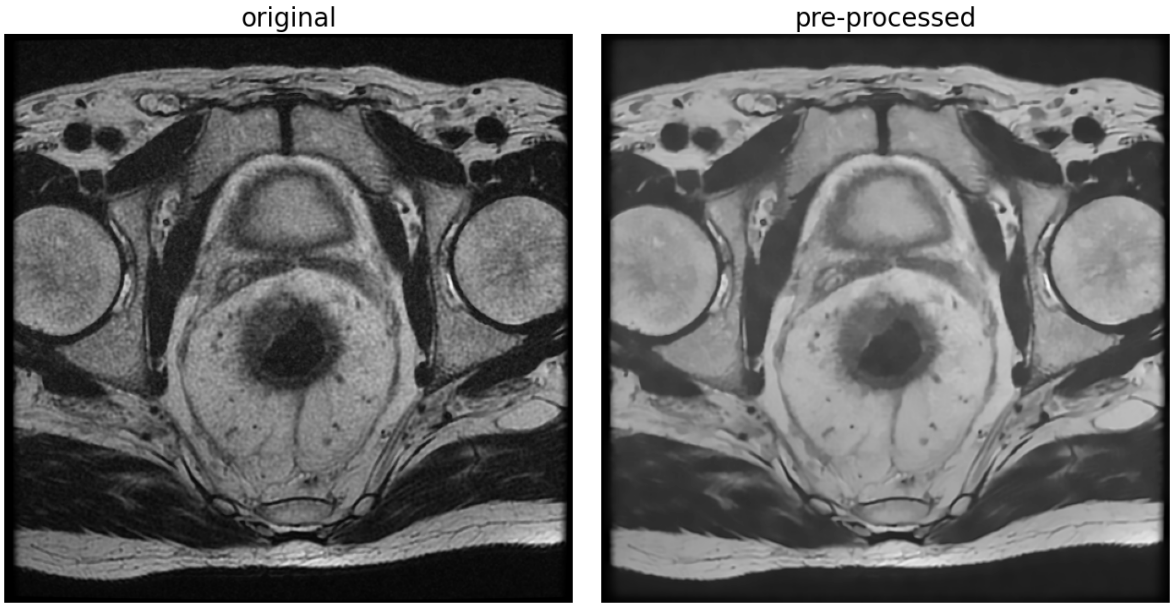

Figure 2.8: *Left)* Original MR image of a patient affected by colorectal cancer. *Right)* The same image after the pre-processing steps.

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. In the end, from 48 patients the remaining ones were 37. 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 (i.e. medical annotations) stored as 8-bit unsigned integers images. The training was performed with a custom data generator which was responsible for providing the right input images and ground-truth images, pre-processing them, performing data augmentation, and splitting the images into training and validation sets. In particular, a total of 488 images was split for the training (391 images) and validation set (97 images). Some data augmentation was performed on the training set of data, to overcome the lack of data and to reduce overfitting. It consisted 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 vertically flipped. The ground-truth images

were also normalized and rescaled to binary floating-point 32-bit, required to work with TensorFlow[23] functions.
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 ground truth 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. As you can see the image size is 512×512 pixels and the images are horizontal or vertical flip. The white area on the ground-truth images represents the tumor area.
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 concept, in Figure 1.4, 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*[26].
The loss function used consists of the combination between Dice loss with the binary focal loss[27]. The former comes from the Dice coefficient or Dice-Sørensen coefficient (DSC):

$$1 - DSC = 1 - \frac{2 \mid X \cap Y \mid}{\mid X \mid + \mid Y \mid} \tag{2.3}$$

Where $\mid X \mid$ and $\mid Y \mid$ are the cardinalities of the two sets (i.e. the ground-truth and the prediction).
The latter instead:

$$FL = -\alpha y(1 - p(y))^{\gamma} log(p(y)) - \alpha(1 - y)p(y)^{\gamma} log(1 - p(y)) \tag{2.4}$$

where y is the the ground-truth, $p(y)$ is the the prediction, α is weighting factor and γ a focusing parameter.
Combining the two methods allows for some diversity in the 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 of the trained model is the Dice coefficient (DSC), which was the most used in literature for this purpose, thus for having a metric of reference.

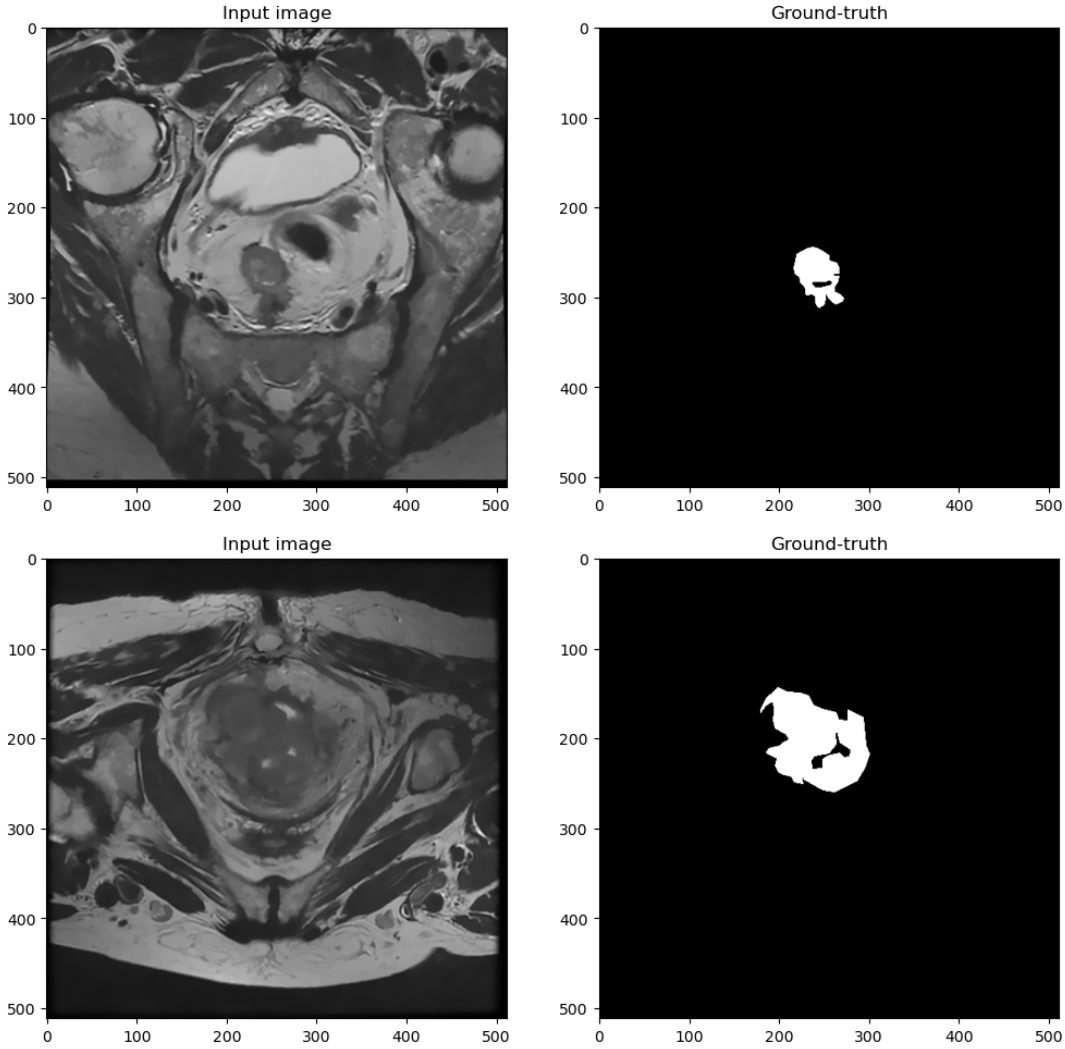

Figure 2.9: Example of input and ground-truth images from the training set. The images are randomly horizontal or vertical flipped. The white area on the ground-truth images represents the tumor area.

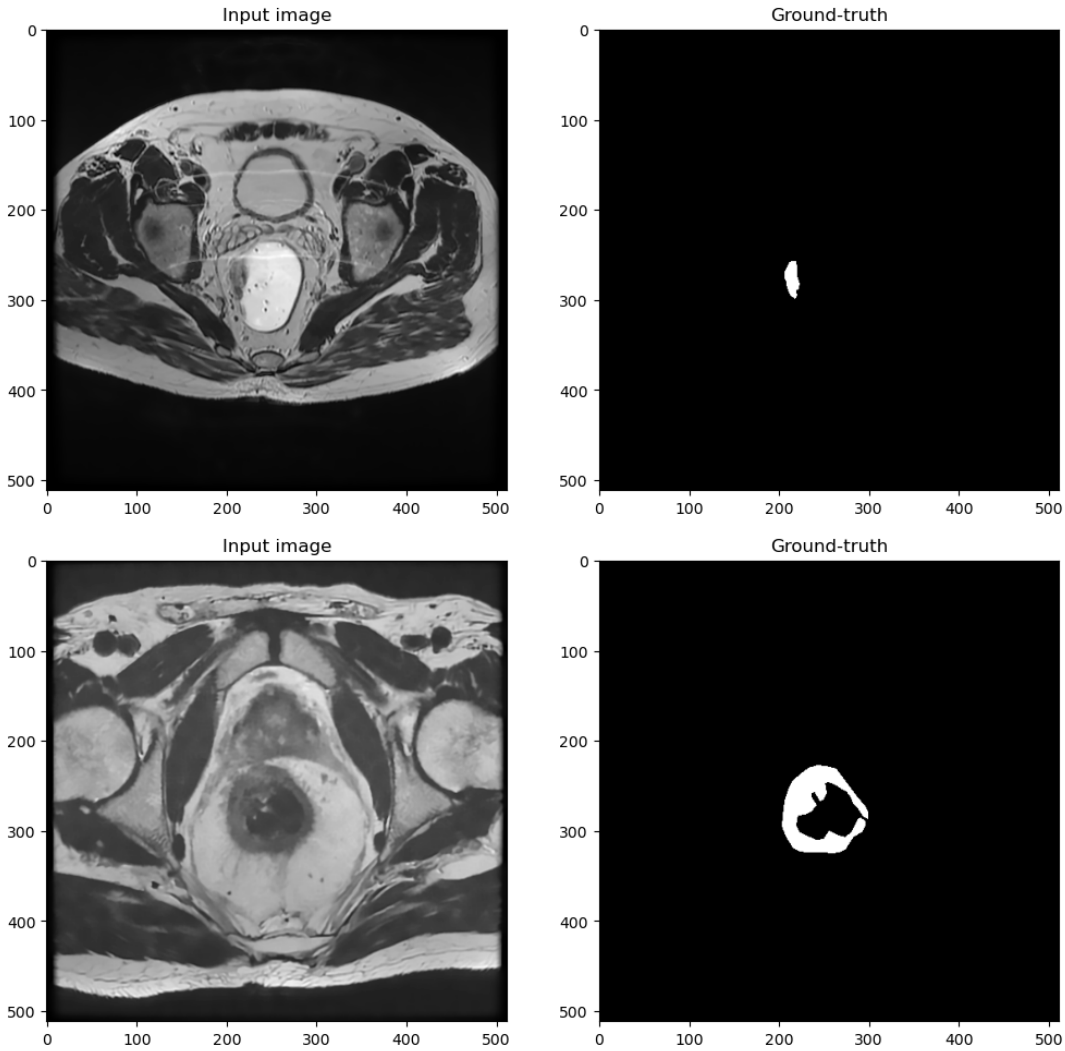

Figure 2.10: Example of input and ground-truth images from the validation set. The white area on the ground-truth images represents the tumor area.

