UNIVERSITÉ DE FRIBOURG
UNIVERSITÄT FREIBURG

eXascale Infolab

UNIVERSITY OF FRIBOURG

BACHELOR THESIS

Pre-Processing Segmentation Datasets for the Hydra Framework

Author:
Christophe Broillet

Supervisors:
Dr. Giuseppe Cuccu
Prof. Dr. Philippe Cudré-Mauroux

February 14, 2022

eXascale Infolab
Department of Informatics

Boulevard de Pérolles 90 • 1700 Fribourg • Switzerland
phone +41 (26) 300 84 65 • diuf-secr@unifr.ch • www3.unifr.ch/inf

Abstract

Christophe Broillet

Pre-Processing Segmentation Datasets for the Hydra Framework

Early cancer detection is a crucial point nowadays to save human lives. Computer aided diagnosis (CAD) systems, which use machine learning techniques, can help radiologists to detect the very first stages of cancer, but also to locate and segment malignant tumors in medical images. Deep learning models, such as convolutional neural networks, are today useful and powerful for computer vision, which makes them the perfect candidates for CAD systems. However, these models require a lot of data, which is often difficult to obtain in the medical field, because of privacy issues. This work focuses on addressing the data scarcity by preprocessing new datasets to an existing computer aided diagnosis system, namely the Hydra framework, while adding the segmentation task to this CAD system. The new datasets contain CT or PET scans of the lungs, and the head and neck, but also ultrasounds of the breast. The preprocessing of the datasets are shown and explained, and are also provided in three new scripts. These datasets are ready to be used in a segmentation task. This task is useful for radiologists, as it shows precisely where the tumors are located, and gives information about their shape. This work develops a new module that creates segmentation masks of medical images, given the cloud of points that contour the tumor. After using this module, the datasets contain the medical images (inputs) and the segmentation masks (labels). The Hydra framework is now ready to train the segmentation task in a supervised learning manner.

Keywords: Machine Learning (ML), Deep Learning, Convolutional Neural Network (CNN), Computer Aided Diagnosis (CAD) System, Segmentation, Cancer

Contents

List of Figures

Chapter 1

Introduction

Computer aided diagnosis (CAD) systems are a way to help radiologists in their work. These systems can detect, locate and segment tumors in medical images, which can ensure a cancer to be detected as early as possible, and to start a treatment quickly, using machine learning techniques. The goal of this work is to improve an existing CAD system by preprocessing and preparing new datasets, while introducing a new segmentation task.

1.1 Cancer

The human body is composed of trillions of cells, that work for all the tasks the body needs. These cells multiply and grow themselves naturally, in order to replace old or damaged cells, which will die in a natural manner. This whole process is called cell division. The disease in which the cell division become uncontrollable is called *cancer*.

In general, cancer can start from any part of the body. Damaged or abnormal cells can also multiply themselves in cancer, which is not the case in a healthy body. An abnormal aggregation of these cells is called a *tumor*. Tumors that are cancerous are called *malignant*, while non-cancerous tumors are called *benign*. When tumors spread through the body to create new tumors, the cancer propagates and these new tumors are called *metastasis* [1].

In 2020, there were about a total of 10 millions deaths from cancer of men and women of all ages, where the lung cancer has clearly the biggest mortality (about 1.8 million). For men only, the most mortal cancers are first the lungs, followed by the liver, the colorectum, the stomach and the prostate. For women, the leading type of cancer is the brest cancer. After that comes the lungs cancer, the colorectum, the cervix, and the stomach. Table 1.1 shows the estimated number of deaths by cancer, for men and women. These numbers are taken from the Global Cancer Observatory, a division of the World Health Organization (WHO) [2].

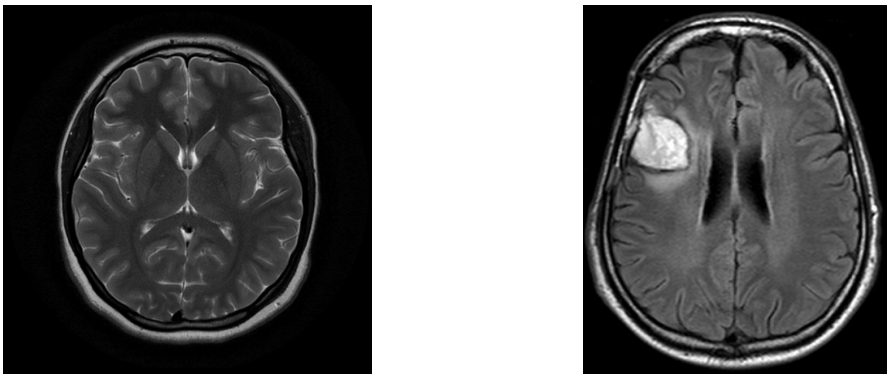
1.1.1 Risk factors and diagnoses

According to the WHO, cancer is a leading cause of death all around the world, with about 10 millions deaths in 2020. There are many risk factors that can cause cancer such as the use of tobacco or alcohol or unhealthy diet. Probability can be minimized when these risk factors are taken with attention. The risk of cancer also increases with age, as the cell replacement mechanism tends to corrupt with old age. The mortality of the cancer can be reduced by an early diagnosis, as the treatment can begin early and can be modified if necessary.

The main method used to detect cancer is called *screening*. The idea is to call patients to go for a screening of the whole body or only on specific parts, even if

Cancer location	Number of deaths for *men*	Number of deaths for *women*
Lung	1'188'679	607'465
Colorectum	515'637	419'536
Stomach	502'788	266'005
Liver	577'522	252'658
Prostate	375'304	-
Breast	-	684'996
All cancers	*5'528'810*	*4'429'323*

TABLE 1.1: Estimated number of deaths for a given cancer location in 2020, for some of the most mortal cancers, for men and women.



(A) Healthy brain (B) Brain with a tumor

FIGURE 1.1: Comparison of two magnetic resonance images of the brain. The brain on the left is healthy, whereas the brain on the right has a clearly visible tumor at the top left. These images are coming from the Kaggle brain dataset [4].

patients have no symptoms. Expert medicine will then analyse the resulting images, or test results. Common screening are for example mammography for women, or prostate-specific antigen (PSA) test for men [3]. Figure 1.1 shows for example a comparison of the magnetic resonance images of an healthy brain 1.1a and a brain with a malignant tumor 1.1b.

1.2 Motivation

As seen in the previous section, cancer is very dangerous and leads to a significant amount of deaths per year, but mortality can be reduced by an early diagnosis, or a performant and precise screening. This work has been done to ensure a better performance of early diagnoses. Since 2018, the cantonal hospital of Fribourg (H-FR) has been collaborating with the eXascale Infolab at the University of Fribourg (UniFR) to create and develop a CAD system that can help the radiologists in their work, to analyze and find early stages of cancer as soon as possible. This project has been called *Hydra*.

Hydra has been created by using machine learning techniques, as a part of artificial intelligence. The software looks at medical images, and predicts whether there is or not a tumor in the image. In that way, radiologists can be warned earlier to start the treatment of a cancer, which will hopefully result in saving lives. To train such a method, a huge amount of data is needed. Worldwide medical institutions have

given public access to their anonymized data. Researchers can then use this data to do similar work as the presented one.

This work is a part of the full Hydra framework. The project has been started with the two past works of Clement and Jobin [5] and Lee and Cuccu [6], who first created the machine learning model that will be used to detect and locate possible tumor in a medical image. They have use three different medical datasets, and try different methods to train the software in an optimal manner. These three datasets are the PROSTATEx [7] dataset, the Lung CT Challenge [8] dataset and the Kaggle brain [4] dataset.

This thesis will present first in Chapter 2 some machine learning concepts and backgrounds that has to be known before getting in the project. It includes a small introduction on machine learning, and how to construct mathematical models that will be train on the data, with some powerful techniques such as neural networks. In Chapter 3, an introduction about the types of medical images will be made. The way these images are handle by a computer will also be shown. Then, the three new datasets will be presented. Their content as well as their origin will be explained. In Chapter 4, the contributions (see Section 1.3) of this work will be developed. Finally, a conclusion about this work and the future work that has to be done can be found in Chapter 5.

1.3 Contributions

To improve the Hydra framework, this work will add three more datasets that will be used. These datasets are also publicly available. The more data is used, the more Hydra will be performant. Before using this data, it has to be preprocessed first. Indeed, the software can not read directly the raw medical images, but they have to be modified so that they can be readable by the software.

The main contributions of this work, in the continuation of the Hydra project, are then:

- A new module that creates segmentation masks from tumor contours
- Three new preprocessing scripts that will preprocess the three new datasets
- Minor improvements on the past work

The Hydra framework is written in the Python programming language, that is a well used and referenced language for machine learning projects. The full code of the contributions made in this work is publicly available on GitHub[1].

[1] https://github.com/ChristopheBroillet/Hydra_Segmentation

Chapter 2

Machine learning basics

This chapter highlights and explains the machine learning basics to understand the Hydra framework. First, a little overview and history of machine learning is shown. Then, the supervised learning paradigm is explained, by introducing some mathematical concepts, such as loss functions. After that, the applications and the basics of neural networks are presented. Deep learning will then quickly be introduced, to finally come to well known convolutional networks, that are used in computer vision and for a computer aided diagnosis applications. When having all this solid mathematical and conceptual background, the Hydra framework will finally be understandable, and presented in the last section.

2.1 Overview

Machine learning uses algorithms to learn about given data, and improves on finding solutions to a problem[1]. Solutions in machine learning consist of trained mathematical models, generic functions accompanied with a parameter set. Machine learning assumes that for each given dataset, there is an *underlying function* that generates the data. The goal of machine learning algorithms is then to find a solution that best fits the data, and that is the closest to the real underlying function, by learning through the data. Solutions are always *approximations* of the real underlying function, limited by the data the machine knows, or that the user gives to it, which is only a subset of the whole possible data. After this period of learning, many applications can be done. Prediction, classification, clustering and anomaly detection can be cited as some of the most used generic applications of machine learning.

Machine learning has first started back in the 1950's. Firstly, the statistical methods or main concepts of machine learning were created, as in the work of Hebb [9], who based his work in the synaptic transmission of signals of electrical impulses, that will be used further in neural networks. In 1957, one of the fundamental algorithm of machine learning, the perceptron, was invented by Rosenblatt [10], who published his work in 1958. The perceptron is a binary classifier algorithm, that is at the very base of all neural networks. Further, in 1963, Michie and Chambers [11] developed a machine made of matchboxes that learns how to play Tic-Tac-Toe and plays against humans. The machine makes a decision move depending on the current position of the boardgame. A turning point in the history of the machine learning was back in 1988, when LeCun et al. [12] published their work about the backpropagation algorithm, that is still used today to train neural networks.

Today, machine learning has a plenty of real world applications, such as email spam detection, recommender systems, face recognition, or language translators, to only cite them. Machine learning is comprised of three main paradigms, that have

[1] https://www.ibm.com/cloud/learn/machine-learning

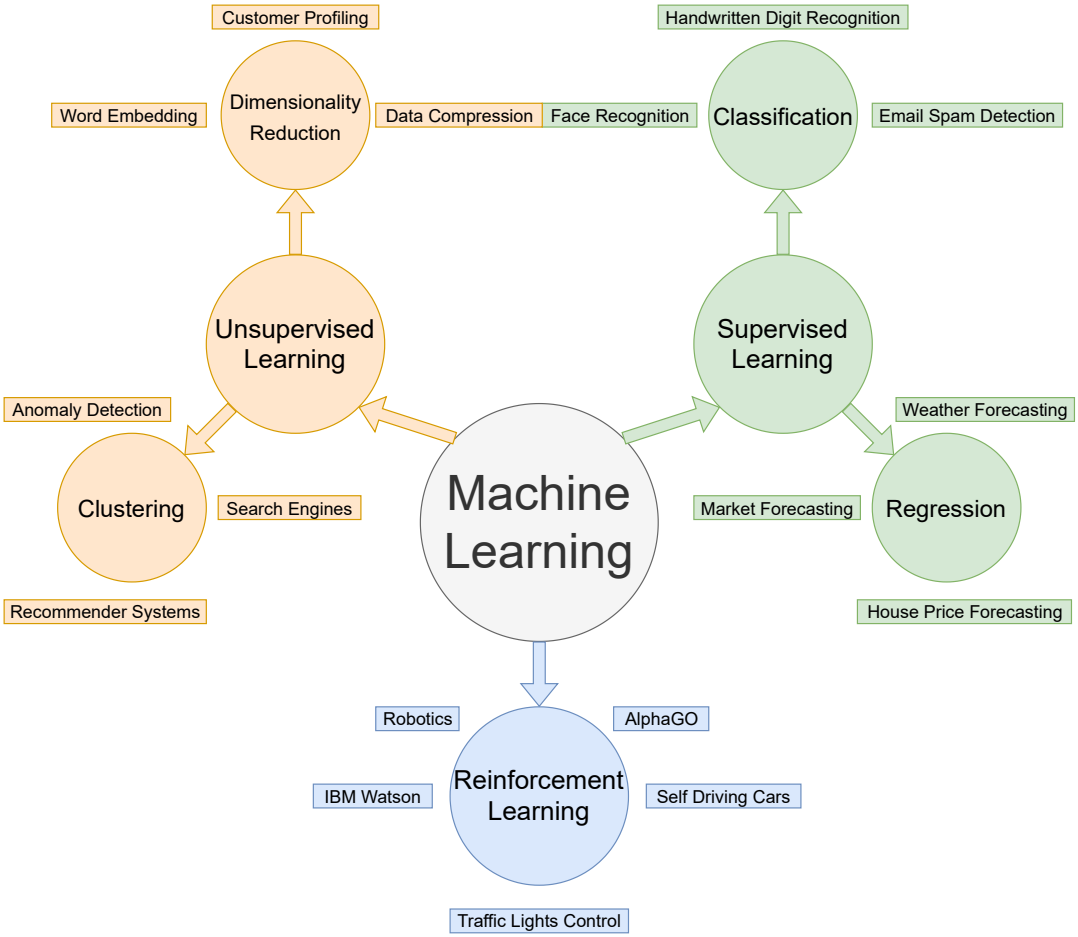


FIGURE 2.1: Some applications of the three main machine learning paradigms.

different behaviour and can be choosen depending on the problem that needs to be solved[2]. These paradigms are:

1. Supervised learning
2. Unsupervised learning
3. Reinforcement learning

Figure 2.1 shows the three main paradigms and some of their applications. The rest of the work will focus on the supervised learning paradigm, which is explained in detail next.

