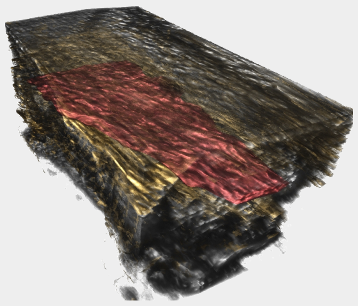
Computer Aided Medical Procedures
Prof. Dr. Nassir Navab



Dissertation

Hybrid Convolutional and Graph-Convolutional Framework for Segmentation Refinement

Roger David Soberanis Mukul

Fakultät für Informatik
Technische Universität München

Technische Universität München
Fakultät für Informatik
Lehrstuhl für Informatikanwendungen in der Medizin

Hybrid Convolutional and Graph-Convolutional Framework for Segmentation Refinement

Roger David Soberanis Mukul

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzende(r): Prof. Dr. Julia Schnabel

Prüfer der Dissertation: Prof. Nassir Navab, Ph.D.

Prof. Dr. Bjoern H. Menze

Die Dissertation wurde am 27.09.2021 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 05.02.2022 angenommen.

Roger David Soberanis Mukul
Hybrid Convolutional and Graph-Convolutional Framework for Segmentation Refinement
Dissertation, Version 0.0

Technische Universität München
Fakultät für Informatik
Lehrstuhl für Informatikanwendungen in der Medizin
Boltzmannstraße 3
85748 and Garching bei München

Abstract

Artificial neural networks (ANN) are learning models inspired by biological neurons. Although the definition of a single artificial neuron can be simple (they take a linear combination of their inputs and then apply a non-linear function), the potential of the networks for learning complex functions is clear, as they are considered universal approximators. For many years, image processing methods based on ANN relied on feature extraction processes tailored to the task at hand until recently, when the combination of large datasets and computational power boosted the implementation of deep convolutional neural networks (CNN). The CNNs can be considered learnable filters and can learn a feature extraction function directly from the data using backpropagation (the learning algorithm for ANN). As a result, it leads to end-to-end models for image-related tasks, like classification or segmentation. Since these models are composed of many layers of CNN and ANN, it is usual to refer to its definition and learning processes under the concept of deep learning. The recent advances in deep learning resulted in models with human-like performance in tasks like object recognition and semantic segmentation in real-work images. In the medical computer science community, it has inspired the development of architectures for medical image processing and computer-assisted diagnosis. However, in contrast with real-world image analysis, medical images present a challenging environment for convolutional neural networks. For example, in the medical image segmentation problem, the similarity between tissues and anatomical variations between patients can lead to errors in the predictions.

This work address refining the prediction of CNN models for the organ segmentation task. In this regard, the analysis of high confidence predictions has brought promising results in their ability to identify potentially correct predictions. We believe that the uncertainty analysis of CNN's can drive the refinement of the predicted segmentation. Still, we need to define the proper mechanism to exploit this information. We have found inspiration in recent works on semi-supervised graph learning. Graph convolutional networks (GCN) are an analogy to the CNN applied to graph data. With the information from the uncertainty analysis, it is possible to define a partially labeled graph representation of the image data and formulate the refinement process as a semi-supervised learning problem on the graph's nodes. Hence, we discuss a post-processing step that integrates the ideas from uncertainty analysis and semi-supervised GCNs. It can work independently from the CNN architecture without requiring additional information other than the model and its corresponding input. For the organ segmentation refinement problem, we show how this composed framework can help on the pancreas and spleen segmentation tasks. We believe that our work contributes to the community by discussing new directions for uncertainty analysis and graph learning models on medical data. We hope it can motivate the development of new graphs and uncertainty-oriented solutions that lead to better results that can help the medical community obtain an early and opportune diagnosis.

Zusammenfassung

Künstliche neuronale Netze (KNN) sind Lernmodelle, die von biologischen Neuronen inspiriert sind. Obwohl die Definition eines einzelnen künstlichen Neurons einfach sein kann (sie nehmen eine lineare Kombination ihrer Eingaben und wenden dann eine nichtlineare Funktion an), ist das Potenzial der Netzwerke zum Erlernen komplexer Funktionen ersichtlich, da sie als universelle Approximatoren gelten. Bildverarbeitungsmethoden auf Basis von KNN beruhten viele Jahre auf ein für die jeweilige Úufgabe zugeschnittenes Verfahren zur Merkmalsextraktion, Erst vor weniger Zeit als Ðowohl die Masse an Daten als Úuch die Rechenleistung zunahm, Ðahm die Implementierung Ðogenannter Deep Convolutional Neural Networks (CNN) zu. Die CNNs können als lernbare Filter betrachtet werden und können mithilfe von Backpropagation (dem Lernalgorithmus für KNN) direkt aus den Daten eine Merkmalsextraktionsfunktion lernen. Als Ergebnis führt es zu End-to-End-Modellen für bildbezogene Aufgaben wie Klassifikation oder Segmentierung. Da sich diese Modelle aus vielen Schichten von CNN und KNN zusammensetzen, ist es üblich, ihre Definition und Lernprozesse unter dem Begriff Deep Learning zu zusammenzufassen. Die jüngsten Fortschritte in Deep Learning führten zu Modellen mit menschenähnlicher Leistung bei Aufgaben wie Objekterkennung und semantischer Segmentierung in realen Bildern. In der medizinischen Informatik hat es die Entwicklung von Architekturen für die medizinische Bildverarbeitung und computergestützte Diagnose inspiriert. Im Gegensatz zur realen Bildanalyse stellen medizinische Bilder jedoch eine schwierige Umgebung für neuronale Faltungsnetze dar. Beispielsweise können beim medizinischen Bildsegmentierungsproblem die Ähnlichkeit zwischen Geweben und anatomische Variationen zwischen Patienten zu Fehlern in den Vorhersagen führen.

In dieser Arbeit beschäftigen wir uns mit der Verfeinerung der Vorhersage von CNN-Modellen für die Organsegmentierungsaufgabe. In dieser Hinsicht hat die Analyse von Vorhersagen mit hoher Konfidenz vielversprechende Ergebnisse hinsichtlich ihrer Fähigkeit geliefert, potenziell korrekte Vorhersagen zu identifizieren. Wir glauben, dass die Unsicherheitsanalyse von CNNs die Verfeinerung der vorhergesagten Segmentierung vorantreiben kann. Dennoch müssen wir den passenden Mechanismus definieren, um diese Informationen zu nutzen. Wir haben uns dabei von neueren Arbeiten zum semi-überwachten Graphenlernen inspirieren lassen. Graph Convolutional Networks (GCN) sind eine Analogie zum CNN, das auf Graphdaten angewendet wird. Mit den Informationen aus der Unsicherheitsanalyse können wir eine teilweise beschriftete Graphendarstellung der Bilddaten definieren und den Verfeinerungsprozess als semi-überwachtes Lernproblem an den Knoten des Graphen formulieren. Daher schlagen wir eine unbestimmtheitsgetriebene GCN-Verfeinerung als Nachbearbeitungsschritt vor, die unabhängig von der CNN-Architektur arbeiten kann, ohne zusätzliche Informationen außer dem Modus und der entsprechenden Eingabe zu benötigen. Wir zielen bei der Aufgabe zur Verfeinerung der Organsegmentierung auf CT-Daten und validieren unseren Vorschlag anhand der Segmentierungsaufgaben für Bauchspeicheldrüse und Milz. Die Ergebnisse zeigen eine bessere Leistung gegenüber dem Ansatz zur Verfeinerung von Zufallsfeldern dem aktuellen Stand der Technik. Wir glauben, dass unsere Arbeit einen Beitrag zur Gemeinschaft leistet, indem wir eine neue Anwendung der Unsicherheitsanalyse und des grafischen Lernens auf medizinischen Daten präsentieren. Wir hoffen, dass es die Entwicklung neuer Diagramme und unsicherheitsorientierter Lösungen motivieren kann, die zu besseren Ergebnissen führen, und die der medizinischen Gemeinschaft helfen können, eine frühe und günstige Diagnose zu erhalten.

Acknowledgments

I want to express my gratitude to the examiner committee, Prof. Julia Schnabel and Prof. Bjorn Menze, for taking the time to review and evaluate my dissertation. To Prof. Nassir Navab and Prof. Shadi Albarqouni, for all the mentoring and support during the doctorate. To all the CAMPers and friends I met during these years. To my mother, Leny Mukul, and my sisters, Catalina Soberanis, and Sara Soberanis, for all the patience. I want to thank Misericordia Herrero, her sons, and Martin Enriquez for sharing housing with me. They became a second family for me. Finally, I want to thank the Consejo Nacional de Ciencia y Tecnologia, (Conacyt Mexico) for supporting four years of my doctorate.

Contents

Part I

Deep Learning

1 Introduction

Artificial intelligence (AI) has the potential to support people in different aspects of their life. It can be seen in recent history when computational learning models and deep learning led to intelligent voice assistants, data analysis for consumer-tailored marketing, and the design of consumer-level autonomous driving systems. The medical domain is an area where AI can make a major impact. Computer-assisted diagnosis can help physicians to bring a rapid, tailored, and opportune diagnosis to patients. AI can be employed in different steps of the medical analysis pipeline. For example, the identification of anatomies in medical images. This process can be understood as an image segmentation problem. Medical image segmentation consists in the division of the image into two or more regions of interest. An example of it is the division based on the organ boundaries in computer tomography (CT) images (Fig. 1.1). Segmentation is an important pre-processing step in many computer-aided methods, including computer-aided diagnosis, planning, and navigation. However, the process can be time-consuming for humans, and in the case of medical data, it requires a level of medical knowledge. For instance, in organ segmentation, the process requires identifying the organ boundaries in each slice of a CT volume. In this task, each volume can be composed of between 200 and 400 slices. Similarly, the person in charge of this task requires previous knowledge in anatomy, to precisely delineate each organ of interest. All this makes a challenging task, especially, if multiple patients should be analyzed. For these reasons, it has been of interest to define automatic or semi-automatic methods for performing this task, with the minimum human intervention.

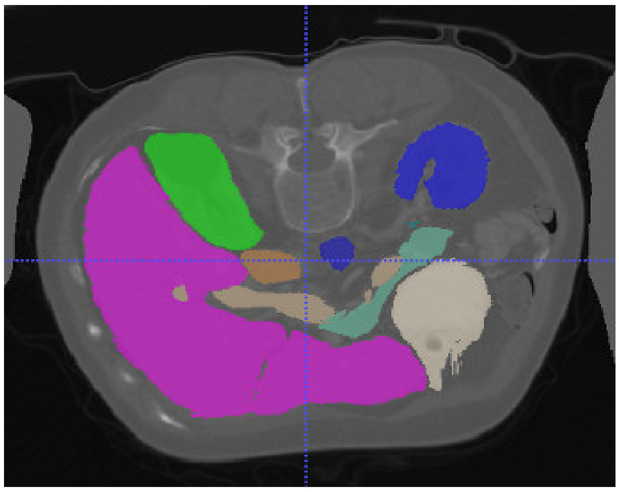


Fig. 1.1. The image segmentation problem consists in separate the pixels of the image into regions, according to one specific criterion. In the example, the image is segmented according to the organ to which each pixel belongs. The different colors define the different regions or organs. Image from the beyond the cranial vault dataset (https://www.synapse.org/#!Synapse:syn3193805/wiki/)

Different models have been proposed employing machine learning methods. Supervised models use a set of training examples for learning a model for the given task. Such examples are composed of a set of inputs together with their corresponding labels. The objective is to use the training set to approximate a mapping from the input to the label space. The models should work for any input, even if they are not part of the training set. For many years, machine learning models rely on feature extraction, to find the most suitable representation of the data for the task at hand, until the development of deep learning.