2.1.3 Segmentation

Once trained, the CNN model was used for the segmentation of the MRI scans of each patient. As mentioned previously, before the segmentation, the scans are pre-processed to reduce noise, for the gamma correction, and to check if the image size is 512×512. During the pre-processing, as we know, the scans are also rescaled to binary floating-point 32-bit, required to work with TensorFlow, since the model is trained on images of that kind.
The segmentation is done for each stack of slices of the patient using the trained CNN model to obtain the prediction.
Then, using OpenCV[28] 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.

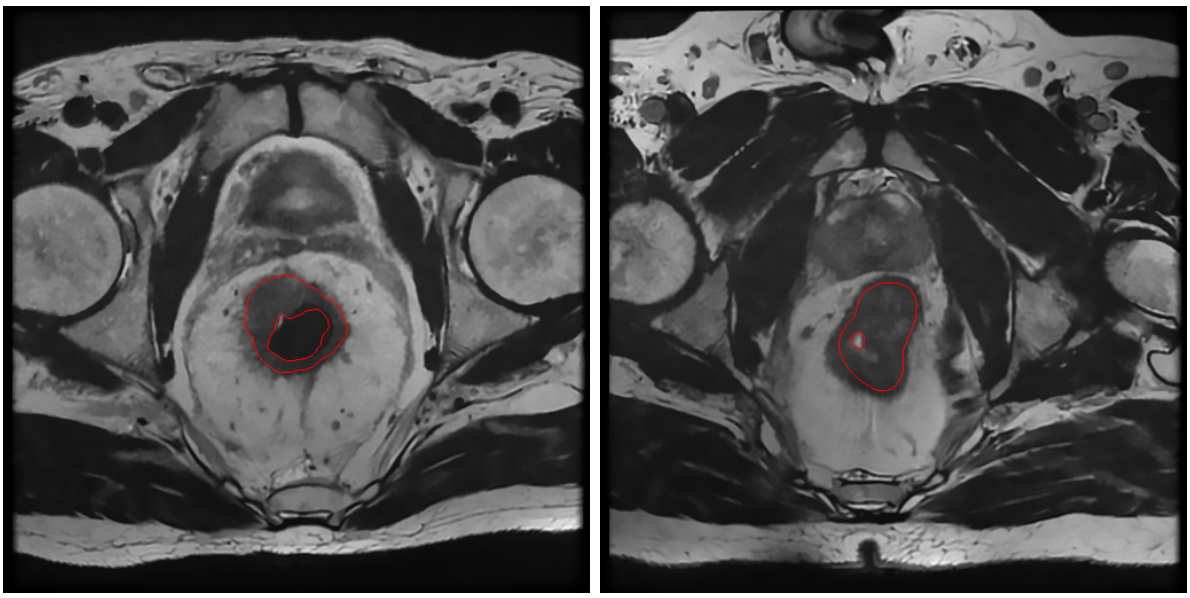

Figure 2.11: Images of colorectal cancer with identified tumor areas from two different patients. The red contour represents the border of the predicted tumor area.

2.1.4 Features Extraction

This step consists of the extraction of the radiomic features from the images. For this purpose the MRI scans of each patient and the segmented images are saved, by using SimpleITK library [29], as 3D images in a particular format (.nrrd) required by Pyradiomics library [30] to extract features from 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, the features have been read, looking for some correlation between features since the goal is to reduce as much as possible the number of irrelevant features. As you can see from Figure2.12, showing the features correlation heatmap, there is the presence of highly correlated (and anti-correlated) features, highlighted by the red (and blue) color. 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.
In order to reduce the number of features, Principal Component Analysis (PCA) was performed on the data frame containing the extracted 100 features for each patient. The number of resulting features is 6, corresponding to the 90% of the total variance. Since Principal Components (PCs) are linear combinations of the original variables, it is not possible to recover the original features name. However, thanks to the attribute `components_` of the Scikit-learn library[31], that outputs an array of shape [`n_components`, `n_features`], it is possible to get how components are related to the original features and each coefficient represents the correlation between a particular component and feature. In figure 2.13, you can see for each principal component (on the right), which is the original feature (on the left) having the maximum correlation coefficient with. In particular:

- 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

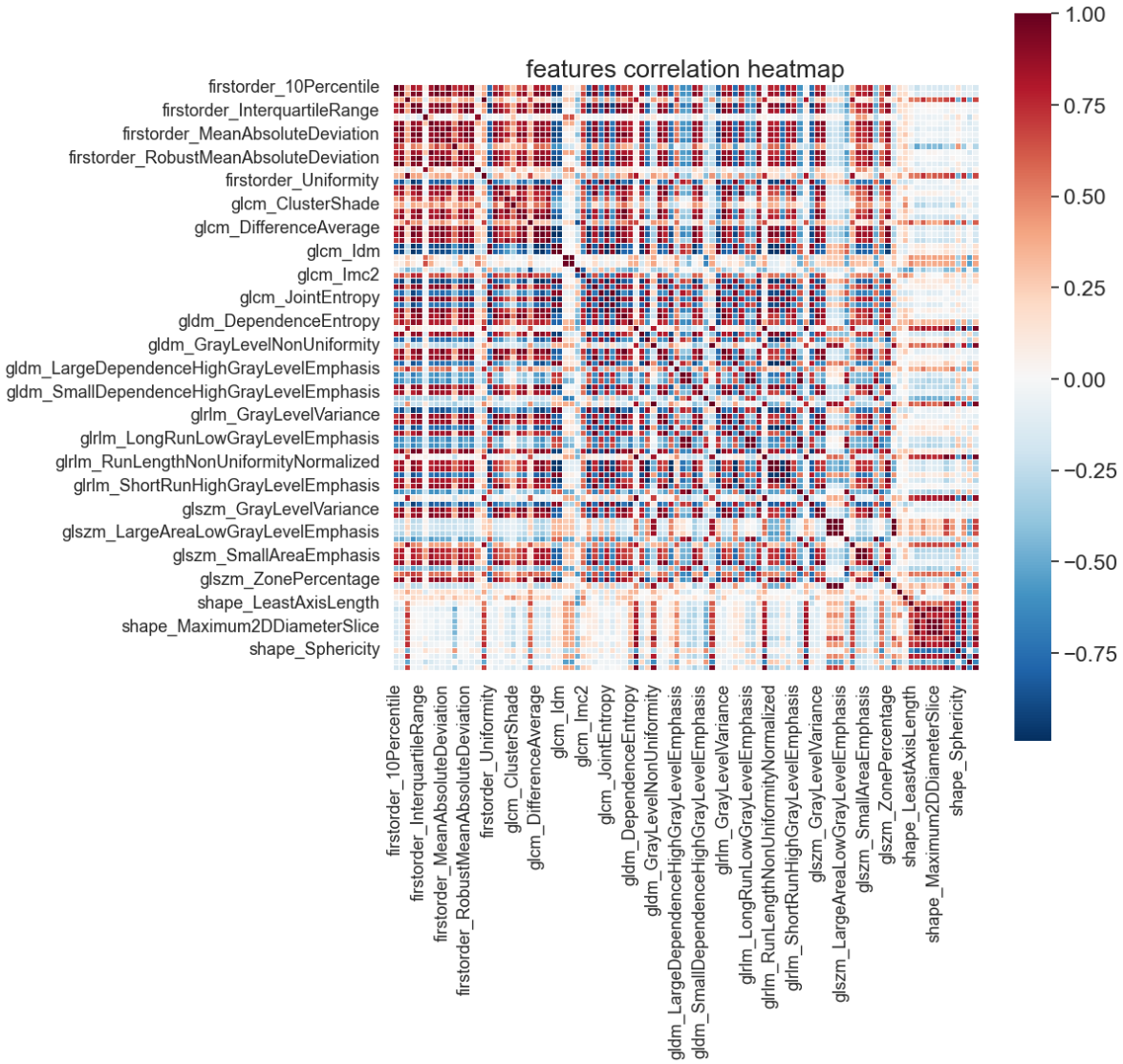

Figure 2.12: Features correlation heatmap. The red color indicates a correlation, the white color indicates no correlation and the blue color indicates anti-correlation between features. *Notes:* not every feature name has been plotted since 100 features caused the font size to be too small.

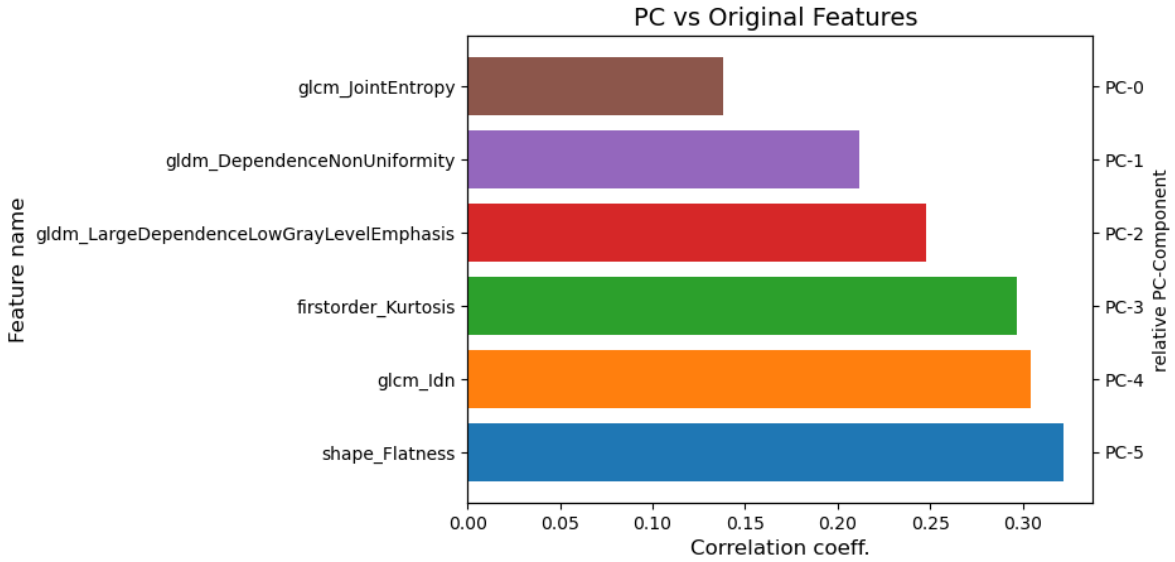

Figure 2.13: For each principal component (on the right), you can see which is the original feature (on the left) having the maximum correlation coefficient with.

