UCLouvain

epl

École polytechnique de Louvain (EPL)

Polytechnic University of Catalonia (UPC)

Unsupervised domain adaptation for bladder segmentation by U-net in Cone Beam CT

Author : **Anna CUXART**
Supervisor : **Benoît MACQ**
Readers : **Benoît MACQ, Eliott BRION, John LEE**
Academic year 2019–2020
Master in Advanced Telecommunications Technologies (UPC)

POLYTECHNIC UNIVERSITY OF CATALONIA (UPC)
UNIVERSITÉ CATHOLIQUE DE LOUVAIN (UCL)

Abstract

by Anna Cuxart Garcia

Introduction Radiotherapy is a medical treatment used to control or kill cancerous cells in cancer patients. At the beginning of the treatment planning process, the patient takes a CT scan in order to plan his radiation dose and, sometimes, he can take a Cone Beam CT scan some days after, right before receiving his radiation, to adjust his couch position for the delivery. The main difference between CT and CBCT scans is that the first one has a higher quality and contrast, and the second one is taken directly at the isocenter. As the treatment planning takes several days, when the patient receives his radiation his organs might not be in the same position as they were at the beginning, so the healthy tissues around the tumor area can receive more radiation than what it was planned and get damaged. Our aim is to implement an automatic segmentation of the bladder in CBCT 3D images using deep learning, in order to get a clearer idea of the position of those organs.
Materials and Methods In order to implement the segmentation we performed unsupervised domain adaptation between CT (the source) and CBCT 3D images (the target), as we didn't have a large labeled CBCT dataset but we did for CT, as their segmentation is already part of the treatment planning process. We have used a subset dataset of 120 patients: 60 CT and 60 CBCT of the male pelvic region. We have implemented a deep learning network using Unet as a segmenter and a regular CNN as a domain discriminator (an adversarial network), which also includes a gradient reversal layer. We have used the Dice score coefficient and the Hausdorff distance in order to evaluate the performance of our network and compare it with some previous works developed in the same field.
Results We have performed three main experiments, for which we have obtained the following DSC and Hausdorff distance (in voxels): (i) lower boundary: 0.383 ± 0.260 and 36.47, (ii) upper boundary: 0.717 ± 0.177 and 27.44, (iii) unsupervised domain adaptation: 0.623 ± 0.149 and 39.18. With this implementation we have closed the gap between training the network only on CT and only on CBCT by a 72%.
Conclusions Cone Beam CT image segmentation using unsupervised domain adaptation proves to be an improving methodology in radiotherapy and presents different applications in other fields.

Acknowledgements

First of all, I would like to thank my project supervisor Benoît Macq from the ICTEAM Research Group at Université Catholique de Louvain for his trust and help, and the opportunity to work with him. Special thanks to his PhD Eliott Brion, for guiding me through all the project, providing all the information needed and always being accessible for help. Thanks also to Jean Léger for helping me with their 2D neural network model, and to Sébastien Lugan for his dedication to solve the technical issues in order to properly work in remote.

I want to thank CHU-UCL-Namur (Dr. Vincent Remouchamps) and CHU-Charleroi (Dr. Nicolas Meert) for providing the data for this project, as well as Sara Teruel Rivas for her technical support in the data acquisition and annotation and to Dr. Genevieve Van Ooteghem for the verification of the contours.

I would like to thank also Prof. Ferran Marqués from UPC for highly recommending me to come at UCL to develop my thesis with ICTEAM. Thanks also to Prof. Enric Monte for agreeing to be my supervisor and tutor at UPC.

Lastly, thanks to my family and friends for encouraging me to stick to my goals and never give up, despite unforeseen situations.

Contents

Abbreviations

UDA	**U**nsupervised **D**omain **A**daptation
ANN	**A**rtificial **N**eural **N**etwork
AI	**A**rtificial **I**ntelligence
CNN	**C**onvolutional **N**eural **N**etwork
NN	**N**eural **N**etwork
CT	**C**omputed **T**omography
CBCT	**C**one **B**eam **C**omputed **T**omography
TP	**T**rue **P**ositive
FP	**F**alse **P**ositive
FN	**F**alse **N**egative
TN	**T**rue **N**egative
SGD	**S**tochastic **G**radient **D**escent
GRL	**G**radient **R**eversal **L**ayer
GT	**G**round **T**ruth
DSC	**D**ice **S**core **C**oefficient
SMBD	**S**ymmetric **M**ean **B**oundary **D**istance

Chapter 1

Introduction

1.1 Radiotherapy

Radiation Therapy, also called Radiotherapy, is a medical treatment used to control or kill cancerous cells. It uses high doses of intense radiation in order to kill cancer cells and reduce the patient's tumor.

The main goal of Radiotherapy is to damage the DNA of the cancer cells, in order to slow their growth and avoid further dividing. The given radiation reacts with the water of the cells, damaging their genetic material that controls cell growth. Mostly, all regular cells can recover their functional behaviour after the radiation treatment, but that's not the case for cancer cells. Unfortunately, it is quite a slow process, as it takes several days and weeks of constant treatment before the DNA is damaged enough for the cancer cell to die, and the healthy cells need to recover from the radiation between the sessions.

There are two kinds of radiation therapy, and they can be given to the patient depending on different aspects: the specific kind of cancer, where is it placed, how deep should the radiation go into the body and the health of the patient amongst others. There are the external beam and internal beam, and their main difference is the source of the radiation given to the patient. The first one uses an external machine outside the body, while the second one uses an implant that is placed inside the patient's body trough surgery to provide a more focused and controlled radiation.

External beam is the most common one, and it can be used to treat different kinds of cancers, as well as to relieve pain that this illness can cause when it spreads to other parts of the body [1] [2][3][4].

1.1.1 Treatment planning

As there are different kinds of radiation and different methodologies, an important part of radiotherapy is the treatment planning. As a previous step of the radiation delivery, a group of medical experts (the radiation oncologist and the radiation therapists) analyze the specific case of the target patient and plan his whole treatment: where is his affected area placed in his body, which kind of radiation should he receive, the daily radiation dose, in which position should he receive it, etc.

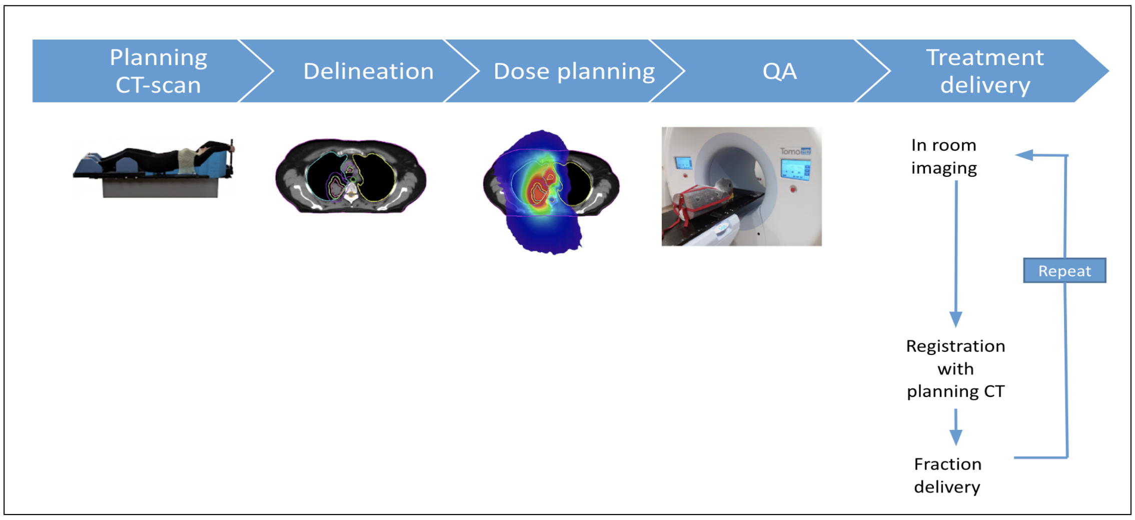


FIGURE 1.1: In radiotherapy, during the treatment planning, the patient takes first a CT scan in order to plan his radiation dose. On the last step, right before he receives his radiation, he can take a second scan, the Cone Beam CT, in order to adjust his position in the machine for the radiation delivery. - *Source: Xavier Geets, Assessment of the quality of a treatment plan (unpublished).*

In the case of external radiation, the physicist and the physician will firstly execute a simulation method in order to examine where and how the patient's treatment should be applied.

In the very first step of the simulation the patient will lay down, as still as possible, in a table of examination. In order to know where the radiation should be applied, the therapists scan the patient using a special X-Ray machine, performing the Computed Tomography (CT) scan (Planning CT-scan in 1.1).

The CT scan uses a combination of different computer-processed X-Ray images taken from different angles of the patient. The x-rays utilize radiation from a radioactive contrast injected into the body of the patient to create cross-sectional (tomographic) images.

In the particular case of the CT, the emitter of x-rays rotates around the immobile patient and the detector, placed in diametrically opposite side, picks up the image of a body section (beam and detector move synchronously).

This simulation and scanning process may take several hours, as the medical experts want to assure that they get a clear image of the affected area, so they can plan how to provide the radiation while damaging the least amount of healthy tissue.

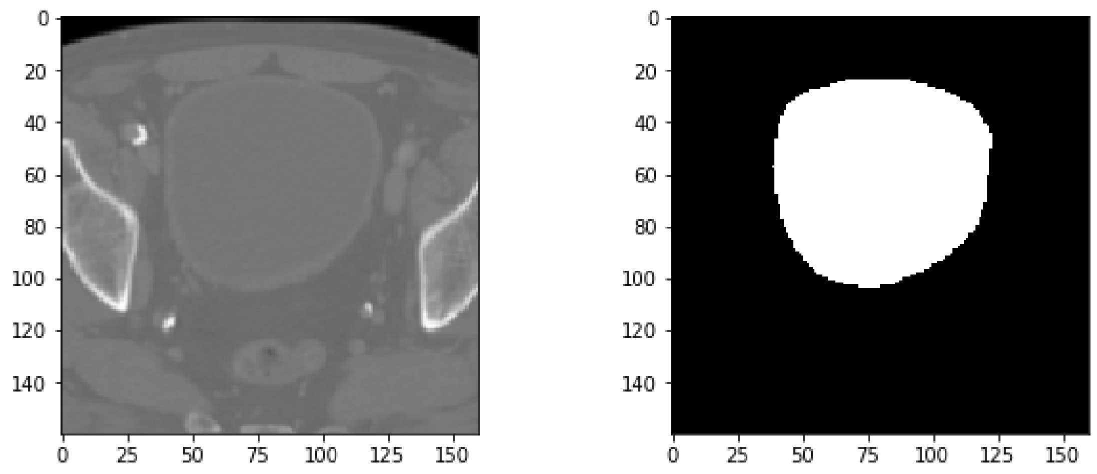


FIGURE 1.2: Slice of a 3D CT scan image of the male pelvic area (left) with his bladder mask segmented by hand (right). Pixel spacing (mm) = [1.2, 1.2, 1.5].

Once the medical experts have all the CT scan images from the patient, they will carefully delineate manually the affected regions by the cancer, contouring precisely the organs and the tumors (Delineation in 1.1). They will use them to plan the radiation dose: adjusting where should it be projected and choosing the adequate intensity, so the main tumor can receive the main radiation but the healthy tissues around are minimum affected (Dose planning in 1.1).

Formerly, they will check the performance of the planning by doing a Q&A test using the radiation machine on a bag of water, simulating the human body (QA in 1.1).

Thereupon, some days after, the patient will be ready to start his first radiation session. Unfortunately, the treatment can not be given with the same machine that did the

CT scan, as it is way bigger and complex, and there are many engineering constraints. Therefore, this time, the patient will be placed in a second treatment position (Treatment delivery in 1.1).

Firstly, he will sit down in a couch of examination. Then, in order to adjust his position to the adequate one to receive the planned radiation, one possible option for the patient is to be scanned using the Cone Beam Computed Tomography (CBCT). CBCT it's a variant type of CT scan that uses a cone-shaped x-ray beam and two dimensional detectors instead of fan-shaped x-ray beam and one dimensional detectors.

The main advantages of CBCT compared to CT is the reduction of time and radiation during the scanning process with the patient. So this second time, the scanning will be faster and the patient will receive a smaller amount of radiation, which is important regarding his delicate health. A second advantage is that, for CBCT scans, the image can be taken at the isocentre, the station where the patient will receive his radiation treatment.

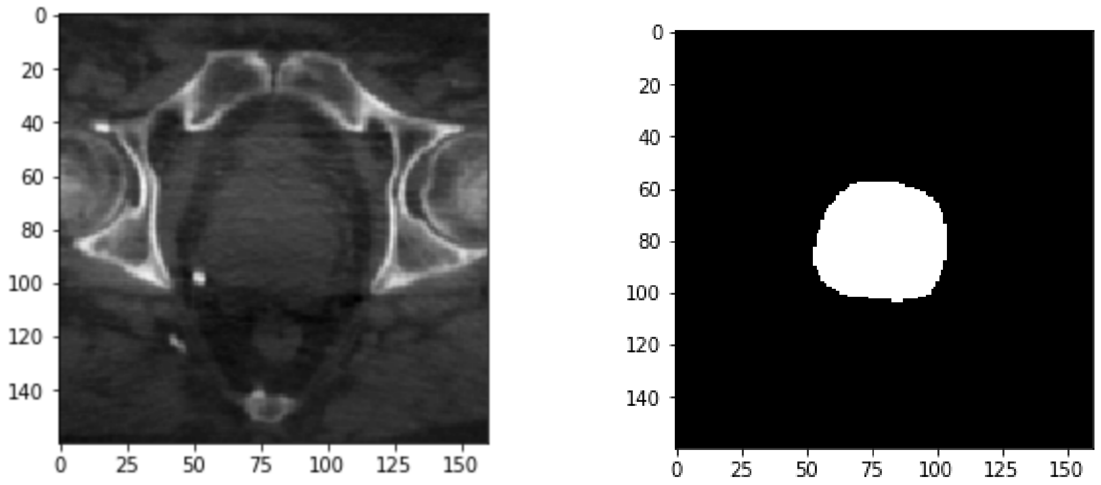


FIGURE 1.3: Slice of a 3D Cone Beam CT scan image of the male pelvic area (left) with his bladder mask segmented by hand (right). Pixel spacing (mm) = [1.2, 1.2, 1.5].