2.2 Supervised learning

As its name suggests, the learning is done in a supervised manner, like a teacher will supervise his students. To be used, supervised learning algorithms need a set of data, called the *training dataset*, i.e. the data from which the underlying function is learned. The training data is composed of tuples of two elements. The first element is an input/entry element, and the second is the known answer corresponding to

[2] https://towardsdatascience.com/what-is-machine-learning-891f23e848da

the input element, called the *label*[3]. The goal of supervised learning algorithms is to find the relation between the input data and their corresponding labels, or in mathematical terms, to find the underlying function $f : X \to Y$ such as:

$$f(x) = y \tag{2.1}$$

where $x \in X$ is an element of the input space X and $y \in Y$ its corresponding label in the decision space Y. The so called training dataset D is then a set of tuples containing an element x_i of the input space with its corresponding label y_i, so mathematically $D = \{(x_i, y_i), i \in \{1, ..., n\}\}$. To best approximate f, i.e. to learn from the training data, the algorithm starts with a solution h_1, and constructs a prediction $\hat{y}$ based on the input x, such as:

$$h_1(x) = \hat{y} \tag{2.2}$$

Then, it compares the true label y with the prediction $\hat{y}$. Depending on this comparison, the algorithm modifies a bit the solution h_1 into the improved solution h_2. This process is iterated, and the solution is improved and will approximate better and better the true underlying function f. The quality of the prediction of the model is verified by using *loss functions*.

2.2.1 Loss functions

A loss function estimates the cost or severity of predicting the actual prediction $h(x_i) = \hat{y}_i$ rather than the real value y_i. With that information, the algorithm knows if it is improving in a good direction, and have an idea how to minimze this loss. There are many loss functions that have different form, and that are more or less representative on specific applications. Given a training data point (x_i, y_i) and a model h to train, here are some often used loss function examples. Firstly, the *squared error loss* is given by:

$$L(h(x_i), y_i) = (y_i - h(x_i))^2 \tag{2.3}$$

This is one of the simpliest and most used loss function for a regression problem. A small variation of this loss function that is often used is to add a factor $\frac{1}{2}$ to simplify computing the derivative. As second example loss function, the *hinge loss* is used for binary classification, by penalizing the wrong prediction, but also the right ones that are not confident. It has the form:

$$L(h(x_i), y_i) = \max(0, 1 - y_i \cdot h(x_i)) \tag{2.4}$$

Finally, a similar loss function to the first loss function (2.3) is the *absolute error loss*, given by:

$$L(h(x_i), y_i) = |y_i - h(x_i)| \tag{2.5}$$

This third loss function is more robust than the squared error because it is generally not affected by outliers. But on the other side, the squared error tends to adapt the model regarding the outliers. The loss function gives the information about how bad or good is one prediction. For an entire training dataset, the sum of the losses of all points in the dataset is required and will be more representative. This sum is called the *empirical risk*, and will be used also further to tune neural networks. The empirical risk is given by:

$$\hat{R}(h) = \sum_{i=1}^{n} L(h(x_i), y_i) \tag{2.6}$$

[3] https://deepai.org/machine-learning-glossary-and-terms/supervised-learning

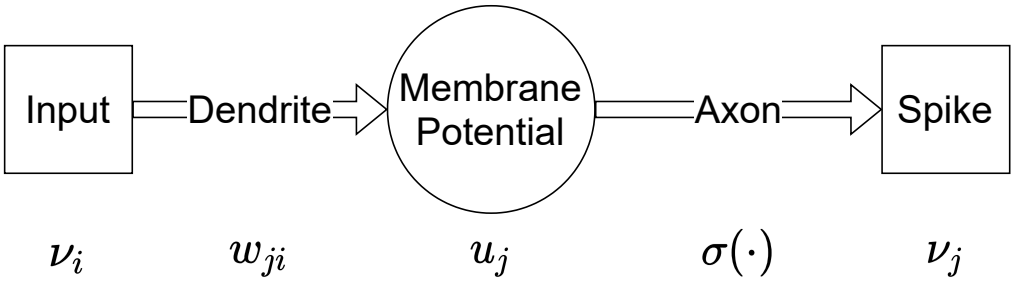


FIGURE 2.2: Schema showing how an artificial neuron works. The input ν_i of the neuron is first weighted by some weight w_{ji} that represents the connection beetween i and j. The result is the membrane potential u_j, that is finally activated by an activation function $\sigma(\cdot)$. The output of the neuron is given by ν_j.

The empirical risk can be averaged by a factor $\frac{1}{n}$ in certain cases, for example with the backpropagation algorithm (see further in Section 2.4).

A key point in the training phase is not to learn too much from the given data. Indeed, as stated in introduction of this chapter, the training data is only a subset of the real whole data. When the algorithm learns the available data too closely, it specializes on this precise data and is not able to predict well the future unseen data, resulting in a bad generalization. This phenomenon is called *overfitting*.

Finally, supervised learning is mainly used for classification or regression problems, and is one of the most used paradigms in machine learning. A class of algorithm that are very strong with supervised learning are neural networks. In the following section, neural networks will be presented in detail.

2.3 Neural Networks

Nowadays, an extremely important concept that is used in machine learning is the *neural network*. Neural networks are inspired by biological brains, that connect neurons with synapses. The concept of the natural neural network and the artificial neural network is to pass some information by connections. Neural networks are commonly trained using supervised learning. Figure 2.2 shows the form of an artificial neuron, in comparison with biological neuron technical terms.

An artifical neural network is composed of many neurons distributed in *layers*. A neural network can have a variable number of layers depending on the task. Each layer is more or less connected to each other by neurons. Different shape of connections make different type of neural network. In each layer, there is one or several neurons. Each neuron has one or more inputs, that are the output of the neurons of the previous layer, but only one output. The neuron will take its inputs, arrange them according learned weights, and activate using an *activation function*, and finally give the output. Mathematically, this is represented by the following formulas:

$$\begin{aligned} u_j &= \sum_{i=1}^{k} w_{ji} \cdot \nu_i \\ \nu_j &= \sigma(u_j) \end{aligned} \tag{2.7}$$

where u_j is the membrane potential of the neuron, ν_i are the inputs of the previous layer, w_{ji} the synaptic weight of the neuron of the connection i to j, and ν_j the output of the neuron, after activation with $\sigma(\cdot)$. There are several activation functions that

can be used, for example the *logistic* function:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.8}$$

As second example, the *hyperbolic tangent*:

$$\sigma(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1 \tag{2.9}$$

Or the *rectified linear unit* (ReLU) function:

$$\sigma(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases} \tag{2.10}$$

And at last example, the *exponential linear unit* (ELU):

$$\sigma(x) = \begin{cases} \alpha(e^x - 1) & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases} \tag{2.11}$$

These activation functions are again choosen depending on the task the neural network needs to do. For example, the logistic function will be more used for a binary classification task, as its output range is from 0 to 1, and can represent a probability. Compared to this, the hyperbolic tangent has the advantage to have an output range from -1 to 1. The two other examples, the ReLU and ELU are some of the most common choices in deep learning and convolutional networks. Both have their output range that goes towards infinity, but they differ in the lower bound. ReLU puts its negative values to zero immediately, whereas ELU will have a lower bound equal to α. ELU has so the advantage to treat smoothly the negative values but at the cost to be slower to compute than the ReLU[4].

The functioning of a neural network is the following. An input is given to the network, and it goes through several layers inside the network. At each layer, more or less outputs go out, and enter the next layer. At the other side of the network, in the last layer, the output of the network goes out, and is the final output, given the very first input. The output can have a very different form regarding the task that has to be achieved. For example, if a neural network is constructed to recognize if there is a cat in an image, then the input will be an image, and the output will be a binary output, with probabilities of true (there is a cat) or false (there is no cat).

Figure 2.3 represents a neural network with two inputs x_1 and x_2, one hidden layer of two neurons n_1 and n_2, and one output layer of one neuron n_3. The weights are represented as w_{ji}. The computations inside the network go as follows. First, the weights are putting in a weight matrix, one matrix between each layer. The matrix between the inputs and the hidden layer is W_{in}, as the matrix between the hidden layer and the ouput layer is W_{hid}. Then, the inputs are stored in a vector x, the intermediate results are in the vector n_{hid} and the output in n_{out}. The activation

[4] https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

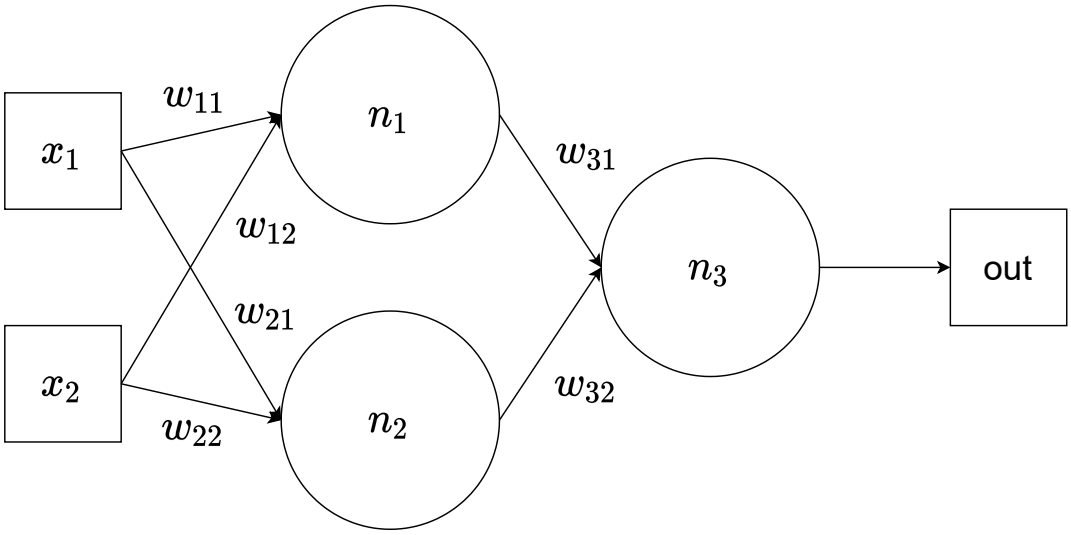


FIGURE 2.3: Schema representing an example neural network. It contains one hidden layer of two neurons n_1 and n_2, and one output layer of one neuron n_3. x_1 and x_2 are the two inputs of the network, whereas the neuron n_3 gives the output of the network. The w_{ji} are the six weights.

function in each neuron is $\sigma(\cdot)$. These definitions are given by:

$$\begin{aligned} W_{in} &= \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix}, \qquad W_{hid} = \begin{pmatrix} w_{31} & w_{32} \end{pmatrix} \\ x &= \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \qquad n_{hid} = \begin{pmatrix} n_1 \\ n_2 \end{pmatrix}, \qquad n_{out} = \begin{pmatrix} n_3 \end{pmatrix} \end{aligned} \tag{2.12}$$

With the help of these definitions, the formulas to compute the intermediate results can be written using matrices multiplication from linear algebra:

$$\begin{aligned} n_{hid} &= \sigma(W_{in}x) = \begin{pmatrix} \sigma(w_{11}x_1 + w_{12}x_2)\sigma(w_{21}x_1 + w_{22}x_2) \end{pmatrix} \\ n_{out} &= \sigma(W_{in}n_{hid}) = \begin{pmatrix} \sigma(w_{31}n_1 + w_{32}n_2) \end{pmatrix} \end{aligned} \tag{2.13}$$

And the full equation for the output of the example network can be written using the two equations above:

$$\text{out} = n_3 = \sigma(w_{31}\sigma(w_{11}x_1 + w_{12}x_2) + w_{32}\sigma(w_{21}x_1 + w_{22}x_2)) \tag{2.14}$$

Mathematically speaking, a neural network is an universal function approximator. That means that a neural network can approximate any function from a theoretical perspective. In that way, neural networks can be used for quite any applications, given a network large enough. Neural networks are parametrized by their structure, i.e. how the layers are disposed, their activation functions and their weights. Different values for the weights can lead to very different outputs, thus the weights need to be well tuned. To give values to the weights in an optimal manner, a well-known algorithm is used: the *backpropagation*. This is the main algorithm to train the network and to tune its weights, in the way to do a specific application.

2.4 Backpropagation

As stated in Section 2.1, the backpropagation was invented in the end of the 1980's, by simultaneous works of different researchers. The backpropagation algorithm is used with neural networks, and will adapt or optimize the weights of the network. Backpropagation is composed of two phases. The first one is called the *forward pass*, when the inputs are propagating from the front of the network to the back, i.e. the network makes a prediction. Then, the prediction is compared to the ground truth label, and the error is computed with a loss function. In the second phase called the *backward pass*, the error is propagating from the back of the network to the front. For each layer, the derivative of the error is computed. These derivatives help to know which weights contributed to the error, and help adapting them in a focused manner.

For a network, the backpropagation algorithm computes the error as the risk over all points of the dataset. The error depends on the weights set w of the network. Indeed, different weights lead to different predictions, thus different errors. Let the error $E(w)$ of the network be:

$$E(w) = \frac{1}{|S|} \sum_{i \in S} L(f_w(x_i), y_i) \tag{2.15}$$

with $f_w(\cdot)$ the function representing the network with specific weights w, $L(\cdot, \cdot)$ a choosen loss function, and S a sample of the training data. This sample S can be different regarding the type of training that is made. For example, S can be the whole training dataset and the training is called to be in batch mode. In contrast, the sample can be choosen to be $|S| = 1$, that is called in online mode. But the most common way to train nowadays is the so called mini batches, by choosing $|S| \ll n$, that is in a small subset at each step of the learning. In general for backpropagation, the loss function used is the squared loss (2.3) with a $\frac{1}{2}$ factor. The error function (2.15) for backpropagation will then be:

$$E(w) = \frac{1}{2|S|} \sum_{i \in S} (f_w(x_i) - y_i)^2 \tag{2.16}$$

Note that in the following paragraphs and formulas, the symbols in superscript within parenthesis will represent the number of the current layer, as the symbols in subscript will represent the number of the neuron it refers. As example, the notation $v_n^{(l)}$ will refer to the output after activation of the n-th neuron of the layer l.