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 neo-adjuvant therapy[2]. 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 Classifier has been made. The classifier consists of a *pipeline* object which is made by the given estimators from the Scikit-learn library:

- `StandardScaler`: to standardize the data
- `PCA`: to perform PCA as described previously
- `SVC`: Support Vector Classifier

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. In the end, the total number of patients was 32. Moreover, since the lack of much data TRG values were binarized into two main classes: 0 and 1. Class 0 means a complete response to the neo-adjuvant 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 that are not equally distributed.
The Classifier model was then trained with Cross-validation (using 10 folds) to avoid the presence of *bias* during the split into training and test set. The Matthews correlation

coefficient (MCC) was used as a measure of the quality of classifications between the true label and the model prediction. It takes into account true and false positives and negatives and is generally regarded as a balanced measure that can be used even if the classes are of very different sizes. 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. In particular, for Cross-validation it was used the `StratifiedKfold` (i.e. object that provides train/test indices to split data in train/test sets) from Scikit-learn that gives the median value, $median = 0.55$, of the MCC distribution (Figure 2.15), obtained after 500 simulations.

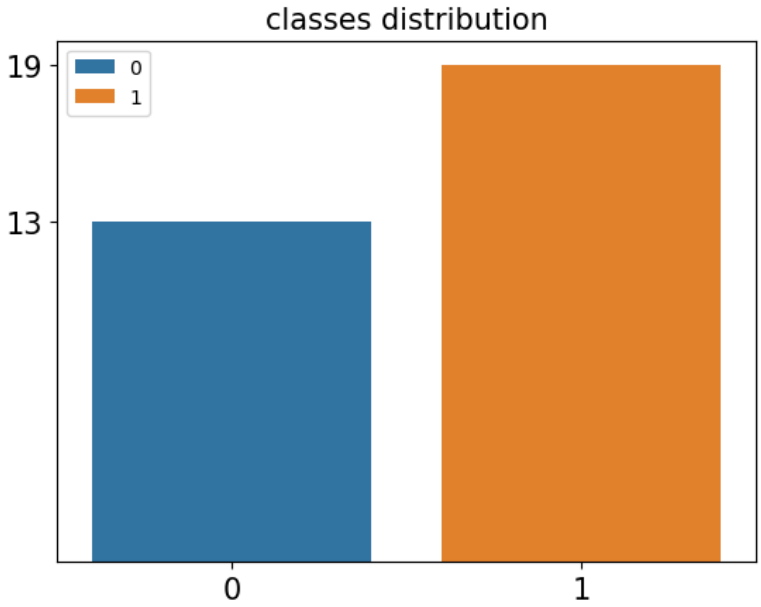

Figure 2.14: Distribution of the classes. Class 0 means a complete response to the neo-adjuvant chemo-radiotherapy (TRG values $\in$ [0, 1]) while class 1 means a moderate response (TRG values $\in$ [2, 3])

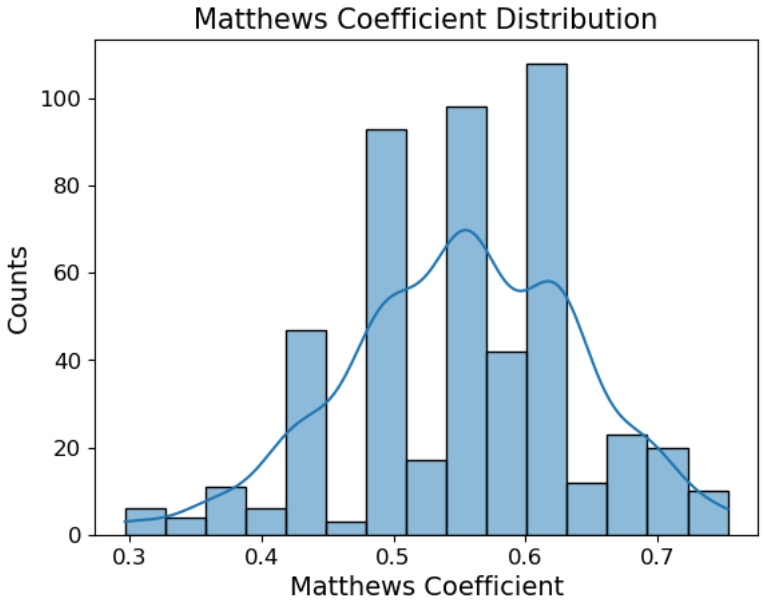

Figure 2.15: Matthews Correlation coefficient distribution. Obtained after 500 simulations, changing the `random_state` (i.e. the randomness) of the `StratifiedKfold` (i.e. object that provides train/test indices to split data in train/test sets) from Scikit-learn.

2.2 Implementation

I have implemented the pipeline described before 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 as anticipated in the description section.
The whole code is open-source and available on GitHub [32] and the relative documentation, made using Sphinx, 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 scripts that provide a fast way to handle DICOM series and ROI 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 bash, 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 to show the implementation of the main steps of the described pipeline.

2.2.1 Pre-processing

As we said in the Description section, the pre-processing consists of the application of a smoothing filter and a 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

def get_slices(dir_path)

    files = glob.glob(dir_path + "/*.dcm")

    # ordering as istance number
    z = [float(pydicom.read_file(f, force=True).get(
        "InstanceNumber", "0") - 1) for f in files]
    order = np.argsort(z)
    files = np.asarray(files)[order]

    slices = [read_slices(f) for f in files]
    slices = np.asarray(slices)

    return slices
```

Listing 2.1: `get_slices` implementation

For this purpose, I used the NUMPY [33] and PYDICOM [34] 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 *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 steps described in the Description section:

- normalization and rescaling
- denoising
- gamma correction

For this purpose, I used the OpenCV [28] and Scikit-Image [25] libraries:

```
import cv2
from skimage.restoration import denoise_nl_means, estimate_sigma

def rescale(img):
    rescaled = cv2.normalize(img, dst=None, alpha=0, beta=1, norm_type=
   cv2.NORM_MINMAX, dtype=cv2.CV_32F)
    return rescaled


def denoise(img, alpha=10):

    patch_kw = dict(patch_size=5, patch_distance=6,)
    sigma_est = np.mean(estimate_sigma(img))
    denoised = denoise_nl_means(img, h=alpha * sigma_est, sigma=
   sigma_est, fast_mode=True, **patch_kw)
    return denoised


def gamma_correction(img, gamma=1.0):
    igamma = 1.0 / gamma
    imin, imax = img.min(), img.max()

    img_c = img.copy()
    img_c = ((img_c - imin) / (imax - imin)) ** igamma
    img_c = img_c * (imax - imin) + imin
    return img_c



def pre_processing_data(slices, alpha=10):

    imgs = []
    for layer in range(slices.shape[0]):
        img = slices[layer, :, :]
        if slices.shape[1:3] != 512:
            resized = cv2.resize(img, (512, 512))
        else:
```

```
            resized = img
        rescaled = rescale(resized)
        denoised = denoise(rescaled, alpha)
        gamma = gamma_correction(denoised)
        imgs.append(gamma)

    images = [np.expand_dims(im, axis=-1) for im in imgs]
    images = np.array(images)

    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, thus increase/decrease the brightness of the image, depending on gamma. 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 a was performed using TENSORFLOW[23] and SEGMENTATION-MODELS API[35]. The core of the training process is the custom `DataGenerator`, which provides data, split them into training and validation set, perform pre-processing and data augmentation on the training set. 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` that are the paths of the directories containing the relative DICOM input and labels 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)
```

```
16
17          assert source_files.size == labels_files.size
18
19
20          source_files, labels_files = self.randomize(source_files,
    labels_files)
21
22          idx = np.arange(0, source_files.size)
23          np.random.shuffle(idx)
24
25          self._source_trainfiles = source_files[idx[int(source_files.
    size * validation_split):]]
26          self._labels_trainfiles = labels_files[idx[int(labels_files.
    size * validation_split):]]
27
28          self._source_valfiles = source_files[idx[:int(source_files.size
     * validation_split)]]
29          self._labels_valfiles = labels_files[idx[:int(labels_files.size
     * validation_split)]]
30
31          self.subset = subset
32
33          if self.subset == 'training':
34              self._num_data = self._source_trainfiles.size
35          elif self.subset == 'validation':
36              self._num_data = self._source_valfiles.size
37
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'
5  model = sm.Unet(BACKBONE, input_shape=(IMAGE_HEIGHT, IMAGE_WIDTH, 1),
       encoder_weights=None, activation='sigmoid')
6 
7  optimizer =  'adam'
8 
9  dice_loss = sm.losses.DiceLoss()
10 focal_loss = sm.losses.BinaryFocalLoss()
11 loss = dice_loss + (1 * focal_loss)
12 
13 def dice_coef(y_true, y_pred):
14     intersection = tf.reduce_sum(y_true * y_pred, axis=[1, 2, 3])
15     total = tf.reduce_sum(y_true, axis=[1, 2, 3]) + tf.reduce_sum(
       y_pred, axis=[1, 2, 3])
16     dice = tf.reduce_mean((2. * intersection + smooth) / (total + 1.))
17     return dice
18 
19 metrics = [ dice_coef ]
20 
21 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.

```
1
2  def predict_images(slices, model, pre_processing=False, t=0.1):
3
4      if pre_processing:
5          prep_slices = pre_processing_data(slices)
6      else:
7          prep_slices = slices
8
9      predicted_slices = np.zeros_like(prep_slices)
10
11     for layer in range(slices.shape[0]):
12         img = prep_slices[layer, ...]
13         predicted_slices[layer, ...] = model.predict(img[np.newaxis,
       ...])[...]
14         predicted_slices[layer, ...] = np.where(predicted_slices[layer,
        ...] <= 0.1, 0, predicted_slices[layer, ...])
15
16
17     return predicted_slices
```

Listing 2.5: prediction function implementation

2.2.3 Features Extraction and Analysis

For this purpose, I used the PYRADIOMICS library[30] 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, using SIMPLEITK library [29], required by PYRADIOMICS. From each patient a total of 100 radiomic features were extracted. The extraction settings were stored in a `Params.yaml` file required by PYRADIOMICS.

```
1  from radiomics import featureextractor
2  import pandas as pd
3  import numpy as np
4  import SimpleITK as sitk
5
6  params  = '../extras/Params.yaml'
7  extractor = featureextractor.RadiomicsFeatureExtractor(params)
8
9  features = {}
10
11 for patient in good_patients:
12
13     dirs = glob.glob(src + '/' + patient + '/*_NRRD')
14
15
```

```
    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 dictionary, which the key is the caseID of patient and the the name of the examination directory containing the dicom series. I started to visualize and sort the features for each patient and stored 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[31], 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 since each coefficient represents the correlation between a particular component and feature.