Once the CBCT scan is finished, the whole couch structure in which the patient is seated will move, adapting him to the position that was planned to provide the radiation. In order to do so, the CBCT scans are compared and aligned with the previous CT images.

The main constraint of this process is that there is a large difference between CT and CBCT images quality. CBCT images are taken faster and with less radiation, so their quality is lower. Therefore, once both images are aligned using the bones position, there can be some changes regarding the position of the organs that go unnoticed.

For example the bladder could be full, and also some organs can accumulate air inside for some days, as the case of the rectum, modifying the position and the shape of the other neighbor organs. So even when both images are properly aligned, some organs can be in a different position than when the treatment was planned using the previous CT images. Therefore, even though the dose radiation has been prepared to focus on the tumor cancer cells, as now some organs have moved they can receive a higher amount of radiation and be damaged as well.

And here is where we want to take part with our project: we believe that, by automatically segmenting CBCT images right after they are taken we can help to readjust the radiation planning for the patient just before he receives his treatment. Then it would be adapted to the new distribution of the organs, and that would improve the administration of the dose. Actually, this would have two interesting applications:

(a) *Online re-planning*: by automatically segmenting CBCT images right when they are taken, there could be an online re-planning of the previously designed radiation distribution in order to adjust it to the actual position of the organs.

(b) *Patient selection for off-line re-planning*: using the automatic segmentation and comparing it to the previos scan, there could be a selection in order to distinguish those patients which the displacement of the organs is too severe for an online re-planning. So they could restart the dose planning process for them.

In order to implement this automatic segmentation, we aim to use deep learning methodologies, and in the following sections we will explain why.

1.2 State of the Art and previous work

In the previous section we saw that there's promise to deliver a more precise radiation treatment by segmenting CBCT images. Lets take a look then at the actual state of the art in the field of medical image segmentation.

Nowadays we can find the so called ***Non-rigid registration*** method, that is used to align medical images based on localized deformations, and they are commonly used to align medical images between patients, as they cope well with anatomical, physiological and pathological variabilities [5]. Their main constraints though are the followings:

- They are too sensitive to large deformations, so their performance can be weaken by some affected organs.
- Their segmentation process is quite slow, as they solve their optimization problem with iterations.
- They are patient dependent, as the alignments are based and done only between medical images of the same patient.

On the other hand, nowadays we can find many applications of artificial intelligence regarding image processing. More specifically, we can find multiple use cases of ***deep learning*** being used for medical image segmentation, and the main reasons are the following:

- Deep learning algorithms are robust, as they are trained with hundreds of examples from different patients to learn how to make specific decisions and segment the desired areas of an image.
- They are fast, as their back end mathematics are based of matrix multiplications, and they are pretty efficient on the GPUs.
- They are autonomous meaning that, during the training process there are a lot of parameters to tune, but once it is finished, for the following patient's registration there will be no need to adapt any parameters manually.

That's why we think that approaching this automatic segmentation problem using deep learning can be an interesting field of research.

However, we are not starting from scratch. There has been multiple previous research studies based on automatic segmentation for CBCT images using deep learning, as well as using unsupervised domain adaptation methods for medical image segmentation.

This project is build upon three previous projects that have been developed in the same field.

- *Cross-domain data augmentation for deep-learning-based male pelvic organ segmentation in cone beam CT* by Jean Léger, Eliott Brion, Paul Desbordes, Christophe De Vleeschouwer and John Aldo Lee [6].

  This first paper describes the work done by the ICTEAM doctorands in UCLouvain, by applying deep learning to implement data augmentation in order to extend their training set to automatically segment male pelvic organs such as the bladder, the rectum and prostate on CBCT radiotheraphy scans.

- *Unsupervised domain adaptation by backpropagation* by Yaroslav Ganin and Victor Lempitsky [7].

  One subcategory of data augmentation is the so called unsupervised domain adaptation, where you have labeled data of a similar nature but from a different domain. This second paper proposes an architecture to implement it by using a regular feed-forward neural network and upgrade it with a new gradient reversal layer.

- *Unsupervised domain adaptation in brain lesion segmentation with adversarial networks*, by Konstantinos Kamnitsas [8].

  In the same line as the previous one, this last paper proposes another architecture to implement unsupervised domain adaptation. In this case they introduce the idea of using adversarial neural networks for medical image segmentation, as they are more invariant to the differences in the input data and they don't require a labeled test domain set.

1.3 Introduction to machine learning

As technology advances we have been hearing about machine learning, but what is it exactly? What can it be used for? How does it work?

As Tom Mitchell describes in his book *Machine learning* [9]: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

If we apply this definition to our project, we could state the following: our experience E is our dataset (Chapter 3.1), composed by CT and cone beam CT images, the task T that we want to perform is to automatically segment the bladder in cone beam CT images and we are going to evaluate the performance P with two measures described in 3.5.

Nowadays there are many different models and methods in order to train a computer with machine learning, depending on the task that you want him to learn, the amount of examples that you can provide, the capacity and memory that the computer has, amongst many others [10] [11] [12].

As machine learning algorithms learn from examples and are able to make future predictions, they have plenty of applications in our day to day life [13], such as:

- Social Networks: from targeting the best ads for each user to recognise your friend's face when you upload a new photo on Facebook.
- Finance: Predicting monetary frauds online, for example by tracking your credit card.
- Healthcare: Helping doctors to predict their patients diagnostics and speeding their processes.

It is not unusual to get overwhelmed by all artificial intelligence concepts and confuse them, so we also wanted to emphasize on distinguishing machine learning and deep learning.

The two main differences between them are the following: first, for deep learning, at the beginning of the training process we set the values of the weights (the filters of the convolutions) randomly and during the training the network will variate and update them automatically. Secondly, the word *deep* comes from the fact that we stack several (hidden) layers between the input layer and the output, therefore the network is able to learn more complex structures in the data [14].

1.4 Unsupervised Domain Adaptation

Usually, when we work with machine learning challenges, we assume that the training and the test data have the same probability distribution. In the main scenario we have the source domain

$$D_s = \{X_s, P(X_s)\}$$

$$P(X_s) = \sum_j p(x_{si}, y_j)$$

where Xs is the feature space of the source data and P(Xs) is the marginal distribution of those features, computing its joint probability distribution with the ground truth labels of the source data, y. As a goal, a predictive function

$$fs(x) = Ps(y/x)$$

is learnt by training and configuring the hyper-parameters on the data (Xs, Ys). This function will aim to approximate the optimal function f's(x) that generated Ys from Xs. On the other hand, we may want to apply this method to a different dataset, but it is important to pay attention at its domain. If the new dataset's domain is different from the previous one, we will call it target domain

$$Dt = \{Xt, P(Xt)\}$$

where Xt is the feature space of the target data and P(Xt) is the marginal distribution of those features.

This scenario is typically observed in biomedical applications, due to variations in the image acquisition, where training and test data may differ in their resolutions, contrast, noise level (so their marginal distributions differ P(Xs) $\neq$ P(Xt)) or on the type of sequences (data differences Xs $\neq$ Xt).

That is why it is interesting to develop methods that can learn from accessible datasets and can be adapted to new domains without additional training data.

Domain adaptation (DA) is a subclass of transfer learning (TL) that assumes that both datasets have the same ground truth (Ys = Yt), and the only difference between them is the domain. Its goal is to train a learning function $f_a(\Delta)$ that can perform successfully on both domains.

Unsupervised domain adaptation (UDA) [8] is a subclass of DA that assumes the availability of a labeled source domain database S = {Xs, Ys} and an unlabeled target domain database T = (Xt). In this environment, the main goals are first learn a representation $h_a(x)$ that maps Xs and Xt to a feature space invariant to the differences between the two domains; and secondly, learn a function $f_{ah}(x)$ such that

$$f_a(x) = f_{ah}(h_a(x))$$

approximates $f'_s(\Delta)$ and it is closer to $f'_T(\Delta)$ than any function $f_s(\Delta)$ learned only using the source data (Xs, Ys).

In this project, we are going to work with a source domain database composed by labeled CT images and a target domain database composed by unlabeled CBCT images.

1.5 Objective

Our main goal of this project is to accomplish the automatic segmentation of CBCT radiotherapy images using deep learning. Why? Because they are used in some processes to treat cancer, in order to analyze where the radiation has to be applied. And we think that, by segmenting the images, the treatment applied to the patient could be executed more precisely, so the healthy tissues and organs around the tumor area would be less affected.

How are we going to do it?

We are going to build a system based on neural networks architecture, and we will feed it with different examples of already segmented CBCT images, providing him the original image and the mask or segmentation. From this data, the algorithm will learn the different features of the images and their regions, so afterwards he will be able to make decisions by its own and predict or propose segmentations for new images. The system will autonomously learn how to do the segmentations, specifically bladder segmentations.

The main constraint is that CBCT images have a low contrast and quality, therefore they are not easy to segment. And the second constraint is that we do not have a dataset of already segmented CBCT images to train a deep learning algorithm, and in order to obtain it we would steal a lot of time of doctors. What we sure have is a dataset of segmented CT images from previous treatments, which are similar to our target but from another domain.

Hence, we propose to implement the automatic CBCT image segmentation NN by using unsupervised domain adaptation. UDA is a deep learning technique that will allow us to work with a CT dataset, with already segmented examples, and a CBCT dataset, without segmented examples in order to find a joint representation of the two datasets that doesn't depend on the domain. This way, we will be able to train our neural network with both datasets, make it learn the features of both images that are domain independent and use them to teach the network to segment the CBCT images based on the already segmented CT examples.

In the following sections, we will proceed to explain in more detail what is exactly machine learning and an accurate definition of unsupervised domain adaptation.

1.6 Contribution of this master thesis

Our main proposal is to implement unsupervised domain adaptation for cone beam CT images (CBCT) using deep learning and, in comparison with the previous works, the main novelty that this master thesis presents is that it combines the following three features at the same time: (i) we are working with a 3D dataset of CBCT images, (ii) we are implementing an architecture that involves adversarial networks with the gradient reversal layer and (iii) we are using Unet as the segmenter of our network.

(i) We are working with a 3D dataset Compared to the previous works such as [6] and [8], in this project we are using a 3D dataset of CT and cone beam CT images. With this, we aim to work closer to the radiotherapy treatment cases described in this chapter, shortening the gap between the experimental work and the clinic applications.

(ii) We are implementing an architecture that involves adversarial networks with the gradient reversal layer As described in [8], we are going to perform unsupervised domain adaptation by implementing an adversarial neural network. But, in order to implement the backpropagation, we are going to join a gradient reversal layer to it, as defined in [7].

(iii) We are using Unet as the segmenter of our network In contradistinction to [8], for this project we are going to use the CNN Unet as the segmenter network of our architecture. Unlike the completely fast forward network that they present, which only plays with downsampling pass, we are going to work with this much more complex network that includes skipping connections and relies on data augmentation.

Moreover, while our use case is in radiotherapy, our work proposal could be used for any application in image segmentation beyond medical environments, such as face recognition, self-driving vehicles and satellite imagery amongst many others.

1.7 Structure of this master thesis

The organization of this master thesis is the following: first, in Chapter 2, we present some background information, by defining what is machine learning and presenting the previous works in which this project is based. In Chapter 3, we introduce the dataset that we have used and we explain in more details the methods that we have employed for this work, such as the environment, the hyperparameters of our network, the architecture that we have designed and the metrics that we have used to evaluate our results. In Chapter 4, we define the experiments that we have performed and share the obtained results, which we discuss in Chapter 5. And finally, in Chapter 6, we terminate with our main conclusions and possible developments for a future work.

1.8 Summary of Chapter 1

Radiation therapy is a medical treatment used in cancer patients that gives a high dose of intense radiation to kill or control the cancer cells and reduce the patient's tumor. Sometimes, during the treatment process, two different kinds of scans are taken, first the computed tomography (CT) and then the cone beam computed tomography (CBCT (section 1.1.1)). As they are taken in different days, the organs of the patient around the tumor area can be displaced, and therefore receive a higher amount of radiation than what it was firstly planned. Previous works (section 2.2) show that automatic segmentation of CBCT images can improve this situation, by curately showing the new position of the organs. Therefore, in this project we aim to implement CBCT image segmentation of the bladder by using unsupervised domain adaptation with adversarial neural networks and Unet.

Chapter 2

Background

In this section we present some detailed definitions of what is machine learning and how it works, followed by an accurate description of the previous works that we have been following.

2.1 Machine learning

Our segmentation methodology will be based on convolution neural networks (CNN), as in the previous projects and in line with the last trends in the scientific community [8]. In this section, we are going to show concepts related to Neural Networks and some new techniques that could aid in the improvement of results.

The algorithmic structures of artificial neural networks (ANN) allow models that are composed of multiple layers of processing to learn data representations with multiple levels of abstraction. This set of layers are composed by neurons and perform a series of linear and non-linear transformations to the input data in order to generate an output close to the expected (the label).

The main scenario, supervised learning, consists in obtaining the parameters of these transformations: the w_i (weights) and the b (biases); with the aim that the output of the neuron (y) differs as little as possible to the expected output. For each artificial neuron we have *"m"* input signals (x), as usually the x_0 input is assigned the value +1.

Neural networks use the error, or loss, of their outputs in a process called *Backpropagation*, whereby these weights and biases are updated.

$$y = \sum_{i=0}^{m} W_i x_i + b \tag{2.1}$$

A group of neurons form a layer, and multiple layers form a network, as for example the multilayer perceptron 2.1. This basic NN is composed by a minimum of three layers: one *Input Layer* (the first, where we will feed our data), one *Output Layer* (the last) and at least one *Hidden layer*. All the neurons of each layer are connected with a certain weight to every node in the following layer, and the amount of *Hidden layers* will depend on the complexity of the task that we want it to learn, more difficulty more layers.

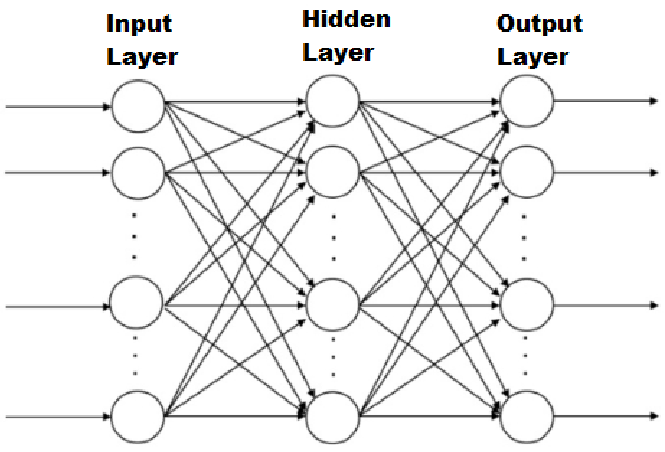


FIGURE 2.1: A Neural Network is composed by, at least, three layers: the input, the output and the hidden layer between them. The multilayer perceptron in the smallest version with three layers [15].