The goal of the backpropagation is to minimize the error function (2.16) of the network, to make the network as strong as possible in its predictions. To do that, weights must be tuned and adapted to make the best results. To know in which direction the weights must be changed regarding the error, backpropagation uses *gradient descent*. This method iteratively finds a local mimimum or maximum of a differentiable function. In this method, the gradient of the increasing error is $\nabla_w E(w)$, and so the direction of the decreasing error will be in the opposite direction of the gradient, i.e in $-\nabla_w E(w)$ direction. A learning rule for a single weight can then be derived as:

$$w_{ji}^{(k)} \leftarrow w_{ji}^{(k)} - \eta \cdot \nabla_w E(w) \tag{2.17}$$

where η is called the learning rate. This rule shows that the weights are updated regarding the most decreasing error direction. As the error function $E(w)$ is differentiable with respect to the weights w, gradient descent can be applied and that is

what backpropagation does. Mathematically[5], the error gradient with the weight for neuron j in layer k connected to incoming neuron i of previous layer is defined as:

$$\nabla_w E(w) = \frac{\partial E}{\partial w_{ji}^{(k)}} \tag{2.18}$$

Remember that in Figure 2.2, $u_j^{(k)}$ was denoted to be the membrane potential of neuron j in the layer k, that is the value before going in the activation function. Let $n^{(k)}$ defines the number of neuron in the layer k, $u_j^{(k)}$ is computed by the sum:

$$u_j^{(k)} = \sum_{m=1}^{n^{(k-1)}} w_{jm}^{(k)} v_m^{(k-1)} \tag{2.19}$$

The partial derivative of the error with respect to one weight is computed using the chain rule, and is then given by:

$$\frac{\partial E}{\partial w_{ji}^{(k)}} = \frac{\partial E}{\partial u_j^{(k)}} \frac{\partial u_j^{(k)}}{\partial w_{ji}^{(k)}} \tag{2.20}$$

The first term of (2.20) is in general called the backpropagated error, and is written as:

$$\delta_j^{(k)} = \frac{\partial E}{\partial u_j^{(k)}} \tag{2.21}$$

Whereas the second term is calculated to be $v_i^{(k-1)}$ the output after the activation of the neuron i in the layer $k-1$, i.e. the previous layer:

$$\frac{\partial u_j^{(k)}}{\partial w_{ji}^{(k)}} = \frac{\partial}{\partial w_{ji}^{(k)}} \left(\sum_{m=1}^{n^{(k-1)}} w_{jm}^{(k)} v_m^{(k-1)} \right) = v_i^{(k-1)} \tag{2.22}$$

In summary, the derivative of the error (2.20) regarding a weight can be written in the compact form:

$$\frac{\partial E}{\partial w_{ji}^{(k)}} = \delta_j^{(k)} v_i^{(k-1)} \tag{2.23}$$

On one hand, for the output layer l, i.e. the last layer of the network, (2.23) will depend directly on the label y and the prediction $f_w(y)$. By applying the chain rule and using the derivative of (2.16), the derivative of the error that is (2.23) will become:

$$\frac{\partial E}{\partial w_{ji}^{(l)}} = \delta_j^{(l)} v_i^{(l-1)} = (f_w(y) - y) \cdot \sigma' \left(\sum_{m=1}^{n^{(k-1)}} w_{jm}^{(l)} v_m^{(l-1)} \right) \cdot v_i^{(l-1)} \tag{2.24}$$

[5] https://brilliant.org/wiki/backpropagation/

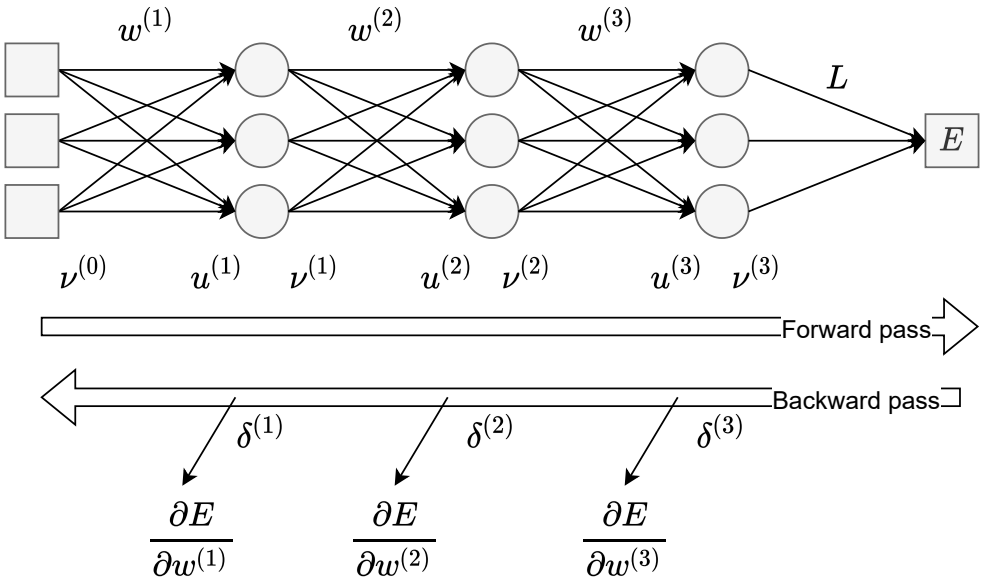


FIGURE 2.4: Schema representing the backpropagation algorithm steps. First, the forward pass is done, where the input $\nu^{(0)}$ go through the network, generating an output prediction. This prediction $\nu^{(3)}$ is compared with the actual target label, and the error E is computed using the loss L. During the backward pass, the errors $\delta^{(i)}$ are computed, and used in the calculation of the partial derivatives. This derivatives help to adjust the weights $w^{(i)}$ of the neural network.

On the other hand, for a hidden layer k, the error term (2.21) is first computed by again using the chain rule:

$$\delta_j^{(k)} = \frac{\partial E}{\partial u_j^{(k)}} = \sum_{m=1}^{n^{(k+1)}} \frac{\partial E}{\partial u_m^{(k+1)}} \frac{\partial u_m^{(k+1)}}{u_j^{(k)}} \tag{2.25}$$

But the first term in this sum represents also an error term (2.21). By using the definition and derivative of (2.19), the error term for hidden layers will become:

$$\delta_j^{(k)} = \sum_{m=1}^{n^{(k+1)}} \delta_m^{(k+1)} \cdot w_{jm}^{(k+1)} \cdot \sigma'(u_j^{(k)}) \tag{2.26}$$

Finally, by taking the result of (2.26), the derivative of the error (2.23) for hidden layers will be:

$$\frac{\partial E}{\partial w_{ji}^{(k)}} = \sigma'(u_j^{(k)}) \cdot \nu_i^{(k-1)} \cdot \sum_{m=1}^{n^{(k+1)}} \delta_m^{(k+1)} \cdot w_{jm}^{(k+1)} \tag{2.27}$$

This last equation shows that the error at layer k is also depending on the error at layer $k+1$, that means that the computations of these errors must be done backwards. That is where the backpropagation algorithm takes its name. Figure 2.4 shows the two steps of the backpropagation: the forward pass where the prediction is made, and the backward pass where the errors are computed.

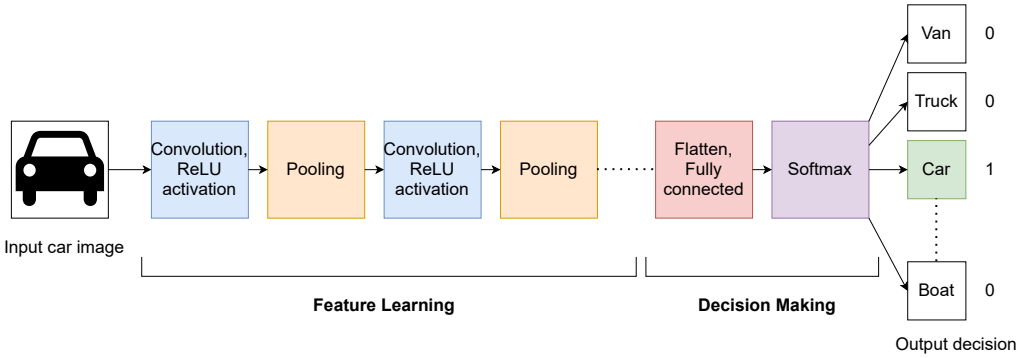


FIGURE 2.5: General architecture of a convolutional neural network. The first part represents a feature extractor, that analyzes the input image by several convolutions and poolings to extract features. These features are fed to the second part, that is a decision maker. In this example, the decision maker is a multi-class classifier that will output the probability that the image is a given vehicle, with the help of the last softmax layer that outputs a vector of probability to be in a given class.

2.5 Deep learning

In the recent years, some of the best results of the field come from *deep learning*. Deep learning is a learning technique that uses a neural network containing many layers. That makes the network, as the name suggests, a deep network. Deep learning starts to appear in the 2010's. Before that time, it was not possible to develop deep learning. Main factors that have helped the development of deep learning were the multilayers neural networks, *big data*, and the power of computations of the machines that are now enough strong to compute deep learning algorithms. The foundation of deep learning is contained in the work of Goodfellow et al. [13], that was published in 2016.

Deep learning is used in many applications such as translators, image processing, natural language processing, and others. A major problem with deep learning however, is that the deep networks need a lot of data to be trained, which is not always easy to obtain. For example, in medical applications, the privacy of the patients data makes it difficult to collect enough data to use with deep learning. Indeed, personal information of the patients are sensible, and hence must be removed before giving public access to the data, as in the Cancer Imaging Archive.

2.5.1 Convolutional neural networks

From image classification to object detection, computer vision is used in different applications such as face recognition, autonomous cars or for military purpose. One of the most popular neural network model used for computer vision is the *convolutional neural network* (CNN). One of the first CNN was when LeCun et al. [14] used backpropagation to recognize handwritten zip codes. Convolutional neural networks take an image as input, and can do several tasks, such as classification, localization or segmentation (see Section 2.6).

The general structure of a convolutional neural network is composed of a first part called *feature extractor* or feature learning part, that is a combination alternating *convolutional layers* and *pooling layers*. The second part is one or more fully connected layers that is a *decision maker*. Figure 2.5 shows the general architecture of a CNN.

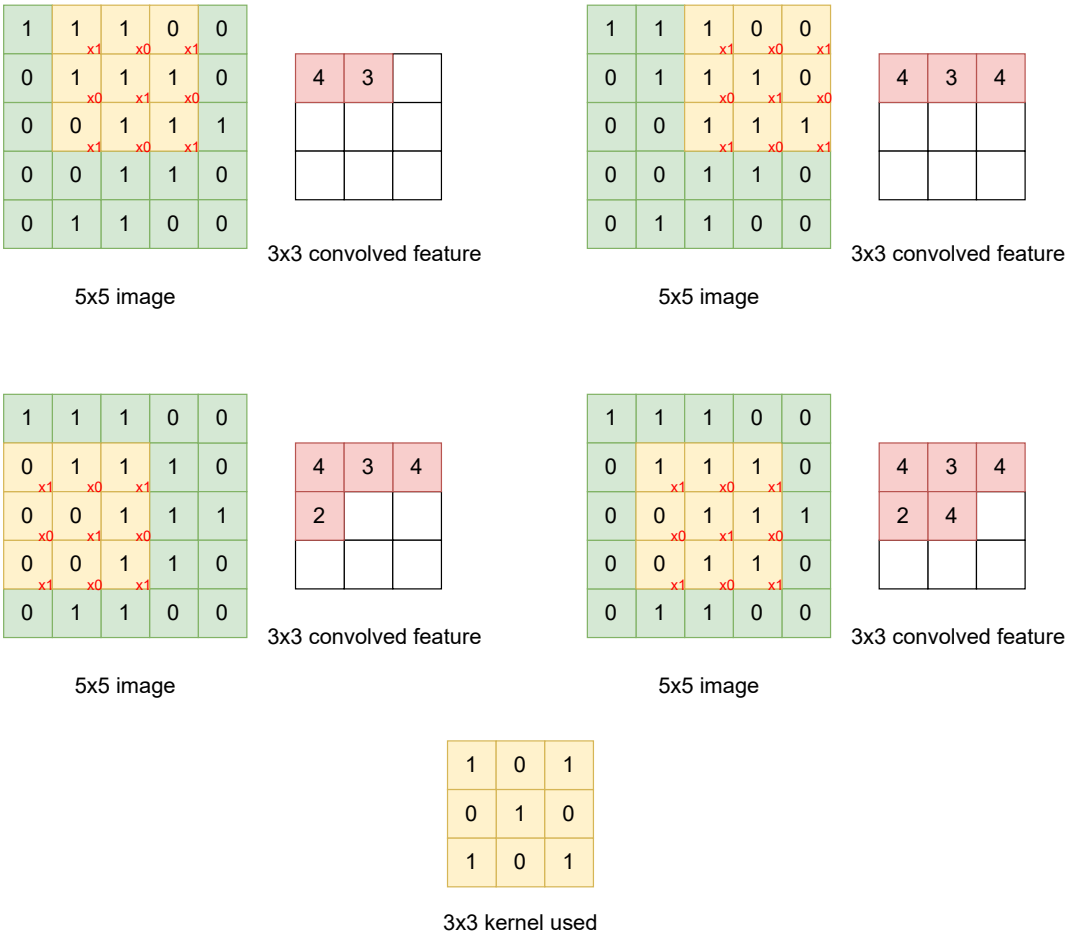


FIGURE 2.6: Four example steps during a convolution. A 3x3 window (yellow) containing the kernel values is sliding on an 5x5 image (green) and contructs a 3x3 convolved feature result (red) by an element-wise multiplication.