In recent years, the machine learning community has adopted deep learning as the standard choice for solving different image and non-image-related problems. Deep learning is composed of models based on artificial neural networks (ANN), specifically, convolutional neural networks (CNN). These CNN integrate several convolutional and fully connected layers and allow both learning the given task together with efficient feature extraction, all in an end-to-end way. Initial models were proposed for classification, however, researchers have extended the architectures for localization and segmentation tasks. Modern architectures are motivated by computer vision problems, including object classification [42, 74], localization [50], and semantic segmentation [47] in real-world images.

1.1 Segmentation and Refinement

These advances in computer vision have motivated the use of CNN for solving medical-related problems. Many medical-related problems share a common objective with computer vision tasks. For instance, it would be of interest to classify and localize anomalies in skin images or endoscopy sequences. The segmentation of anatomical regions can also obtain benefits from CNN models. However, the characteristics of medical images can represent a challenge to learning models. In contrast with real-work images, where objects boundaries are well defined, medical images present low contrast in the boundaries and high similarity between the tissue of the different anatomies (Fig. 1.2). Also, it is known that deep learning models are data-hungry and require thousands of labeled samples to train. As we discussed before, the manual segmentation of medical data is not a trivial task and requires a certain level of knowledge. It can limit the amount of available data for model training. All this can lead to errors in the segmentation, resulting in missing regions (false negatives) or the segmentation of incorrect sections (false positives). All this makes the definition of refinement methods an interesting topic. In this domain, conditional random fields (CRF) [41] is one of the standard methods for segmentation refinement.

CRF employs the model's prediction, together with pixel intensity and positional relationships to perform the refinement. The method is employed as a post-processing step for each individual image and does not require ground truth (labeled data) to operate. In this sense, additional information, like an indicator of prediction correctness, can bring a useful input for a refinement strategy. However, this is information is not currently considered, and it is not directly available. It brings us to two main questions: How can we estimate the correctness of the model prediction? and how can we use this information in a refinement strategy? To answer the first question, we rely on uncertainty analysis, and for the second question, we found an answer on graph learning models, and we define the refinement process as a semi-supervised node classification problem.

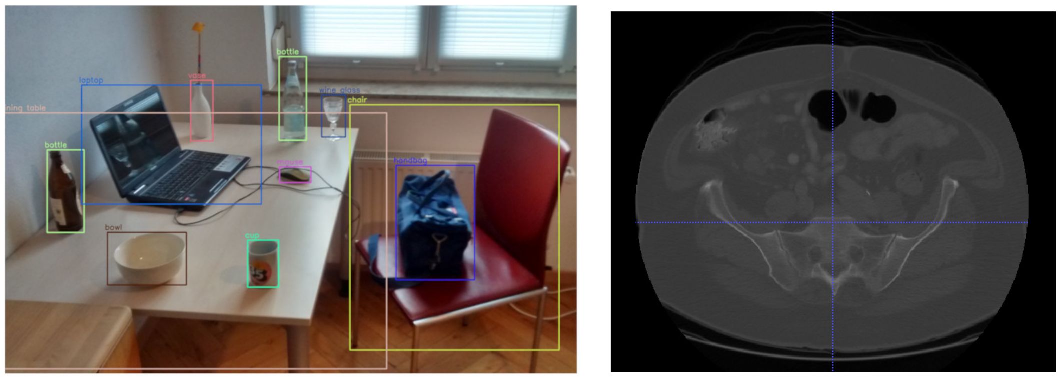


Fig. 1.2. A real-world image (left) compared with a slice of a CT image (right). Real-world images present a clear definition of objects, compared with medical images. Similarly, for most of the real-world images, non-trained annotators can help in the generation of ground truth for model training. As an example, it would be easy to find and define the contour of the object laptop in the left image, but, would it be possible for anyone to indicate if the pancreas is present in the right image? Left image: results from the YOLO object detector (source: Wikipedia). Right image: A slice from the medical image decathlon dataset (http://medicaldecathlon.com/).

1.2 Uncertainty and Graph Convolutional Models in Segmentation Refinement

Given a trained CNN, an input image, and their corresponding prediction, the proposal first performs the uncertainty analysis of the CNN for the given input. Our objective is to propose a method that works on inference time, as a post-processing step. An important point we considered is that in inference time, the only information available is the model, the prediction, and the input. In this sense, we assume that no ground truth information is available. To obtain the uncertainty information under this scenario, we rely on the Monte Carlo dropout method [26]. This method allows obtaining the model's confidence and uncertainty using its dropout layers. It enables applying it to a wide range of models, without retraining, and with no additional information than the previously mentioned. We argue that the model uncertainty includes important information about the correctness of its prediction [20, 51]. Low uncertainty regions might indicate a potentially correct prediction, while high uncertainty outputs might be potentially incorrect. Including this information in the refinement process should be beneficial for the correction of the potentially misclassified outputs.

Once we have the model's uncertainty, we use this information to formulate a partially label graph. The nodes of the graph are defined by pixels (or voxels) of the images. Connections are done based on local neighborhoods and randomly chosen long-range nodes. The uncertainty is employed to define the labels of the node. High uncertainty nodes take the CNN prediction as label, while low uncertainty nodes are considered to be unlabeled. This defines a partially labeled graph, where we can formulate a semi-supervised node classification problem [40]. We can then use state of art graph convolutional networks to train a model using the labeled part of the graph. Finally, the refinement is done by evaluating the graph on the trained graph convolution model.

The proposal is tested in two segmentation problems: Pancreas and Spleen segmentation. We trained a 2D U-Net model on each task, and perform the refinement of a set of CT volumes. Experiments show that our model can outperform the classical conditional random field refinement method on both segmentation tasks.

The document is structured in two parts. In the first part, we describe the basics of supervised learning, neural networks, and deep learning. We introduce the concept of supervised learning and artificial neural networks in section 2. It follows with an introduction to the convolutional neural networks and their main building blocks considering some of the architectures proposed for image classification problems (section 3). Here, we discuss the extension of CNN to the segmentation task and the definition of fully convolutional neural networks in section. We review some basic architectures, like the U-Shaped Models, and present some examples of their application in the medical field. We present the main idea of Monte Carlo Dropout for uncertainty analysis, and we conclude this part, with a discussion of the spectral graph convolutional neural networks in section 4.

The second part describes our method and discusses the formulation and the construction of the partially label graph, based on uncertainty information (section 5). We also present our experiments and results in the segmentation of two different organs: the pancreas and spleen. We perform a sensitivity analysis of our proposal (section 6) and we finalize with some final remarks and possible future directions for graph-based refinement strategies in section 7.

2 Supervised Learning and Artificial Neural Networks

Machine learning (ML) involves the processes and algorithms that allow a computer program to learn from experience. Broadly speaking, ML studies the processes that enable a computer to improve its performance in a given task, as its experience increases [49]. The literature gives different learning processes. In this thesis, we will focus on learning neural networks in a supervised learning scheme. In this learning scheme, experience comes from a set of training examples, each composed of input and output pairs. The kind of tasks attended by this learning scheme look for finding a mapping between the input values and desired outputs. One example is the image classification problem. Experience is expressed as the difference between the program's predictions and the real outputs. The computer earns experience by "training" with different input-output examples. We will dive into more detail on supervised learning in section 2.1.

Artificial neural networks (ANN) are a type of learning model inspired by biological neural networks. Their formulation comprehends a linear combination of their input values, followed by a non-linear activation function (see section 2.2). During the past few years, we have witnessed the adoption of convolutional neural networks (CNN) and deep learning for solving a wide variety of computational learning problems. In the computer vision community, CNN's are the preferred choice to address tasks like object classification, object localization, and semantic segmentation. Deep CNNs are capable of obtaining high-performance results, which has led to a growing academic and technical literature, presenting different architectures, learning techniques, and studies, together with their potential areas of applications.

The high performance of CNNs has inspired their adoption in different fields complementary to computer vision. The medical computer science community is not agnostic to these tendencies. Researchers have investigated the application of CNNs in diverse medical problems involving computer-assisted diagnosis and navigation, prediction of disease progression, classification, localization, and segmentation of medical structures, to name some examples. Many of these jobs rely on CNNs for medical image analysis. This requires the models to work in a challenging environment that involves fewer annotated datasets, the high similarity between background and foreground, and multiple imaging modalities (for example, MRI, CT, endoscopic sequences, ultra-sound, and x-ray, to cite some image modalities). Despite these difficulties, CNN's have achieved top results in many medical tasks, gaining high popularity in this domain.

In this chapter, we introduce some of the main ideas behind machine learning and deep learning. We present some concepts and notations used in the following chapters of the document. Our work has the main focus on the analysis of medical images. Hence, even

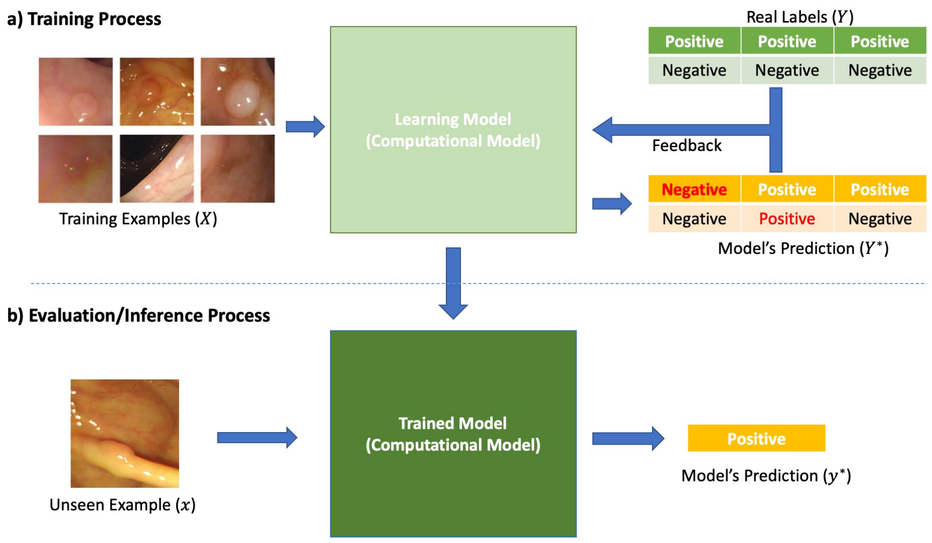

a) Training Process
Training Examples (X)
Learning Model (Computational Model)
Real Labels (Y)
Positive Positive Positive
Negative Negative Negative
Feedback
Negative Positive Positive
Negative Positive Negative
Model's Prediction (Y^*)
b) Evaluation/Inference Process
Unseen Example (x)
Trained Model (Computational Model)
Positive
Model's Prediction (y^*)

Fig. 2.1. The endoscopy image classification task as an example of supervised learning. The model's input corresponds to a set of endoscopy images where some of them contain neoplasia. The top row of the Training Examples images in a) correspond to neoplasia examples (positive class), while the bottom row contains normal tissue (negative class). a) In the training process, the training set $\{X, Y\}$ is used to improve the model's predictions Y^*. b) During the evaluation, the model should predict the correct label even for examples not in X.

though the material presented is general to different data and learning problems, the discussion centers on learning from image information.

2.1 Supervised Learning

In the previous section, we mentioned we will focus on neural networks and supervised learning. We will start our discussion with the last concept since most of the methods that will be presented follow a supervised learning strategy. As presented by [49], we can define a learning problem in terms of the task at hand, the source of experience, and the performance metric. Considering the supervised learning scheme, a computational model improves its performance in a particular task by earning experience from a set of training examples composed of input data and output targets [49, 56, 67]. We can define the main task of supervised learning, as finding a mapping between the input's and output's distributions, using the available training data (observed samples). Such mapping should allow obtaining the correct output even for input data not in the training set (unobserved samples). The main assumption here, however, is that both, observed and unobserved inputs, come from the same distribution.