2.1.1 Neural network hyper-parameters

Neural networks have a lot of parameters that need to be adjusted in order to obtain better results. In the following sections we will cover the most important ones.

2.1.1.1 Epochs

The number of epochs [16] means the number of times that we will pass all the training data through the neural network during the training process. The number of epochs is usually increased until the accuracy of the validation data starts to decrease, even when the accuracy of the training data continues to increase (as this is when we could detect a potential overfitting).

2.1.1.2 Loss

One of the most important hyper-parameters is the loss function. We will use it to estimate the error by comparing and measuring the performance of our prediction with the expected result. The main goal is to keep is as small as possible, as we don't want divergences between the estimated and the expected values. Therefore, during the process of training the model, the weights of the interconnections of the neurons will be adjusted for a higher accuracy, by reducing the loss towards its minimum.

The choice of the best function of loss resides in understanding what type of error is or is not acceptable for the problem in question.

2.1.1.3 Activation Function

Each neuron of the neural network has an activation function that defines the output of the neuron. It is used to introduce non-linearity in the modeling capabilities of the network. We can find several options for activation functions, but the most common are: *Linear*, *Sigmoid*, *Tanh*, *ReLU* and *Softmax* [15].

The choice of a specific activation function depends on several factors: (a) the kind of data that are being processed, (b) the kind of prediction that we seek to obtain, (c) which part of the network we are in, and (d) in which kind of layer we are going to apply it.

2.1.1.4 Optimizer

During the training process of the neural network, the main goal is to automatically adjust the network parameters (weights and biases) in order to minimize the loss function. We want to optimize the network to obtain the best results.

We can use different optimizers, and the most common ones are the following: *Stochastic Gradient Descent*, *RMSprop*, *Adagrad*, *Adadelta*, *Adam*, *Adamax* and *Nadam* [15].

2.1.1.5 Learning Rate

In order to adjust and update the parameters of the network (weights and biases), the learning rate controls how much these components change per epoch, being α in the gradient descent equation [17]:

$$W_{ij} \leftarrow W_{ij} - \alpha \frac{dError}{dW_{ij}} \tag{2.2}$$

The learning rate decay (α) is used to decrease the learning rate as epochs go by to allow the learning to advance faster at the beginning than towards the end ("fine-tuning"). As progress is made, smaller and smaller adjustments are made to facilitate the convergence of the training process to the minimum of the loss function.

2.1.2 Convolutional Neural Networks

As their name self describes, artificial neural networks are an artificial simulation of the animal's biological neural network, with the main goal of being trained, learned and be capable to make decisions by mimicking the human brain distribution. We can program different computers as if they were a network of millions of brain cells, the so called artificial neurons, and train them to learn how to make automatic decisions [18] [19].

There are many different classes or types of ANN, depending on the task we want the computers to learn, but the most common ones are the following:

- MLP: Multilayer perceptron are the vanilla neural networks, composed by at least 3 layers of artificial neurons (input layer, minimum 1 hidden layer, and one output layer). Their main characteristic is that each neuron of the network is fully connected to all the neurons of the previous and following layers.
- RNN: Recurrent neural networks are basically designed to work with sequence prediction problems. Their main characteristic is that they have a recurrent connection on their hidden states, the artificial neurons of the hidden layers, in order to ensure that the sequential information is captured in the input data.
- CNN: Convolutional Neural Networks.

Convolutional Neuronal Networks [20] [21] appeared at the late 90s, and their name comes from their use of Convolutional Layers A.1, which look for spatial relationships inside the input data, so they can learn to recognize patterns across the space. Some of their applications can be system recommendation or natural language processing, but nowadays they are mostly used for image and video classification and recognition, and they have had a huge impact in the area of computer vision.

The main characteristic of CNNs is that they encode certain properties of the input data into the architecture, so they can learn to recognize specific components of an image, from lines and curves to more complex figures like faces, objects or animals.

In this project, we will use CNNs to extract features from both CT and CBCT images in order to automatically segment the affected areas and organs of the patients.

However this kind of features are quite complex, requiring that we first extract the simpler ones such as lines and curves in the first layers, and as we get to the deeper ones we will get to more complex features like circles and textures.

as well as the architecture we have designed for this project and the datasets that we have used.

2.2 Previous research

In the previous section we have seen more detailed definitions of machine learning and how neural networks work. Formerly, we are going to present some previous projects that have been done in the same research area.

As mentioned in 1.2, this project is based in three previous works that have been developed in this field. Two of them [8] and [7] propose to implement an unsupervised domain adaptation with two interesting features:

2.2.1 Unsupervised domain adaptation with adversarial networks

The first paper [8] describes an architecture to implement unsupervised domain adaptation to segment brain lesion images by using adversarial networks.

The authors propose to use two different networks to achieve the UDA segmentation: the segmenter and the domain discriminator (the adversarial network). The first one is trained using only the source dataset, and the second one is trained with both source and target datasets, in order to learn those features that are invariant to their domains. With them, after the training, the segmenter is finally able to segment the target images without any example of their original segmentations.

2.2.2 Unsupervised domain adaptation with backpropagation

The second paper [7] describes an architecture composed by three connected networks: the feature extractor, the label predictor and the domain classifier. And their interesting extra feature is that between the domain discriminator and the feature extractor they place a gradient reversal layer, which allows the network to be trained using backpropagation.

Lets dig a little bit deeper into their proposal to understand how it works and how we adapted it to our project. In the following figure we can have a closer look at the architecture of the model:

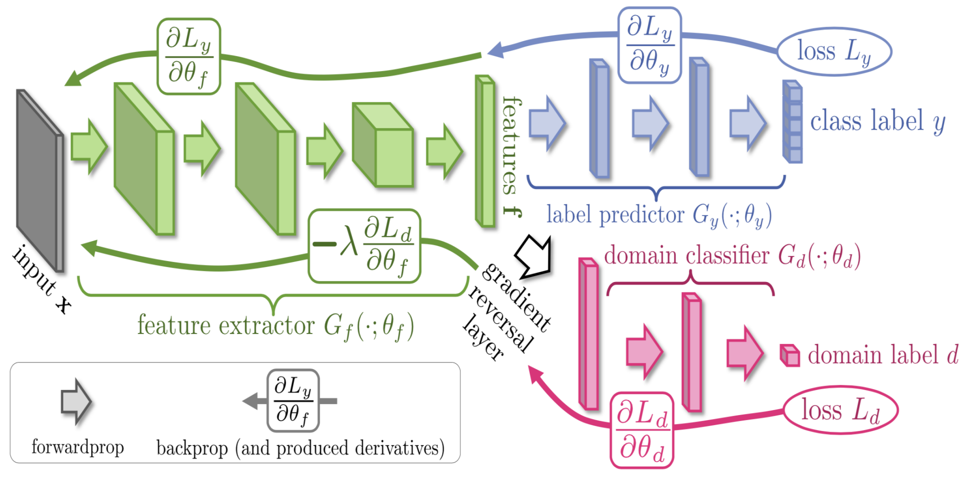

$\frac{\partial L_y}{\partial \theta_f}$
$\frac{\partial L_y}{\partial \theta_y}$
loss L_y
class label y
features $\mathbf{f}$
label predictor $G_y(\cdot;\theta_y)$
input $\mathbf{x}$
$-\lambda \frac{\partial L_d}{\partial \theta_f}$
gradient reversal layer
domain classifier $G_d(\cdot;\theta_d)$
feature extractor $G_f(\cdot;\theta_f)$
domain label d
$\frac{\partial L_d}{\partial \theta_d}$
loss L_d
forwardprop
backprop (and produced derivatives)

FIGURE 2.2: The unsupervised domain adaptation by backpropagation in [7] presents the following neural network architecture: a feed-forward network composed by a feature extractor and a label predictor, which allows the segmentation task, and a domain classifier that allows the unsupervised domain adaptation task. This last network is connected to the feature extractor by a gradient reversal layer, in order to compute the backpropagation by multiplying the gradient by a negative constant. This bans the domain classifier to distinguish the domain of the input features, finding therefore the domain-invariant features.

This structure is composed by three different parts: the *feature extractor*, the *label predictor* and the *domain classifier*. The two first ones combined result as the standard feed-forward NN architecture and the third, which is connected to the *feature extractor* with a *gradient reversal layer*, contributes with the UDA function of the network.

During the training time, they will feed the network with both samples from our source domain (labeled CT images) and target domain (unlabeled CBCT images), and with the domain classifier they will distinguish their domain. Also, the labels of the source samples will be known during this period, but the ones from the target will rest unknown, as we don't have them. So our main goal is to predict the target labels for the CBCT images during test time.

But how does this neural network work? Let's have a closer look at it. For each image at the input, the network will predict its label (the segmentation of the bladder) and its domain (source or target). And in order to do that it goes trough the following process:

First, the input image x goes trough the *feature extractor* part and gets mapped by G_f to a feature vector f.

$$f = G_f(x; \theta_f) \tag{2.3}$$

Then, f goes trough the *label predictor* network and gets mapped by G_y to the segmentation label y.

$$y = G_y(f; \theta_y) = G_y(G_f(x; \theta_f); \theta_y) \tag{2.4}$$

And finally, f also goes trough the *domain classifier* network and gets mapped by G_d into the binary domain label d.

$$d = G_d(f; \theta_d) = G_d(G_f(x; \theta_f); \theta_d) \tag{2.5}$$

Hence, during the training time, they want to minimize the loss corresponding to the label predictor (so the network learns how to properly segment the source / CT images) and to maximize the loss related to the domain classifier (in order to learn which features from source and target / CT and CBCT are invariant to the domain). And, to achieve that, they are going to optimize the parameters from each one of the three parts of the network. They will to find the parameters from the *label predictor* that minimizes the *label predictor's loss* (L_y), the parameters from the *domain classifier* that minimizes the *domain classifier's loss* (L_d) and the parameters from the *feature extractor* that minimizes the *label predictor's loss* and maximizes the *domain classifier's loss.* And λ will be the trade-off parameter that is going to control the interaction between the two losses.

$$\begin{aligned} E(\theta_f, \theta_y, \theta_d) &= \sum_{i=1..N, d_i=0} L_y(G_y(G_f(x;\theta_f);\theta_y), y) - \lambda \sum_{i=1..N} L_d(G_d(G_f(x;\theta_f);\theta_d), y) \\ &= \sum_{i=1..N, d_i=0} L_y^i(\theta_f, \theta_y) - \lambda \sum_{i=1..N} L_d^i(\theta_f, \theta_d) \end{aligned} \tag{2.6}$$

The main objective is to find the parameters $\theta_f, \theta_y, \theta_d$ that correspond to a saddle point of 2.6:

$$(\hat{\theta}_f, \hat{\theta}_y) = \arg\min_{\theta_f \theta_y} E(\theta_f, \theta_y, \hat{\theta}_d) \tag{2.7}$$

$$\hat{\theta}_d = \arg\max_{\theta_d} E(\hat{\theta}_f, \hat{\theta}_y, \theta_d) \tag{2.8}$$

In order to seek those parameters, we are going to use the following stochastic updates, which are close to the Stochastic Gradient Descent, with μ as a learning rate:

$$\theta_f \leftarrow \theta_f - \mu(\frac{\partial L_y^i}{\partial \theta_f} - \lambda \frac{\partial L_d^i}{\partial \theta_f}) \tag{2.9}$$

$$\theta_y \leftarrow \theta_y - \mu \frac{\partial L_y^i}{\partial \theta_y} \tag{2.10}$$

$$\theta_d \leftarrow \theta_d - \mu \frac{\partial L_d^i}{\partial \theta_d} \tag{2.11}$$

The main difference with the SGD updates is that in this case we have the factor -λ, that will allow the network to learn the domain invariant features. However, this backpropagation rule is a inconvenient when it comes to implementation, as it is not a standard learning rule and it doesn't exist in the libraries.

So, as they propose in [8], we added a Gradient Reversal Layer to the network. The GRL acts as an identity transformation during the forward propagation, and as a constant product on the backward.

$$R_\lambda(x) = x \tag{2.12}$$

$$\frac{\partial R_\lambda}{\partial x} = -\lambda \mathbb{I} \tag{2.13}$$

Therefore, we can redefine the total loss equation in 2.6 as:

$$\tilde{E}(\theta_f, \theta_y, \theta_d) = \sum_{i=1..N, d_i=0} L_y(G_y(G_f(x;\theta_f);\theta_y), y) + \sum_{i=1..N} L_d(G_d(R_\lambda(G_f(x;\theta_f));\theta_d), y) \tag{2.14}$$

With this new version, we will be able to implement the stochastic updates of 2.9, 2.10 and 2.11 to find the features that are domain invariant and discriminative at the same time.

Therefore, for our project, we have decided to implement both features in order to perform unsupervised domain adaptation. We will distribute our architecture into a segmenter and a domain discriminator, using as well the gradient reversal layer. With them, we will be able to test our network with the CBCT target domain dataset and get the predicted segmentations, based on the learning process with the CT label predictions and the domain discriminations.

Chapter 3

Materials and methods

3.1 Dataset

For this project we have worked with two datasets, the source (CT images) and the target (CBCT images), and they are both composed by a *train* and a *test* group. Hence we have a total of four groups: *CT_train*, *CT_test*, *CBCT_train* and *CBCT_test*, and each group is constituted by 30 3D images and 3D bladder masks of size 40x40x32 and pixel spacing (mm) = [4.8, 4.8, 6], one for each different patients.

The original datasets were formed by bigger images, of size 160x160x128 and pixel spacing (mm) = [1.2, 1.2, 1.5], but they were too large to fit our network. As we are working with 3D images, when we tried to extract their features on the first step of our network, there were too many parameters to compute and the network saturated. Hence we decided to downsample them, and in order to make the network function we had to downsample by a factor of four: from 160x160x128 to 40x40x32.