```
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[31]. 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)
    #M_coeffs[i] = MCC
    M_coeffs.append(MCC)

data = M_coeffs
median = np.argsort(data)[len(data)//2]
print(median)
data[median]

skf = StratifiedKFold(n_splits=n_splits, shuffle=True, random_state=
    median + 1)
y_pred = cross_val_predict(pipeline, X, y, cv=skf)

target_names = ['class 0', 'class 1']
print(classification_report(y, y_pred, target_names=target_names))
```

Listing 2.9: Prediction of response implementation

Chapter 3

Results

This Chapter will be aimed to show and discuss the results of the developed pipeline. First of all, a brief description of the Dataset provided by the IRCCS Sant'Orsola-Malpighi Policlinic will be provided. Then, the accuracy of the segmentation and of the prediction of response. Finally, the outputs of the implementation will be shown.

3.1 Dataset Description

The main dataset used for this project was provided by IRCCS Sant'Orsola-Malpighi Polyclinic. It consists of MRI from 48 patients affected by colorectal cancer undergoing neo-adjuvant 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 medical 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 Accuracy

In this section, I will discuss the results of the accuracy achieved by the implemented pipeline, for the segmentation and for the prediction of response.
First of all the training process, then the comparison between the results of segmentation and the medical annotation (ground-truth) will be provided. Then, for the prediction of response, the classification report, confusion matrix and ROC curve will be shown.

3.2.1 Segmentation

The accuracy metric for the segmentation results is given by the Dice Similarity Coefficient (DSC) evaluated on the validation set. 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 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 result obtained for the training set of data while the green one represents the result obtained for the validation set. As you can see, the blue and green curves almost overlap to the end, meaning the model generalizes quite well. In fact, when the distance between the two curves starts to increase, the model stops generalizing, resulting in the phenomenon of data overfitting.
In Table 3.2, you can see the comparison between the state of the art and the implemented pipeline about the accuracy. It must be said that the literature about MRI colorectal cancer segmentation 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: *mucinous* cases lower the performances
- Medical annotations
- Loss function

In fact, as showed by Jovana Panic at al.[36] mucinous cases can considerably affect the performances. *Mucinous* consists of tumor subtype 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. [5] showed that the DSC of the same model trained using medical annotations made by different experts can give different DSC scores: DSC=0.68 (expert 1) and DSC=0.70 (expert 2). Another sensitive factor is the loss function. As shown by

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

Yi. Jie Huang et al.[37], depending on the network's architectures and the loss functions you can have different performances.
Despite having not so much data and the presence of *mucinous* cases my implementation got a score of DSC = 0.71 which is consistent and among the highest of literature.