For example, consider the following image classification problem. Suppose we have a collection of endoscopic images. Some of the images contain healthy tissue. However, a percentage of this dataset includes abnormal tissue growths, or neoplasias (see Fig. 2.1). We can define this task as the classification of images in healthy tissue (negative label) or neoplasia (positive label). The input data consist of a numerical parametrization of the image. The objective is to find a mapping between the images and the two available classes (the learning model in Fig. 2.1a). We are expecting that the learned model can also estimate the output for other endoscopic images, not in the training set (Fig. 2.1b). Similarly, we expect that all the input images for the model come from endoscopic studies (same distribution as the training set). This example incorporates all the elements present in a supervised learning problem.

We still need to formally define the supervised learning problem, and we have not mentioned, yet, anything in relation to the learning models. Let's first define the mathematical notation for the components involved in this learning scheme. We will discuss about the learning model in the upcoming section. Formally, consider $x \in \Omega$ to be an observation of a phenomenon or event, with Ω the set of all possible observations. The dimensionality of x depends on the nature of the events. For our image classification example, input RGB color images can be represented as three-dimensional arrays, and hence $x \in \Omega \subset \mathbb{R}^{r \times c \times 3}$, for images with dimensions $r \times c$ and three color channels.

Each $x \in \Omega$ is associated to a given target or label $y \in \mathcal{Y}$. This can represent each object category (neoplasia positive or negative) in our image classification example. The relationship between x and y can be expressed by an unknown function $h : \Omega \to \mathcal{Y}$:

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

The objective then, is to find an approximation of h. Considering a training set $S_t = \{(x_i, y_i) | x_i \in \Omega, y_i \in \mathcal{Y}, i = 1, ..., N\}$, the supervised learning task consist in using S_t to find a function $f(x; \theta) : \Omega \to \mathcal{Y}$ such that

$$f(x; \theta) \approx h(x), \forall x \in \Omega \tag{2.2}$$

With θ the model's parameters of f. This means that, even if the only information available is contained in S_t, the approximation $f(\cdot; \theta)$ must properly generalize to all other "unseen" data points $x_u \in \Omega - S_t$.

In addition to the training set S_t, we can distinguish three main components in the supervised learning task: the function's representation f or learning model; the learning method that allows the model to improve its performance from S_t; and the performance metric, that evaluates how good is the approximation f.

There exist different computational algorithms proposed for learning a model from a training set. As examples, we can mention support vector machines, Adaboost, or Random Forests [12, 43, 49]. However, the discussion of this document will concentrate on artificial neural networks, since this kind of model currently leads the state of the art in different image processing tasks. Similarly, the methodology proposed in this thesis is inspired by different advances in the convolutional neural networks field.

As a final remark, we can distinguish between two kinds of supervised learning problems, considering the nature of the label space $\mathcal{Y}$. Discrete values of $y \in \mathcal{Y}$ define a classification problem Our neoplasia detection example corresponds to this kind of learning task, with $|\mathcal{Y} = 2|$ (binary classification). For continuous values of $y \in \mathcal{Y}$, we say the learning task corresponds to a regression job. For example, if we define the task as estimating the size of the neoplasia, we will be working with a regression problem. Most of the tasks addressed by this thesis correspond to classification employing neural networks. In the upcoming sections, we will discuss the basics of supervised classification with artificial neural networks.

2.2 Artificial Neural Networks and Classification

So far, we have presented the concepts involved in the supervised learning process. However, we have not discussed the description of the learning model or the method that allows accumulating experience for improving its performance. In this section, we review the artificial neural networks (ANN) as a learning model and discuss the methods for training them. Our first focus is on fully connected neural networks composed of multilayered perceptron networks (MLP). With this, we pave the way for discussing, in the following sections, one of the more successful learning models for image processing, the convolutional neural networks.

Let's consider a supervised binary classification problem, with training set $S_t = \{(x_i, y_i) | x \in \Omega \subset \mathbb{R}^m, y \in \{-1, 1\}, i = 1, \ldots, N\}$. We can start defining the formulation of a single neuron. Note that the input to our neuron is composed of a vector of size m. Artificial Neurons have a biological inspiration. They represent an analogy to the complex interconnections of biological neurons. Broadly, an artificial neuron takes as input a set of real-valued numbers (or a real-valued vector), applies a transformation, and outputs a single real value. In this sense, we can define the general representation of an artificial neuron as a function $o(z; \omega, b)$:

$$o(z; \omega, b) = \sigma(\omega^T z + b) = \sigma(w_1 z_1 + w_2 z_2 + \cdots + w_k z_k + b) \tag{2.3}$$

where the parameter $\omega \in \mathbb{R}^k$ is a vector defining the weights for each entry of the input vector $z \in \mathbb{R}^k$ and $b \in \mathbb{R}$ defines a bias. With this, the neuron first applies a linear transformation to the input vector. This transformation is followed by the evaluation of the activation function σ, which leads to the neuron's output. It is graphically expressed by Fig. 2.2a. Note that the neuron's input z is not necessary given by the input data. It is because the input for any particular neuron may come from the input vector x or other neurons in the network, depending on the position of the neuron in the network. Consequently, the output of a particular neuron can feed other units in the ANN or can serve as the final model's output. Neurons that take their input from other nodes in the network are called hidden units (or hidden neurons). On the other hand, neurons that define the model's output are known as output units.

ANNs are composed of a set of interconnected neurons of the form of Eq. 2.3. Fig. 2.2b shows a typical architecture of a feed-forward fully connected neural network. They can be represented by acyclic directed graphs. Neurons inside the network are organized in layers, and layers are classified as input, hidden, and output layers (see Fig. 2.2). Neurons in the input layer receive our input data. The output layer gives us the model's prediction (or class estimation). We do not interact directly with hidden layers, as they take inputs from previous layers in the network at the time that their outputs feed the neurons in the next layers. Each neuron have their own set of weights and bias. The input vector for a particular neuron is given by the output of all the neurons in the previous layer. The output layer can have any

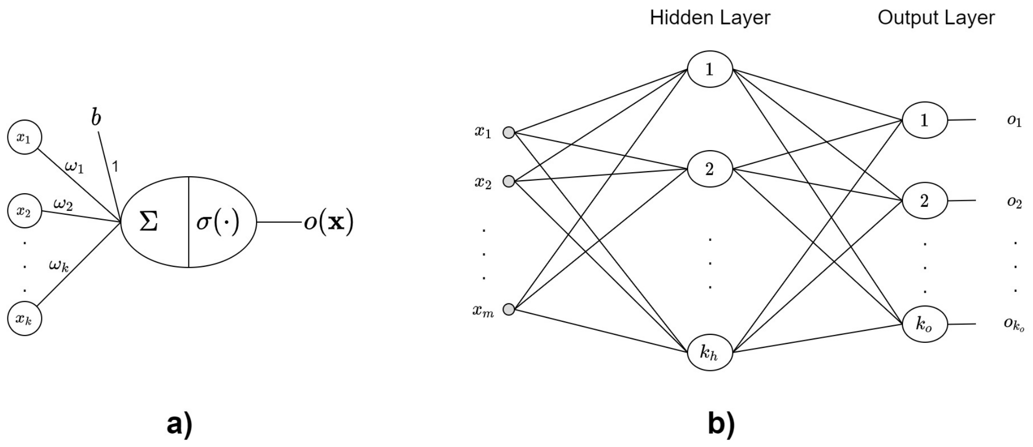


Fig. 2.2. A common graphical representation for artificial neurons and artificial neural networks. a) A single neuron, takes the weighted addition of the elements of the input vector $\mathbf{x} = [x_1, \dots x_k]$ considering the weights $\omega = [\omega_1, \dots, \omega_k]$, adds a bias factor b, and apply a function $\sigma(\cdot)$. In the image, the bias is represented as an additional input with weight 1. b) A two-layered feed forward neural network represented as an acyclic graph. An ANN is composed by the connection of multiple neurons, organized in layers. Each node in the graph represents a single neuron. The ANN contains a hidden layer with k_h neurons and a output layer with k_o units.

number of neurons, allowing us to model multi-class classification problems by including one output neuron for each available class.

We can define a mathematical formulation for the two-layered neural network represented in Fig. 2.2b. We consider the $m-$dimensional input vector $\mathbf{x} \in \mathbb{R}$. We will also consider the single hidden layer to be composed of k_h neurons and an output layer with k_o units. Then, we can define the output of a given hidden neuron $f_{h,i}$ as:

$$f_{h,i}(\mathbf{x}) = \sigma(\omega_{h,i}^T \mathbf{x} + b_{h,i}) \tag{2.4}$$

for $i = 1, \dots, k_h$, and $\omega^{h,i}$ the neuron's $m-$dimensional real-valued weight vector with real-valued bias $b_{h,i}$. Evaluating all the neurons in the hidden layer will result in a new vector $\mathbf{f_h} = [f_{h,1}, \dots, f_{h,k_h}]^T$. We can interpret this as a transformation of the input data from $\mathbb{R}^m$ to $\mathbb{R}^{k_h}$. Now, each output unit $f_{o,j}$ will follow a similar path, leading to the expression:

$$f_{o,j}(\mathbf{f_h}) = \sigma(\omega_{o,j}^T \mathbf{f_h} + b_{o,j}) \tag{2.5}$$

where $j = 1, \dots, k_o$ iterates over the output units. The weights vectors $\omega_{o,j}$ are k_h-dimensional, and the input to each neuron corresponds to the outputs of the previous layer. The overall output vector of the entire network is given by $f(\mathbf{x}; \theta) = [f_{o,1}, \dots, f_{o,k_o}]^T$. We set θ to contain all the weights and biases of the network.

An alternative notation can be defined if we put together all the weighting vectors of a given layer into a matrix form. For the hidden layer, consider the $k_h \times m$ matrix $\mathbf{W_h}$, containing the weighting vector $\omega_{h,i}$ in its row i for $i \in [1, k_h]$. Also, we can collect all the biases into a single vector $\mathbf{b}_h = [b_{h,1}, \dots, b_{h,k_h}]^T$. With this, we can define the output of the hidden layer as:

$$f_h(\mathbf{x}) = \sigma(\mathbf{W_h}\mathbf{x} + \mathbf{b_h}) \tag{2.6}$$

In this case, the activation function σ is applied in an element-wise way to the vector $\mathbf{W_h x}+\mathbf{b_h}$. Defining the $k_o \times k_h$ weighting matrix $\mathbf{W_o}$ and k_o-dimensional bias vector $\mathbf{b}_o$ similarly, the output layer $f_o(\cdot)$ follows the same formulation. Then, we can write the final output of the two-layered ANN as:

$$f(\mathbf{x}, \theta) = f_o(f_h(\mathbf{x})) = \sigma(\mathbf{W_o}\sigma(\mathbf{W_h x} + \mathbf{b_h}) + \mathbf{b_o}) \tag{2.7}$$

In general, all the layers of an ANN follows the formulation of Eq. 2.6. A neural network or arbitrary number of layers can be represented by successive compositions of function of the form of Eq. 2.6.