For this project, we haven't used any validation set, and here's why. A validation set is used to twist and adjust the hyperparameters of the network before executing the final test set, so it gives us an idea on how our network performs better. In our case though, we have been following some previous works and projects from the same field, so we already had some references of the performance on a network that has a good dice value on medical image segmentation. Mainly, we have been following [8] results (detailed in the Discussion chapter) as a reference to guide our results interpertation and, regarding our hyperparameters, most of them were already selected in the previous project [6] with high dice values as well ($.874 \pm .096$).

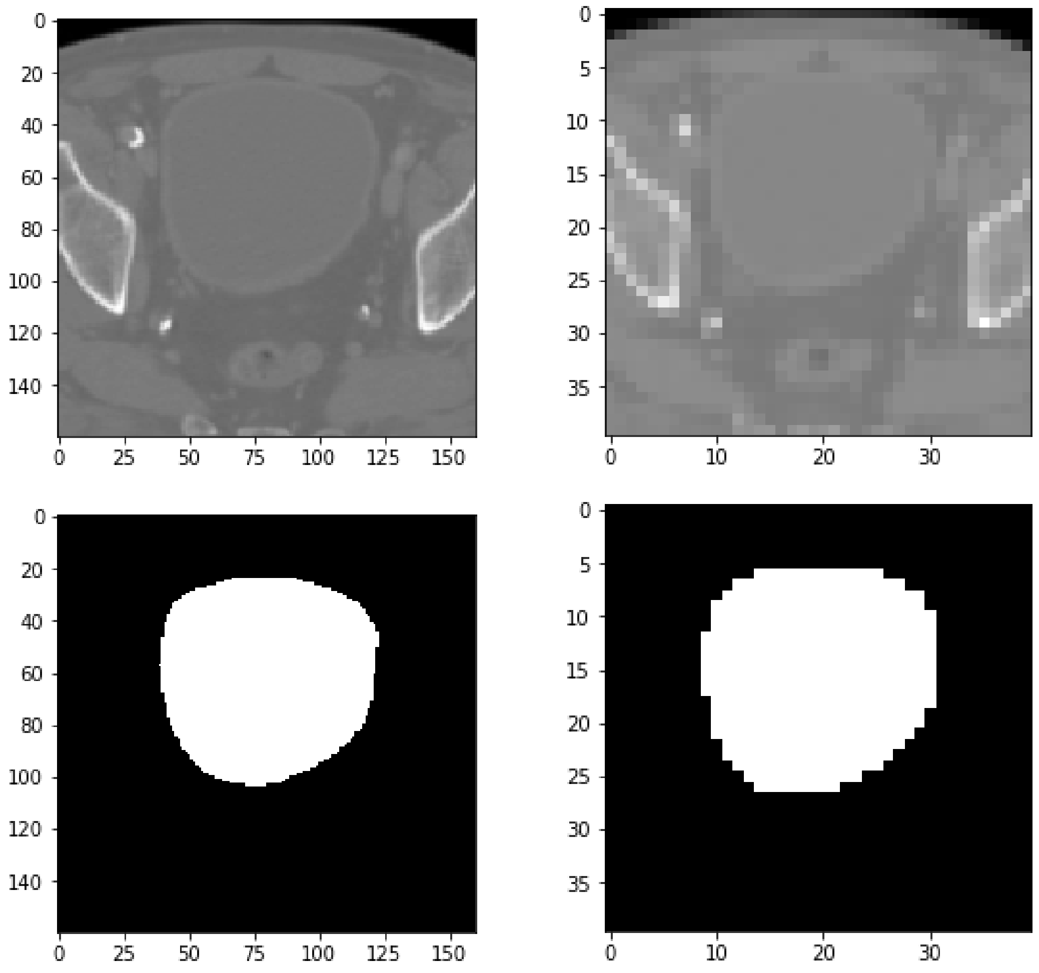


FIGURE 3.1: Slice of CT scan image of the male pelvic region and its mask. Top left: original image, bottom left: original bladder mask segmented by hand (both with pixel size = [1.2, 1.2]). Top right: downsized image, bottom right: its downsized bladder mask (both with pixel size = [4.8, 4.8]).

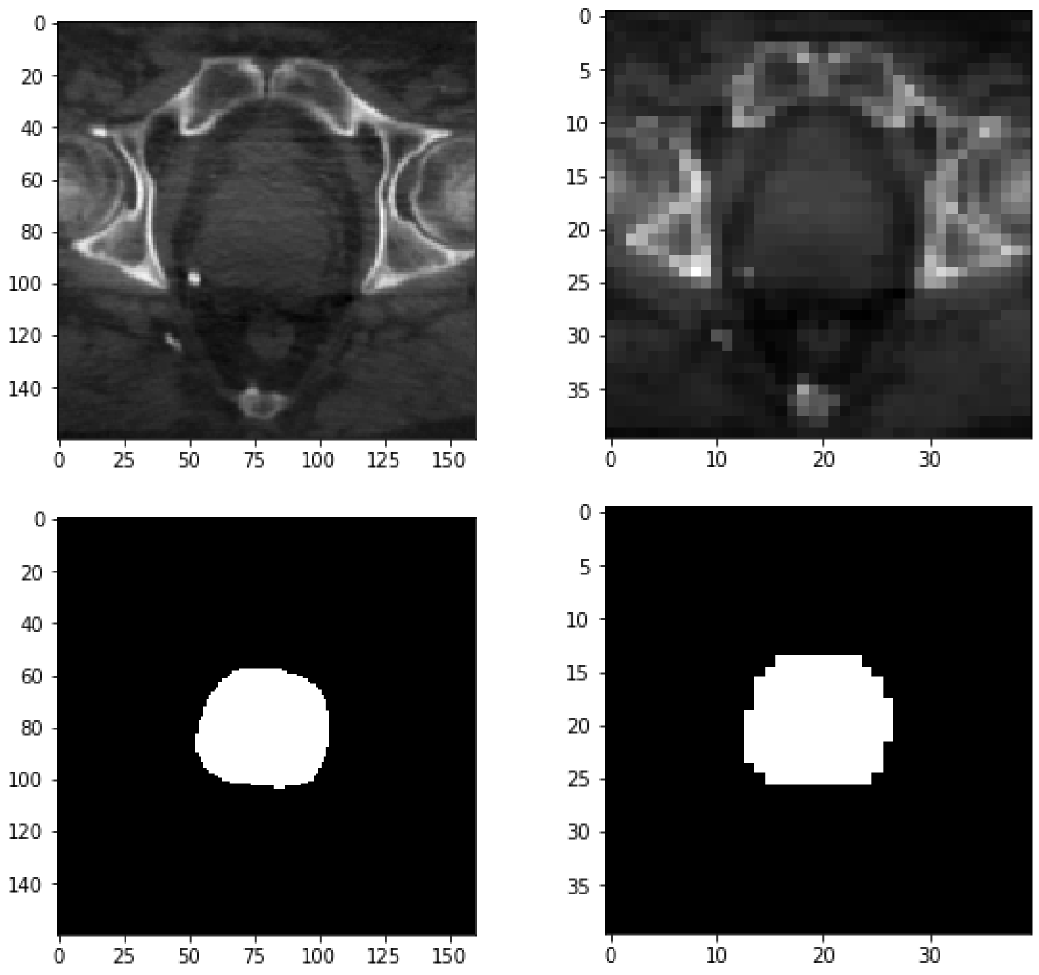


Figure 3.2: Slice of Cone Beam CT scan image of the male pelvic region and its mask. Top left: original image, bottom left: original bladder mask segmented by hand (both with pixel size = [1.2, 1.2]). Top right: downsized image, bottom right: its downsized bladder mask (both with pixel size = [4.8, 4.8]).

We are aware that the resolution with which we are working is very low, definetley not cliniclal aplicable. However it allows fast experimentation and our hope is that the conclusions from working with downsampled data can also be applied to the original dataset of full resolution.

3.2 Environment

For this project we have used `Ubuntu` [1] and `Anaconda` [2] as the virtual environment manager, and `Jupyter notebook` [3] as a development tool, and the whole project was written in `Python` [4]. We have trained and tested our CNNs with `nVidia cuDNN` [5] acceleration on a computer equipped with a GPU `nVidia GeForce GTX 1080 Ti` [6] 11Gb, and our training time is 11 minutes.

Regarding other packages that we used, `TensorFlow` [7] and `Keras` [8] are the most important ones due to their powerful handling of deep learning.

You can find our whole code containing the development of our 4 experiments in the following repository on Github:
`https://github.com/annacuxart/unsupervised_domain_adaptation_unet_CBCT_segmentation`

3.3 Hyperparameters

Each neural network has its own characteristic elements, and their initialization, optimization and tuning are key to make it optimal and powerful.

In the previous sections we have been talking about the *Parameters* of the network, the coefficients that the network optimizes using different algorithms and strategies, for example the weights W and the biases b. These parameters are initialised by us, but during the whole training process they are chosen and updated by the algorithm itself, in order to reduce the loss and improve the network.

Another characteristical element of a neural network is its *Hyperparameters* 2.1.1, and the main difference with the other ones is that we set them at the beginning of the training process and they rest untouched, they are not optimized by the network. In order to set them, we require a minimum knowledge of deep learning, as they determine the structure of our network and how is it going to be trained.

[1]https://ubuntu.com/
[2]https://www.anaconda.com
[3]http://jupyter.org/
[4]https://www.python.org
[5]https://developer.nvidia.com/cudnn
[6]https://www.nvidia.com/en-sg/geforce/products/10series/geforce-gtx-1080-ti/
[7]https://www.tensorflow.org/
[8]https://keras.io/

For this project, we have selected the following Hyperparameters, following the implementations of [6]:

- **Learning rate:** at first we setted the same learning rate for both networks, but after several training trials we realized that the segmenter and the domain discriminator had different learning speeds, so we finally opted for splitting them as it follows.
  - *Segmenter learning rate:*

    $$\mu_S(p) = \frac{\mu_0}{(1+\alpha p)^\beta} \tag{3.1}$$

    where p = (current step) / n_{steps}, $\mu_0 = 1e$-2, $\alpha = 10$, $\beta = 1$ and $\eta_{steps} = 1e4$.
  - *Domain discriminator learning rate:*

    $$\mu_D(p) = \psi \times \mu_S(p) \tag{3.2}$$

    where $\psi = 1e$-2.
- **Total Loss:** as descrived in 2.14, we will compute our total loss by adding both segmenter's and domain discriminator losses, taking into account that the second one includes the gradient descend layer 2.12 and 2.13.

  $$\tilde{E}(\theta_s, \theta_d) = \sum_{i=1..N, d_i=0} L_s(G_s(x;\theta_s)) + \sum_{i=1..N} L_d(G_d(R_\lambda(x;\theta_d))) \tag{3.3}$$

  In this equation, L_s and L_d stands for the segmenter and discriminator's losses respectively. They are both computed with the cross entropy of the predictions on a training batch, the first one using a *sigmoid* as an activation function, and the second one using a *softmax*. The subindex i stands for all the samples in the batch, and $d_i = 0$ means that the segmenter's loss is only computed using the samples from the target domain, contrary to the discriminato's loss, which uses data from both domains. x corresponds to the input data, which is first mapped by the segmenter G_s with its vector of parameters θ_s. At the same time, the input data goes trough the gradient reversal layer R_λ and enters the domain discriminator G_d, with its parameters θ_d. Finally, both losses are computed and added up to obtain the total loss of the network.

- **Batch Size:** as we are working with 3D images, which translates to a lot of parameters, we decided to feed them into the network by small batches at a time. So we finally chose a batch size = 4.
- **Number of Feature Maps:** feature maps get their name by their function of mapping of where a certain kind of feature is found in the image, a high activation means a certain feature was found. In our case, we have chosen 12.
- **Domain discriminator architecture:** another hyperparameter that we can adjust is the architecture of the domain discriminator network. In our case, following the examples of the original papers [8], we decided to implement a Convolutional Neural Network, described in more detail in 3.3.
- **Lambda:** as described in [8], this hyperparameter adjusts the strength with which the segmenter is adapting its features in order to counter the domain discriminator, and it also controls the strength with which the discriminator is adapting his features. For our project we have chosen to use the same lambda schedule one as the original Kamnitsas paper proposed.

$$\lambda(p) = \begin{cases} 0 & if \quad p < p_1 \\ \lambda = \lambda_{max} \times \dfrac{p_{curr} - p_1}{p_2 - p_1} & if \quad p_1 < p < p_2 \\ \lambda_{max} & if \quad p_2 < p \end{cases} \tag{3.4}$$

Where $p_1 = 0.2$, $p_2 = 0.7$ and $\lambda_{max} = 3$.

3.4 Our architecture

In order to implement the unsupervised domain adaptation to segment CBCT images, we have taken the architecture of [6], which was based on 2.2.1 and 2.2.2, and we have adapted it to work with 3D images instead of 2D. We have implemented two connected Neural Networks: the *segmenter* and the *domain discriminator*.

But we have done a modification that has not been seen yet. In our case, we have used the CNN *Unet* [22] as the model for the segmenter network, and a regular fast-forward CNN for the domain discriminator.

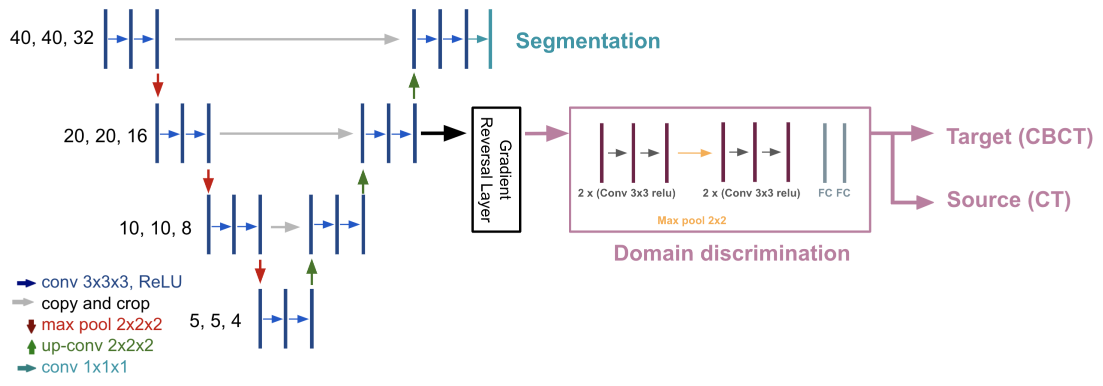


Figure 3.3: Our architecture is composed by: Unet structure as a segmenter to predict the labels, and a regular CNN as a domain discriminator (the adversarial network) in order to implement the unsupervised domain adaptation, finding those features that are invariant to the domain. The domain discriminator is connected with a gradient reversal layer to the 6th level of the segmenter, in order to compute the UDA with backpropagation, as discussed in 2.2.