To extact features of an image, CNNs alternate convolutions and poolings. Convolution is a mathematical operation between two objects that indicates correlated features of these objects. In computer vision, the two objects used in a convolution are the input image itself and a *kernel* or *mask*. A kernel is a two dimensional matrix. The form and the values of the kernel will define the effect of the convolution when applied to the image. For example, kernels can detect ridges, vertical or horizontal lines, or sharp the image. To perform a convolution, a window (subarea of the image) containing the kernel values will slide over the image iteratively. At each step, a convolved feature is outputed. Convolutions are characterized by parameters such as the size of the kernel, the stride i.e. by how much the window is moved in a single iteration, and paddings that can add pixel in the border of the image. Depending on these parameters, the outputed convolved features can have the same size, or a reduced dimensionality compared to the input image. Figure 2.6 shows 4 steps during a convolution, that applies an element-wise multiplication between the image and the kernel.

After a convolution, a pooling is performed. The pooling will reduce the dimensionality of the image, and extract the most dominant features that are invariant under translation and rotation. The pooling also uses sliding windows over the convolved image, that will output a specific characteristic depending on the type of

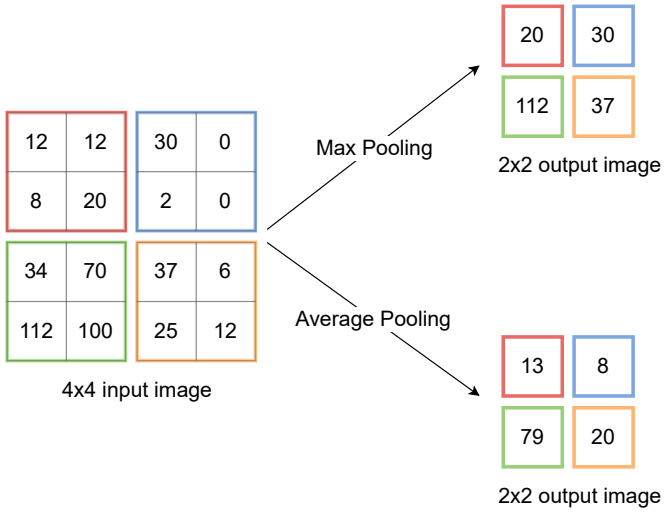


FIGURE 2.7: Two of the main types of pooling. Here, a 2x2 kernel is used. The max pooling extracts the maximal value of the kernel, while the average pooling extracts the average of all values within the kernel.

pooling. The two most used poolings are the *max pooling*, which extracts the maximal value in the kernel, or the *average pooling* that extracts the average value of all values in the kernel. In case of CNN, the max pooling is typically prefered. Figure 2.7 shows how the max and average pooling are performed.

2.6 Computer aided diagnosis tasks

In the last decades with machine learning and artificial intelligence coming more and more popular and powerful, improvements have been made in the medical field too. With the possibility to use computers to analyze several types of images for many purpose, medical images can be analyzed as well. Techniques can help radiologists in their work, as a second opinion or to detect primary signs of cancer that humans can miss. These techniques are called *Computer Aided Diagnosis* (CAD) systems [15]. With the development of deep learning, CAD systems become more and more precise and powerful, by using convolutional neural networks to analyze medical images and perform different tasks, such as classification, localization and segmentation.

2.6.1 Classification

Classification is the task to partition images depending on their characteristics (features). Classification can be binary, i.e. with only two classes, or multi-class. CAD systems can perform this task for example to classify images in a binary manner: The positive class regroups images containing one or more malignant tumors, while the negative class contains images having no malignant masses. The CNN of a CAD system can be trained using supervised learning, but it requires ground truth medical diagnosis. The medical images are the inputs of the network, while the medical diagnoses (true or false) are the labels. By feeding a significant amout of data, the CNN is train and can output a probability that an image contains a malignant tumor. The classification task performed by CAD systems can help radiologists to detect primarily stages of cancer.

(A) CT scan of the lungs (B) Segmentation mask (C) Tumor colored in red

FIGURE 2.8: Example image of a computed tomography scan of the lungs and its segmentation mask from the LIDC-IDRI dataset [18]. The detected malignant tumor is colored in red in the right image.

2.6.2 Localization

Localization is the task to points the position of an object in an image. In the medical field, a precise location of a malignant tumor is required when doctors are ablating (removing) it. CAD systems can also perform localization: given a medical image in input, they can tell a position where they find a malignant tumor. When training in a supervised learning manner, CAD systems need images as inputs but also ground truth positions (as coordinates for example) of malignant tumors within the image as labels. Finally, an overlay can for example display the location of the tumor outputed by the CAD system.

2.6.3 Segmentation

In the medical field, *segmentation* is the process to delimit or contour a specific body part. On one hand, CAD systems can segment organs: for example, Litjens et al. [16] analyzed the results of the 2012 *Prostate MR Image Segmentation* (PROMISE12) challenge organized by the Medical Image Computing and Computer Assisted Intervention society (MICCAI)[6]. The goal of this challenge was to develop CAD systems that can perform the segmentation of the prostate, given MRIs of the prostate as inputs. On the other hand, CAD systems can segment malignant tumors: for example, Wang et al. [17] develop a CAD system capable of segmenting brain tumors. CAD systems can do segmentation in two or three dimensions. Like classification and localization, segmentation can also be trained with supervised learning. The inputs are the medical images, and the labels are the so called *segmentation masks*. These masks are monochrome (binary) images that have the same size as the input image. Each pixel of the segmentation mask corresponds to a pixel in the medical image. If the pixel is black, the malignant tumor is not present in that position, while if the pixel is white, the tumor is present. Figure 3.1 illustrates an example of a medical image (input) in 2.8a with its corresponding segmentation mask (label) in 2.8b, and both are display in a same image in 2.8c, where a malignant tumor is colored in red.

[6] http://www.miccai.org/

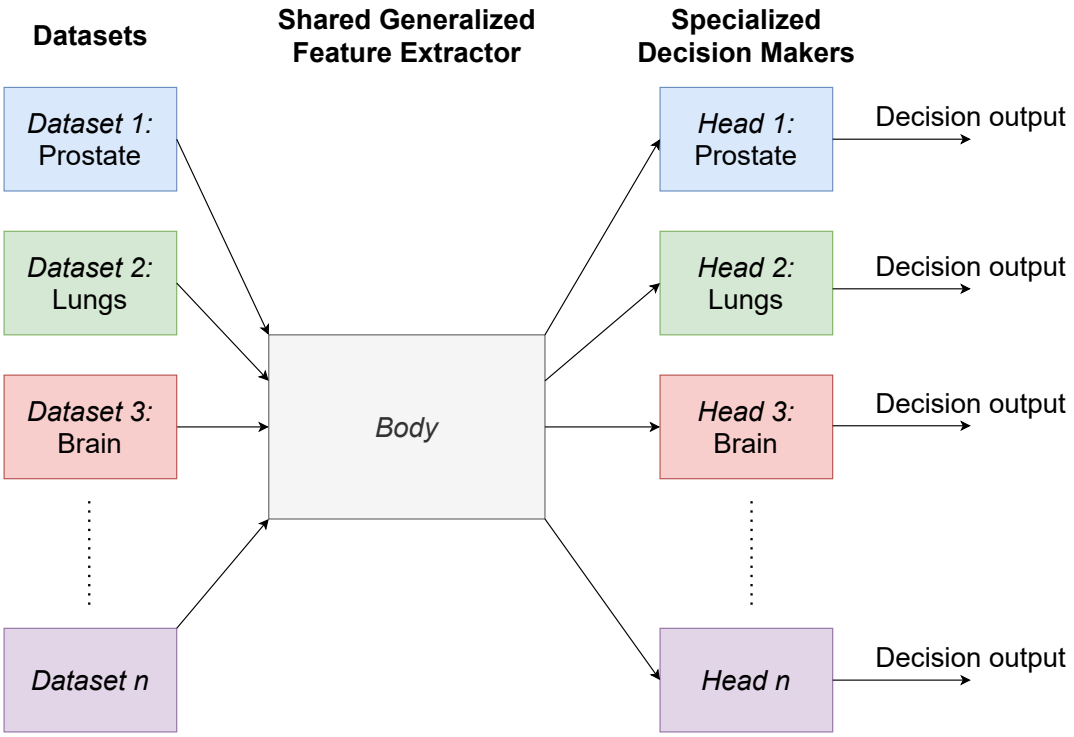


FIGURE 2.9: Generic Hydra model. The body is a feature extractor and is common for all type of organs. The model has then many heads that are the decision makers, each head being specialized for one specific organ. The input dataset enters the network first in the shared feature extractor (the body), and goes to the specialized decision maker attached depending on the organ (the head). The decision goes out of the head.

2.7 Hydra framework

As stated in the motivation of this work, the main goal of this project is to develop a CAD system with the collaboration of the H-FR. This CAD system can help the radiologists in their work. At this time, the Hydra project has been first trained to do tumor detection on the prostate, the lungs and the brain. The name Hydra comes from the shape of the network. It is composed of a body with many heads, like the mythical creature: The body is the feature extractor of the network, i.e. the part where the image is analyzed and convolved and is the same for all type of organs, because tumors share common characteristics. The multiple heads are a set of decision makers returning the probability or not to find a tumor. Each head is specialized for a precise organ. Currently, there is one head for the prostate, one for the lungs and one for the brain. When predicting a specific organ, the body of Hydra will be taken, and the head for the specific organ will be attached. In that way, the feature extraction is general enough to have a good intuition for the decision, and the decision maker is specialized enough to give the right decision for a specific organ. Figure 2.9 shows the generic form of the Hydra CNN model.

This type of architecture avoids the problem of publicly available medical data scarcity stated in the Section 2.5. It has two advantages:

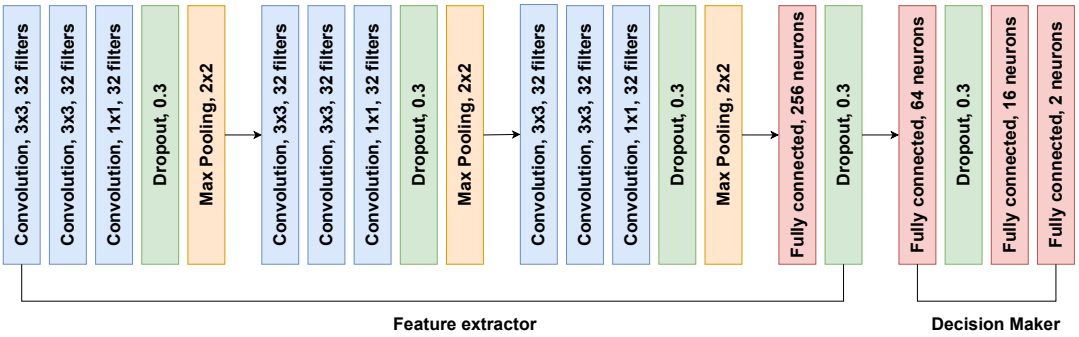


FIGURE 2.10: Full Hydra model, with its layers. The feature extractor is composed of 3 blocks of convolutions ending by a dropout and a max pooling layer. The decision maker is a block of fully connected layers.

1. Hydra can learn features from tumors for any organs. Even if data is sparse for a new organ, Hydra uses what it has previously learned from other organs to extract features. In that way, the data used to train the feature extractor is augmented, by aggregating data of any organs. The feature extraction is then generalized.

2. Each decision maker of Hydra is specialized only for one specific organ. Thus, the heads do not need a lot of data to be train. The scarsity of the data for an organ avoids the generalization of the heads, that could lead to incorrect predictions.

The current convolutional neural network model used for Hydra is based on the VGG-16 model architecture [19], where 16 is the number of layers in the model. The Hydra model contains 9 convolutional layers separated in 3 convolutional blocks of 5 layers each, and 4 fully connected layers at the end. The activation function used after the convolutions is the ELU (2.11) (see Section 2.2). Dropout layers are used to prevent overfitting, as training large neural network on relatively small training datasets could overfit the training dataset. To avoid that, dropout layers skip or ignore randomly some neuron's inputs and outputs during the process. This has the effect to preventing the neurons to co-adapt themselves too much during the training [20]. Figure 2.10 shows the current full model Hydra with its separated blocks.

Chapter 3

Datasets

This Chapter presents the three datasets that have been choosen to be part of this work. The first section, after a small introduction, explains how the choice of these datasets was made. Then, the types of medical images that are used in this work will be described. These types are ones of the most common used. At the end of the second section, a little detour is taken to present the DICOM standard, which is a common standard for medical imaging. Finally in the last section, the three datasets will be presented.

3.1 Introduction and choices

Deep learning models require a lot of data to be efficient (see Section 2.5). However, in research of deep learning methods in the medical field, data is often scarce. Indeed, lack of publicly available medical data is an issue because of the personal and sensible data of the patients. The data needs to be anonymized before publishing. Today, there exist such publicly available data, provided by the following ressource: the Cancer Imaging Archive (TCIA) [21] is funded by the National Cancer Institute (United States). The data is separated in collections, that have a common disease, the same medical images type (see Section 3.2) or is at the base of a common research goal.

The first part of this work was to choose different datasets to preprocess. Different conferences, like the Medical Image Computing and Computer Assisted Intervention society (MICCAI)[1], the Medical Imaging with Deep Learning (MIDL)[2] or the Machine Learning for Healthcare Conference (MLHC)[3] offer many publications in the scope of this work. About 25 publications coming from these conferences were read in detail, but only 5 were retained [17, 22, 23, 24, 25]. The available datasets of these 5 publications went on a common list. Finally, the work can start with three new datasets choosen from this list. Two of them come directly from TCIA.

3.2 Types of medical images

Depending on the body part to look at, a different type of image or scan can be used. There exist many types of medical images, depending on the hardware, the technology used or the parameters for a same setup. The medical images types are sometimes called the *modality* of the image. Modalities used in this work are:

[1] http://www.miccai.org/
[2] https://2021.midl.io/
[3] https://www.mlforhc.org

- Magnetic resonance image (MRI)
- Computed tomography (CT) scan
- Positron emission tomography (PET) scan
- Ultrasound (US) image

These four modalities will be explained in detail in the following paragraphs. While CT, PET scans and US images come in the datasets presented in this work, the MRI modality comes from previous datasets preprocessed for the *Hydra* framework in the work of Clement and Jobin [5].