2.2.1 Activation Function

We have defined an artificial neuron as a family of functions that takes the input vector $\mathbf{x}$, performs the weighted addition of the input's entries, and applies an activation function to obtain the neuron's output. We also discussed how these neurons can be connected to obtain an ANN. In short, considering a particular layer in the ANN containing k_l neurons, the equation that defines any layer in the network is given by:

$$\mathbf{f_l} = \sigma(\mathbf{W_l f_{l-1}} + \mathbf{b_l}) \tag{2.8}$$

this is the same formulation as in Eq. 2.6, however, instead of representing the hidden layer, we look for describing a general definition of an ANN layer. In this regard, now $\mathbf{W_l}$ represents a $k_l \times k_{l-1}$ weighting matrix, and $\mathbf{b_l}$ a k_l-dimensional bias vector. The $k_{l-1}-$dimensional vector $\mathbf{f_{l-1}}$ is the output of the previous layer, with $0 < l < L$. The ANN has a total of L layers and $\mathbf{f_0} = \mathbf{x}$.

In this section we discuss the activation function $\sigma(z)$ of the artificial neuron. We can start defining the activation function to be the sign function (see Fig. 2.3a):

$$\sigma(z) = \text{sign}(z) = \begin{cases} 1 & \text{if } z > 0 \\ -1 & \text{otherwise} \end{cases} \tag{2.9}$$

which leads to the definition of the following individual artificial neuron:

$$o(\mathbf{z}; \omega, b) = \text{sign}(\omega^T \mathbf{z} + b) \tag{2.10}$$

The neuron of Eq. 2.10 defines the well known perceptron unit. ANN composed by perceptron neurons are referred as multi-layered perceptron networks (MLP), where each layer follow the analogous formulation:

$$\mathbf{f_l} = \text{sign}(\mathbf{W_l f_{l-1}} + \mathbf{b_l}) \tag{2.11}$$

We are assuming that the activation function perform a element-wise evaluation when their input is a vector. Note that defining sign$(\mathbf{z})$ as the activation function, the neuron performs a non-linear transformation over the input vector. The linear operation $\omega^T x + b$ in equation

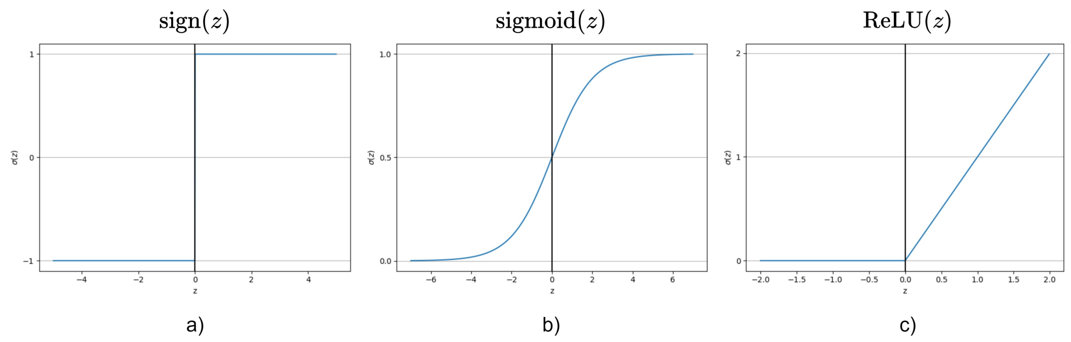


Fig. 2.3. Three commonly used activation functions: a) The sign(z) activation function employed in the original perceptron neuron; b) the sigmoid(z) function, a differentiable activation function; c) the rectified linear unit ReLU(z) activation function. This function is widely employed in current convolutional neural networks.

(2.10) defines a hyperplane that divides the input space. In this sense, the perceptron behaves similarly to a thresholding rule over $\omega^T x$. It will assign the value of 1 or -1 if $\omega^T x$ is above or below $-b$, respectively.

Perceptron units can be a suitable classifier for binary classification problems where the labels define two distinguishable clusters in the input space. Datasets with this property are said to be linearly separable. The learning task consists of finding the parameters ω and b that properly split the input space, according to the available classes. Unfortunately, real-world problems are rarely linearly separable. A single perceptron can represent linearly separable data problems like Boolean functions. It is possible to find the correct weight values to represent basic Boolean models like the AND, and OR functions. However, coming to more complex representations like the XOR function requires an ensemble of neurons to be represented. More complex real-word problems will require a multi-layered representation. There exist methods to learn the weights of a signed perceptron neuron, but given the limitations in the expressivity of a single neuron, it is of most interest methods for learning networks of many neurons. We understand by expressivity, the capability of a model to represent different functions. At this point, there is a drawback with the sign activation function. The main learning method for learning ANN weights requires the computation of the derivatives of the activation function. The sign function presents a discontinuity in the point $z = 0$, making it not suitable for the learning algorithm. For this reason, the sign activation function is rarely used in modern architectures.

Different alternatives have been proposed in the literature. The sigmoid units [12, 49] are an alternative to the sign activation function. The sigmoid follows the definition:

$$\sigma(z) = \frac{1}{1+e^{-z}} \tag{2.12}$$

The graph of this function is presented in Fig. 2.3b. The sigmoid is a continuous non-linear function that remembers a smooth version of a step function. It allows computing its derivative, which will be necessary for training purposes. Similar to the sign activation, the sigmoid can be used to separate the input space into two groups. It makes it ideal for classification problems. Indeed, a single neuron would have the following definition:

$$o(\mathbf{z}, \omega, b) = \frac{1}{1 + \exp\left(-\omega^T \mathbf{z} - b\right)} \tag{2.13}$$

which resembles a logistic linear regression model. In contrast to the sign function, the sigmoid has values bouned between 0 and 1. It presents an asymptotic behavior towards these values when the input grows to the negative or positive sides, respectively. Similarly, we can use the value of the sigmoid as an indicator of the model's confidence, since points with a high distance from the hyper-plane $\omega\mathbf{z} + b$ will presents outputs close to 0 or 1. Values close to the plane will result in the sigmoid giving a value close to 0.5. The asymptotic behavior of this function can represent a disadvantage. The learning algorithm for neural networks (backpropagation) employs the magnitude of the gradient to iteratively update the model's weights. Having an almost constant behavior in the positive and negative extremes of the activation function can lead to small gradients. This saturation problem can also result in small updates for the weights at some point in the training process [48]. Initial ANNs were composed of sigmoid units. In recent ANN models, the use of sigmoid units is reserved for the output layers in binary classification tasks. Modern architectures employ a different function for the hidden layers: the rectified linear unit.

The Rectified Liners Unit (ReLU) offers an alternative as activation function [42]. This is expressed in Eq. (2.14) and has become almost a standard choice, specially for deep convolutional models.

$$\sigma(z) = \text{ReLU(z)} = \max(0, z) \tag{2.14}$$

The function max(0, z) simply outputs the biggest value between o and z. As we can see in Fig. 2.3c, the ReLU activation function behaves as a linear function for positive values of its input but collapses to zero for negative values. The growing nature of the linear part can provide strong gradients for training. This function does not have the saturation problem as in the sigmoid units. Similarly, it presents a constant gradient, due to its linear behavior. These translate into an improved training time according to [42]. Additionally, the constant zero value for negative inputs can potentially set the activation of some neurons to zero. This can have both, benefits and drawbacks. On the first side, having zero output for some units in the network can contribute to the efficiency of the network. On the other hand, this can also lead some neurons to never activate, giving a constant output of zero, and avoiding those units from training, due to the also constant zero value of theirs gradients. To avoid this problem, some variations of the ReLU, like the leaky ReLU have been proposed [48]. Instead of having a constant zero value for negative inputs, leaky ReLU assigns a small fraction of their input. This can prevent some neurons from never activate, but also, can reduce gains in efficiency that a zero activation can offer, representing a trade-off to consider in design choices. As we have mentioned before, ReLU units are a popular choice for hidden layers of the network. At this point, it is worth mentioning that we will extend the definition of multilayered perceptron (MLP) as an umbrella term for any fully connected ANN of the form of Eq 2.8, no matter the choice of the activation function.

2.2.2 Loss Function and Model Training

We have fully described the ANN as a learning model for supervised tasks. Now we can introduce the mechanism employed for learning from the training set. We can consider the ANN as a parametrized family of functions defined by their set of weights $\mathbf{W}_l$ and biases $\mathbf{b}_l$, with $l \in [0, 1]$. The training process consists in finding the proper values for the parameters of the model. The backpropagation algorithm is a learning method still used in modern architectures. It is based on the gradient descent algorithm that updates the model parameters base on the gradient of an error metric, known as loss function. The learning task is formulated as an optimization problem looking to minimize the loss function by searching in the parameters space. Here is the importance of having a model composed of differentiable activation functions. In a general definition, the loss function takes the model's prediction and the ground truth or label for the corresponding input data and outputs the error in the model's prediction. For a model $f(\mathbf{x}; \theta)$, and training example $(\mathbf{x}, y)$ composed of the pair input vector and label, the loss function gives the error $\mathcal{E}_\theta$ of the model when predicting the label for $\mathbf{x}$:

$$\mathcal{E}_\theta = \mathcal{L}(f(\mathbf{x}, \theta), y) \tag{2.15}$$

The loss function outputs a minimum value (usually zero) for a perfect prediction of the input's label. Its value increases according to the difference between the prediction and the ground truth label. Backpropagation is a widely used algorithm for training neural networks. Given a training set S and a model $f(\mathbf{x}, \theta)$, the objective is find the model's parameters that minimize the prediction error:

$$\theta_{\text{optimal}} = \arg\min_{\theta} \mathcal{E}_\theta \tag{2.16}$$

with $(\mathbf{x}, y) \in S$. An ideal model must have a value of $\mathcal{L}(f(\mathbf{x}, \theta), y) = 0$, not only for each $(\mathbf{x}, y) \in S$, but also for all $(\mathbf{x}, y)$, $\mathbf{x} \in \Omega$, $y = h(\mathbf{x})$. Remember that $h(\mathbf{x})$ is the hypothetical unknown function that relates inputs with labels, and that we want to approximate.

As we mentioned, a gradient descent-based method can be employed to learn the weights of the model. For a differentiable model with a set of parameters $\theta = \{\theta_i : i \in [0, n_p]\}$, the gradient descent method updates each parameter θ_i with a fraction of the gradient of the loss function with respect to the corresponding parameter θ_i. Note that, for neural networks, the set θ contains all the weights and bias, of all the neurons in the model. The algorithm iteratively updates each parameter according to the rule:

$$\theta_i = \theta_i - \alpha \frac{\partial \mathcal{L}}{\partial \theta_i} \tag{2.17}$$

in Eq. 2.17, α is hyperparameter (a parameter of the learning method) known as learning rate, that defines the fraction of the gradient employed to update the weights. Gradient descent is an optimization method that can find a global minimum value for convex functions. For non-convex loss functions, the method can optimize up to a local minimum value. Unfortunately, this is the case of most complex models, like ANN. The non-linear activation function causes the loss function to be non-convex. Despite this drawback, gradient-based optimization methods can still find good-enough weights for the different learning tasks.