As we can see in 3.3, we have build a combination of a modified Unet (see more details in 3.4.1) and a CNN. Our Unet has a depth of 4 layers instead of 5 in order to adjust it to the dimensions of our input data, as we will explain in the following section. The second network, the domain discriminator, is implemented with two convolutional layers with 3x3x3 filters and a ReLU, a MaxPooling layer of 2x2x2, another two convolutional layers and two final fully connected layers (for more details of each layer see Appendix A).

In the original paper of [8] they proposed different structures, by connecting the domain discriminator at different levels of the expanding part of Unet. In our case we decided to connect it at the 6th level as a start, but for future steps it would be interesting to compare it with the results obtained at different levels, even a combination of them.

And, as descrived in 3.3, we will compute the total loss of our network as the loss of the segmenter plus the loss of the domain discriminator using backpropagation. Let's now take a look at the definition of the original Unet itself.

3.4.1 Unet

Unet [22] is a known CNN that relies on data augmentation A.4.1 to take more advantage of the available labeled data samples.

This network is composed by convolutional layers with 16 3x3 filters with stride 2, non padded and activated by *ReLU*. After each convolution layer there is a 2x2 max-pooling layer that reduce the amount of features.

Its name appeals to its "U" shape, due to its contracting path, that captures the context in the data, and its expanding path, that allows precise localization, and it is commonly used for biomedical image segmentation, such as [23] and [24].

3.5 Metrics

Finally, in order to evaluate our results and compare them with previous related works, we are going to use the following metrics: dice score coefficient and Hausdorff distance. The first one is overlap based and the second one is distance based, so they provide complementary information about the quality of the segmentation results:

- **Dice Score Coefficient:** When working with neural networks for image segmentation, the *DICE* score (also known as the "similarity coefficient") [25] is a commonly used metric. It quantifies how closely the network's outputs match the training dataset comprising hand-annotated ground truth segmentation.

  In our case, it will let us compare the automatic image segmentation that the NN estimates with the real segmentation, the one done by hand by the Doctors. It will compute the size of the overlap of the two segmentations divided by the total size of the two affected areas (the ones that we are trying to segment). Therefore, it will provide us the information of the networks accuracy, so we can know how is it working and we can make the adequate arrangements.

$$DICEScore = \frac{2 * TP}{2 * TP + FP + FN} \tag{3.5}$$

TP stands for the number of True Positive pixels, FP is the number of False Positive pixels and FN is the number of False Negative pixels, as illustrated in 3.4.

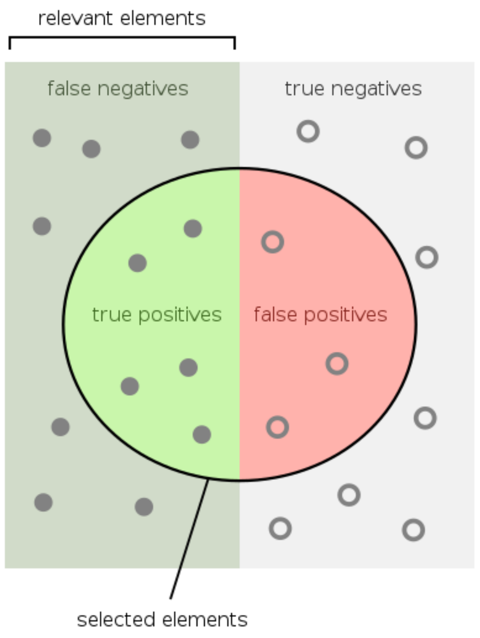


FIGURE 3.4: The classification chart [26] is commonly used in medicine, in order to perform binary classification tests. It shows the original positive and negative samples (relevant elements and right side respectively) and the prediction that it's been made (selected elements). With this, we can know which samples have been correctly predicted as positive (true positive), which ones have been wrongly predicted as negative (false positive), which ones have been wrongly predicted as negative (false negative) and which ones have been correctly predicted as negative (true negative).

- **Hausdorff distance:** Besides the Dice Score Coefficient metric, in order to evaluate our results we also computed the Hausdorff Distance [27] between the original segmentations and our predicted ones.

  The *Hausdorff Distance* is a mathematical way to compute, using the Euclidean distance, the maximum distance of a set to the nearest point in another set, as it follows:

$$h(A, B) = \max_{a \in A} \{ \min_{b \in B} \{ d(a, b) \} \} \tag{3.6}$$

In our case, we have computed this metric for each one of our experiments, selecting the patient which DSC was closer to the mean. As our data consists in 3D images, we have measured the shortest distance of any point in the original segmentation to any point of the predicted segmentation, in the three dimensions.

Chapter 4

Experiments & Results

As we have described in the previous sections, our main goal is to train our network with the source labeled dataset (CT images) and the target unlabeled one (CBCT images), in order to be able to automatically segment the target images with the segmenter, based on the information of the domain invariant features that the domain discriminator provides. In order to achieve that, we have divided this project into four experiments.

1. *The Lower Bound*
2. *The Upper Bound*
3. *The Unsupervised Domain Adaptation*
4. *The Unsupervised Domain Adaptation with* $\lambda = 0$

The first two experiments aim to get information about the worst and the best case scenario in this work, those being train the network only using the segmenter without the domain discriminator, and feeding it with only CT examples and only CBCT examples respectively.

The third experiment is the main one, where we train the network using both segmenter and domain discriminator in order to implement the unsupervised domain adaptation process. In that case, we are going to train it with both datasets, even though not both parts of the network will have access to them. And the last one is a variation of the third, to understand how the hyperparameter lambda affects the network. That being said, let's proceed to describe each one of them in more detail and discuss their results.

4.1 Lower Bound - train on CT

With this first experiment we wanted to observe the lowest result that we could expect from our network working only with the segmenter, hence we fed it only with samples from the source CT dataset and we tested it with both CT and CBCT datasets.

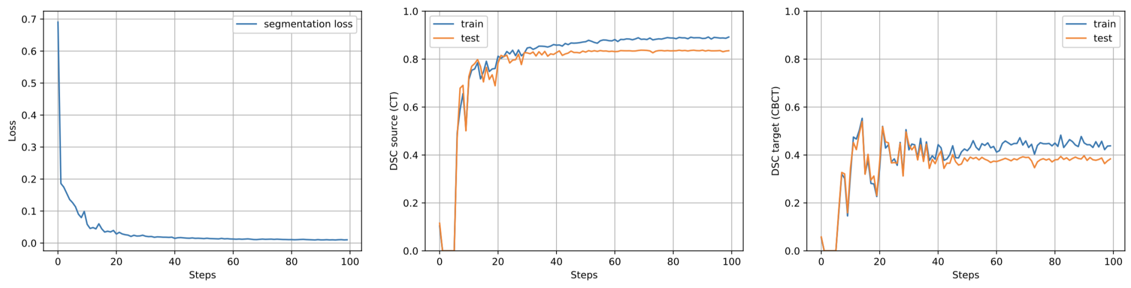


Figure 4.1: With this experiment, the lower bound, we wanted to have a worst case scenario for our project, by training the network on CT using only the segmenter. We can see how we obtained high dice scores on CT and low ones on CBCT (0.383 on the test set, even though in this experiment both CBCT train and test are seen as test datasets), as they are both from different domains and the segmenter itself is not capable to segment CBCT by only seen CT samples. At the left we can see how the segmentation loss decreases during the training process.

Observing the results from 4.1, we can say that the Dice Score Coefficient of the CT dataset (second graph) presents good results, as the final DSC value gets past 0.8 and that's a result that we have seen in previous works that were also based in CT and CBCT image segmentation. We can appreciate as well that the CT train dataset has a higher DSC value than the test CT dataset. That's the behaviour we would expect, as the network has never seen the CT test dataset before, but as it is similar to the train one it is able to segment it successfully.

Regarding the DSC results of the CBCT datasets (third graph), we can see that the segmenter doesn't work as accurate as on CT. That's because CBCT images are not in the same domain as the CT, and therefore, as the segmenter is trained only on CT, its performance on this second group decreases.

4.2 Upper Bound - train on CBCT

With this second experiment we wanted to obtain our upper bound, we wanted to know how the network would behave in the best case scenario again, using only the segmenter: training it with CBCT and testing with CBCT. Therefore, we trained it only with CBCT samples, and we tested it on both CT and CBCT datasets, to compare the results obtained with the first experiment.

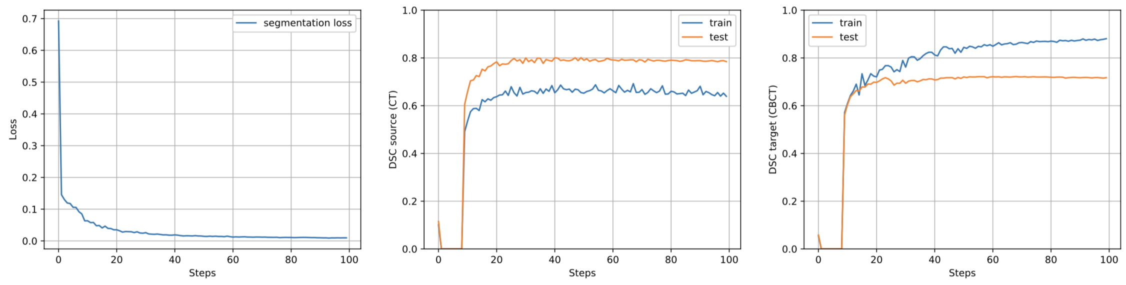


FIGURE 4.2: With this experiment, the upper bound, we wanted to have the best case scenario for our project, by training the network on CBCT using only the segmenter. We can see how we obtained high dice scores on CBCT (0.717), and surprisingly high ones on CT as well. At the left we can see how the segmentation loss decreases during the training process.

By taking a look at the right graph of 4.2, at the DSC results on the CBCT dataset, we can appreciate that they have increased in respect to the lower bound. And the CT ones work fine as well, even though the performance decreases in 0.1.

We can settle that the segmenter achieves better results when training on CBCT and testing on CT that not the opposite. One guess to explain this behaviour is that it might be easier to segment CT images because they have a higher quality, a better contrast. Hence the gap between the DSC of source and target datasets when we train with CBCT samples is not as large as when we train on CT.

In this case we can see that the CT test results are higher than the CT train, and that's as well the expected behaviour. In this experiment, as we are training only with CBCT, both CT train and test datasets are seen as test types, as the network hasn't seen either of them before. Thus it is a random behaviour which one gets greatest values, in this situation it could have been the CT train as well.

One aspect that is surprising is that DSC results on CT are always higher than CBCT ones, even when we train the network on CBCT. And again, our guess is that it might be because CT images have a higher quality and maybe are easy to segment, but we are not sure about it.

4.3 Main experiment - Unsupervised domain adaptation

With this experiment we developed the main goal of this project: implement the unsupervised domain adaptation with Unet and an adversarial neural network. For this we trained our full network, including the segmenter and the domain discriminator, with both CT and CBCT datasest and we tested it on CT and CBCT to, again, compare the results with the lower and upper boundaries.

So, even though we train the whole network with both datasets, the segmenter and the domain discriminator have different access to them. The first one has only access to the labeled CT dataset during the trainning period, and to the unlabeled CT and CBCT datasets for the testing. But the domain discriminator, during the training period, has access to both CT and CBCT datasets, with their labeled domains, as it aims to classify them by this feature.

Now, let's take a look at the results we have obtained:

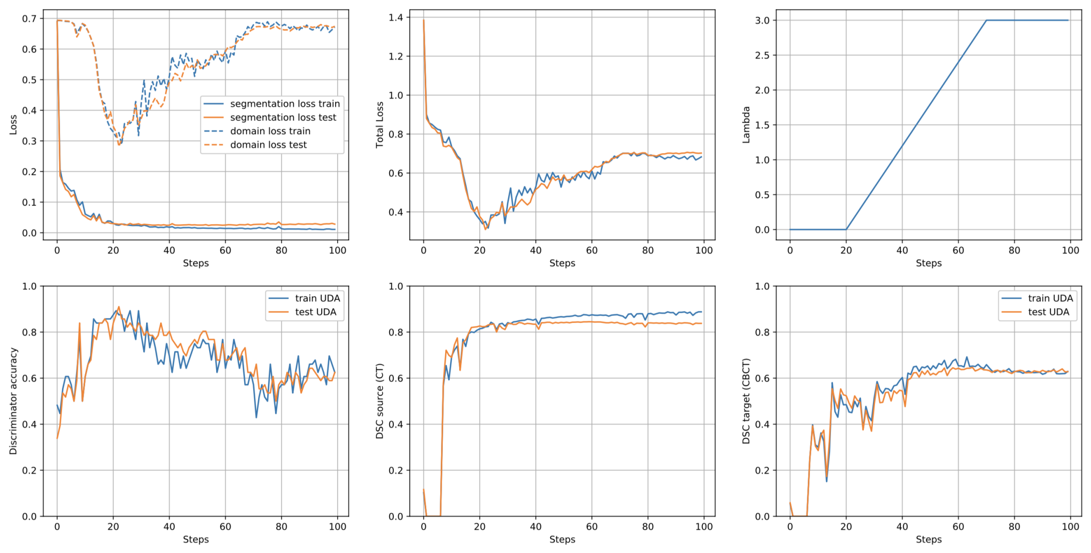


FIGURE 4.3: This is the main experiment of our project, the unsupervised domain adaptation, and for this we used our whole network (the segmenter and domain discriminator), and we trained it with CT and test it on CBCT. We can observe how our CBCT dice scores at the bottom right graph (0.623) improved in comparison with the first experiment (0.383), but they are not as high as our upper bound (0.717).

On the following lines we are going to discuss each one of the graphs that appear in 4.3, to understand their behaviours.

4.3.1 Losses

Starting with the top left graph, we can see both segmenter and domain discriminator losses, for the train and test sets. At the first stage, first 20 iterations, both losses decrease as both networks are being trained independently, so they learn how to perform their tasks properly step after step. Afterwards, from the 20th step and forward, our main goal is to improve the segmenter for CBCTs and to weaken the discriminator.

So the segmenter's loss decreases as we expected, and the discriminators loss augments because we force him to, in order to find the domain invariant features of the samples.

The *segmenter's loss* is computed by doing the cross entropy [28] between the original labels of the samples and the ones predicted by the segmenter, with a sigmoid [15] as an activation function 2.1.1.3.