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 the art and the implemented pipeline about MRI colorectal cancer segmentation accuracy.

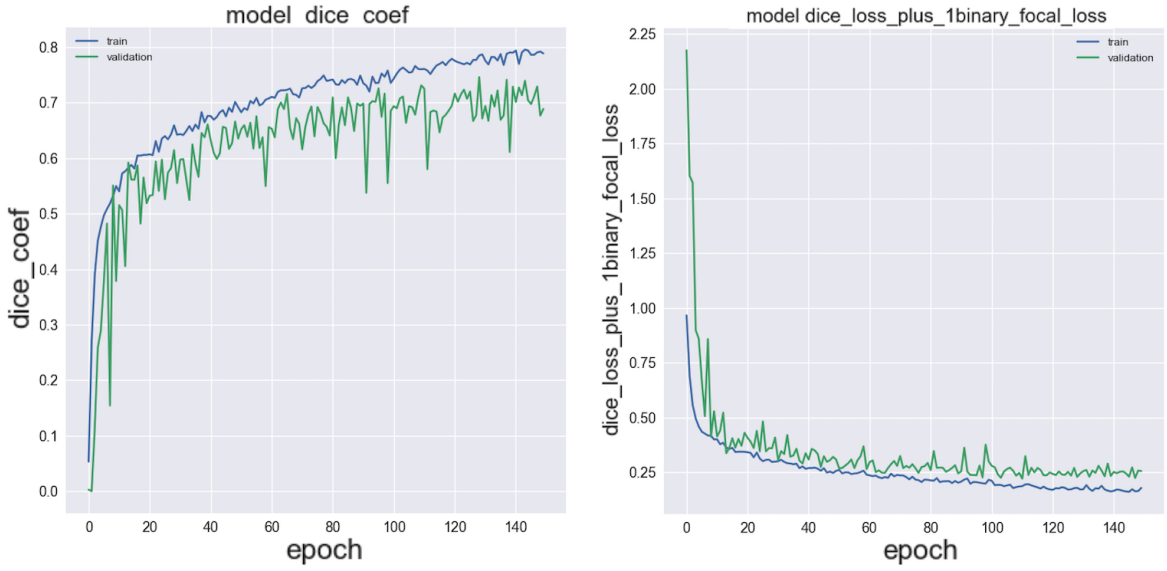

Figure 3.1: Training (blue) and validation (green) model history plots.*Left*): model dice coefficient as a function of the epochs.*Right*): model loss as a function of the epochs.

Comparison with Manual Annotations

To check the pipeline performances, I have also compared the obtained segmentation with the manual annotation (Ground-truth). 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 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. The segmentation performed by the model is consistent with the ground-truth one. 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 correct Region of Interest (ROI). Some of them 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.

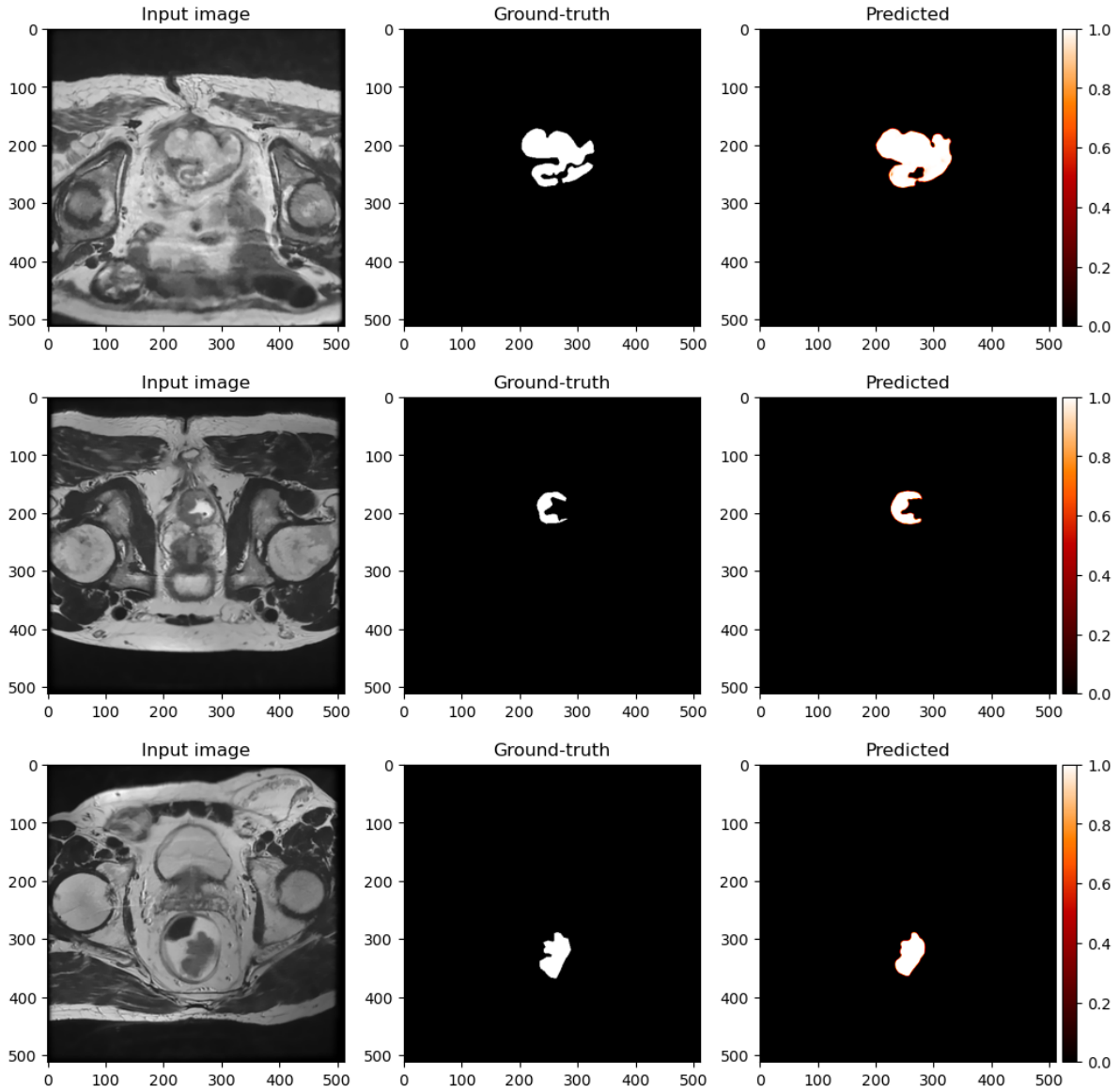

Figure 3.2: Comparison between the ground-truth image and the predicted one by the CNN model. The first column represents the input image. The second one represents the ground-truth image. Finally, the third one represents the predicted tumor area. The prediction is plotted as a probability density map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

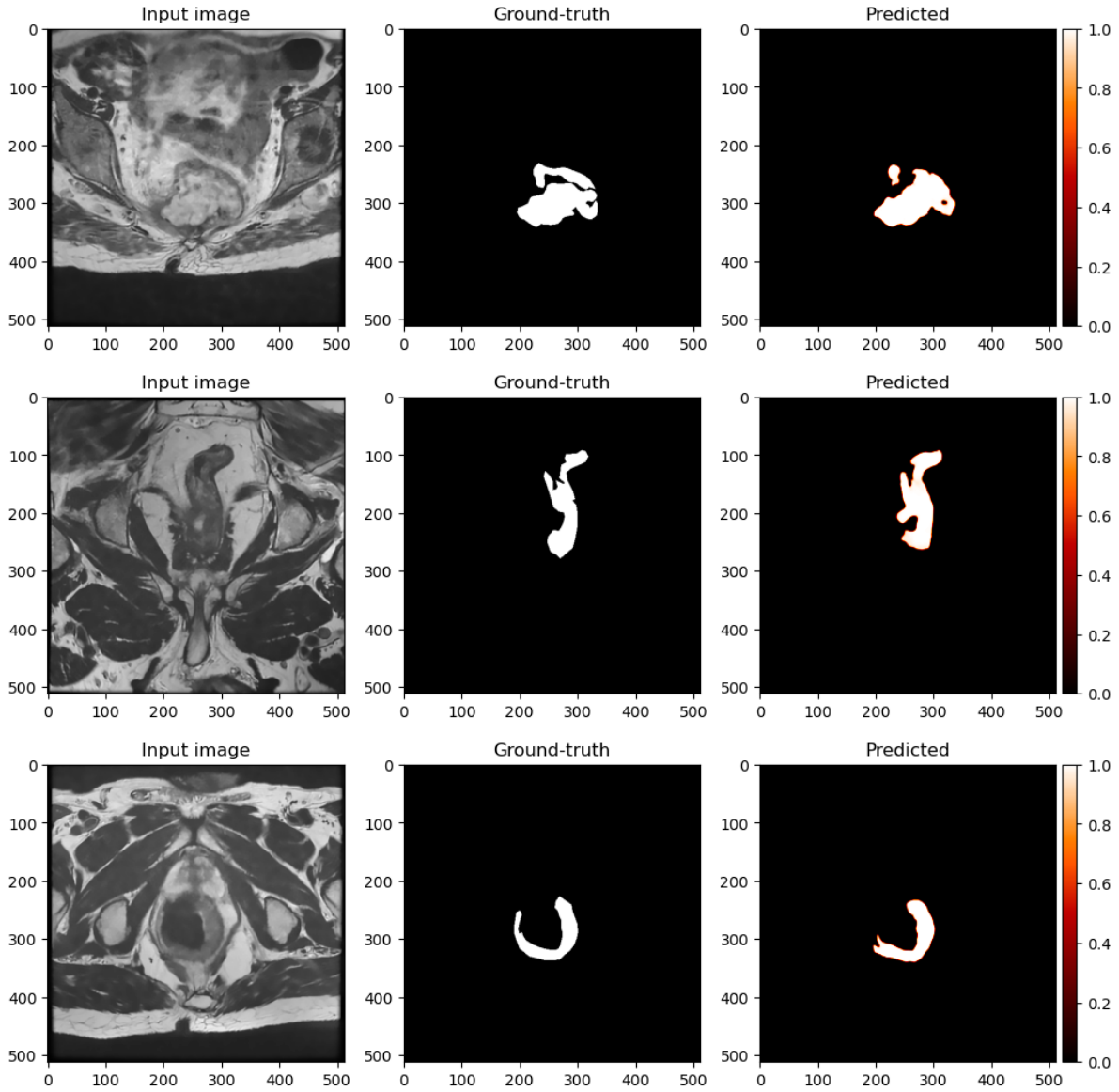

Figure 3.3: Comparison between the ground-truth image and the predicted one by the CNN model. The first column represents the input image. The second one represents the ground-truth image. Finally, the third one represents the predicted tumor area. The prediction is plotted as a probability density map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

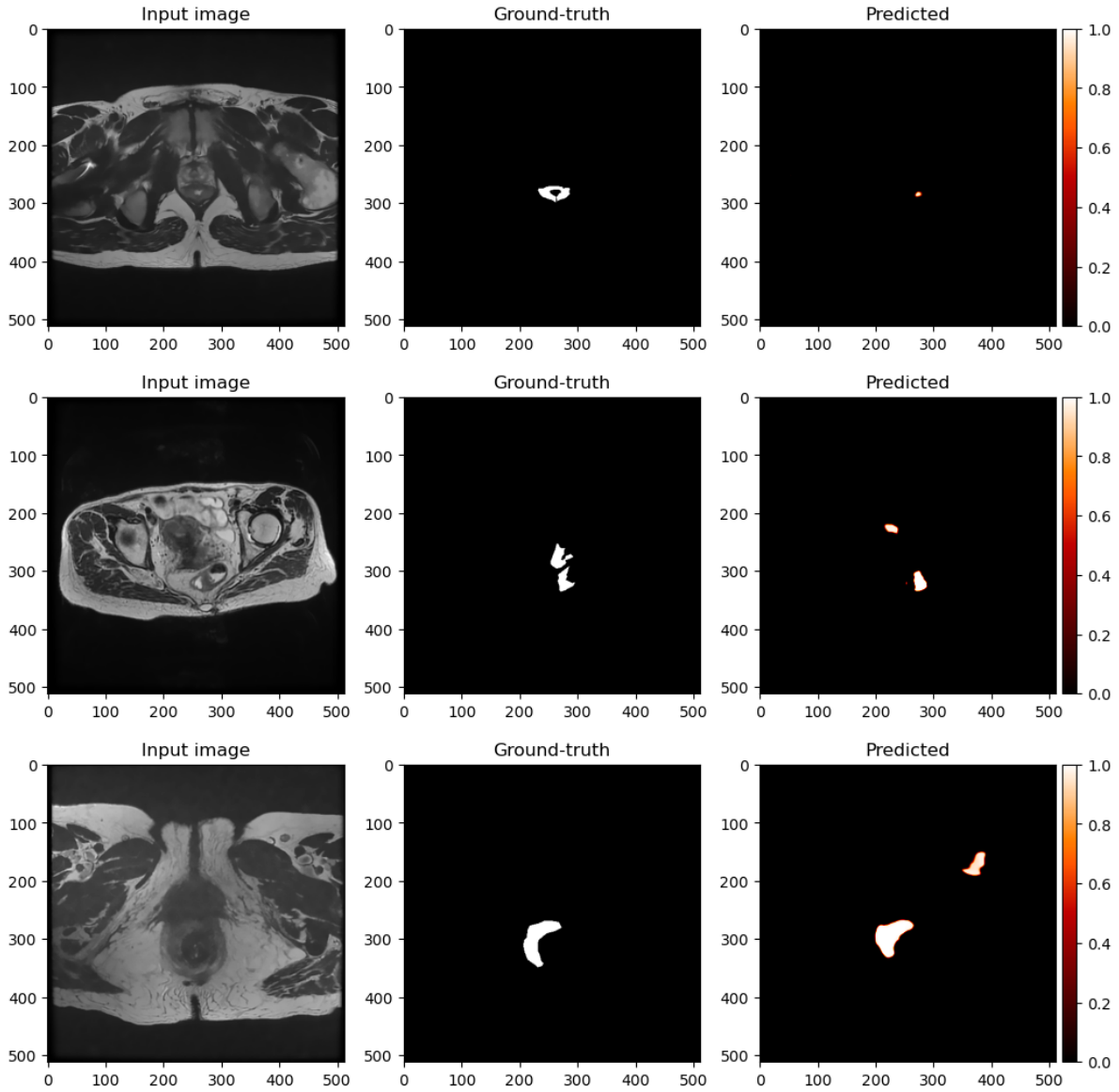

Figure 3.4: Comparison between the ground-truth image and the predicted one by the CNN model. The first column represents the input image. The second one represents the ground-truth image. Finally, the third one represents the predicted tumor area. The prediction is plotted as a probability density map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

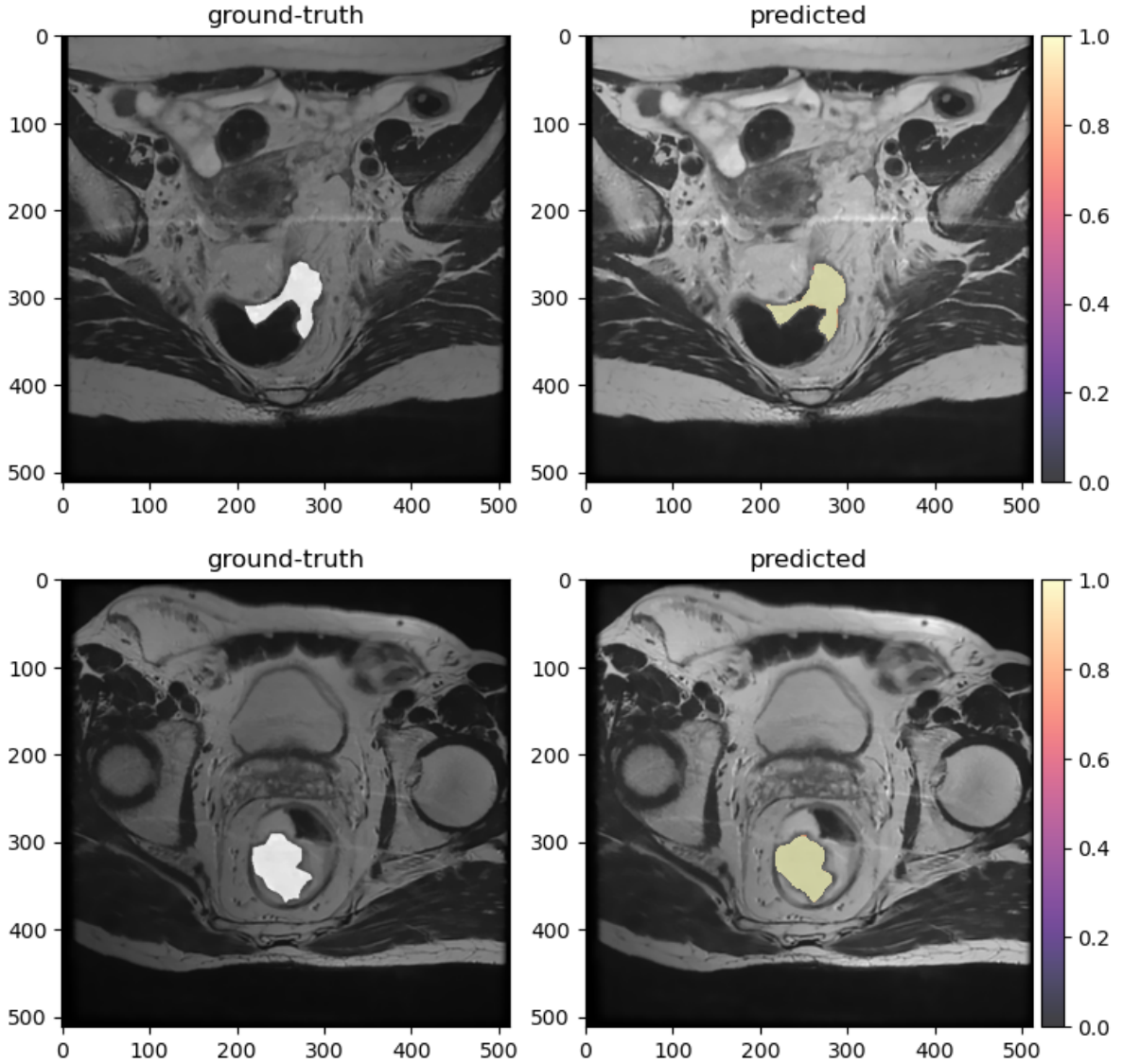

Figure 3.5: Comparison between the ground-truth image and the prediction over the original image. The prediction is plotted as a probability density map between 0. and 1. as shown by the sequential colormap. From training set.

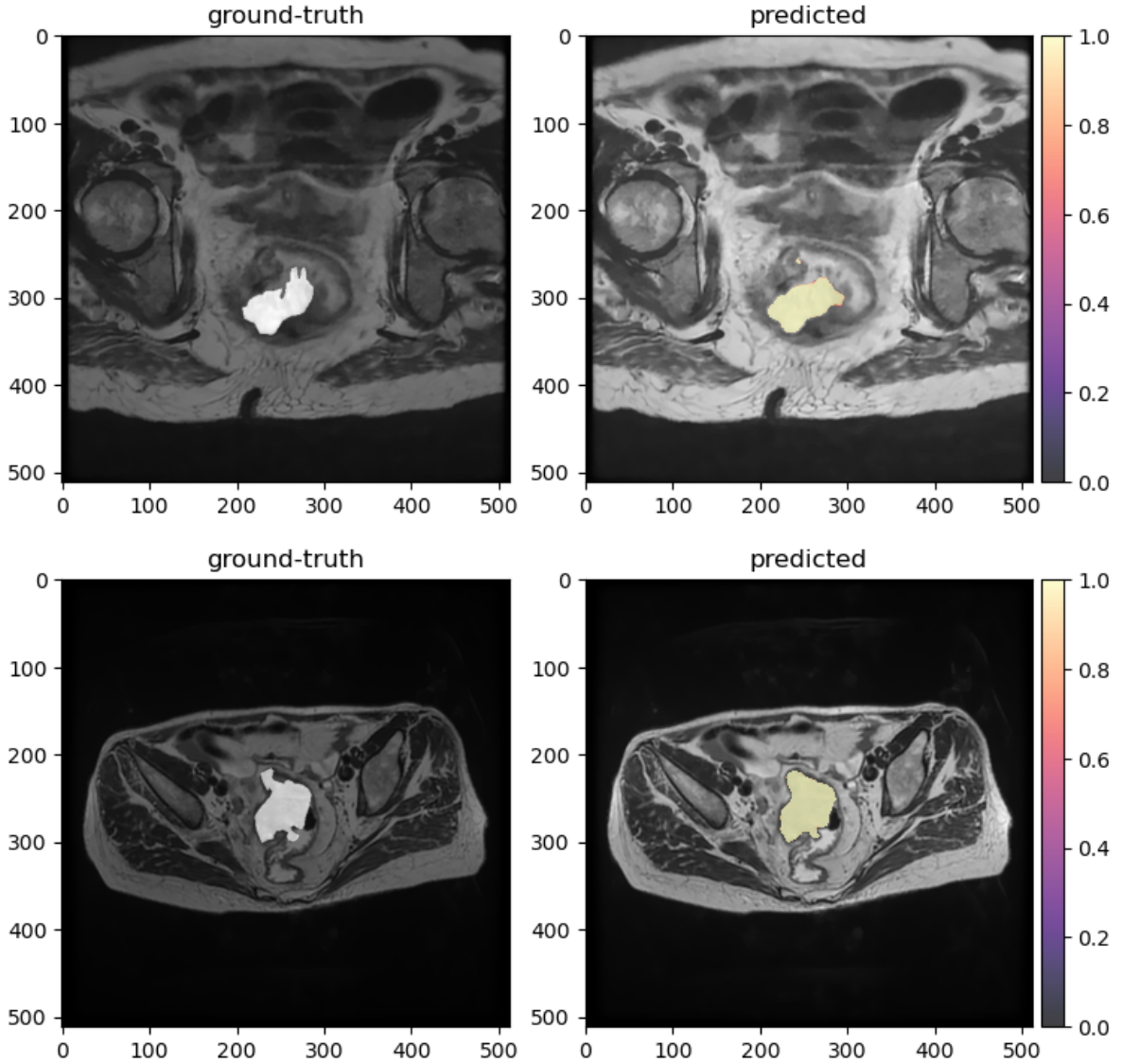

Figure 3.6: Comparison between the ground-truth image and the prediction over the original image. The prediction is plotted as a probability density map between 0. and 1. as shown by the sequential colormap. From training set.

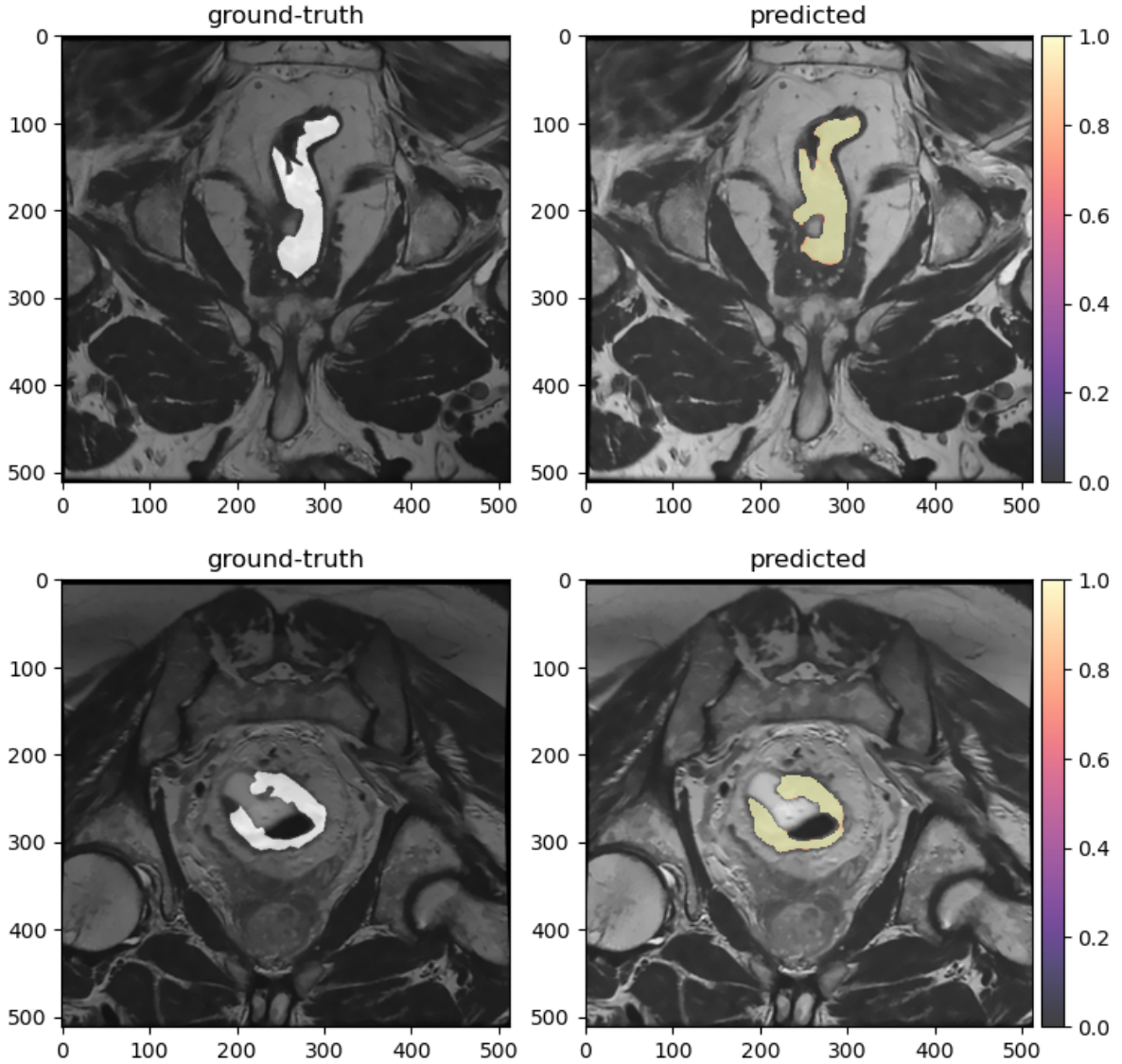

Figure 3.7: Comparison between the ground-truth image and the prediction over the original image. The prediction is plotted as a probability density map between 0. and 1. as shown by the sequential colormap. From validation set.

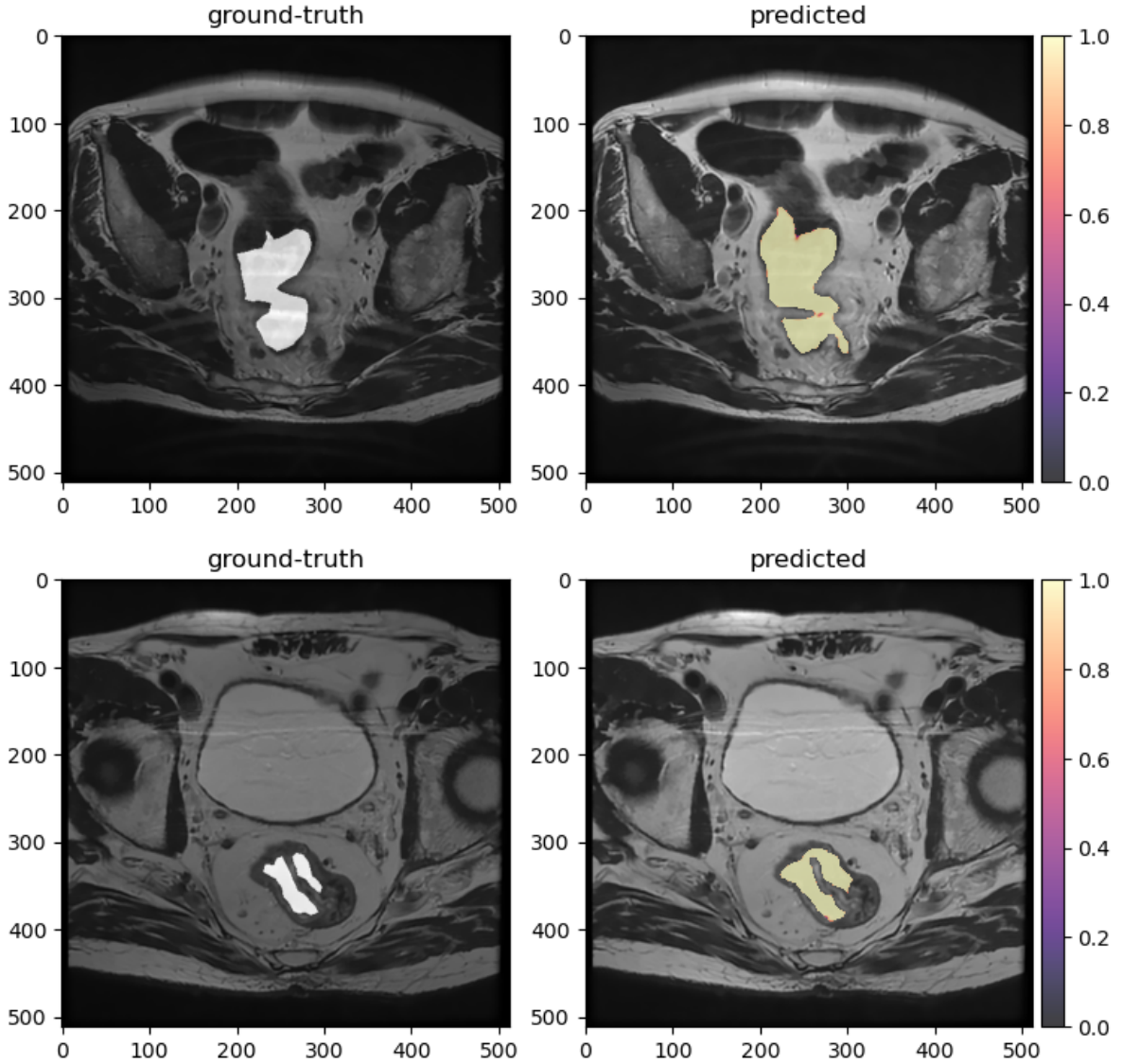

Figure 3.8: Comparison between the ground-truth image and the prediction over the original image. The prediction is plotted as a probability density map between 0. and 1. as shown by the sequential colormap. From validation set.

3.2.2 Prediction of Response

In order to measure the accuracy of the classification for each class (Class 0 for complete response and Class 1 for moderate response), I used different metrics:

Precision

Intuitively, it is the ability of the classifier not to label as positive a sample that is negative:

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

Recall

Intuitively, it is the ability of the classifier is the ability of the classifier to find all the positive samples:

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

F1-Score

It is the harmonic mean of the precision and recall:

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

where tp is the number of true positives and fn the number of false negatives.

The results are shown in Table 3.3, where for each class you get the relative score. The *support* column indicates the population for that class. As you can see, for Class 0 the scores are lower compared to Class 1 but also the *support* is lower since the cases of a complete response were less compare to the number of case of a moderate one. The sum of the *support* is 32 corresponding to the number of patients involved into the analysis.

	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

I also computed the confusion matrix (or error matrix), in Figure 3.9, to evaluate the accuracy of the classification. Each row of the matrix represents the instances in an actual class while each column represents the instances in a predicted class. As you can see from the confusion matrix, on the diagonal you have the number of well-classified instances while on the anti-diagonal the wrong-classified ones. 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.

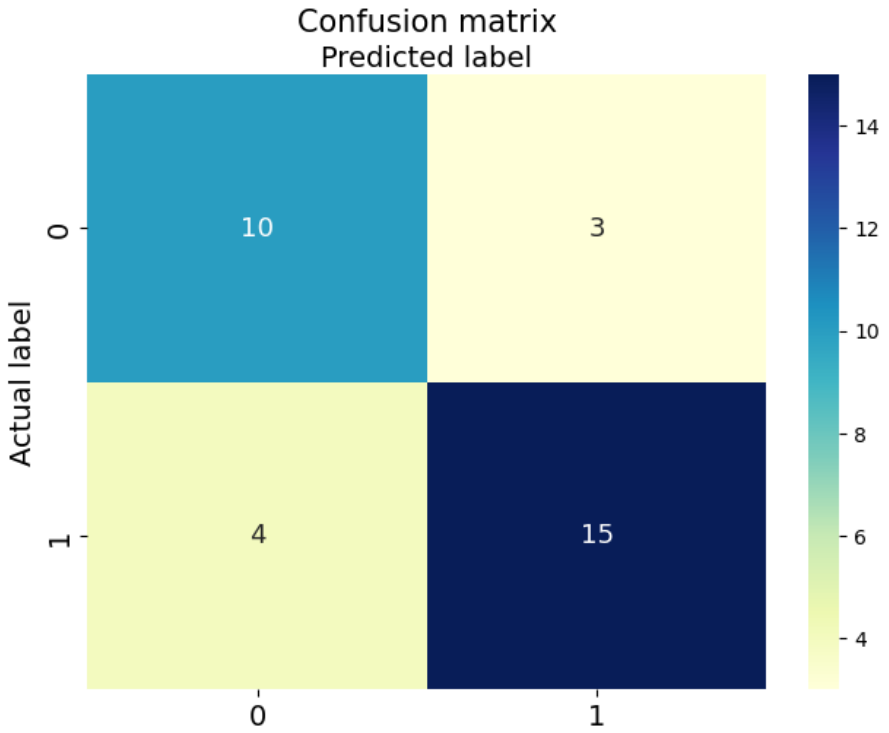

Figure 3.9: Confusion Matrix. The rows of the matrix represents the instances in an actual class while each column represents the instances in a predicted class. Class 0 means a complete response while Class 1 a moderate one.

The diagnostic ability of the classifier was also measured by the Receiver Operating Characteristic curve, or ROC curve. The 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: Area Under 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.

In Figure 3.10, the ROC curve for the classifier is shown. As you can see, the AUC both for classes is 0.82, greater than 0.5 which would correspond to a random classifier, that would give no benefit. I computed the average curve for Class 0 and Class 1, both considering class imbalance (micro-average) and not considering it, so giving the same weight to the classes (macro-average). Also for this case the AUC is greater than 0.5, meaning that the classifier prediction gives more benefit than a random one.

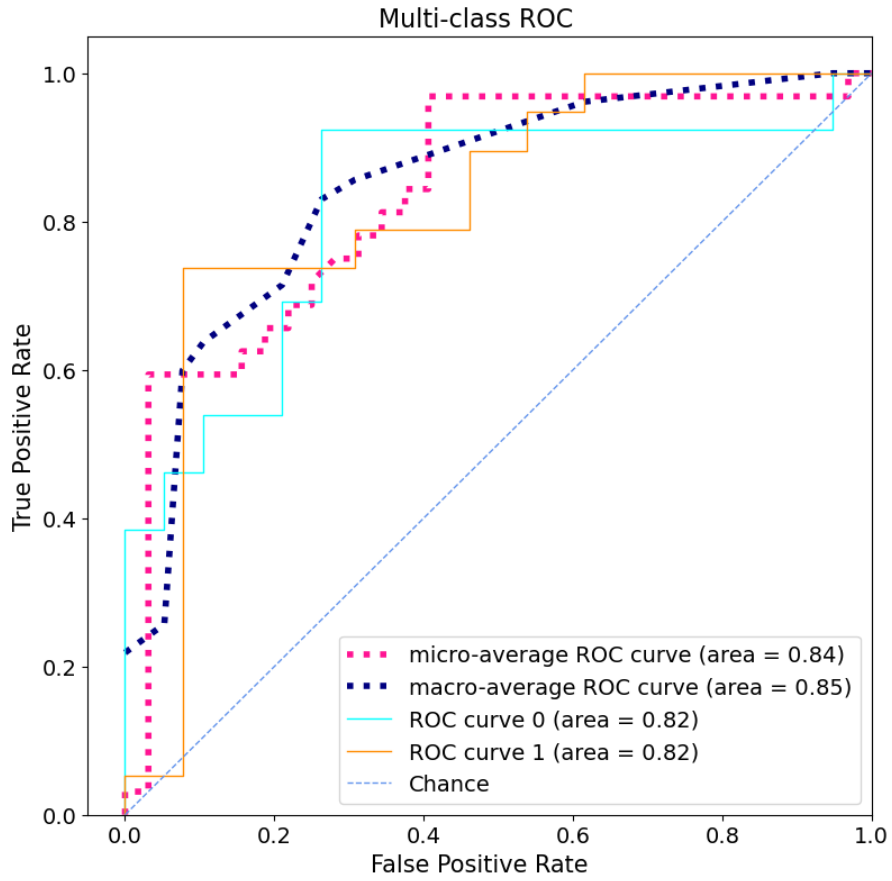

Figure 3.10: Receiver Operating Characteristic curve, or ROC curve.

3.3 Outputs

Once you have installed the implemented pipeline package, available on GitHub[32], you can start to segment the images directly from your bash. There are different outputs options depending on the needing.

Quick Start

The input `dir` is the path of the dir containing the DICOM series

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

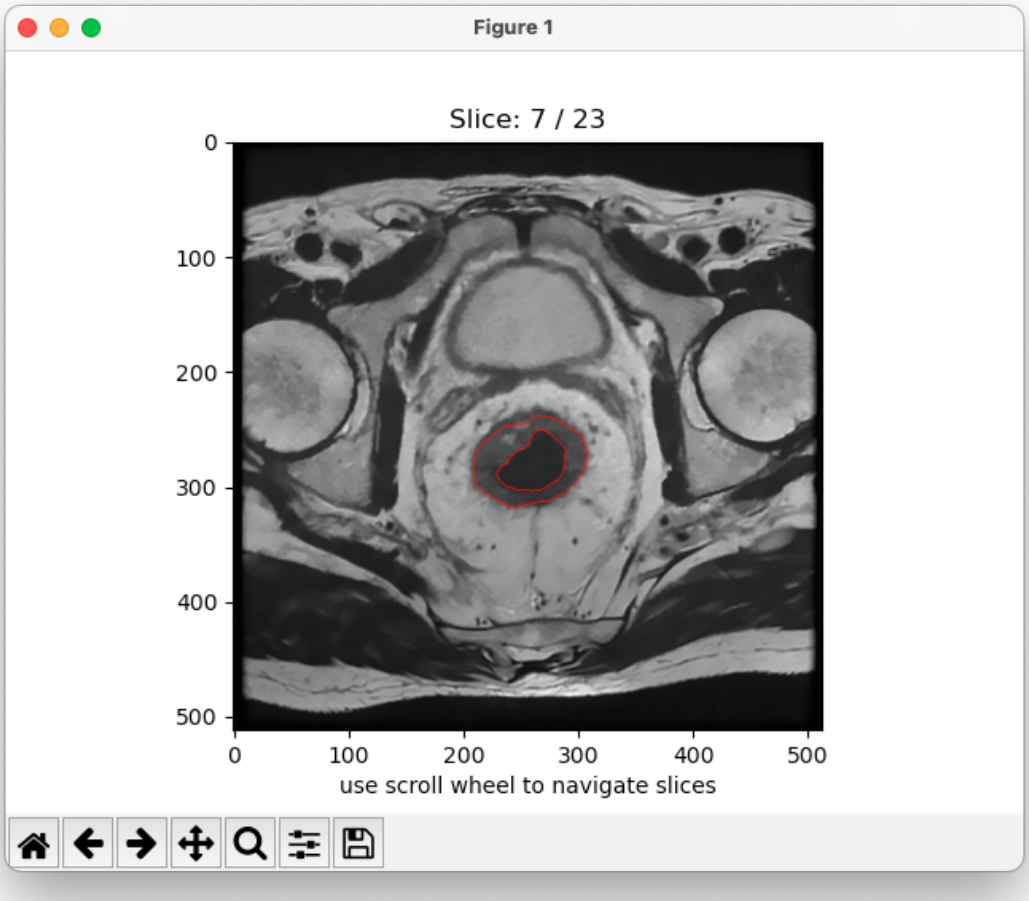

Figure 3.11: Identified tumor area inside the red contours.

mask

When enabled plot the predicted binary [0, 1] mask of each slice.

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

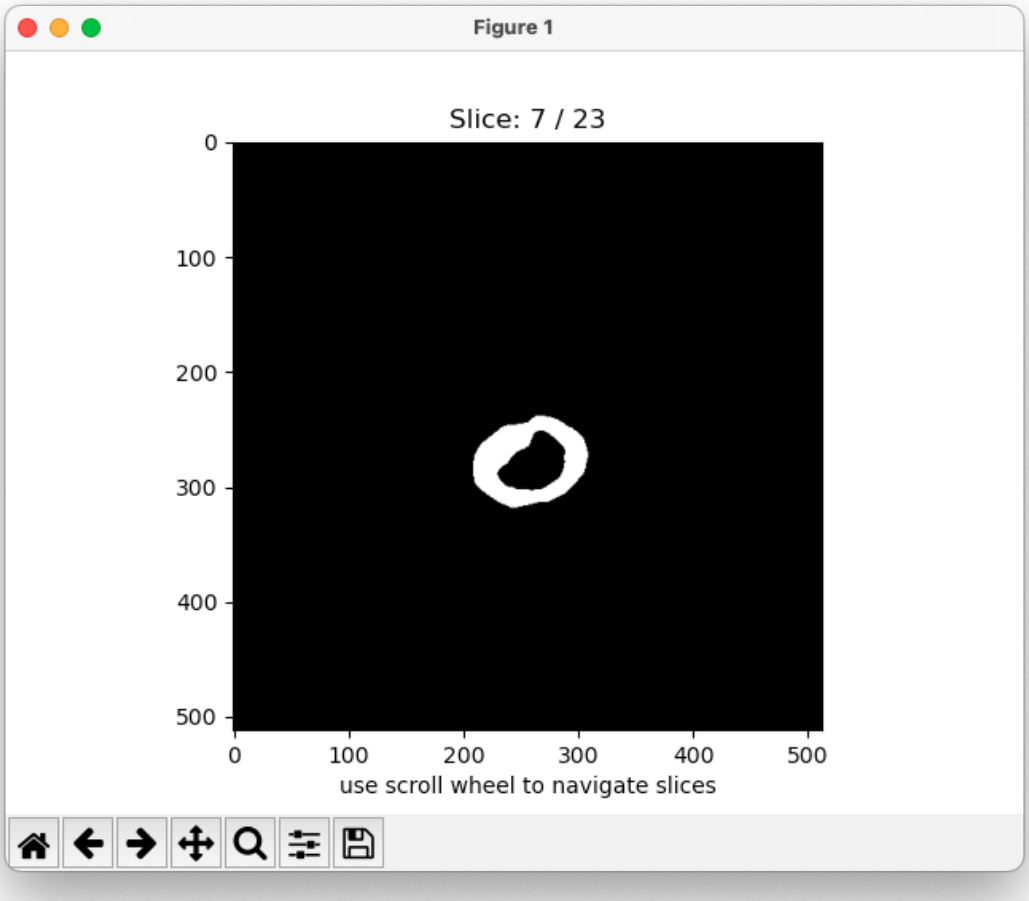

Figure 3.12: Binary mask of the segmented tumor area.

density

When enabled plot the predicted probability map between 0. and 1. of each slice over the original image.

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