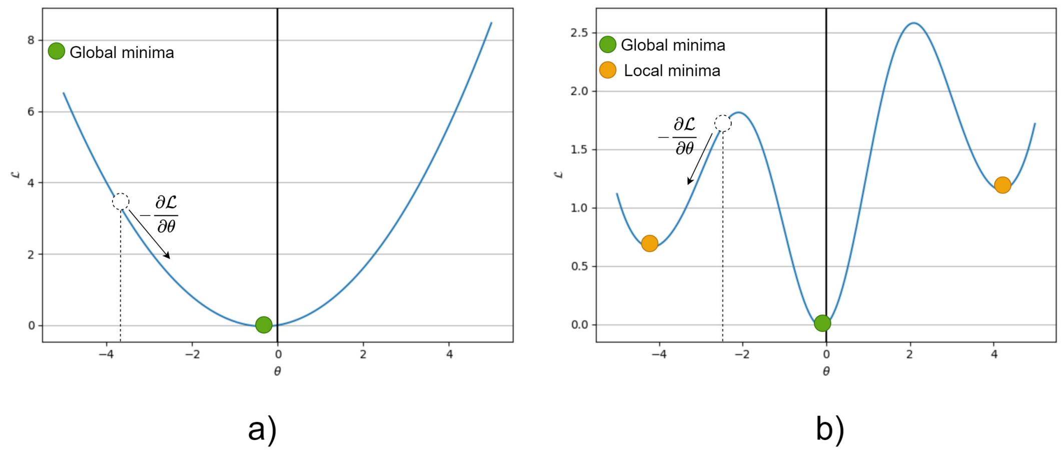


Fig. 2.4. Toy examples of a) a convex and b) non-convex function.The dashed circle represents the parameter values of the current iteration of a gradient-based learning algorithm. The algorithm searches the parameter space moving in the opposite direction of the gradient (represented by an arrow). In a convex function, the method can reach a global minimum. However, in no-convex functions, the method can guarantee convergence up to a local minimum.

Given that an ANN is composed of a successive composition of functions of form of Eq. 2.3, we can employ the chain rule to compute the derivative of the loss function with respect to a particular parameter. However, it is necessary to consider the differences between outputs units and hidden units. Output units give the model´s prediction, and hence, they have direct interaction with the loss function. The gradient for an output unit j of the form $\sigma(\omega_j^T \mathbf{z}_j + b_j)$ can be computed as:

$$\frac{\partial \mathcal{L}}{\partial \theta_{ji}} = \frac{\partial \mathcal{L}}{\partial g_j}\frac{\partial g_j}{\partial \theta_{ji}} = \frac{\partial \mathcal{L}}{\partial g_j} z_{ji} = \frac{\partial \mathcal{L}}{\partial \sigma_j}\frac{\partial \sigma_j}{\partial g_j} z_{ji} \tag{2.18}$$

with $z_{ji} \in \mathbf{z}_j$ the entry i of the input vector $\mathbf{z}_j$, $\theta_{ji} \in \omega_j$ is the weight assigned to z_{ji}, and $\sigma_j = \sigma(g_j)$ is a short to indicate that the derivative is with respect to the activation function, with $g_j = \omega_j^T \mathbf{z}_j + b_j$.

Hidden units do not interact directly with the loss function. But their contribution to the error is done through the neurons in the next layer. It is because hidden units are employed as inputs to neurons in the following layer. Again, it is possible to apply the chain rule, to compute the gradient for hidden units. Using the same notations as in Eq. 2.18, the gradient for weights in a given hidden unit is computed as:

$$\frac{\partial \mathcal{L}}{\partial \theta_{ji}} = z_{ji}\frac{\partial \sigma_j}{\partial g_j} \sum_{k \in \text{nxt}(j)} \frac{\partial \mathcal{L}}{\partial g_k}\frac{\partial g_k}{\partial \sigma_j} = z_{ji}\frac{\partial \sigma_j}{\partial g_j} \sum_{k \in \text{nxt}(j)} \frac{\partial \mathcal{L}}{\partial g_k}\theta_{kj} \tag{2.19}$$

considering that the hidden unit j is in the layer l, nxt(j) refers to the units in layer $l + 1$. Before obtaining any gradient, it is necessary to evaluate the model with the current input to compute the output of all the neurons in the model. Also, note that for computing the gradient of hidden units in layer l, it is necessary to first compute the gradients of the next layer $l + 1$. It means that the gradients propagate backward starting from the output units. This algorithm is known as Backpropagation. It is a well-studied algorithm, and its full formulation can be found in the literature [12, 49].

The gradient can be computed using only one, all, or a portion of the training examples. It is common to employ only a randomly selected portion of the training examples to compute the gradient. This random portion is known as a batch. In ANN training jargon, an updating step of the weights (gradient computation with parameter update) is known as an iteration. When the model has seen all the training examples in the training set (i.e., all the examples have been used to update the weights), we have concluded an epoch. One epoch has several iterations, with the number of iteration depending on the batch size. Models are usually trained for many epochs.

As we mentioned earlier, one of the method's drawbacks is the convergence to a minimum local in non-convex function (see Fig. 2.4). This is the case for ANNs. To reduce the effect of local minimum, it is possible to add a momentum factor to the update rule. The main idea of momentum is to accumulate a portion of the gradient of previous iterations. The accumulated gradient can then help to continue the update process in the presence of local minimum. While the direction of the gradient does not change, the momentum term will accumulate a portion of the previous values "accelerating" the learning process in the current direction of the gradient. Momentum updates the weight i of a unit j according to the following $\Delta\theta_{ji}(t)$ value in the iteration t [49]:

$$\Delta\theta_{ji}(t) = \eta\Delta\theta_{ji}(t-1) + \frac{\partial\mathcal{L}}{\partial\theta_i} \tag{2.20}$$

The last expression replaces the gradient in Eq. 2.17 and introduce the momentum hyperparameter η. Different loss functions have been proposed, for example, the mean squared error (MSE) is defined as:

$$\mathcal{L}(f(x,\theta),y) = \frac{1}{2}\sum_m (f(x,\theta)-y)^2 \tag{2.21}$$

with m the total number of samples. This loss function is generally employed in regression tasks. For classification purposes, the binary cross-entropy (BCE) is a usual choice for binary classification problems. It is defined by:

$$\mathcal{L}(f(x,\theta),y) = -\frac{1}{m}\sum_m y\log(f(x,\theta)) + (1-y)\log(1-f(x,\theta)) \tag{2.22}$$

Other errors functions have been proposed in the literature, however, we will introduce them when discussing the real world applications of the neural networks.

2.2.3 Data Representation and Feature Extraction

We have discussed all the elements necessary for supervised learning using ANN, including the model definition, error metrics, and learning process. Before finalizing this section, we discuss the representation of the data employed as input for the models. As we have mentioned before, we assumed that the input for MLP ANNs is a vector of m elements. It means that we need to define a parametric representation of the data, especially when our inputs are images. We can start by briefly describing how an image is computationally represented. An image is represented by their intensities, and each pixel is described as a numerical value going from

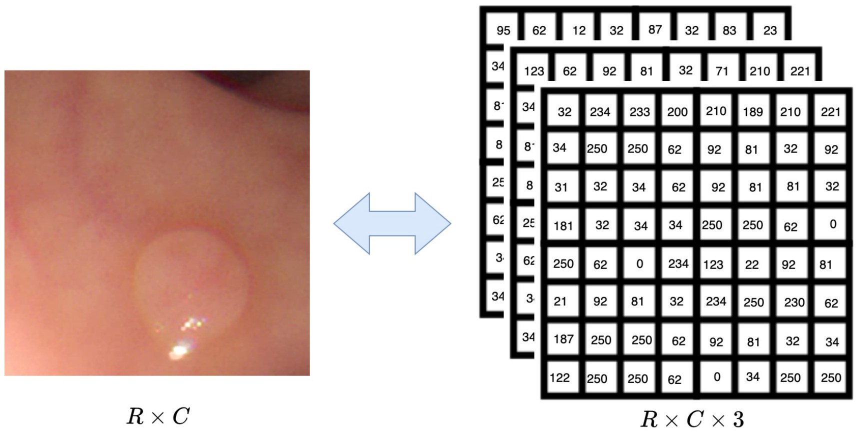


Fig. 2.5. A $R \times C$ color RGB image is represented as a $R \times C \times 3$ matrix. Each entry of the matrix indicates the intensity of the corresponding pixel and color channel. Grayscale images are represented similarly by 2-D matrices, where the entries indicate the gray intensity. The intensity values for every pixel are represented by a number between 0 and 255, in increasing order.

zero (black) to 255 (white). In this sense, it is natural to represent an image as a matrix with dimensions $R \times C$ corresponding to the height and width of the image, in pixels (Fig. 2.5).

We can take as an example the image classification problem. Again, we can consider the input to be an endoscopy frame from a video sequence. The binary classification task is to decide either neoplasia is present in the image or not. We can parametrize the presence of the abnormality with the output label $y = 1$, and the absence of the same can be represented with $y = 0$. We can choose an MLP neural network as our learning model.

Now, we need to define what will be the input to our neural network. We can directly use the matrix representation of the image. Since traditional neural networks require a vector as the input, we can stack all the columns of the three color channels of the image I with size $R \times C$ to generate a vector $\mathbf{x} \in \mathbb{R}^m$, $m = R \cdot C \cdot 3$ (see Fig. 2.6a). Even though we can use this representation as input for the model, this might not be ideal. The raw values of the image can contain patterns that are highly similar between the two classes. For example, in our endoscopy classification problem, the pixels of the tissue inside the neoplasia might be similar to the healthy tissue. Instead, it is common to compute an intermediate representation of the image, to include meaningful information into the input vector. This process is known as feature extraction.

Instead of using the pixels values, we can obtain a different representation of the input image. We can think of feature extraction as a function $ft(\mathbf{x})$ (See Fig. 2.6b) that takes the raw input vector and outputs a new vectorial representation of the data. This new representation potentially has a different dimensionality, and it is expected to highlight important properties of the image related to the given task.

For example, we can assume that the contour of the neoplasias in our tissue classification task has a different color distribution. We can think that comparing the color distribution of the images, we should note these differences in images containing neoplasia, compared

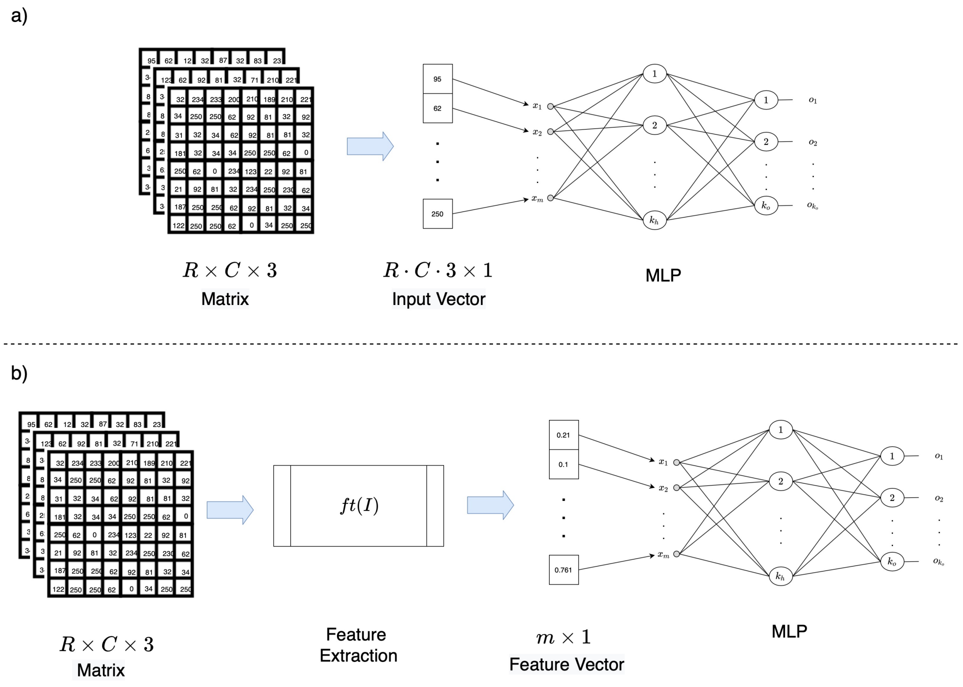


Fig. 2.6. Model input data: a) We can use the raw intensity values of the image as input for the model. However, for complex problems, intensities values alone might not be enough to differentiate between the different classes; b) Instead, it is possible to compute an intermediate representation of the image, employing meaningful "features" of the classes. This process is known as feature extraction.

with images of healthy tissue. Then, we can employ the normalized image histogram as an input feature for our model. The histogram is a vector of dimension m, where m in this case represents the total number of intensity values that a pixel can take. In a standard image, this is $m = 256$, for a range that goes from 0 to 255. Each entry of the vector contains the number of times the corresponding intensity appears in the image. The normalized histogram divides this vector by a value z in a way that the sum of all its elements equals one. We can think of the normalized histogram as the probability of finding a particular intensity in the image. If I is a particular image with $I(i, j)$ its intensity at the pixel position (i, j), then we can define the histogram vector as:

$$\text{hist}(k) = \frac{1}{z} \sum_{i,j} [I(i, j) = k] \tag{2.23}$$

We will use the notation $[\text{arg}]$ to indicates a value of one if their argument ($I(i, j) = k$ in this case) is true, and zero otherwise. In this equation, z is the normalization factor that ensures $\sum_i \text{hist}(i) = 1$. In this case, z equals to the total number of pixels in I. For a color image, we could compute the histogram for every color channel, and compose them into a single input vector.