3.2.1 Magnetic resonance image

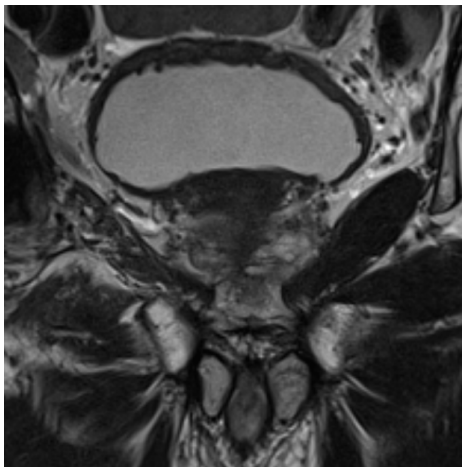
The first modality, the *Magnetic resonance image*, produces three-dimensional images from the body, as a stack of 2D images. It uses a strong magnetic field, that will force the protons in the body to align with the field. With a magnetic field that has a frequency dependance, protons will align and dealign as the time passes. This motion of the protons and the energy released during the process is then captured by some sensors, that produce the image. MRI is more often used for soft tissues like muscles, ligments or tendons. For example, MRI is used for the knees, the shoulders or even the brain. To get the image, patients lie on a bed that goes inside a MRI scanner, that has the shape of a closed tunnel, and they stay fixed during the process [26]. Figure 3.1a shows a MRI of the prostate.

3.2.2 Computed tomography scan

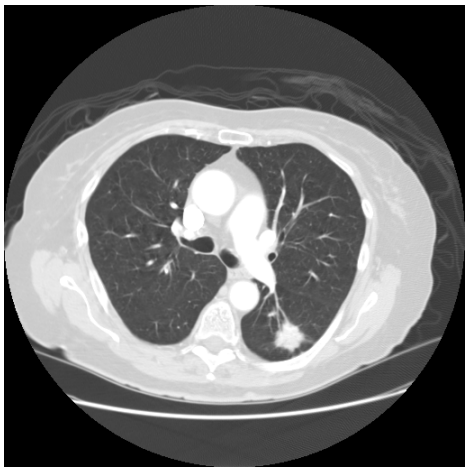
The second type is the *computed tomography scan*. These scans produce two-dimensional images that can then be stacked to form a full-body image, as for MRI. CT scans use X-rays, that are emitted and go through the body, and are then captured by detectors on the other side. A 2D image is then constructed. CT scans can better see hard tissues, like bones. It can then be used to see bones fractures, but also to image the lungs or the brain. To construct the image, patients lie on a bed that moves slowly through a scanner that has a ring shape. The scanner will emit the X-rays as the patient goes through [27]. Figure 3.1b shows a CT scan of the lungs.

3.2.3 Positron emission tomography scan

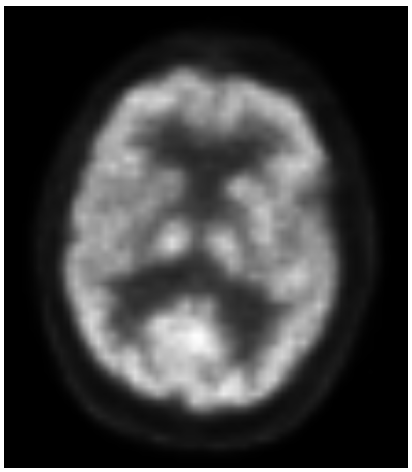
The third type mentionned is the *positron emission tomography scan*. While the two previous types were non-invasive techniques, the PET is invasive. The patient receives first an intravenous injection of a safe radioactive substance, called the *radiotracer*. The radiotracer is absorbed by the body cells: cells that are diseased will absorb more radiotracer than the healthy ones. This is because the radiotracer accrues in cells that need a lot of glucose. As tumor cells need a significant amount of glucose, it is well absorbed by tumors. By going through a PET scanner, detectors can see which parts have absorbed more of the radiotracer, and can construct 2D images that will be stacked to form a full body three-dimensional image [28]. Figure 3.1c shows a PET scan of the brain.



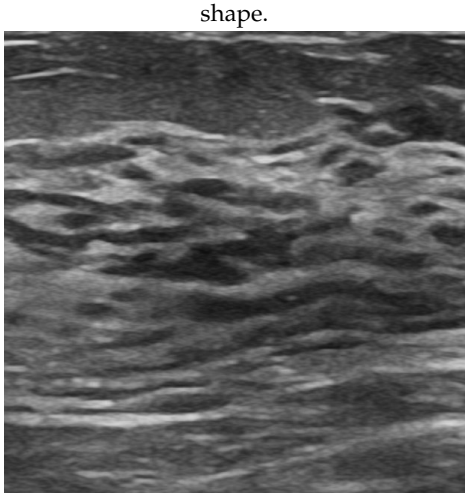
(A) Example of a magnetic resonance image of the prostate. This image comes from the PROSTATEx dataset [7] and has been cropped from the original image.



(B) Example of a computed tomography scan of the lungs, taken from the LIDC-IDRI dataset [18]. At the bottom right of the image, a cancerous tumor can be seen, with its typical spider-web-shape.



(C) Example of a positron emission tomography scan of the brain from the HNPC dataset [30].



(D) Example of an ultrasound image of the breast, taken from the BUSI dataset [31]. This image has been cropped from the original image, to focus on the cancerous tumor.

FIGURE 3.1: Example figure that shows four different modalities of medical images.

3.2.4 Ultrasound image

Finally, the last type listed above is the *ultrasound images*. This is also a non-invasive technique that uses ultrasound waves emitted from a device named transducer, that is directly touching the skin of the patient. The transducer first projects sound waves within the body, which travel and are then reflected by the different body parts. The transducer listens to the reflected waves, and by measuring the amplitude of the waves and the time they take to reflect, the transducer can compute a 2D image of the tissues and organs. The ultrasound image is then produced in realtime, and can capture the motion of the body parts. This type of image is widely used for several echography, or for breast imaging for example [29]. Figure 3.1d shows an ultrasound image of the breast (mammography).

3.3 DICOM standard

Different types of medical images need different hardware, scanners but also the screens on which the image is displayed, or the sensors. All this hardware setup will lead to compatibility problems. Indeed, hospitals around the world have not the exact same hardware to display or create the medical images. When patients go from one hospital to another, images could be displayed differently. To solve this problem, a standardization for medical image is introduced as the *Digital Imaging and Communications in Medicine* (DICOM) standard for medical images and their information. Born in 1993 under the collaboration of the American College of Radiology (ACR) and the National Electrical Manufacturers Association (NEMA), the DICOM standard has revolutionized the medical world, both for doctors and patients. It has enabled advanced medical imaging system, and is one of the most deployed healthcare messaging system. DICOM is the `ISO 12052:2017` standard [32].

DICOM files can be recognized by their `.dcm` file extension. The structure of a DICOM file can be represented as a dictionary structure, each entry being a key-value pair. Keys are called *DICOM tags* and are tuples of two hexadecimal numbers, uniquely identified, which correspond to a certain value. A tool that explains and describes all DICOM tags of all type of images is available at the DICOM standard browser[4]. These hundred of different tags contain all medical information: from the patient information, to the size of the image, to the type of image, to some numeric identifiers, to settings of the scanner who takes the image. Each DICOM file contains all these tags but also one or more 2D images (slices), stored in a specific DICOM tag, named `Pixel Data`. Figure 3.2 shows an example of DICOM tags available in a CT scan.

To solve the compatibility problems mentioned above, DICOM provides some information in its tags. For example, it provides slope and intercept values to rescale the image (tags `(0028,1052)` and `(0028,1053)`). Additional information like the pixel spacing (tag `(0028,0030)`) or bits allocated (tag `(0028,0100)`) are also useful to normalize images from a dataset. Section 4.1 shows more technical details to convert medical images.

3.4 Lung Image Database Consortium image collection

The first dataset that will be used in this work is the Lung Image Database Consortium image collection (LIDC-IDRI) dataset [18]. This dataset is the result of the collaboration of seven academic centers and eight medical imaging companies, under the initiation of the *National Cancer Institute* (NCI), with the help of the *Food and Drug Administration* (FDA) (United States). The goal of this collaboration and this dataset is "to develop, train and evaluate computer aided-diagnosis methods for lung cancer detection and diagnosis" [33].

The dataset contains three different types of scans: computed tomography (CT), digital radiography (DX) and computed radiography (CR) scans of the lungs. These scans cover about 1000 different patients, for a total of more than 240'000 images. All images in the dataset are in the DICOM format. In the LIDC-IDRI dataset, four radiologists have participated together to analyze and mark the whole set of CT scans. Xie et al. [22] and Ding et al. [23] have used this dataset for a detection task, while in this work it is used for a localization and a segmentation task (see Section 4.3).

[4] https://dicom.innolitics.com/ciods

```
(0018, 1130) Table Height                       DS: "225.0"
(0018, 1140) Rotation Direction                 CS: 'CW'
(0018, 1150) Exposure Time                      IS: "1426"
(0018, 1151) X-Ray Tube Current                 IS: "330"
(0018, 1152) Exposure                           IS: "70"
(0018, 1160) Filter Type                        SH: 'BODY FILTER'
(0018, 1170) Generator Power                    IS: "39600"
(0018, 1190) Focal Spot(s)                      DS: "0.7"
(0018, 1210) Convolution Kernel                 SH: 'STANDARD'
(0018, 5100) Patient Position                   CS: 'HFS'
(0020, 000d) Study Instance UID                 UI: 1.3.6.1.4.1.14519.
(0020, 000e) Series Instance UID                UI: 1.3.6.1.4.1.14519.
(0020, 0010) Study ID                           SH: ''
(0020, 0011) Series Number                      IS: "2"
(0020, 0012) Acquisition Number                 IS: "1"
(0020, 0013) Instance Number                    IS: "100"
(0020, 0032) Image Position (Patient)           DS: [-325, -325, 347.5]
(0020, 0037) Image Orientation (Patient)        DS: [1, 0, 0, 0, 1, 0]
(0020, 0052) Frame of Reference UID             UI: 1.3.6.1.4.1.14519.
(0020, 1040) Position Reference Indicator       LO: 'XY'
(0020, 1041) Slice Location                     DS: "347.5"
(0028, 0002) Samples per Pixel                  US: 1
(0028, 0004) Photometric Interpretation         CS: 'MONOCHROME2'
(0028, 0010) Rows                               US: 512
(0028, 0011) Columns                            US: 512
(0028, 0030) Pixel Spacing                      DS: [1.269531, 1.269531]
(0028, 0100) Bits Allocated                     US: 16
(0028, 0101) Bits Stored                        US: 16
(0028, 0102) High Bit                           US: 15
(0028, 0103) Pixel Representation               US: 0
(0028, 0303) Longitudinal Temporal Information M CS: 'MODIFIED'
(0028, 1050) Window Center                      DS: "40.0"
(0028, 1051) Window Width                       DS: "350.0"
(0028, 1052) Rescale Intercept                  DS: "-1024.0"
(0028, 1053) Rescale Slope                      DS: "1.0"
(3253, 0010) Private Creator                    LO: 'Varian Medical Systems VISION 3253'
(7fe0, 0010) Pixel Data                         OW: Array of 524288 elements
```

FIGURE 3.2: DICOM file content example for a CT scan. On the left, DICOM tags which are tuples of two hexadecimal numbers, have an associated name to be readable by humans. On the right, the value of each DICOM tag is displayed. The `UID` tags values have been cut for readability.

The work of analysing this dataset has been separated in two different phases. During the first phase, the radiologists read all the CT scans in a blind manner, i.e. the radiologists have never seen the images before, and they do not know anything about the past history of the patients. Each radiologist was on his own, with no additional information, and had to mark the lesions and class them in three different categories:

1. Nodules >= 3 mm diameter
2. Nodules < 3 mm diameter
3. Non-Nodules >= 3 mm diameter

For the first category, the radiologists drew the contour of this type of nodule in each slice of CT scan where the nodule appeared. Then, they were asked to subjectively give their opinion about some characteristics related to the nodule, for example the difficulty to detect the nodule, its internal structure, how well its margins are defined, or its malignancy. All characteristics are given in a scale between 1 and 5.

For the last two categories, the radiologists just gave the three-dimensional center of mass of these (non-)nodules.

```
<unblindedReadNodule>
    <noduleID>0</noduleID>
    <characteristics>
            <subtlety>5</subtlety>
            <internalStructure>1</internalStructure>
            <calcification>6</calcification>
            <sphericity>5</sphericity>
            <margin>4</margin>
            <lobulation>1</lobulation>
            <spiculation>5</spiculation>
            <texture>4</texture>
            <malignancy>4</malignancy>
    </characteristics>
    <roi>
            <imageZposition>-105.0</imageZposition>
            <imageSOP_UID>1.3.6.1.4.1.14519.5.2.1.</imageSOP_UID>
            <inclusion>TRUE</inclusion>
            <edgeMap>
            <xCoord>311</xCoord>
            <yCoord>361</yCoord>
            </edgeMap>
            <edgeMap>
            <xCoord>310</xCoord>
            <yCoord>362</yCoord>
            </edgeMap>
            <edgeMap>
```

LISTING 3.1: XML content example for a nodule >= 3 mm of the LIDC-IDRI dataset [18]. There is a nodule ID, some characteristics and a `roi` (region of interest) tag that contains the contour points (x, y) of the nodule. The `imageSOP_UID` tag has been left out for readability.

During the second phase, the results of the first phase were complied and send back to the four radiologists. They then read the marking work of the other radiologists. Finally, with the help of the four diagnoses, they were been asked to make a final decision about the marking and the characteristics.