And the *domain discriminator's loss* is also computed with the cross entropy with the original domain labels and the classified ones, using a softmax [15] as an activation function.

Following to the second graph at the first row, we can observe that the *Total Loss* is the result of the *segmenter's loss* plus the *domain discriminator's loss*, which takes into account the Gradient Reversal Layer, just as it was described in 2.14.

4.3.2 Lambda

Moving to the third graph in the first row, we can see the hyper-parameter *Lambda*. It adjusts the strength with which the segmenter is adapting its features in order to counter the domain discriminator, and it also controls the strength with which the discriminator is adapting his features. For our project we have chosen to use the same lambda schedule one as the original paper proposed [8].

For the first 20 steps we set lambda to 0, meaning that we train the segmenter and discriminator independently. Then, when the domain discriminator starts to learn how to distinguish between CT and CBCT, we start to counter it, so the segmenter gets information about which features are invariant to the domain. For this we will increase lambda following the paper's linear schedule as described in 3.4.

Lambda max is a parameter that we can adjust, we chose to set it to 3. Finally, after the 70th step, we fix lambda to lambda max, in order to focus only on refining the segmenter's features for CBCT.

4.3.3 Discriminator Accuracy

We can define the discriminator's accuracy, first graph of the second row, as the amount of samples that the discriminator is capable to distinguish by its domain.

$$Domain \quad Discriminator's \quad Accuracy = \frac{True \quad Positives + True \quad Negatives}{All \quad samples} \tag{4.1}$$

Regarding its graph we can see that for the first 20 iterations, while lambda stays at 0 value, the accuracy increases as the discriminator is being trained independently, so it starts to learn how to properly distinguish between the two domains (CT and CBCT).

After the 20 iterations, we force him to stop learning, so the accuracy starts to decrease. At this point, we want to know which features are independent from the domain, so we can provide that information to the segmenter to let it know how to operate with CBCT images.

4.3.4 Source DSC - CT

If we take a look at the second graph of the second row we can see the results of the Dice Scores for CT tests and CT train datasets.

Something that is surprising here is that, in this case, the results are not very different from our first experiment, the Lower Bound. And that is not something we would expect. In this situation we also do the training with CT, but we are constraining a little bit with the network, we only let it do the segmentation based on the features that are invariant to the domain. So it is interesting to see that even with that restriction, the network gets up to a 0.8 Dice Score when segmenting CT images, as intuitively we would expect it to be lower.

One guess could be that maybe it is because the lambda max is not that high, and if we increase it there would be more intervention between the two learnings, and maybe it would go lower. But the truth is that we don't know why it behaves like this, and it would definitely be an interesting field to research for further steps of this project.

4.3.5 Target DSC - CBCT

Finally, on the bottom right graph we can find the Dice Scores results for the CBCT datasets. As it was expected, this ones are placed between the Lower and the Upper bound, as in this case the network is trained on CT but the features learned are the ones invariant to the domain of the samples. So it is the foreseen behaviour, dice scores between the 2 bounds.

4.4 Unsupervised domain adaptation with lambda = 0

As a last experiment, we computed the unsupervised domain adaptation as in the 3rd experiment but in this case with

$$\lambda = 0 \tag{4.2}$$

With this experiment, we aimed to compare the behaviour of our network when its two parts, the segmenter and the domain discriminator, work and are trained independently.

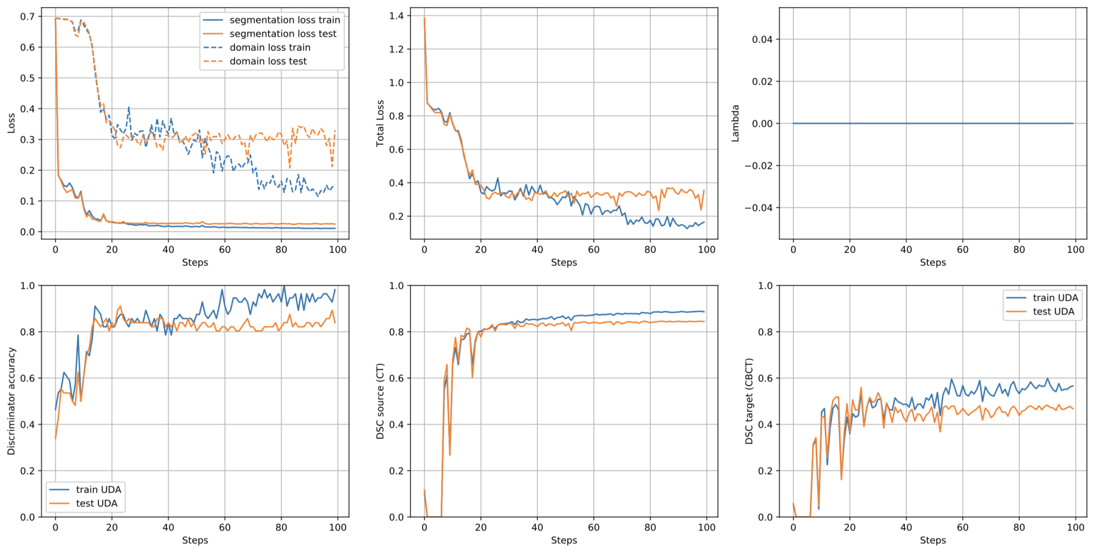


FIGURE 4.4: This is a variation of the UDA experiment, by setting $\lambda = 0$. With this, we make the segmenter and domain discriminator train independently, and we can see how the dice score of CBCT decreases to 0.467.

We can see that, in this case, both segmenter's and discriminator's losses decrease along the iterations, and the domain discriminator's accuracy increases. That's the results we would expect, as with $\lambda = 0$ both networks are trained independently, and there's no aim to carry through the unsupervised domain adaptation.

We can also appreciate that the Dice Scores for the CT dataset are close to the ones in the first experiment, the Lower Bound, when we trained the network only on CT examples. And, for the CBCT Dice Score, we can see that it is as well close to the first experiment. And that's the behaviour we would expect, as not the segmenter is not looking for the features that are independent to the domain, so he doesn't know how to segment the CBCT dataset.

4.5 Results examples

In the previous sections we have seen and discussed the analytical results of our project, hence now let's take a look at some examples of the predicted CBCT segmentations and compare them with the original ones.

In the following figures 4.5 we can compare the original handmade segmentations by the doctors (in green) with the predicted automatic segmentations by our network (in red). We have shared the results of the same 3D CBCT image from the same patient (the one which results were close to the mean) at three different dephts computed in the main three experiments: slide 0, slide 4, slide 9 and slide 12.

For each slide, we can first appreciate the original CBCT image itself (first column in the left), at a 40x40 resolution and pixel space = [4.8, 4.8]. In the second column from the left we can see the first segmentation, corresponds to the Lower Bound experiment, where we trained the network only on CT and we tested it on CBCT. As we can see, the predictions are far from acceptable, and that's what we would expect, as in this case the network is only using the segmenter, and without the UDA technique it is not capable to segment the CBCT images, that differ the domain from the train dataset CT.

Following to the right, in the third column from the left, we can see a notable improvement, as the predicted segmentation almost fits the original one. This is due to the fact that on the second experiment, the Upper Bound we are training the network only using CBCT images, therefore it is easier for the network to segment images from the same domain.

Lastly, in the fourth column at the right, we can observe the predicted results from our last experiment, the Unsupervised domain adaptation. In this case we are using both parts of our network, the segmenter and the domain discriminator, and even though we are training the segmenter only on CT, we can see the improvements in comparison with the first experiment (second column). In this case, the network is focusing on the domain invariant features from the input data, so even being trained on CT it is almost capable to imitate the segmentation on CBCT.

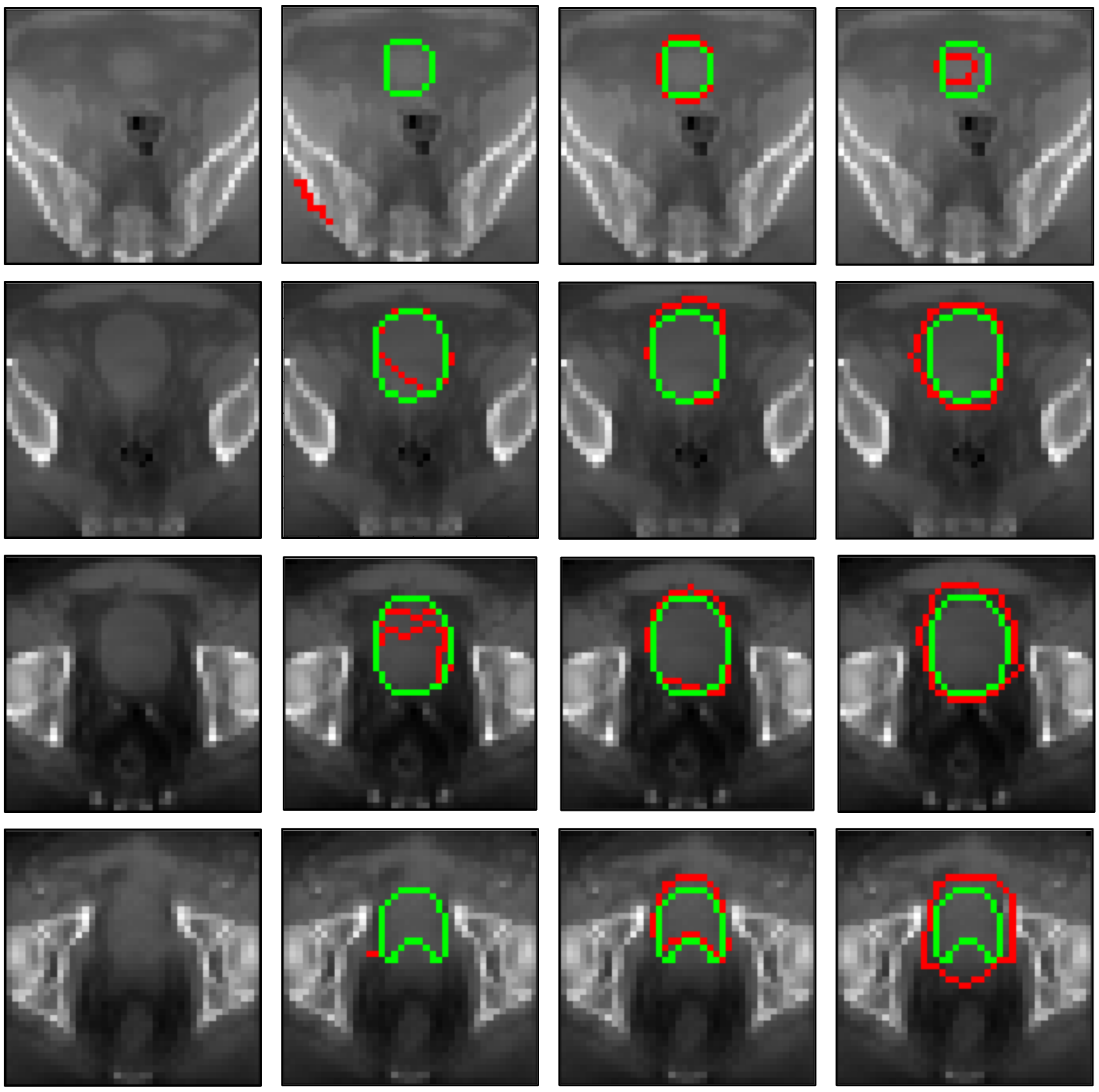
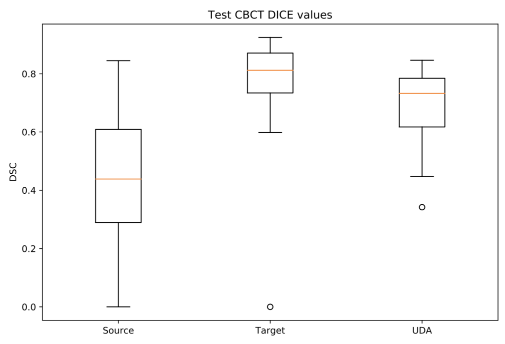


FIGURE 4.5: Comparison between the original bladder segmentations (in green) and the predicted segmentations (in red) of CBCT images. These examples correspond to our three first experiments of one patient with results closer to the mean: lower bound (second column), upper bound (third column) and unsupervised domain adaptation (last column). The first column corresponds to the original CBCT images. For each experiment (column) their respective DSC scores and hausdorff distance values (in voxels) are: 0.378 and 40.93, 0.892 and 33.17, 0.780 and 48.22. Each row corresponds to a different slice of the 3D image, from top to bottom: slice number 0, 4, 9 and 12 (in total each 3D image has 32 slides). We can observe how the second column represents our worst case scenario, the third one represents the best and the last one represents the UDA with a performance between them. As we can see, their DSC values follow the same pattern: the lower bound gets the lowest DSC with 0.378, the upper bound gets the highest with 0.892 and the UDA stays in between with 0.780. What surprises us here is the UDA hausdorff distance, as it should be between both bounds but instead goes higher than the upper bound.

In the following figure 4.6 we can observe the variations of the Dice Scores obtained for the CBCT test dataset, as well as the hausdorff distances computed.

If we take a closer look at the DSC values, from left to right, we can appreciate that the *source* results are the most spread, and that's what we would expect from the Lower Bound experiment, as it is the worst case scenario of our project. Secondly, moving to the best case scenario, we can see that for the *target* experiment the Dice values are the most concentrated, as in this case we train on CBCT and test on CBCT. Finally, the distribution of the *unsupervised domain adaptation* experiment is settled between the source and the target experiments.

Regarding the right boxplot, corresponding to the Hausdorff distance computed between the original segmentations and the predicted ones, we can see that the source distances are higher than the target, and that's the result we would expect, as they are the worst and best case scenarii. What surprises us here is the results from the unsupervised domain adaptation distances, that are higher than the source experiment and have a higher deviation. We are not sure about the reason of this behaviour, but we have to mention that the distance is computed in voxels. So in order to adjust it to mm, we should take the distances from the axis X, Y and Z and multiply them by their pixel spacing, which are respectively 4.8, 4.8 and 6 mm.