The histogram introduces a different representation of the image. This representation is useful when the intensity distribution between the images of the same class is similar. This assumption can be true for some of the endoscopy images, however, most of them will contain other elements, like reflections, bubbles, or motion blur derived from the image acquisition process. The histogram is not the only feature that can be employed to represent the information contained in an image. Given the different challenges involved in image analysis, it is common to use complex features tailored to the problem at hand.

Different image features have been employed in the literature, previous to deep learning. Let's take, for example, the detection of *Plasmodium* parasites in blood smear microscopic images. *Plasmodium* parasites are responsible for the malaria disease [59]. It is possible to compute gray-label features, like contrast, texture-based features, or histogram-based features from the images. For the histogram-based features, even though the histogram itself can serve as a feature, it is possible to compute additional features from it, like the mean intensity, standard deviation, skew, or entropy [59]. It is also a common practice to pre-process the image to reduce the number of regions to analyze in a process known as image segmentation. We will discuss in detail segmentation in later chapters. For now, it is enough to mention that this process will divide (or cluster) the pixels of an image if two or more groups. For our example, it would be possible to separate the cells in the image from the rest of the sample. All these objects will contain regular blood cells, together with parasite objects, in the case of infection. We can classify each of these objects of interest to determine if they represent *Plasmodium*. Morphology features could be computed from these structures, as feature extraction methods [59].

We can find more examples in the task of polyp detection in endoscopic still frames. Polyps are abnormal growths in the colon tissue and can precursors of colorectal cancer. Indeed, the images of our neoplasia examples are polyps. Early detection increases the chances of survival for this type of cancer, motivating the development of computational methodologies for this problem. The literature has proposed different features for polyp detection, some of them based on texture information, for example, the color wavelet covariance, texture spectrum histogram, texture spectrum, and color histogram statistics [32], and the local binary pattern [5, 32]. Most of these features are based on the differences in intensity between a central pixel and its surrounding eight-pixel neighborhood. The histogram of oriented gradients (HOG) is also a choice for the polyp detection problem [34].

These features are no exclusive to *Plasmodium* or polyp detection tasks, and they can be computed or "extracted" for different problems. We can decompose these detection problems roughly in two steps. For a given image I, we first compute a set of feature representations $ft_1(I), ft_2(I), \ldots, ft_m(I)$. The concatenation of these representations in a single vector leads to the feature vector $\mathbf{x}$. This vector is the input for our classifier $f(\mathbf{x}; \theta)$. The second step in the detection problem consists of training the classifier.

Some feature representations can perform better than others, depending on the problem at hand. Finding suitable features to extract can be difficult. Some features might be precisely handcrafted for a particular task. We are not going deeper in computing these data representations since, recently, their use has become secondary, mainly because deep learning and the convolutional neural networks have changed the feature extraction task.

3 Deep Convolutional Neural Networks

Data representation was an essential step in image processing tasks. Linearly separable data is an ideal input for any model, however, this is not the case for most of the data in the world. To reduce the complexity of the inputs, it was common to change the representation of the input data, to make the differential attributes of each class more notable for the learning model. As we discussed previously, it is possible to compute different features that describe particular properties of the image. In many cases, the features are chosen or specifically designed for the problem at hand. The selection of features can be a challenging design problem by itself. Currently, the feature extraction process is included in the ANN design. It allows the model, not only to learn the task but also to define the best representation for the input data. The computational model that allows the feature extraction is the convolutional neural network (CNN). They are a kind of ANN that leverages the 2D spatial information from the image and computes a representation that is suitable for the task. The construction of CNN allows bringing to the computed features a grade of invariance upon transformation in the input image, like translation and small deformations. Convolutional neural networks define a powerful learning model that can handle different learning tasks in an end-to-end way, taking an image as input, obtaining an input representation, and giving the classification results with one single model.

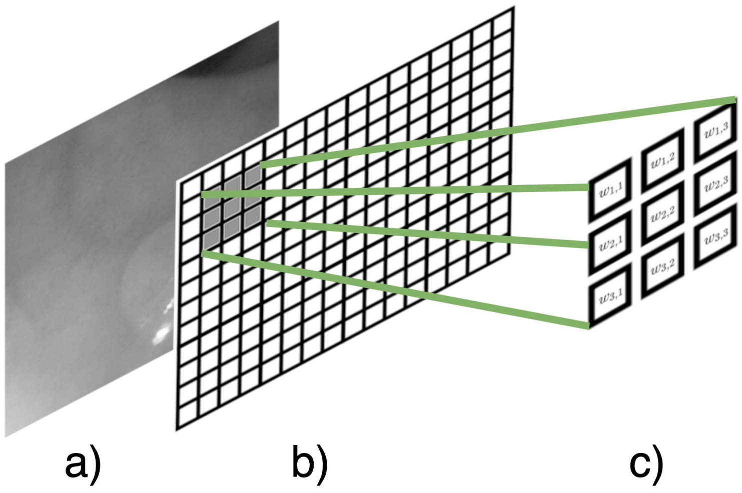


Fig. 3.1. a) An input image, and b) its grid representation. Each element of the grid b) is a numerical value representing the pixel intensity at that position; c) the weights of a single 3×3 convolutional neuron. The neuron computes the weighted sum of the pixel values inside the gray region in b). Neurons in a CNN are organized in planes. All neurons of the same plane share weights, but takes different regions of the image as inputs. It allows performing the same filter, at different image locations.

3.1 Convolutional Units

Features present a rich representation of input data. However, they need to be selected or designed according to the problem to attend. This paradigm has changed with the adoption of deep convolutional neural networks (CNN). CNN's are capable of use backpropagation to learn both a suitable data representation from the raw images and a classifier for predicting input categories.

Like neural networks, the convolutional units have inspiration on models defined to represent a biological system, specifically, the vertebrae's vision system [23]. The basic components of the CNNs are the convolutional units. They have similar behavior as MLP neurons, in the sense that first compute a linear operation in the input and apply a non-linear function. However, the main difference is that the linear operation works in a grid-like structure instead of a vector. Convolutional units can work directly on the image, without changing its representation (e.g. vectorizing it). Fig. 3.1 illustrates a convolutional unit. The figure shows an input image (a) and its matrix representation represented by a grid (b). The 3×3 grid (c) represent the weights of the convolutional neuron. The gray region in the image's grid in Fig. 3.1b is the input to the neuron. CNN's neurons operate on small gird-like regions of the image. Each weight of the neuron multiples its corresponding region in the image. After this, bias value is added, and the results go through a non-linear activation function. Note that, for simplicity, the bias and the activation function are not represented in the image.

Neurons in an ANN are organized in a plane, known as a feature map [12]. Each neuron of the feature map takes a different region of the image as input, and all neurons in the same feature map share their weights. The region covered by a neuron is called the receptive field. In our example in Fig. Fig. 3.1, the receptive field as a size of 3×3. A CNN neuron can be interpreted as a feature detector. Then, the neurons of a feature map detect the same feature but at different positions of the image [12]. It is common to organize the neurons in the feature map so that the receptive fields overlap with a separation of one pixel. This resembles the convolutional operation (giving their name to these units) with the kernel given by the shared weights of the feature map. Considering this last, we can formulate the equation of a feature map as:

$$o(\mathbf{x}; \mathbf{w}, b) = \sigma(\mathbf{w} * \mathbf{x} + b) \tag{3.1}$$

where not $\mathbf{x}$ is an input image (or a grid-like structure), and $*$ represents the convolutional operation with $\mathbf{w}$ a $K \times K$ convolutional kernel. The output feature map is a grid-like structure, where the size is not necessarily the same as the input. A convolutional layer is composed by a group of features map. We can extent the definition presented in 2.8 and define the convolutinal layer in a similar way.

$$\mathbf{l_i} = \sigma(\mathbf{W_i} * \mathbf{l_{i-1}} + \mathbf{b_i}) \tag{3.2}$$

Now the parameter $\mathbf{W}$ represents a multidimensional array or tensor, of dimension $K \times K \times D_{i-1} \times D_i$, for two-dimensional input images. D_{i-1} is the number of features maps of the previous layer, and in the case of the first layer, it represents the number of channels of the input image. D_i is the number of output features maps of the current layer. Another way to describe $\mathbf{W}$ is considering it as a collection of D_i different $K \times K \times D_{i-1}$ learnable

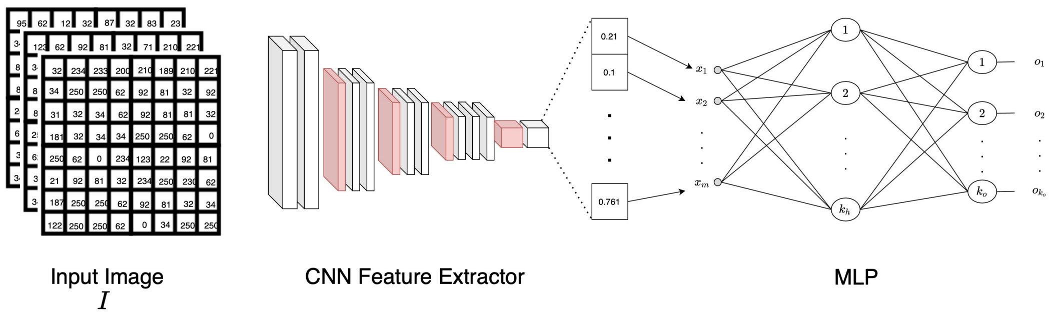


Fig. 3.2. A general representation of a convolutional neural network for the classification task. The channels of the input image go directly into a CNN composed of groups of convolutional layers (white blocks) and downscaling operations (red blocks). The receptive field (the input region for a single convolutional neuron) increases with the depth. The final convolutional layer is used as a feature vector for an MLP to perform the final classification. The model can be trained end-to-end using backpropagation.

convolutional kernels. The output l_i is now a tensor, and in the case of 2D inputs (images), it will take the shape $R_i \times C_i \times D_i$. The non-linear function σ is applied element wise to each entry of the output. A given convolutional layer learns to extract a set of D_i features from the input matrix. Like the MLP ANN, a CNN is composed of successive convolutional layers. Since the feature maps of the convolutional units integrate the information from a region of the image, staking multiple layers allows the network to increase the region covered by the receptive field, with respect to the input image. Deeper layers aggregate information coming from larger regions of the input image. Additionally, CNN models include a downscaling step, given by a max pool operation. It brings a degree of invariance to transformation of the input image to the CNN.