All diagnoses, characteristics and annotations of the radiologists are stored in XML files. There is one XML file per CT series. An example of XML file for a nodule of the first category (Nodules >= 3 mm) is showed in the Listing 3.1. This example contains a nodule ID, the characteristics mentioned before, and the contour of the nodule, which is contained in the tag named `roi` (region of interest). The contour contains all the x and y coordinates of the points that make the contour of the nodule, and the universal identifier (UID) of the corresponding slice.

3.5 Head-Neck-PET-CT dataset

The second dataset that will be used in this work is the Head-Neck-PET-CT (HNPC) dataset [30]. It is a compilation of images, diagnoses and other medical information that are coming from four different institutions in Québec, Canada. 92 patients are coming from the *Hôpital Général Juif* (HGJ) in Montréal, 102 from the *Centre Hospitalier Universitaire de Sherbrooke* (CHUS), 41 patients from the *Hôpital Maisonneuve-Rosemont* (HMR), and finally 65 from the *Centre hospitalier de l'Université de Montréal* (CHUM).

The dataset contains computed tomography (CT) and positron emission tomography (PET) scans of the head, including the brain, to the neck. Both images types are given in the DICOM format. The HNPC dataset contains 300 different patients, that all have histologically proven head and neck cancer. The dataset contains a total of about 120'000 images. This patients data was used first in the work of Vallières et al. [34], that has been the trigger of the creation and compilation of the HNPC dataset.

The HNPC dataset contains, for each patient, several series. For each CT and PET scans series mentionned before, there is a corresponding radiotherapy structure set (RTStruct) series, that is a single DICOM file containing the contour points for several regions of interest (ROI) inside a DICOM tag. To illustrate, Figure 3.3 shows a part of the `structure set ROI` DICOM tag, that contains ROI names and their correponding ROI number used further in the RTStruct, as a reference. The RTStruct file contains then, for each slice of its corresponding images series, the contour points of the ROIs and also a universal identifier that uniquely identifies the corresponding slice. These RTStruct series are the result of the work of radiologists coming from the four different institutions, that draw the contour of the ROIs, including tumors. The dataset contains also two other series named radiotherapy plan (RTPlan) and radiotherapy dose (RTDose). These two series contain more medical information about the patients, such as the planning of the therapy or the scheduling for the doses given to the patients.

With the HNPC dataset, two excel files are attached. The first of the two excel files is called `INFO_clinical_HN.xlsx` and contains textual medical information about all the patients, including for example the stage of the cancer, its primary location, the time of the diagnosis or the length of the treatment. In the second excel file, named `INFO_GTVcontours_HN.xlsx`, the gross tumor volume (GTV) names are listed under the same name as they appear in the RTStruct, as ROI name (see the DICOM tags `(3006,0026)` of the Figure 3.3).

```
(3006, 0020)  Structure Set ROI Sequence  27 item(s) ----
   (3006, 0022) ROI Number                          IS: "1"
   (3006, 0024) Referenced Frame of Reference UID   UI: 1.3.6.1.4.1.14519.
   (3006, 0026) ROI Name                            LO: 'BODY'
   (3006, 0036) ROI Generation Algorithm            CS: 'MANUAL'
   ---------
   (3006, 0022) ROI Number                          IS: "2"
   (3006, 0024) Referenced Frame of Reference UID   UI: 1.3.6.1.4.1.14519.
   (3006, 0026) ROI Name                            LO: 'Brain'
   (3006, 0036) ROI Generation Algorithm            CS: 'MANUAL'
   ---------
   (3006, 0022) ROI Number                          IS: "3"
   (3006, 0024) Referenced Frame of Reference UID   UI: 1.3.6.1.4.1.14519.
   (3006, 0026) ROI Name                            LO: 'Brain Stem'
   (3006, 0036) ROI Generation Algorithm            CS: 'MANUAL'
   ---------
   (3006, 0022) ROI Number                          IS: "4"
   (3006, 0024) Referenced Frame of Reference UID   UI: 1.3.6.1.4.1.14519.
   (3006, 0026) ROI Name                            LO: 'Brain Stem PV'
   (3006, 0036) ROI Generation Algorithm            CS: 'MANUAL'
   ---------
   (3006, 0022) ROI Number                          IS: "5"
   (3006, 0024) Referenced Frame of Reference UID   UI: 1.3.6.1.4.1.14519.
   (3006, 0026) ROI Name                            LO: 'COUCH'
   (3006, 0036) ROI Generation Algorithm            CS: 'MANUAL'
   ---------
   (3006, 0022) ROI Number                          IS: "6"
   (3006, 0024) Referenced Frame of Reference UID   UI: 1.3.6.1.4.1.14519.
   (3006, 0026) ROI Name                            LO: 'CTV 1'
   (3006, 0036) ROI Generation Algorithm            CS: 'MANUAL'
```

FIGURE 3.3: DICOM content example for an RTStruct series of the HNPC dataset [30]. DICOM tags can be seen on the left as tuples of two hexadecimal numbers, and their corresponding value can be seen on the right. The `UID` tags values have been cut for a better readability of the image.

3.6 Breast Ultrasound Images dataset

The third and last dataset used in this work is the Breast Ultrasound Images (BUSI) dataset [31]. This dataset is a collection of ultrasound images from the breast. The images were all collected in 2018, and they come from 600 different female patients, aged from 25 to 75. Originally, 1100 ultrasound images were collected at the *Baheya hospital* in Egypt, under the DICOM format. Before constructing the dataset, radiologists have marked and reviewed annotations in the images.

The BUSI dataset contains, after reduction, a total of 780 ultrasound images, cropped and converted into the PNG format. The ground truth images, i.e. the segmentation masks, are given along the original images. These segmentation masks have been created by the radiologists over a year of work. For some images, there are more than one segmentation mask, if there is more than one tumor. The breast ultrasound images are separated in three categories: normal, benign and malignant. Each of these categories is a complete folder given in the dataset, containing images and their segmentation masks.

Chapter 4

Segmentation and preprocessing scripts

This chapter presents the two core contributions of this work. First, a small introduction about the conversion of images is shown. The first main contribution is then explained. It consists of a new module that creates automatically the segmentation masks given the contour of a tumor. The second contribution is the preprocessing of the data itself. For each of the three datasets presented in the Chapter 3, the preprocessing is explained in detail. All code is written in the Python programming language and publicly available on GitHub[1].

4.1 Images conversion

In computer science, images are stored as tables of numbers. Each cell in the table corresponds to one pixel in the image, and contains a certain value (number) that represents the pixel intensity, i.e. the strength of the pixel in a the given type of image. An image can have one or more channels, i.e. layer of pixels, that are stacked to form the image. Three of the most used types of image are:

1. The *monochrome image*, also called binary image, contains a binary intensity that is either on or off, and can be interpreted as black and white image. The pixels have either the value 0 (black) or the value 1 (white). The monochrome image is thus encoded in 1 bit per pixel, and has only one channel.

2. The *greyscale image* contains only intensities on a grey scale. The difference with the monochrome image is that the grey color has many shades and not only 2. The pixel intensity represents then the level of grey of the pixel, and the value depends on the encoding. For example, if a greyscale image is encoded in 8 bits per pixel, the range of the pixel intensity will be $[0, 255]$. Here, the black pixel corresponds to the value 0 and the white to the 255 value. The greyscale image has also only one channel.

3. The *red green blue (RGB) image* is a color image. To represents the colors, shades of red, green and blue are added. The pixel intensity depends on the encoding too. The RGB image contains 3 channels, one for the red, one for the green and one for the blue. These 3 channels are then stacked to form the RGB image. In that case, a pixel is not represented by only one value, as in the two previous types, but by a tuple of three values, one for each RGB colors.

[1] https://github.com/ChristopheBroillet/Hydra_Segmentation

As stated in Section 3.3, the DICOM standard aims to standardize all medical images and data between medical institutions worldwide. DICOM converts images with information contained in DICOM tags. The conversion described below is applied on the `pixel data (7fe0,0010)` DICOM tag, that contains one or more medical images, and will result in converted output images. The conversion is done in two steps that are explained below, while the corresponding code of this work is written in the `normalize_dicom` module available on GitHub[2].

4.1.1 Hounsfield correction

The *Hounsfield units* (HU) are universally used units in CT or PET scans, to display a standardized image. The Hounsfield scale is based on the measure of the radiodensity, i.e. the ability of electromagnetic waves to go through a given material. The arbitrarily defined radiodensity of the distilled water (at standard temperature and pressure) is 0 HU, while radiodensity of air is -1000 HU [35].

To convert a CT or PET scan, the raw data pixels are first converted using the Hounsfield correction. The equation used to convert original data to HU is given by:

$$HU = m \cdot P + b \tag{4.1}$$

where HU is the output value in Hounsfield unit, m is the rescale slope given in the `rescale slope (0028,1053)` DICOM tag, P the input value of the pixel and b the rescale intercept given in the `rescale intercept (0028,1052)` DICOM tag.

The use of the Hounsfiels units have helped radiologists to interpret and diagnose disease on CT or PET scans. Indeed, scanners take images that have in principle between 12 and 16 bits per pixel, that is between 4096 to 65536 shades of grey. But human eyes struggle to distinguish this amount of shades, while medical displays support encodings from 8 to 10 bits per pixel nowadays. To avoid these two problems, CT and PET scans are converted using the Hounsfield units [36].

4.1.2 Normalization

After this first correction in HU, another linear transformation needs to be applied. This second transformation is necessary in order to keep a normalized output range value $[y_{min}, y_{max}]$ for the pixels. To get this output range, the following rules[3] will be applied to the resulting HU from the first correction for CT and PET scans. For other modalities, they will be applied directly on the pixel values of the image:

$$\begin{aligned} &\text{if } HU \leq c - 0.5 - \frac{w-1}{2}, \text{ then } y = y_{min} \\ &\text{else if } HU > c - 0.5 + \frac{w-1}{2}, \text{ then } y = y_{max} \\ &\text{else } y = \left(\frac{HU - (c - 0.5)}{w - 1} + 0.5 \right) \cdot (y_{max} - y_{min}) + y_{min} \end{aligned} \tag{4.2}$$

with HU the input pixel value, c the window center given in the `window center (0028,1050)` and w the window width in the `window width (0028,1051)`, y_{min} and y_{max} the minimum and maximum value of the output range, and y the output value.

[2] https://github.com/ChristopheBroillet/Hydra_Segmentation/blob/main/preprocessing_scripts/Head-Neck-PET-CT/normalize_dicom.py

[3] These rules are taken from the DICOM standard browser:
https://dicom.innolitics.com/ciods/ct-image/voi-lut/00281050

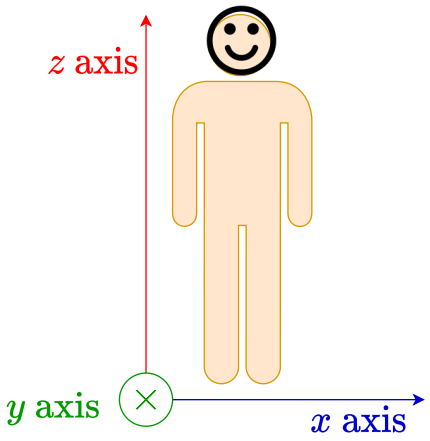


FIGURE 4.1: Example of an orientation of the patient-base coordinate system. The x axis points from the right to the left of the patient, the y axis from the face to the back of the patient (the axis enters the plane) and the z axis points from the feet to the head of the patient.

In this work, the output range value will be $[0, 255]$, as the output image will be a greyscale image with 8 bits per pixel. In this case, $y_{min} = 0$ and $y_{max} = 255$.

4.2 The `segmentation_mask` module

In this work, a new module is provided: the `segmentation_mask` module. The goal of this module is to create a segmentation mask for a medical image, given the contour points of the segmentation of the tumor. These contour points are given as ground truth with the dataset they belong to. With these segmentation masks, the deep learning model can be fed with both medical images and segmentation masks. As explained in the Section 2.5, the medical images are the inputs of the network and the segmentation masks are the labels. The `segmentation_mask` module contains two new functions described in the following paragraphs.

4.2.1 The `mm_to_imagecoordinates` function

In some cases, like in the Head-Neck-PET-CT dataset, the segmentation points are given in the Patient-Based Coordinate System (PBCS), that are in absolute distance given in millimeters within the patient body. The PBCS is a 3D orthonormal right-handed system, i.e. where the cross product of a unit vector along a x axis and y unit vector gives a z unit vector. Figure 4.1 shows one of the most used PBCS orientation.

The preprocessing and conversion of the images need the row and column indices of the image. The first function of the module, called `mm_to_imagecoordinates`, is used to convert units from the PBCS given in mm, to the image plane coordinate system (IPCS) given in indices. The formula that will be used to convert from one system to the other is taken from the DICOM standard browser[4]:

$$\begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix} = \begin{bmatrix} X_x\Delta_i & Y_x\Delta_j & 0 & S_x \\ X_y\Delta_i & Y_y\Delta_j & 0 & S_y \\ X_z\Delta_i & Y_z\Delta_j & 0 & S_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \\ 0 \\ 1 \end{bmatrix} \tag{4.3}$$

[4] https://dicom.innolitics.com/ciods/ct-image/image-plane/00200032

The point (P_x, P_y, P_z) represents the coordinates of the point along the three x, y, z axis in the PBCS. It is the point that will be converted in the IPCS. The point given by (S_x, S_y, S_z) is the upper-left pixel in the PBCS, that will be the origin $(0, 0, 0)$ of the converted image in the IPCS. It is contained in the `image position (patient) (0020,0032)` DICOM tag. Then, the DICOM tag `image orientation (patient) (0020,0037)` gives the $(X_x, X_y, X_z, Y_x, Y_y, Y_z)$ values. The (X_x, X_y, X_z) are the direction cosines (i.e. cosines of angles) between the x axis of the Figure 4.1 and the row direction of the IPCS. It expresses the change of direction in the PBCS when moving from one column to the next in the IPCS. Similarly, the (Y_x, Y_y, Y_z) are the direction cosines between the y axis of the Figure 4.1 and the column direction of the IPCS, and it expresses the change of direction in the PBCS when moving from one row to the next in the IPCS. For example, if the orientation of the PBCS is exactly the one presented in the Figure 4.1, then $(X_x, X_y, X_z, Y_x, Y_y, Y_z)$ will be equal to $(1, 0, 0, 0, 1, 0)$. Further, the `pixel spacing (0028,0030)` DICOM tag gives the column spacing Δ_i between the center of two adjacent pixels in mm, and Δ_j the row spacing. Finally, i is the *column* index and j the *row* index of the converted image.