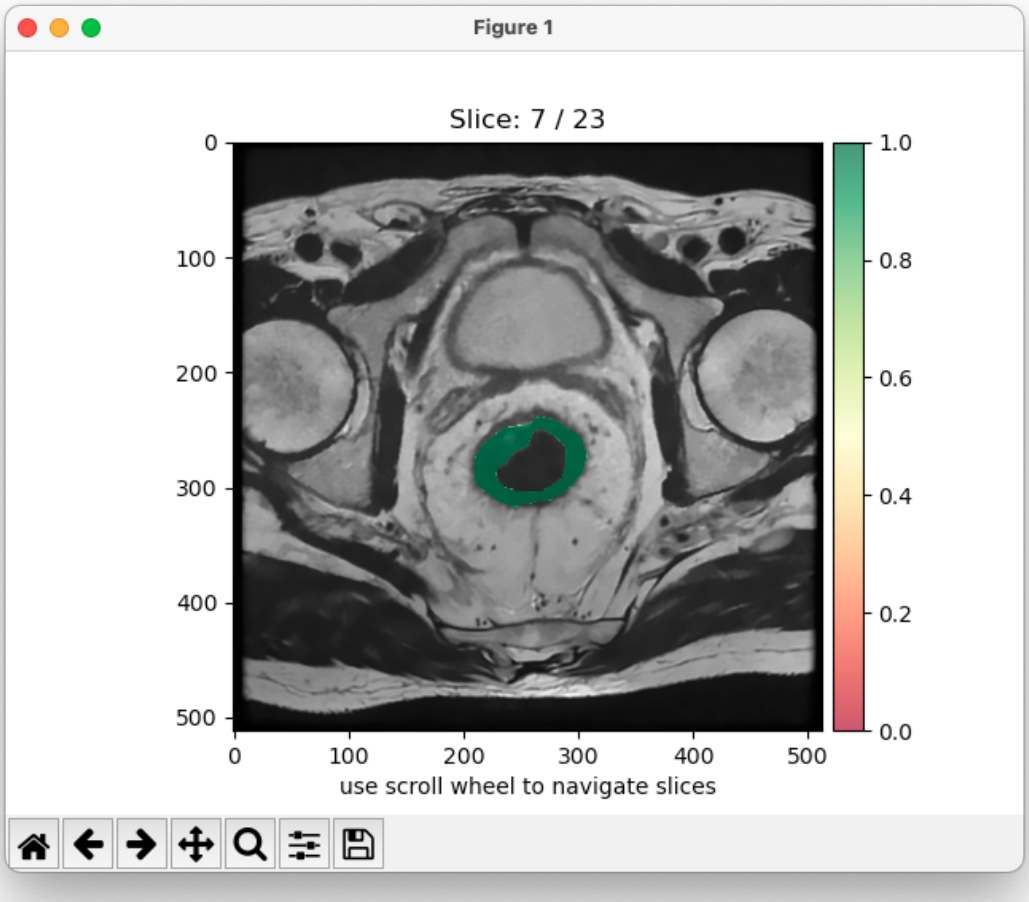

Figure 3.13: Probability map of the segmented tumor area.

3D

When enabled plot the 3D mesh of the segmented areas.

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

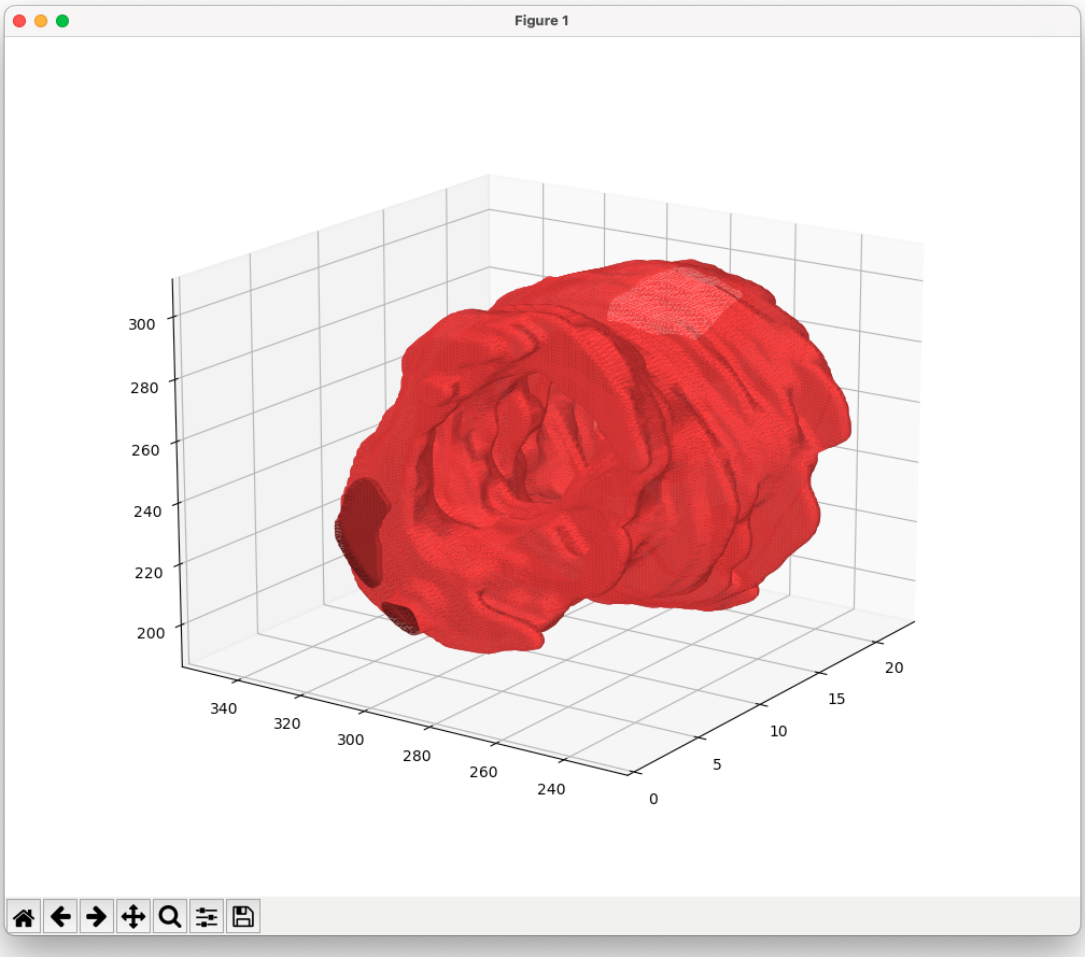

Figure 3.14: 3D mesh of the segmented areas.

Conclusions

In this work of thesis, I have developed and implemented an automated pipeline to predict the response to neoadjuvant chemo-radiotherapy of patients affected by colorectal cancer.
The starting point was the MRI scans. Firstly, I started with the exploration of data coming from the IRCCS Sant'Orsola-Malpighi Policlinic dataset. It consists of MRI scans from 48 patients affected by colorectal cancer undergoing neo-adjuvant radiochemotherapy. Within the scans, also manual annotations made by expert clinicians were provided. Data were pre-processed applying filters on the images, to denoise and enhance the brightness. This preliminary step was done to help the supervised training of a Convolutional Neural Network to perform automatic segmentation. However, from the dataset, only 37 patients were selected because for some of them there was a misregistration between the images and the medical annotations (ground-truth). The training was performed over a total of 488 images, split into training and validation set. By comparing the segmentation coming from the trained model and the one coming from manual medical annotations, it resulted that the model performs consistently with the medical annotations, even if for some restricted cases it failed to segment correctly the images. The metric used to evaluate the segmentation is the Dice Similarity Coefficient (DSC). The evaluation on data coming from the validation set, results in a DSC=0.71, which is consistent and among the highest of literature.
From the segmented images, for each patient, I extracted 100 radiomic features and stored them into a dataframe containing 37 rows, one for each patient and 100 columns, one for each feature. The features were analyzed in order to train a classifier to obtain a 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. These data were obtained from the clinical database of the patients provided by the IRCCS Sant'Orsola-Malpighi Policlinic. Unfortunately, for some patients TRG data were missing so they were excluded from the analysis. The total number of patients from 37 became 32. To overcame the lack of data, TRG values were binarized into Class 0

and Class 1. Class 0 represents a complete response to chemo-radiotherapy while Class 1 a moderate one. The classifier was made exploiting Principal Components Analysis (PCA) and a Support Vector Classifier. PCA was performed to reduce the number of features from 100 to 6, corresponding to the 90% of the total variance. Then the Support Vector Classifier was trained end cross-validated on the data. The results show that the classification for Class 0 is good for 10 cases over 13 thus 3 are wrong classified, while for class 1 it is good for 15 cases over 19 thus 4 are wrong classified. The performance of the classifier was also tested computing the Receiver Operating Characteristic (ROC) curve, in particular calculating the area under the ROC curve, Area Under Curve (AUC). The AUC resulted higher than the 80 % for both the classes. Even for the average ROC curve, the AUC resulted higher than the 80 %, thus grater than 50 % which correspond to a random classifier (no-benefit classifier).