Fig. 3.2 presents a basic structure of a CNN. The input represents a three-channel color image. The white planes in the CNN feature extractor represent the $R_i \times C_i \times D_i$ feature maps obtained by a given convolutional layer. The red planes represent downscaling operations. As Fig. 3.2 shows, we can add a fully connected MLP after the last convolutional layer. This architecture defines a deep convolutional neural network, which is trainable with backpropagation. This fully trainable architecture has the advantage of optimizing the feature transformation of the input image and the MLP classifier for the defined tasks. CNN only requires the images and labels to train. It is unnecessary to design a feature extraction method for the input since it is implicit in the network design.

3.2 AlexNet

Initial proposals contained few convolutional layers (e.g., LeNet had 3 Conv Layers). One of their main tasks was handwritten digit recognition. They operated on small inputs, e.g., images of 16×16 pixels [45]. Initially, computational resources represented an obstacle to designing deeper architectures on higher resolution images until the development of affordable and efficient consumer-level graphics processing units (GPUs). Deep learning models recently gained popularity when a deep architecture obtains the best top-5 score in the ImageNet

Large Scale Visual Recognition Challenge[1] (ILSVRC). The ILSVRC challenge includes the task of image classification with a total of 1000 object categories. The dataset comprised around 1.2 million training images, 50,000 images for validation, and 150,000 images for testing purposes. All images are taken from the ImageNet dataset, which contains over 15 million labeled images [42]. The large number of training examples in the ImageNet dataset, together with efficient GPU implementation of the convolutional operation, made possible the development of deeper CNN models, containing many hidden layers.

The network proposed by Alex Krizhevsky (known as AlexNet) was trained for the classification task considering the 1000 possible categories. It contains a total of eight layers divided into five convolutional layers and three fully connected MLP layers at the end [42]. Deep architectures have different problems, for example, overfitting. Overfitting occurs when the network performs outstandingly in the training set but cannot generalize to other examples. It means that the model has low performance when predicting any other example, not in the training set. To avoid overfitting issues due to a high number of parameters, and to improve the model‘s performance, Krizhevsky employed diverse design strategies. Some of them are still using in current models.

Before the AlexNet, different models use to employ activation functions like the sigmoid. As we discussed before, some of the disadvantages of these functions are the saturation in the positive and negative extrema of the function, leading to small update signals (gradients), and slowing down the training process. The architecture of AlexNet employs ReLU activation functions, claiming a faster training time compared with saturating nonlinearities. This non-linear function has become one of the standard activation functions for modern architectures.

Also, to improve generalization, Krizhevsky employed local response normalization to the outputs of the neurons in a given layer, which performs an inter-feature map normalization at each point of the current feature map i, considering a local feature map neighborhood of i within the same layer. Current models still relying on normalization, however, it is common to follow a batch normalization strategy [33]. Other design choices include overlapping pooling layers. Pooling layers are employed to compile the information coming from a layer, or group of stacked layers, and reduce the dimensionality of the output feature maps (downsampling). Similar to convolutional units, pooling layers act on small regions of the input and are ordered in a grid-like array. Usually, pooling layers do not overlap, meaning that the regions of action for each pooling unit are different. The AlexNet instead, proposed to employ overlapping pooling units.

Methods to reduce overfitting include the artificial augmentation of data. Spacial augmentations can include rotation, cropping, translations, and flipping of the input image. Intensity transformation includes changes in the intensity, color, and illumination, for example. A popular approach to reduce overfitting is the addition of dropout layers [80]. During training time, this method "drops" a particular hidden neuron with a certain probability, avoiding the participation of dropped units during the forward-backward pass of the training process. This process reduces the neuron co-adaptation and forces each unit to learn a robust feature representation of the inputs. During each training step, this method drops a random number of neurons. It can be interpreted as generating a random submodel and training it by one single

[1] https://image-net.org/challenges/LSVRC/

step. During inference, all the potentially infinite submodels are employed to evaluated to obtain the predictions. It is done efficiently using all the neurons (no dropout in testing time) but multiplying their outputs by 0.5. AlexNet uses dropout in the two first fully-connected layers, with a dropout probability of 0.5.

The neural network trains on two GPUs. When AlexNet was proposed, the GPUs employed were Nividia GTX 580 with three GB of video memory. It makes a total of 6 GB of memory used to train a network with 60 million parameters [42]. The network took a training time of six days in the two GTX 580. It achieved the best top-5 test error rate of 15.3 %, representing a considerable improvement over second place, with a 26.2 %, during the ILSVRC-2012 competition. At the time this thesis is written, consumer-level GPUs reach the 12 GB of memory for a single GPU.

3.3 Common Components in Deep CNNs

The AlexNet is one of the recent architectures that brought the focus on deep learning and deep CNNs. As discussed in the last section, it incorporates different strategies and layers to define a successful architecture. Some of the ideas included in the original AlexNet are still used in modern architectures. Commonly, CNNs models can incorporate, in addition to the convolutional blocks, pooling layers, dropout layers, and batch normalization.

3.3.1 Pooling

A desirable property for the features obtained with CNNs is the invariance to a certain degree of translation and other transformations in the input. In the classification problem, we can take an image and apply a slight translation to elements in the image, and this should not change the class to the image belongs. The pooling operation performs downsampling of the features maps, reducing its size, and hence the memory requirements. It allows increasing the number of features maps in the following layers. However, pooling operations allow the model to learn shift-invariant features.

In practice, the pooling operator is an aggregation operation over a small neighborhood of the features maps [68]. The most common aggregation layer is the non-overlapping max-pooling operations. The operation takes a local neighborhood of $k_p \times k_p$ in the input feature map, and outputs the maximum value inside the neighborhood. Similar to CNNs, max-pooling units operate in different regions of the input map. It is usual to employ 2×2 pooling units, covering all the input (Fig. 3.3). It allows obtaining a feature map that is half the dimension of the input.

3.3.2 Overfitting and Dropout

The large number of parameters in deep models can cause overfitting. As we mentioned, it occurs when a model achieves high performance in the training set that is not reflected when evaluating new samples. The initial proposal for reducing overfitting consists in reducing the

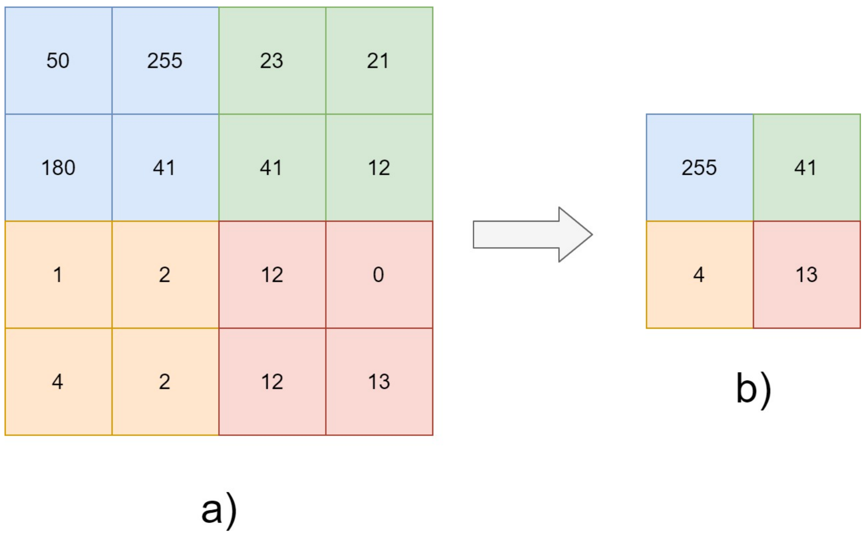


Fig. 3.3. A 2×2 non-overlapping max-pooling operation. a) The input feature map. Each color represent the local neighborhood involved in the pooling operation. b) The results of max-pooling, conserving only the biggest value inside each neighborhood and halving the size of the input feature map.

L_1 or L_2 norms of the model´s parameters by adding this term to the loss function. Other methods involve a second set of examples, knwon as the validation set. The validation set does not take part for training, but it is use to evaluate the model's performance after certain number of epochs. If the performance in the training set is increasing, but the performance on validation start to decrease, we can say that the model is overfitting, and then we can stop the training.

Dropout [80] is also a method proposed to reduce overfitting in deep learning models. Broadly, as we mentioned before, dropout reduce the co-adaptation of the neurons by randomly removing a set of units and their connections during the training step. The idea behind dropout is that one way to improve the model´s performance is by taking the prediction obtained from averaging the outputs of the model under several selections of its parameters. Dropping neurons from the network can be similar to sampling a submodel out of the complete full-neurons model. Another way to interpret it is considerong that a particular neuron should learn how to produce a meaningful feature under a randomly chosen sample of the other units, and hence, reducing complex co-adaptations.

We can describe how dropout works using a single model, like a two-layered MPL, defined as:

$$y_o = \sigma(\mathbf{x}W_1 + \mathbf{b})W_2 \tag{3.3}$$

for this example, we set $\mathbf{x}$ a $Q-$ dimensional input vector, W_1 a $Q \times K$ and W_2 a $K \times D$ weight matrices of the first layer and second layer, and $\mathbf{b}$ the $K-$dimensional bias vector. As usual, σ represents the non-linear activation function.

Dropout considers random binary vectors for each layer. In our example, we can denote as $\mathbf{z_1}$ and $\mathbf{z_2}$ with dimensions Q and K, respectively. These are vectors composed of independent random variables following a Bernoulli distribution. It is denoted as:

$$\mathbf{z}_{1q} \sim \text{Bernoulli}(p_1), \mathbf{z}_{2k} \sim \text{Bernoulli}(p_2) \tag{3.4}$$

with q and k iterating over the entries of $\mathbf{z_1}$ and $\mathbf{z_2}$, respectively. The parameter p_1 and p_2 are known as the dropout rates. They indicate the chance for a particular neuron to be "dropped". It is usual to select the same dropout probability for all the layers in the network, making $p_1 = p_2 = p$ in our two-layered example. During each training iteration, the dropout strategy samples each of the $\mathbf{z}$´s vector, and performs the forward and backward passes with the following sub-model:

$$y_o = [\sigma([\mathbf{x} \circ \mathbf{z}_1]W_1 + \mathbf{b}) \circ \mathbf{z}_2]W_2 \tag{3.5}$$

where $\circ$ is the element-wise multiplication (Hadamard product). Each training iteration samples a different set of vectors. It is equivalent to train several sub-models for one iteration. During testing, dropout is not applied. Instead, the weights are scaled using the dropout rates as $W_1 = pW_1$, and similar for W_2. For deeper networks, or for grid-based models (like CNNs) these formulations are adjusted accordingly.

3.3.3 Batch Normalization

Normalization aims to reduce the variability in the input data, by bringing all the information contained in the image into a similar distribution or a similar scale. Normalization is a common practice for the input data, however, it is also possible to normalize the hidden outputs of the neural network, in order to reduce the change in distribution that the feature transformations could bring. According to [33], changes in the layer's distribution cause neurons to need to continuously adapt to the new distributions. To reduce this distribution change in the network activations, the authors of [33] proposed batch normalization, which works at the training batch level and is included as part of the model´s architecture. The authors claim a lower dependence on the initialization of model parameters and more flexible choices of the learning rate, allowing working with high values. Typically, higher learning rates might cause the learning process to be unstable. Batch normalization has become a recurrent resource in the definition of models.

Input data can be normalized by subtracting the data mean and dividing the result by the standard deviation. Batch normalization performs this operation for the inputs of all hidden layers in the network. The statistics are not computed using the entire training set. Instead, they are computed using only the samples in the training batch, during each training iteration. The operation performed is defined by

$$\hat{\mathbf{x}} = \gamma \frac{\mathbf{x} - \mathrm{E}[\mathbf{x}]}{\sqrt{\mathrm{Var}[\mathbf{x}]}} + \beta \tag{3.6}$$

here, $\mathbf{x}$ represent the input for a particular layer. The parameters γ and β are learnable values that allow the BN layer to learn the identity function. It allows maintaining the expressivity of the model and recovering the original input to the layer by the learning algorithm.