As images are in two dimensions, the first two equations of (4.3) are:

$$\begin{cases} P_x = X_x \Delta_i i + Y_x \Delta_j j + S_x \\ P_y = X_y \Delta_i i + Y_y \Delta_j j + S_y \end{cases} \tag{4.4}$$

The `solve` method of the `linalg` module provided by `NumPy` [37] is used to solve this system of two equations for i and j. The next described function uses the presented function in an optional manner. If the contour points are given in the PBCS, the script needs to call this method to convert the points. If the contour points are already given in the IPCS, then no conversion is required.

4.2.2 The `create_segmentation_mask` function

The second and main function of this module is the `create_segmentation_mask` function. Indeed, this is the function that is primarily called in the preprocessing scripts to start the creation of the segmentation masks. To open and use DICOM files in Python, this function uses the `pydicom` library [38]. The outputed segmentation masks are monochrome images. Indeed, black pixels (value 0) represent no tumor, while white pixels (value 1) represent tumor. As arguments, the function takes:

- The data contour points of the segmentation. The data needs to be a 2D list, i.e. a list of points (x, y) of the segmentation. The data is considered as ground truth.
- The medical image slice that corresponds to the segmentation. This is an opened DICOM file from the `pydicom` library.
- The path to an output folder. This folder will contain the outputed segmentation masks.
- A boolean called `conversion` that indicates if the conversion is required or not. If `conversion` is *True*, the `create_segmentation_mask` function will use the previous described function in Section 4.2.1.

Algorithm 1 shows in pseudocode the different steps to create the segmentation masks, by calling the `create_segmentation_mask` function. The following paragraphs explain those steps in detail.

Algorithm 1 `create_segmentation_mask` function

function CREATE_SEGMENTATION_MASK($img, data, output\_folder, conversion$)
  Create a log_file ▷ `.txt` file
  Initialize $seg\_mask \leftarrow img \cdot 0$ ▷ 0 for black pixels
  for each *point* in *data* **do**
    if *conversion* **then** ▷ *conversion* is a boolean
      $x, y \leftarrow$ `mm_to_imagecoordinates`$(img, point)$
    else
      $x, y \leftarrow point$
    $seg\_mask[y][x] \leftarrow 1$ ▷ Add white pixel
  Initialize *white_boundaries* ▷ Will contain white indices for each row
  for each *white_row* in *seg_mask* **do**
    white_boundaries append $[row\_idx, min\_white\_idx, max\_white\_idx]$
  Sort *white_boundaries* by *row_idx*
    ▷ Two rows at a time, row counter incremented by 1 at each iteration
  for each $first, second$ in *white_boundaries* **do**
    if $first$ and *second* not differ by 1 in their indices **then** ▷ Missing line
      Add missing line with average value of $first$ and *second*
      Write entry in log_file
    ▷ Three rows at a time, row counter incremented by 1 at each iteration
  for each $first, second, third$ in *white_boundaries* **do**
    $average \leftarrow average(first, second)$
    Define $ERROR\_THRESHOLD = 0.05$ ▷ Tolerate to 5% error
    if $\left|\frac{average - second}{average}\right| > ERROR\_THRESHOLD$ **then** ▷ Absurd value
      $second \leftarrow average$
      Write entry in log_file
  Sort *white_boundaries* by *row_idx*
  Fill white rows in *seg_mask* according *white_boundaries*
  return *seg_mask*

At first, an empty mask is initialized, with all pixels value to be 0 (black pixels). This mask will have the same size as the medical image. Then, the contours points given as argument of the function are drawn on the mask as white pixels (value 1). At the same time, a simple log file is created to keep track of the modifications and potential errors occuring during the second step.

The data contour points given with datasets can be partially missing or wrong. Indeed, data is entered by humans and can be the source of mistakes. Data transmission between medical institutions, as well as digitization of old data can also lead to errors. During the second step, the data of the contour points is thus imputed and mitigated if necessary. The presented module first proceeds to imputation, i.e. looks at missing data. It checks if there is a missing line among the contour by looking at the row indices of two consecutive lines. If a line is missing, it looks at the neighbour lines and compute their average white boundaries. The white boundaries delimit the section of a row that contains white pixel. Finally, an entry in the log file is added, specifiying which line was missing.

After imputation of the data, i.e. completing the potential missing rows, the function will then start a data mitigation process, to look for absurd values. It will look at three rows at a time for each iteration step, and compute the average value of

```
Row 220 is missing
Row 223 is missing
Rows 217 - 218 - 219 have min absurd value. Before: 285 - 307 - 284. After: 285 - 284 - 284
Rows 218 - 219 - 220 have max absurd value. Before: 308 - 285 - 296. After: 308 - 302 - 296
Rows 221 - 222 - 223 have min absurd value. Before: 284 - 307 - 295. After: 284 - 289 - 295
```

FIGURE 4.2: Example log file from the `creste_segmentation_mask` function. This file tells the user that the rows 220 and 223 were missing in the original data, and were added by the function. Then, the function mitigated some values that had more than 5% error. The file shows the old and new values.

the first and third line. This computed value will then be compared with the value of the second (middle) line. If the comparison results in a too big difference between the values, defined by a threshold, the middle value is considered as an error or an absurd value. In this work, a too big difference (absurd value) is when values of white boundaries differ from more than 5% than their neighbours. The data is then mitigated by replacing the old value with the neighbour average value, and an entry is added to the log file, with the old and new values.

Figure 4.2 shows an example log file for an image of the HNPC dataset. The HNPC contained some missing lines and absurd values in the data extracted from the RTStruct series, while the LIDC-IDRI dataset had no errors.

The following sections will present and explain the preprocessing Python scripts for the new datasets, in both textual information or in pseudocode. To manage the DICOM files contained in the datasets but also to get the access to the DICOM tags, the scripts will use the `pydicom` library.

4.3 LIDC-IDRI preprocessing

In order to use this dataset for a supervised learning training purpose, the images and the medical diagnoses have to be preprocessed first. Because of simplicity and completeness of the data and their labels in the dataset, this work only considers the first category (Nodules >= 3 mm). The other two categories are left out.

The LIDC-IDRI dataset can be used for two different tasks. The first task will be the *localization* task, to detect if the nodules are cancerous or not, and also locate them within the image. The second task will be the *segmentation* task, where the contour of the cancerous nodule will be drawn. The preprocessing is done in a single step, where the images are converted from DICOM to `NumPy` arrays, and the labels are collected. For each slice, the label for the localization task will be a True/False diagnosis and the location of the center of the nodule, as the label for the segmentation task will be a segmentation mask. For the preprocessing of this dataset, only one script has been written. When running it, the user is first asked to enter the task he wants to perform, either the localization or the segmentation. The script then performs the given task.

Preprocessing for the localization task

The idea of the preprocessing for the localization is to first classify in a binary manner the nodules according their malignancy, given in a 1 to 5 scale by the radiologists, 1 being benign and 5 being malignant. The nodules that have a malignancy of 1 or 2 are treated as False cases, while the nodules with a malignancy of 4 or 5 are

Algorithm 2 LIDC-IDRI preprocessing for the *localization* task

```
Create two folders True and False in output_folder
for each series in dataset_folder do                ▷ Iterate over patients/studies/series
    if series is not CT then
        Continue
    Initialize nodules_df['NoduleID','xpos','ypos','zpos','Diagnosis']
    Get the 4 reading_sessions of the radiologists                  ▷ By parsing XML file
    for each session in reading_sessions do
        for each nodule in session do
            if malignancy in [1,2] then
                diagnosis = False
            else if malignancy in [4,5] then
                diagnosis = True
            else
                Continue
            Initialize x_coords = [] and y_coords = []
            for each slice in nodule do
                for each contour_point in slice do
                    [x, y] ← contour_point                          ▷ Get coordinates
                    x_coords append x
                    y_coords append y
                x_mean ← mean(x_coords)
                y_mean ← mean(y_coords)
                z ← slice
                nodules_df append [NoduleID, x_mean, y_mean, z, Diagnosis]
    for each row in nodules_df do
        for each dicom_image in patient do
            if dicom_image is the slice where the nodule of row appears then
                diagnosis ← row['Diagnosis']
                slice ← normalize(dicom_image)          ▷ normalize_dicom module
                Save slice in output_folder/{diagnosis}
```

treated as True cases. The nodules with malignancy 3 are left aside as the diagnosis is uncertain.

Then, for each of this cases, the corresponding slice is searched and taken. As the contour is given in a XML file for each of these slices, the x and y coordinates are extracted, and the center of the nodule is computed, via a mean calculation. The slices are finally converted into greyscale images encoded in 8 bits per pixel (pixel intensity from 0 to 255) using the conversion steps of Section 4.1. The script does this conversion so that the slice can be displayed correctly on a screen. The center of the nodule is saved in the name of the file, and the True and False cases are separated in two different output folders. For example, a True case can have the name `LIDC-IDRI-0001_nid-0_pos-314-365_29.npy`. The first part, `LIDC-IDRI-0001` is the patient ID, `nid-0` is the nodule ID as the case can have more than one nodule. Then, the position of the center of the nodule `pos-314-365` is given in (x, y) coordinates. Finally, the last number `29` is an added index to avoid duplicate names. The different steps to preprocess the dataset for the localization are shown in the Algorithm 2.

Algorithm 3 LIDC-IDRI preprocessing for the *segmentation* task

```
Create two folders img and masks in output_folder
for each series in dataset_folder do                ▷ Iterate over patients/studies/series
    if series is not CT then
        Continue
    Initialize nodules_df['NoduleID','contourdata','zpos']
    Get the 4 reading_sessions of the radiologists                  ▷ By parsing XML file
    for each session in reading_sessions do
        for each nodule in session do
            Select nodules of malignancy ≥ 4
            Initialize contour_points = []
            for each slice in nodule do
                for each contour_point in slice do
                    [x, y] ← contour_point                          ▷ Get coordinates
                    contour_points append [x, y]
                z ← slice
                nodules_df append [NoduleID, contour_points, z]
    for each row in nodules_df do
        for each image in patient do
            if image is the slice where the nodule of row appears then
                image_slice ← normalize(image)          ▷ normalize_dicom module
                Save image_slice in output_folder/img
                segmentation_mask ← create_segmentation_mask
                Save segmentation_mask in output_folder/masks
```

Preprocessing for the segmentation task

The preprocessing for the segmentation task is very similar to the preprocessing for the localization task, as it uses the same dataset. The script for the localization task took the nodules of the first category (Nodules >= 3 mm) with malignancy 1, 2, 4 or 5. For the segmentation however, the script only takes the cancerous nodules, i.e. with malignancy 4 or 5. The other ones are left out.

Then, the XML parsing is done in the very same way as the localization task, by getting all contour points, but this time the center of the nodules is not computed, as only the contour points themselves are required to do the segmentation task. These contour points are then saved in a list.

Finally, the images are converted into greyscale images encoded in 8 bits per pixel, and the segmentation masks, that are monochrome images, are created using the `segmentation_mask` module. Algorithm 3 shows the steps to preprocess the dataset for the segmentation task.

Algorithm 4 HNPC preprocessing

```
Create two folders img and masks in output_folder
Initialize metadata_df
Read excel file roinames_excel                          ▷ INFO_GTVcontours_HN.xlsx
for each patient in dataset_folder do
    for each visit in patient do
        for each series in visit do
            if series is CT, PET or RTStruct then
                metadata_df append metadata information
Split metadata_df in RT_df, images_df                   ▷ according to modality
for each row in RT_df do
    dicom_file ← row
    Open dicom_file                                     ▷ Get ROI_contour_sequence
    for each contour_sequence in ROI_contour_sequence do
        contour_image ← contour_sequence[0x3006, 0x16][0]          ▷ DICOM tag
        series_images_path ← contour_image[0x8, 0x1155].value      ▷ DICOM tag
        for each image in series_images_path do
            dicom_image ← Search the referenced slice
            image_slice ← normalize(dicom_image)        ▷ normalize_dicom module
            Save image_slice in output_folder/img
            segmentation_mask ← create_segmentation_mask
            Save segmentation_mask in output_folder/masks
```

4.4 HNPC preprocessing

The preprocessing of the Head-Neck-PET-CT dataset is separated in two phases. In the first phase, the preprocessing script iterates over all patients, studies and series. It then checks the modality of the series. If the series is a CT or PET series (image series) or a RTStruct series, the script continues. If the modality is something else, like RTDose or RTPlan, the series is just skipped as the script does not need it to preprocess for the segmentation task.

Then, the script reads the spreadsheet `INFO_GTVcontours_HN.xlsx`, and extracts the names the radiologists gave to the tumors. For both types of image series (CT and PET), the script saves the Patient ID, the series universal identifier (UID), the modality, the path and the roi name of the tumor in a `pandas` [39, 40] dataframe. In addition, for the RTStruct series only, the script also saves the series UID of its referenced image series. In that way, the dataframe contains names, UID and relations between the RTStruct series and their corresponding images. This will make it easier to work with in the next step.

In the second phase, the `pandas` dataframe just created is separated in two other dataframes: one containing only the images series, and the other containing the RTStruct series. An iteration is then started in the RTStruct dataframe. The goal of this iteration is to first take the RTStruct and its corresponding images series, and for each slice of the images series that contains the tumor, it then extracts the contour points that segment the tumor. Finally, the image is as before converted into a `NumPy` array, and the contour points are inputed into the `segmentation_mask` module, which outputs the segmentation mask. The Algorithm 4 shows all this preprocessing.