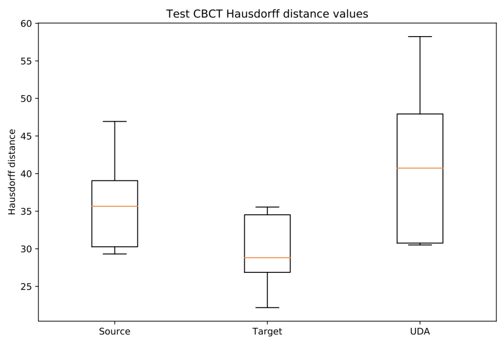


Figure 4.6: Boxplots of the DSC values distribution (right) and Hausdorff distance in voxels (left) for the 30 test CBCT patient.

Chapter 5

Discussion

In the previous section we have been sharing and analyzing the results obtained in our three experiments: the *lower bound*, the *upper bound* and the *unsupervised domain adaptation*. Through the following pages, we will discuss the results obtained in our work, and we will compare them with the papers there were based on and some other related projects.

First, let's compare our results with the original Kamnitsas paper on *unsupervised domain adaptation with Adversarial Networks* [8]:

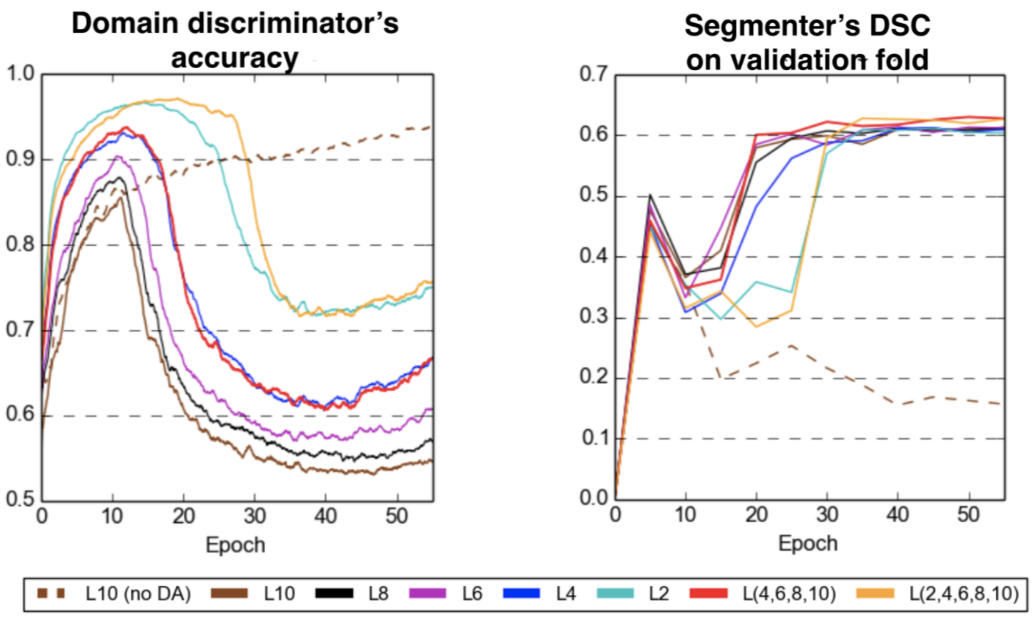


FIGURE 5.1: Original results from the Kamnitsas paper [8] about the domain discriminator's accuracy and the segmenter's dice values.

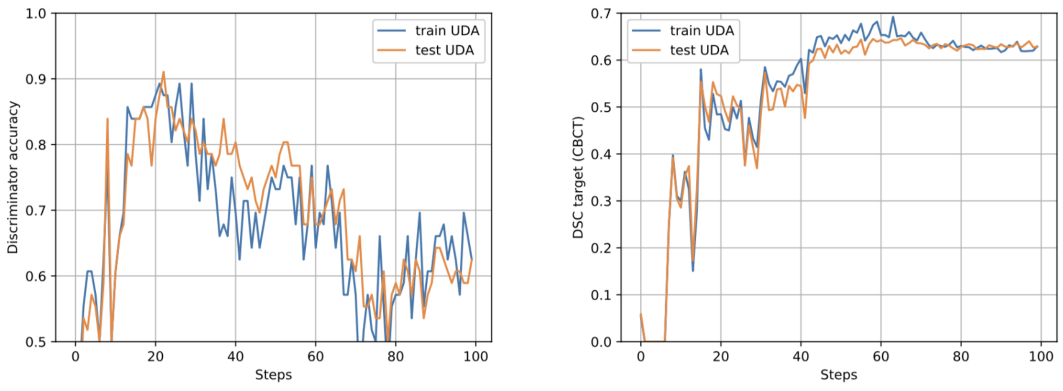


FIGURE 5.2: Results from our unsupervised domain adaptation experiment, showing the discriminator's accuracy and the dice value of the target CBCT dataset across 100 steps. We can observe that they have a similar shapre compared to Kamnitsas results, but ours are more inestable.

First of all, we didn't got to contrast our network's behaviour placing the domain discriminator at different layers of the segmenter as they proposed (see different colors of the legend in 5.1), even though we have that in mind for the possible future steps of this project. In our case, we tested the network with the discriminator placed at the 6th level of Unet 3.3, so those are the results we are going to analyse.

It is important to remind that our main goal is to achieve a high DSC value when testing on CBCT dataset. It is not necessary for the segmenter to work perfectly and the discriminator to do its worst, we just want to find the best combination of the two networks in order to achieve that high DSC score 2.7 and 2.8.

If we take a look at our domain discriminator, what we could expect from it is to not be too close to 1 nor too close to 0. In the first case it wouldn't find the features invariant to the domain, and on the second one it would mean that the segmenter puts too much effort in fooling the discriminator. So we would expect the discriminator's accuracy to be between 0.5 and 1, and that's precisely the range that we've obtained in our results.

Comparing it with the original one presented in Kamnitsas paper, we can set that they have a similar behaviour, yet their curve is smoother than ours. One reason to explain that might be that we have way less data than them, and also we are working with 3D images. Another reason might be that our domains are much more different that the ones of the original paper, as they are doing the adaptation between two MRI datasets. So for them it's the same modality but the samples are taken from two different machines, meaning that the domains are much closer to another. In our case we have a bigger challenge, as CT and CBCT are totally different domains.

Besides, we are working at a lower resolution, as we explained in 3.1, and also we are working with a smaller batch size for the same reason. So we think those factors can contribute to our instability.

The last reason to explain the differences between our domain discriminator's accuracy and theirs is that we are using Unet architecture for our segmenter and they are using a classical CNN without skip connection. In the paper they are using domain adaptation in a completely fast forward network, only playing with downsampling pass, while we are working with a much complex network that includes skipping connections. That's a significant difference and that's the main contribution of our work, as we are studying UDA in a possibly more challenging setting, which might bring instability and more variability amongst others to our code.

In the following table we can find the comparison between our DSC and Hausdorff distance results, with other related projects on CT, CBCT and other radiotherapy segmentation methods.

Study	Method	DSC	HD (voxels)
Ours: lower bound	DL($n_{CBCT} = 30, n_{CT} = 30$)	$.383 \pm .260$	36.47
Ours: upper bound	DL($n_{CBCT} = 30, n_{CT} = 30$)	$.717 \pm .177$	27.44
Ours: UDA	DL($n_{CBCT} = 30, n_{CT} = 30$)	$\mathbf{.623 \pm .149}$	**39.18**
DIR [6]	Rigid image registration	$.714 \pm .149$	$6.93 \pm 4.09^{\star}$
DIR [6]	DIR, RS intensity-based	$.737 \pm .155$	$6.27 \pm 4.08^{\star}$
DIR [6]	DIR, Morphons	$.784 \pm .151$	$5.04 \pm 3.90^{\star}$
Léger et al. (2020) [6]	DL($n_{CBCT} = 42, n_{CT} = 74$)	$.874 \pm .096$	$2.47 \pm 1.93^{\star}$
Schreier et al. (2019) [30]	DL($n_{CBCT} = 300, n_{CT} = 300$)	$.932^{\dagger}$	$2.57 \pm .54^{\dagger\ddagger}$
Brion et al. (2019) [22]	DL($n_{CBCT} = 32, n_{CT} = 64$)	$.848 \pm .085$	$2.8 \pm 1.4^{\star}$
Hänsch et al. (2018) [31]	DL($n_{CBCT} = 124, n_{CT} = 88$)	.88	-
Motegi et al. (2019) [32]	DIR, MIM intensity-based	$\sim .80$	-
Motegi et al. (2019) [32]	DIR, RS intensity-based	$\sim .78$	-
Takayama et al. (2017) [33]	DIR, RS intensity-based	$.69 \pm .07$	-
Woerner et al. (2017) [34]	DIR, Cascade MI-based	$\sim .83$	$\sim 2.6^{*}$
Konig et al. (2016) [35]	DIR, Rigid on bone and prostate	$.85 \pm .05$	-
Thor et al. (2011) [36]	DIR, Demons	.73	-
van de Schoot et al. (2014) [37]	Patient specific model	$\sim .87$	-
Chai et al. (2012) [38]	Patient specific model	.78	-

TABLE 5.1: Dice Score Coefficient and distance comparison between our results and related projects. For our three experiments (lower bound, upper bound and UDA) we have computed the hausdorff distance (HD) in voxles (in order to set them into mm we should multiply each axis by its pixel spacing value). $\star$ SMBD: Symmetric mean boundary distance. † Results reported on a test set containing both CBCT and CT scans. ‡ The authors compute the root mean square boundary distance. $*$ The authors compute the mean boundary distance.

The methods presented in the table stand for the following. DL: deep learning, with their datasets composed by the number of CBCT images (n_{CBCT}) and the number of CT images (n_{CT}), DIR: Deformable image registration and RS: RayStation. If we compare our results obtained with the other works we can clearly see that there's an important gap. However, we believe that they are promising still, and we could expect that at full resolution and with sufficient training data our method could be competitive with others, as we should take into account that our work has faced the following restrictions: firstly, we have trained our network with a smaller dataset, as ours is a subset of the original Léger et al. dataset; also, as described in 3.1, we are working with images of a lower resolution, as we had to downsample the original ones by a factor of 4 because of our GPUs restrictions 3.2.

If we take a look at the Hausdorff distance results, we can observe that the distance in the lower bound is higher than the upper bound, and this is the behaviour that we would expect, as they are respectively the worst and best case scenarios for our project.

What surprises us here is that the distance for the third and main experiment, the unsupervised domain adaptation, is higher than the lower bound, and that is unexpected as it should be between the lower and the upper bound. We are not sure why it behaves like this, but we know how could we improve it as a following steps of this project: we could compute the mean Hausdorff distance in all the layers of the 3D image, by implementing the Symmetric mean boundary distance (SMBD) [6].

$$SMBD = \frac{\bar{D}(M,P) + \bar{D}(P,M)}{2} \tag{5.1}$$

SMBD computes the mean distance $\bar{D}$(M, P) from the contour of the manual segmentation (M) to the predicted segmentation (P) 3D binary masks, then the mean distance from P to M and finally computes the mean of their sum. The distance D(M, P) stands for:

$$D(M,P) = \{min_{x \in \Omega_p} \parallel s \odot (x - y) \parallel, y \in \Omega_M\} \tag{5.2}$$

where Ω_M, Ω_P are the boundaries extracted from M and P, and s^T stans for the pixel spacing in mm (4.8, 4.8, 6). That is why we think it could bring more accurate information about the results, as it is more robust to the outliers than the Hausdorff distance. Also, another way to improve the Hausdorff distance results could be to look at the 95% percentile of all the distances computed, so we wouldn't take into account the outliers as well.

Chapter 6

Conclusions

The main goal of this master thesis is to implement an automatic segmentation of the bladder in CBCT images using deep learning. CBCT images are used in radiotheraphy, a treatment used in patients with cancer, right before the previously planned radiation is delivered to the patient. We believe that by segmenting CBCT images, we can help to improve the dosification of the radiation, conducting its focus to the tumor area so the healthy tissues around it don't receive a higher amount of what they are supposed to.

Unfortunately, we didn't have a labeled dataset of CBCT images, but what we did have was one of CT images, as this other kind of scan is taken previously at the beginning of the treatment planning and needs to be segmented for the process. So we performed unsupervised domain adaptation between the CT dataset (the source) and the CBCT (the target).

In order to do so, we have implemented an architecture that combines the CNN Unet as a segmenter and a regular CNN without skip connections as an adversarial network with the gradient reversal layer, the domain discriminator. With this, we have successfully acomplished our goal, as by executing UDA (training our network with a source CT dataset and testing it on the target CBCT dataset) we have improved our results compared with training the network only with the source dataset using only a segmenter.

If we take a closer look at the last chapter, we can observe that there's a 20% gap between our results and the mean of the previous projects. We have identified our weaknesses and there are three main reasons to explain that gap, as during this project we have faced some remarkable challenges. First of all, we have been working with a small dataset, 120 3D images with a low resolution. Due to our GPU memory limitations 3.2, we had to split the original dataset and downsize it from 160x160x128 to 40x40x32 in order to fit the data into our network. Secondly, we have dealt with a source and target domains that are completely different. Some of the previous works that were also implementing unsupervised domain adaptation were working with two similar domains, as in [8], where they were doing an adaptation between two MRI datasets. And third, we have brought the innovation of using Unet as a segmenter network, which has never been implemented before for this purpose and it has brought some instability to the network.

We trust that, by implementing the following steps, we can improve these first results: firstly, we would like to work with a bigger dataset (for example the original [6] from which we extracted our subset), and in order to do so we would need a GPU with more than 11GB of memory. Also, we would like to do more research about some of the hyperparameters like λ_{max}, as we think they might bring more stability to the network. As we have mentioned, we could also implement the symmetric mean boundary distance instead of the Hausdorff distance, to get more accurate information about the predicted segmentations and reevaluate our results. Finally, we could also implement other data augmentation techniques in order to enlarge our dataset and improve our results.

As a future work, we would like to follow the steps of [8] and adapt the domain discriminator at different levels of Unet, as in this project we opted to place it only at the 6th. Kamnitsas observed that his method was robust at a specific layer and for a specific value of λ_{max}, and we would like to investigate if this applies to our case as well. Also, we would like to extend this project in order to apply the CBCT segmentation to other male pelvic organs, like the rectum and the prostate, as we already have the datasets of images and masks for them.

Hence, even though we obtained lower results than the other previous works shown in Chapter 5, we still believe that using Unet for 3D Cone Beam CT automatic image segmentation can improve the main concern in Radiotherapy regarding the administration of the radiation. By segmenting CBCT images right after they are taken, we could have a clear vision of each organ's position, and we could use it for: (i) do an online re-planning of the radiation's distribution for that patient, refocusing the higher radiation to the tumor area so the healthy tissues are less affected or (ii) do a selection of the patient, in order to know if he can have an online re-planning or if he needs an offline one, meaning that he must restart the radiation planning process.