The pipeline was implemented and developed by using python language on the GitHub platform as an open-source project with the relative documentation. Once you have installed it, you can segment images directly from the bash. The pipeline gives different outputs depending on the needing. It is possible to obtain 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 developing of this project are possible, like embedding more information about the clinical status of the patient. Moreover, increasing the number of patients into the analysis can give more general and significant results. In the end, this project, despite the lack data, provided an automated pipeline able to segment MRI scans of patients affected by colorectal cancer in order and to predict the response to neoadjuvant chemo-radiotherapy using radiomics features with satisfactory results.

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] 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.

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

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

[5] 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.

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

[7] 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.

[8] 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.

[9] URL: https://slidetodoc.com/overview-of-image-processing-operations-spatial-domain-transform/.

[10] *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.

[11] 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.

[12] Carl-Fredrik Westin, Hans Knutsson, and Ron Kikinis. "Chapter 2 - Adaptive Image Filtering". In: *Handbook of Medical Image Processing and Analysis (Second Edition)*. Ed. by ISAAC N. BANKMAN. Second Edition. Burlington: Academic Press, 2009, pp. 19–33.

[13] Johannes Schindelin et al. "Fiji: an open-source platform for biological-image analysis". In: *Nature Methods* 9.7 (2012), pp. 676–682. DOI: 10.1038/nmeth.2019. URL: https://doi.org/10.1038/nmeth.2019.

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

[15] 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.

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

[17] 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.

[18] 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.

[19] 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.

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

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

[22] 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/.

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

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

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

[26] 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`.

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

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

[29] 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.

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

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

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

[33] 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`.

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

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

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

[37] 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`.