Algorithm 5 BUSI preprocessing

```
Create two folders img and masks in output_folder
for each category in dataset_folder do
    if category = normal then
        Continue
    image_paths_list ← Get all original images of category
    for image_path in image_paths_list do
        if image_name matches image_path then                ▷ With a RegEx
            Construct mask_name and mask_path from image_name
            image_slice ← convert(image_path)
            Save image_slice in output_folder/img
            segmentation_mask ← convert(mask_path)
            Save segmentation_mask in output_folder/masks
```

4.5 BUSI preprocessing

The preprocessing steps of BUSI are much shorter than in the two previous datasets. As the images are given in the PNG format, the preprocessing script does no use the `pydicom` library. While the segmentation masks have already been created and given in the dataset, the script does not use either the `segmentation_mask` module. No resizing or rescaling operation is needed, as images and masks are already in the same size. Thus, a simple conversion from PNG to `NumPy` array is required to preprocess this dataset. As the segmentation masks for the images of the first category (normal) are empty, this category is not taken in the preprocessing.

For both remaining categories (benign and malignant), the preprocessing script iterates over the original images. It can then find the corresponding segmentation masks, as they have the same name as the images. Once the image and its mask are found, they are converted into `NumPy` arrays, in format of 8 bits per pixel, 1-channel greyscale. They are finally saved under two distinct folders, one for the images and one for the masks. The preprocessing of the BUSI dataset is already finished. The Algorithm 5 shows the different steps to preprocess the BUSI dataset.

4.6 Discussion

This work improves the Hydra framework in the following ways:

- The new `segmentation_mask` module provides a robust way to create segmentation masks, given a medical image and the contour points of the tumor. The module is ready to be used for other future datasets for the segmentation task, and will give the segmentation masks for each image. Even if the contour points data is not complete or have some absurd values, the module will impute and mitigate the missing data. The `segmentation_mask` outputs two different files. On one hand, the segmentation masks that are monochrome images, will represent the labels during the training of Hydra, in a supervised learning manner. On the other, each mask will have a proper log file containing information about the imputation and mitigation of the data if necessary. The user can check these log files for a debugging purpose, and to see where data contained errors. Finally, the proper segmentation mask is created and returned along with the preprocessed medical image.

- The `segmentation_mask` module can also be used to segment other objects than just tumors. For example, if a CAD system is designed to segment the prostate itself as in the *PROMISE12* challenge (see Section 2.6.3), then this module is ready to output a mask that segments the prostate, and can be easily added to this CAD system.

- Hydra can now uses new datasets. Indeed, after all the preprocessing steps presented in this work that are implemented in three different preprocessing scripts, the three datasets are now ready to be used to train the Hydra model. Firstly, the *lung image database collection* can be used either for the localization task, or the segmentation task. A single script preprocesses this dataset for the two tasks. Then, the *head-neck-PET-CT* and the *breast ultrasound images* datasets can both be used for the segmentation task. At the end, each medical image has its own corresponding segmentation mask as ground truth, and they can be used together to train the segmentation task in a supervised learning manner.

- The `normalize_dicom` has received some minor changes. The code has been cleaned to be more readable by the user, but it basically achieves the same goal: convert the medical images in a standardized manner, with or without the use of the Hounsfield units, and in a way that Hydra can read.

Chapter 5

Conclusion

This work presented the new contributions on a computer aided diagnosis (CAD) system named Hydra, that can detect cancerous tumors. The Hydra framework was born under a collaboration between the H-FR and the eXascale Infolab of the University of Fribourg. This work contributes three scripts that preprocess three new datasets: they modify and transform the data so that it can be used with the Hydra model. The number of datasets for the Hydra framework has been doubled. With these new datasets, this work helps avoiding the problem of scarcity of publicly available medical data.

Another contribution of this work is also the implementation of a new task for the CAD system, named the segmentation. This was possible by developing a brand new module. It creates a segmentation mask given a medical image and a list of contour points that delimit the tumor. The new module has been used on three new datasets: two of them are preprocessed for the segmentation task while the last one is preprocessed for both localization and segmentation tasks.

5.1 Future Work

As the new module that creates the segmentation masks has been developed in this work and is ready to be used, the next goal is to utilize it in the Hydra framework. That means, a new Hydra model has to be constructed on purpose for segmentation task. A model that is used nowadays for segmentation is the U-Net architecture [41]. Hydra can then use this U-Net model instead of the current VGG-16 model.

Another way to improve Hydra's performance is to augment the data already preprocessed in this work: by flipping or rotating for example, the medical images are not the same to the eyes of the machine, and thus will extend the data available for the Hydra framework.

In the next years with improvement and advent of new machine learning techniques, the Hydra framework or other CAD systems will hopefully be used to really help radiologists to detect as early as possible the first stages of cancer, resulting in saving human lives.

Bibliography

[1] National Cancer Institute. *What Is Cancer? - National Cancer Institute*. Sept. 17, 2007. URL: https://www.cancer.gov/about-cancer/understanding/what-is-cancer.

[2] Global Cancer Observatory. *Cancer today*. 2020. URL: http://gco.iarc.fr/today/home.

[3] World Health Organization. *Cancer*. URL: https://www.who.int/news-room/fact-sheets/detail/cancer.

[4] *Brain MRI Images for Brain Tumor Detection*. URL: https://kaggle.com/navoneel/brain-mri-images-for-brain-tumor-detection.

[5] Julien Clement and Johan Jobin. "Prostate Cancer Classification: A Transfer Learning Approach to Integrate Information From Diverse Body Parts". In: (2020).

[6] Jiyoung Lee and Giuseppe Cuccu. "P-Hydra: Bridging Transfer Learning And Multitask Learning". In: (2020).

[7] Samuel G. Armato et al. "PROSTATEx Challenges for computerized classification of prostate lesions from multiparametric magnetic resonance images". In: *Journal of Medical Imaging* 5.4 (2018). Publisher: SPIE, p. 044501.

[8] Justin S. Kirby et al. "LUNGx Challenge for computerized lung nodule classification". In: *Journal of Medical Imaging* 3.4 (2016). Publisher: International Society for Optics and Photonics, p. 044506.

[9] Donald O. Hebb. *The Organization of Behavior: A Neuropsychological Theory*. New York: Psychology Press, May 1, 2002. 378 pp. ISBN: 978-1-4106-1240-3. DOI: 10.4324/9781410612403.

[10] Frank Rosenblatt. "The perceptron: A probabilistic model for information storage and organization in the brain". In: *Psychological Review* 65.6 (1958), pp. 386–408. ISSN: 1939-1471. DOI: 10.1037/h0042519.

[11] Donald Michie and Roger A. Chambers. "BOXES: An experiment in adaptive control". In: *Machine intelligence* 2 (1968), pp. 137–152.

[12] Yann LeCun et al. "A theoretical framework for back-propagation". In: *Proceedings of the 1988 connectionist models summer school*. Vol. 1. 1988, pp. 21–28.

[13] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[14] Yann LeCun et al. "Backpropagation Applied to Handwritten Zip Code Recognition". In: *Neural Computation* 1.4 (Dec. 1989). Conference Name: Neural Computation, pp. 541–551. ISSN: 0899-7667. DOI: 10.1162/neco.1989.1.4.541.

[15] Kunio Doi. "Computer-aided diagnosis in medical imaging: Historical review, current status and future potential". In: *Computerized Medical Imaging and Graphics*. Computer-aided Diagnosis (CAD) and Image-guided Decision Support 31.4 (June 1, 2007), pp. 198–211. ISSN: 0895-6111. DOI: 10.1016/j.compmedimag.2007.02.002.

[16] Geert Litjens et al. "Evaluation of prostate segmentation algorithms for MRI: The PROMISE12 challenge". In: *Medical Image Analysis* 18.2 (Feb. 1, 2014), pp. 359–373. ISSN: 1361-8415. DOI: 10.1016/j.media.2013.12.002.

[17] Wenxuan Wang et al. "TransBTS: Multimodal Brain Tumor Segmentation Using Transformer". In: *Medical Image Computing and Computer Assisted Intervention – MICCAI 2021*. Ed. by Marleen de Bruijne et al. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2021, pp. 109–119. ISBN: 978-3-030-87193-2. DOI: 10.1007/978-3-030-87193-2_11.

[18] Samuel G. Armato et al. *Data From LIDC-IDRI*. In collab. with TCIA Team. Type: dataset. 2015. DOI: 10.7937/K9/TCIA.2015.LO9QL9SX.

[19] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *arXiv:1409.1556 [cs]* (Apr. 10, 2015). arXiv: 1409.1556.

[20] Nitish Srivastava et al. "Dropout: a simple way to prevent neural networks from overfitting". In: *The journal of machine learning research* 15.1 (2014). Publisher: JMLR. org, pp. 1929–1958.

[21] Kenneth Clark et al. "The Cancer Imaging Archive (TCIA): Maintaining and Operating a Public Information Repository". In: *Journal of Digital Imaging* 26.6 (Dec. 2013), pp. 1045–1057. ISSN: 0897-1889, 1618-727X. DOI: 10.1007/s10278-013-9622-7.

[22] Yutong Xie et al. "Transferable Multi-model Ensemble for Benign-Malignant Lung Nodule Classification on Chest CT". In: *Medical Image Computing and Computer Assisted Intervention – MICCAI 2017*. Ed. by Maxime Descoteaux et al. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2017, pp. 656–664. ISBN: 978-3-319-66179-7. DOI: 10.1007/978-3-319-66179-7_75.

[23] Jia Ding et al. "Accurate Pulmonary Nodule Detection in Computed Tomography Images Using Deep Convolutional Neural Networks". In: *Medical Image Computing and Computer Assisted Intervention – MICCAI 2017*. Ed. by Maxime Descoteaux et al. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2017, pp. 559–567. ISBN: 978-3-319-66179-7. DOI: 10.1007/978-3-319-66179-7_64.

[24] Vincent Andrearczyk et al. "Automatic Segmentation of Head and Neck Tumors and Nodal Metastases in PET-CT scans". In: *Proceedings of the Third Conference on Medical Imaging with Deep Learning*. Medical Imaging with Deep Learning. ISSN: 2640-3498. PMLR, Sept. 21, 2020, pp. 33–43.

[25] Ke Wang, Shujun Liang, and Yu Zhang. "Residual Feedback Network for Breast Lesion Segmentation in Ultrasound Image". In: *Medical Image Computing and Computer Assisted Intervention – MICCAI 2021*. Ed. by Marleen de Bruijne et al. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2021, pp. 471–481. ISBN: 978-3-030-87193-2. DOI: 10.1007/978-3-030-87193-2_45.

[26] National Institute of Biomedical Imaging {and} Bioengineering. *Magnetic Resonance Imaging (MRI)*. URL: https://www.nibib.nih.gov/science-education/science-topics/magnetic-resonance-imaging-mri.

[27] National Institute of Biomedical Imaging {and} Bioengineering. *Computed Tomography (CT)*. URL: https://www.nibib.nih.gov/science-education/science-topics/computed-tomography-ct.

[28] Cleveland Clinic. *PET Scan: Tests, Types, Procedure*. Cleveland Clinic. URL: https://my.clevelandclinic.org/health/diagnostics/10123-pet-scan.

[29] National Institute of Biomedical Imaging {and} Bioengineering. *Ultrasound*. URL: https://www.nibib.nih.gov/science-education/science-topics/ultrasound.

[30] Martin Vallières et al. *Data from Head-Neck-PET-CT*. In collab. with TCIA Team. Type: dataset. 2017. DOI: 10.7937/K9/TCIA.2017.8OJE5Q00.

[31] Walid Al-Dhabyani et al. "Dataset of breast ultrasound images". In: *Data in Brief* 28 (Feb. 1, 2020), p. 104863. ISSN: 2352-3409. DOI: 10.1016/j.dib.2019.104863.

[32] Peter Mildenberger, Marco Eichelberg, and Eric Martin. "Introduction to the DICOM standard". In: *European Radiology* 12.4 (2002). Publisher: Springer-Verlag GmbH, pp. 920–927.

[33] Samuel G. Armato et al. "The Lung Image Database Consortium (LIDC) and Image Database Resource Initiative (IDRI): A Completed Reference Database of Lung Nodules on CT Scans: The LIDC/IDRI thoracic CT database of lung nodules". In: *Medical Physics* 38.2 (Jan. 24, 2011), pp. 915–931. ISSN: 00942405. DOI: 10.1118/1.3528204.

[34] Martin Vallières et al. "Radiomics strategies for risk assessment of tumour failure in head-and-neck cancer". In: *Scientific Reports* 7.1 (Aug. 31, 2017), p. 10117. ISSN: 2045-2322. DOI: 10.1038/s41598-017-10371-5.

[35] Uwe Schneider, Eros Pedroni, and Antony Lomax. "The calibration of CT Hounsfield units for radiotherapy treatment planning". In: *Physics in Medicine and Biology* 41.1 (1996). Publisher: Institute of Physics and IOP Publishing Limited, pp. 111–124.

[36] Tom Kimpe and Tom Tuytschaever. "Increasing the Number of Gray Shades in Medical Display Systems—How Much is Enough?" In: *Journal of Digital Imaging* 20.4 (Dec. 2007), pp. 422–432. ISSN: 0897-1889. DOI: 10.1007/s10278-006-1052-3.

[37] Charles R. Harris et al. "Array programming with NumPy". In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. ISSN: 1476-4687. DOI: 10.1038/s41586-020-2649-2. URL: https://www.nature.com/articles/s41586-020-2649-2.

[38] Darcy Mason. "SU-E-T-33: pydicom: an open source DICOM library". In: *Medical Physics* 38.6 (2011). Publisher: Wiley Online Library, pp. 3493–3493. DOI: 10.1118/1.3611983.

[39] The pandas development team. *pandas-dev/pandas: Pandas*. Version 1.3.5. Mar. 18, 2020. DOI: 10.5281/zenodo.3715232. URL: https://zenodo.org/record/3715232.

[40] Wes McKinney. "Data structures for statistical computing in python". In: *Proceedings of the 9th Python in Science Conference*. Vol. 445. Austin, TX, 2010, pp. 51–56. DOI: 10.25080/Majora-92bf1922-00a.

[41] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*. Ed. by Nassir Navab et al. Vol. 9351. Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing, 2015, pp. 234–241. ISBN: 978-3-319-24573-7 978-3-319-24574-4. DOI: 10.1007/978-3-319-24574-4_28.