Also, we believe that this segmentation method that performs unsupervised domain adaptation with an adversarial network and Unet can be useful for many other applications, even outside the medical field. For example, if we switch to the automobile sector, we could apply this technique to self driving cars. In order to be trained to avoid pedestrians and objects in the road they need a large and specific dataset. So, for instance, they could use this method for the training process in different seasons. They could take the summer dataset as a source domain and the winter dataset as a target domain, and apply this UDA method to train the automobile to segment and recognize pedestrians and objects in winter only by seen summer examples.

Therefore, overall, we think this is an interesting and promising project that can still go further for new improvements, and that can be applied to many different areas to perform automatic segmentation with unsupervised domain adaptation, for those circumstances where we don't have enough labeled data to train our network.

Appendix A

Convolutional Neural Networks

In the following sections we are going to introduce the architecture of a CNN, explaining its different layers in detail and other interesting features.

A.1 Convolution Layer

The main purpose of a convolutional layer is to detect features or visual features in images such as edges, lines, color drops, etc. This is a very interesting property because, once it has learned a characteristic at a specific point in the image, it can recognize it later in any part of it.

Another important feature is that convolutional layers can learn spatial hierarchies of patterns by preserving spatial relationships. For example, a first convolutional layer can learn basic elements such as edges, and a second convolutional layer can learn patterns composed of basic elements learned in the previous layer. And so on until it learns very complex patterns. This allows convolutional neural networks to efficiently learn increasingly complex and abstract visual concepts.

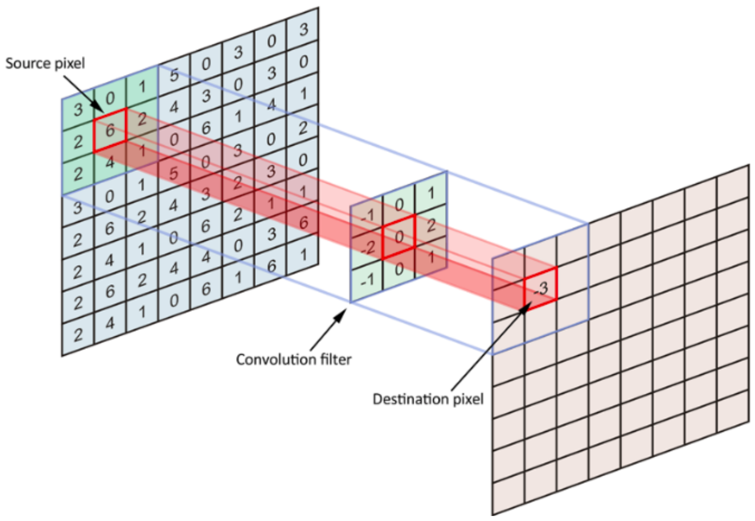


FIGURE A.1: Convolutional Layer [29], where the destination pixels are computed by sliding the convolutional filter trough the source pixels.

There are a few hyper-parameters to consider while using convolutional layers:

- **Filter size**: Amount of neighboring pixels, usually 3x3, 5x5 or 7x7.
- **Channel size**: Number of filters equating to number of "characteristics" that we wish to have, usually 32 or 64.
- **Padding**: Sometimes an output image of the same dimensions as the input is desired, requiring the addition of zeros, before any subsequent windowing. For this, we can use the hyper-parameter padding in the convolutional layers.
- **Stride**: Indicates the number of steps in which the filter window moves. If we set a large stride, the size of the information that will be passed to the next layer will decrease. Usually this is set to 1.

A.2 Pooling layer

Pooling layers A.2 are placed after a convolutional layer, and their main goal is to make their outputs (the so called feature maps) space invariant. Feature maps are very sensible to the position of their input features, and this increases over-fitting. So by applying a polling layer we can down-sample their representation and make them invariant to translations. It will also reduce the number of parameters, which will help to reduce the computational cost.

There are several ways to condense the information. One of the most usual choices is *max-pooling*, which keeps the maximum value of those that were in the input *window*.

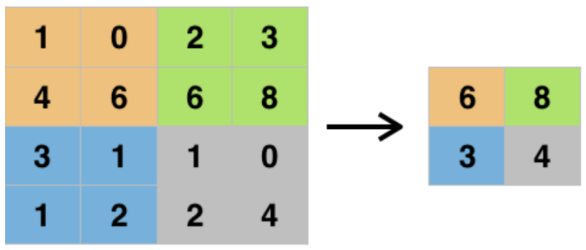


FIGURE A.2: Max Pooling Layer with a 2x2 window [30], where the information of the original image gets condensed.

Average-pooling is also a common choice, where instead of the maximum value the average of the input points group is computed. However, max-pooling tends to work better than alternative solutions [30].

A.3 Fully connected layer

The main objective of a fully connected layer is to take the output of the previous layers (convolutionals and poolings) and use them to classify the input image into a label. To do so, the output result if flattened into a single vector of features, where each value represents the probability that a specific feature belongs to a label.

The fully connected layer will feed the final *softmax* activation function. As its name indicates, this layer interconnects all the outputs from previous layers, mixing all the features together to determine the output of the whole network [31].

A.4 Useful variations and combinations

In the following section, we introduce a few techniques expected to improve performance of neural networks in this environment.

A.4.1 Data Augmentation

The main goal of data augmentation is help to reduce one of the most common problems when working with neural networks: overfitting. With data augmentation we can increase the amount of data available by altering the current images of the dataset to obtain new ones [31].

In our particular case, learning multiple variants from our features could help our network. For example, we could augment our data by rotating or scaling (zooming in) some images. Indeed, the most common augmentation methods are: rotation, flipping, zooming, brightness equalization and contrast equalization.

A.4.2 Transfer Learning

Transfer Learning stands by, instead of training a neural network from scratch, maybe take some knowledge of an already trained network that solves a similar problem. It focuses on storing knowledge while solving one problem and applying it afterwards to a different but related problem [31]. Transfer learning allows the learned model on the ”source task” to be adapted to a similar yet different ”target task”.

Also, the source and target datasets do not need to be from the same distribution, which could be useful for processing datasets of medical images from different scanners (multi-site).

A.5 Batch Size

One of the main constraints that we can face when we work with neural networks is the memory of the computer we are using, as NN require a huge amount of data for the training process. This is a common issue when we work with images, as they are a type of data that require a lot of space.

In terms of memory, there are three steps in the process of training a NN. First we feed the network with our data examples (the images), and they are stored in the computer's Hard Disk memory. Then, they are loaded form the Hard Disk to the RAM memory, where most of the operations take place but, unfortunately, the RAM capacity is smaller than the Hard Disk. That's why we need a data generator that loads the images one by one into the RAM, instead of loading the whole data set at once. Finally, when we train or test our network, the data goes from the RAM to the GPU memory, as it is very fast for matricial computations, but is again smaller than the RAM. So in this case we will need a Batch generator to feed the GPU by small groups of loaded data. The size of this groups will be determined by the Batch Size.

There are also other reasons to use a smaller Batch Size instead of the whole dataset, for example it is directly related with the Stochastic Gradient Descend accuracy [16]. The SGD technique computes the gradient over the entire dataset in order to find, after several loops, the lowest point of a function. So depending on the Batch Size the SGD can be more accurate while taking a longer computational time (for a big Batch Size) or faster, giving quicker updates, but being less accurate (for a small Batch Size). So there is a trade off between the number of updates per time and their quality, and it is usually comprised between 1 and 256 [14].

Appendix B

User Guide

B.1 Environment and dataset

This project has been developed using the tool `Jupyter notebook` and it was written in `Python`. You can find all the details of the environment and the `Github` repository in 3.2, and the dataset that we have used is described in 3.1.

B.2 Train & test

In order to run any of the four experiments you just have to set the parameters *training_mode* and *lambda* to select one of the experiments and then execute the rest of the code cell by cell, as they are all commented and self-described for an easy use:

1. Lower bound: `training_mode = source`, `lambda` commented.
2. Upper bound: `training_mode = target`, `lambda` commented.
3. Unsupervised domain adaptation: `training_mode = dann`, `lambda` commented.
4. Unsupervised domain adaptation training the segmenter and the domain discriminator independently: `training_mode = dann`, `lambda = 0`.

Bibliography

[1] Medical News Today. *What to know about radiation therapy?, September 3, 2019.* URL: https://www.medicalnewstoday.com/articles/158513#types.

[2] Mayo Clinic access date May 2020. *Radiation therapy, March 28, 2018.* URL: https://www.mayoclinic.org/tests-procedures/radiation-therapy/about/pac-20385162.

[3] 2019 National Cancer Institute at the National Institutes of Health January 8. *Radiation Therapy to Treat Cancer.* URL: https://www.cancer.gov/about-cancer/treatment/types/radiation-therapy.

[4] American Cancer Society. *Getting External Beam Radiation Therapy, December 27, 2019.* URL: https://www.cancer.org/treatment/treatments-and-side-effects/treatment-types/radiation/external-beam-radiation-therapy.html.

[5] SimpleElastix. *Non-rigid Registration, Mars 2020.* URL: https://simpleelastix.readthedocs.io/NonRigidRegistration.html.

[6] Jean Léger, Eliott Brion, Paul Desbordes, Christophe De Vleeschouwer, and John Aldo Lee. "Cross-domain data augmentation for deep-learning-based male pelvic organ segmentation in cone beam CT". In: *"Applied Sciences"* (2020). URL: http://hdl.handle.net/2078.1/226998.

[7] Yaroslav Ganin and Victor Lempitsky. *Unsupervised Domain Adaptation by Backpropagation.* 2014. arXiv: 1409.7495 [stat.ML].

[8] Konstantinos Kamnitsas et al. "Unsupervised domain adaptation in brain lesion segmentation with adversarial networks". In: *IPMI.* 2017.

[9] Thomas M. Mitchell. *Machine Learning.* 1st ed. USA: McGraw-Hill, Inc., 1997. ISBN: 0070428077.

[10] Jeremy Jordan. *Machine learning overview., May 29,2017.* URL: https://www.jeremyjordan.me/machine-learning-overview/.

[11] Randy Lao. *A Beginner's Guide to Machine Learning, July 3, 2018.* URL: https://towardsdatascience.com/a-beginners-guide-to-machine-learning-5d87d1b06111.

[12] Machine Learning Mastery. *Machine Learning Mastery, 2019.* URL: https://machinelearningmastery.com/start-here/#algorithms.

[13] Daffodil Software. *9 Applications of Machine Learning from Day-to-Day Life, July 31, 2017.* URL: https://medium.com/app-affairs/9-applications-of-machine-learning-from-day-to-day-life-112a47a429d0.

[14] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* http://www.deeplearningbook.org. MIT Press, 2016.

[15] Jordi Torres. *Deep Learning: Introducción práctica.* UPC, 2018. URL: https://torres.ai/artificial-intelligence-content/first-contact-deep-learning-practical-introduction-keras/.

[16] Jason Brownlee. *Difference Between a Batch and an Epoch in a Neural Network, July 20, 2018.* URL: https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/.

[17] Bottou L. *Large-Scale Machine Learning with Stochastic Gradient Descent.* In: Lechevallier Y., Saporta G. (eds) Proceedings of COMPSTAT ' 2010. Physica-Verlag HD, 2010. ISBN: 9783790826036.

[18] Bernard Marr. *What Are Artificial Neural Networks - A Simple Explanation For Absolutely Anyone, September 24, 2018.* URL: https://www.forbes.com/sites/bernardmarr/2018/09/24/what-are-artificial-neural-networks-a-simple-explanation-for-absolutely-anyone/#7ecd1ec51245.

[19] Wikipedia. *Artificial Neural Network, April 8, 2020.* URL: https://en.wikipedia.org/wiki/Artificial_neural_network.

[20] Jason Brownlee. *When to Use MLP, CNN, and RNN Neural Networks, July 23, 2018.* URL: https://machinelearningmastery.com/when-to-use-mlp-cnn-and-rnn-neural-networks/.

[21] Uniqtech. *Multilayer Perceptron (MLP) vs Convolutional Neural Network in Deep Learning, Dec 22, 2018*. URL: https://medium.com/data-science-bootcamp/multilayer-perceptron-mlp-vs-convolutional-neural-network-in-deep-learning-c890f487a8f1.

[22] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “U-Net: Convolutional Networks for Biomedical Image Segmentation”. In: *arxiv* (2015). URL: https://arxiv.org/pdf/1505.04597.pdf.

[23] Perelman School of Medicine at the University of Pennsylvania. *MICCAI BraTS 2017: Scope — Section for Biomedical Image Analysis (SBIA), Retrieved 2018-12-24*. URL: www.med.upenn.edu.

[24] Retrieved 2018-12-24. Perelman School of Medicine at the University of Pennsylvania. *SLIVER07 : Home*. URL: www.sliver07.orgu.

[25] Wikipedia. *Sørensen–Dice coefficient, April 2020*. URL: https://en.wikipedia.org/wiki/S%5C%C3%5C%B8rensen%5C%E2%5C%80%5C%93Dice_coefficien.

[26] Wikipedia. *F1 score, April 2020*. URL: https://en.wikipedia.org/wiki/F1_score.

[27] Normand Grégoire and Mikael Bouillot. *Hausdorff Distance between convex polygons, May 2020*. URL: http://cgm.cs.mcgill.ca/~godfried/teaching/cg-projects/98/normand/main.html.

[28] Jason Brownlee. *A Gentle Introduction to Cross-Entropy for Machine Learning, October 21, 2019*. URL: https://machinelearningmastery.com/cross-entropy-for-machine-learning/.

[29] Tim Hartley. *When Parallelism Gets Tricky: Accelerating Floyd-Steinberg on the Mali GPU, November 25, 2014*. URL: https://community.arm.com/developer/tools-software/graphics/b/blog/posts/when-parallelism-gets-tricky-accelerating-floyd-steinberg-on-the-mali-gpu.

[30] DeepAI. *Max Pooling, Mars 2020*. URL: https://deepai.org/machine-learning-glossary-and-terms/max-pooling.

[31] ETSETB TelecomBCN Universitat Politecnica de Catalunya. *Deep Learning for Artificial Intelligence, Autumn 2019*. URL: https://telecombcn-dl.github.io/dlai-2019/.