3.3.4 Transfer Learning

Deep learning models require large amounts of labeled data to train. Challenges like the ImageNet classification challenge bring access to big datasets on real-world objects. However, it is not the case for other tasks, especially in the medical domain. It can be expensive to collect labeled data leading to a reduced amount of available training, validation, and testing examples. For this reason, researchers have been interested in the reusability capabilities of the CNN models. In particular, can a model trained, for example, in the large ImageNet dataset, be reused in a related task but with a different dataset. The concept of transfer learning comprises this strategy. It is common to employ models pre-trained on the largest datasets like ImageNet for initializing the weights of CNNs. Then, these CNNs train on the new, and potentially smaller, datasets, for example, medical datasets.

Transfer Learning in deep models is based on the observation that the initial layers of the CNNs learn to extract general features [87]. These features are common to different tasks and datasets. Deeper layers of the network are observed to be more task-specific. In this sense, it is could be possible to initialize a new model with the already learned filters and train the task-specific parts of the network. Typically, a set of k initial layers are selected and copied to the new model. The remaining layers of the new model are initialized as usual. During training, there are different options for processing the selected k layers. These layers can be frozen, meaning that their values are kept during training. This is useful to avoid overfitting when the number of training examples for the new task is low. These initial layers can also be fine-tuned, to adapt them to the new dataset. It is feasible when the number of training examples for the new task is high. If the amount of training examples is large, then it is not necessary to transfer previous models, since, with an adequate amount of samples, the features can be learned from scratch. As we can expect, transfer learning works better when the transferred task and the new task are similar.

3.3.5 How to implement CNNs?

The implementation of AlexNet employed an efficient CUDA version of the convolution operation. Implementing the components and training methods efficiently can be time-consuming. However, given the high interest in deep learning and CNN, diverse frameworks have been released for fast implementation of CNNs. Those frameworks have efficient implementations of the basic building blocks for CNNs, and their corresponding learning algorithms. Currently, two of the most used frameworks are Tensorflow[2] and PyTorch[3]. Both are based on python, which gives them a high versatility and allows the user to focus on the design and testing of new techniques or architectures.

[2]https://www.tensorflow.org
[3]https://pytorch.org

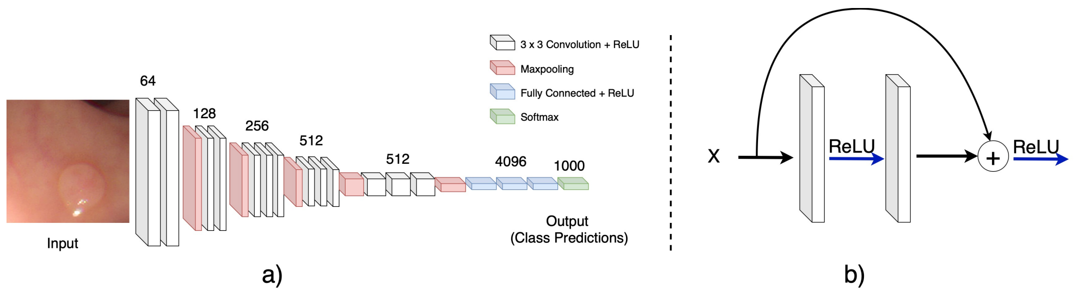


Fig. 3.4. a) The VGG-16 architecture. It is composed of 16 trainable layers. The convolutional part is composed of blocks of successive convolutional layers, with a max-pooling operation at the end. All the layers inside the same block have the same number of feature maps indicated by a number at the top of the layer in the image (e.g., the first two layers have 64 features maps); b) A skip connection representing the residual function of ResNet. The figure presents a residual block composed of a convolution, a ReLU activation, a second convolution, and the aggregation of the input to the block with the output of the second convolution. Finally, a second ReLU is applied.

3.4 Deep Convolutional Neutworks in the Medical Domain: Endoscopic Image Analysis

In this section, we discuss a medical application of CNNs in the medical domain. We have presented the neoplasia classification as an example when explaining the components of ANN. In this section, we formally consider it as a medical problem and discuss how deep learning models are applied to it. We will define this problem as the polyp detection problem. For the moment, we can consider a polyp as an abnormal growth in the colon tissue. We will come into further details later in this section. Before starting to discuss this problem, we would like to briefly introduce three architectures that have been employed in the medical domain. After the success of the AlexNet, the computer vision community proposed different models for image classification. In the next years, researchers explored different sizes of convolutional units, model depth, and methods for initialization, pretraining, and transfer learning from previous models. It leads to the definition of multiple architectures that are still the base of recent models. Examples of these architectures are the ResNet [31] and VGG [74] models, proposed for image classification.

3.4.1 VGG and ResNet

The work presented by Simonyan [74] explores the performance of CNN when the depth varies. Sinonyan presented an architecture with 3×3 convolutional filters and depths from 11 to 19 layers. The networks incorporate Max-pooling after a certain number of convolutional layers. The number of convolutional layers is variable (from 8 to 16). Finally, three fully-connected layers follow the convolutional set and give the final 1000 channels output for the ILSVRC classes (Fig. 3.4a). The number of features maps starts from 64 in the first layer. This number is duplicated after max-pooling until a max value of 512 channels. The name of the team participating in the competition was VGG, from where these architectures take their name.

The deep residual neural network (ResNet) presented in [31] incorporate residual function in the network formulation. The residual functions are defined as $f_l(\mathbf{x}) + \mathbf{x}$, where the function $f_l(\mathbf{x})$ represents the output of weight layer with $\mathbf{x}$ its corresponding input. These residual connections can be interpreted as a shortcut or skip connection that adds information from previous layers to the input of subsequent convolutional units (Fig. 3.4b). Both architectures have been used as main building blocks for more complex models, like the region proposal neural networks, defined for object localization.

3.4.2 Faster R-CNN and Object Localization

ResNet and VGG present powerful CNN architectures for image classification in a global scheme; that is, these models offer a prediction for the entire image and cannot bring information at a local level. Local information not only can give information about the class or object represented in the image. It also can bring the localization of the object inside the image.

Initial object localization problems addressed with CNN's required the regeneration of region proposals. These proposals can be defined as bounding boxes containing a section of the image. Each proposal is then classified by the CNN to define either it contains or not an object. In the case of bounding boxes containing objects, the CNN predicts the object's class. The generation of the region proposals was independent of the neural network and often led to bottlenecks in the object localization problem. Faster R-CNN [57] introduces regions proposal neural network to address this problem.

The main motivation of faster R-CNN is the fact that the activation of the feature maps of deep layers in convolutional neural networks encodes localization information about the predicted object. Under this assumption, faster R-CNN employs this information to also compute region proposals that are later classified to predict object categories. This definition allows employing the same set of convolutional layers for both tasks. Figure 3.5 presents an overview of this architecture. Region proposals are generated by sliding a small network over the output feature map of the last convolutional layer. This network takes an $n \times n$ spatial window as input. Each spatial window is mapped to a lower-dimensional feature map and then used to feed two fully connected layers for bounding box regression and object/non-object classification, respectively. Each region proposal is modeled by four values that represent the coordinates of a bounding box. The network predicts a maximum number of k region proposals, each one with an object and a non-object probability. From this, the regression layer has a total of $4k$ outputs, and the classification layer has $2k$. Each proposal is associated with an anchor centered at the sliding window, and at the same time, each anchor is associated with a scale and aspect ratio. The anchors have the property of translation invariance and can address multiple scales making this kind of models perfect for localization problems in medical images, like the polyp localization task. The region proposal network employs the following combined loss function:

$$\mathcal{L}(\{p_i\},\{t_i\}) = \frac{1}{N_{\text{cls}}}\sum_i \mathcal{L}_{\text{cls}}(p_i, p_i^*) + \lambda \frac{1}{N_{\text{reg}}}\sum_i \mathcal{L}_{\text{reg}}(t_i, t_i^*) \tag{3.7}$$

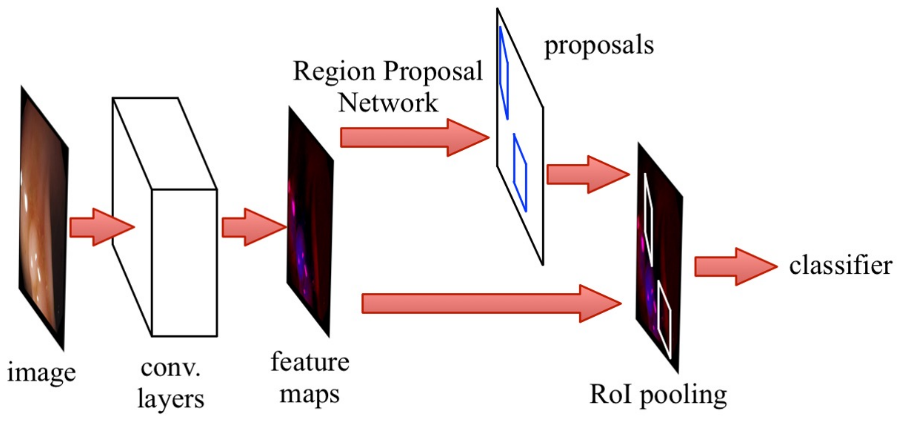


Fig. 3.5. A general overview of the Faster R-CNN architecture. Faster R-CNN employs the feature maps learned by a deep network to feed a region proposal network and a classifier. The region proposal network performs a regression task and outputs a set of object bounding boxes. Each bounding box is parametrized by a collection of default boxes known as anchors. The Classifier takes the bounding boxes and network's feature maps to perform the object classification of the bounding box predictions.

where p_i is the probability for the anchor i to be an object, t_i represents the four parameters that define the predicted object's bounding box. The parameters $p*_i$ and $t*_i$ represent the same but for a ground truth bounding box associated with an object-positive anchor. The loss function $\mathcal{L}_{\text{cls}}$ is represented by the $\log$ loss. The regression loss for bounding box prediction is given by $\mathcal{L}_{\text{reg}}(t_i, t_i^*) = R(t_i - t_i^*)$, with R the smooth L1 loss. Each loss is normalized by N_{cls} and N_{reg}, and λ is balancing parameter. The output of the region proposal network is evaluated by a Fast R-CNN architecture, for object classification.

3.4.3 Polyp Localization

According to the World Health Organization, 8.8 million people worldwide died from cancer in 2015. Early diagnostic helps to increase the survival rate for patients; however, this implies the detection of symptomatic patients as early as possible [8, 72, 73]. Colorectal cancer (CRC) is the third most commonly diagnosed cancer in the United States, where 135,430 new cases were expected in 2017 [72], and 97,220 new cases were estimated for 2018 [73]. Survival rates for CRC reach 95% when detected in the early stages [8]. For this, the gold standard for CRC screening is the endoscopic analysis of the colon, known as a colonoscopy. During this study, the endoscopist explores the colon cavity looking for abnormal growths of tissue known as polyps (Fig. 3.6), which may develop into tumors. Polyps with a certain shape, kind, and size (e.g., those with sizes between 5 mm and 1 cm) should be removed for closer examination [15].

The detection problem has been a topic of research, given the importance of early detection in patients' recovery. Automatic polyp detection represents a challenging problem. Algorithms should work in an environment with artifacts like specular reflections and motion blur. It is also a highly dynamic environment, where structures' shapes are affected by endoscopic tools or natural colon movements, leading to variations in the colon shape. It is in addition to the

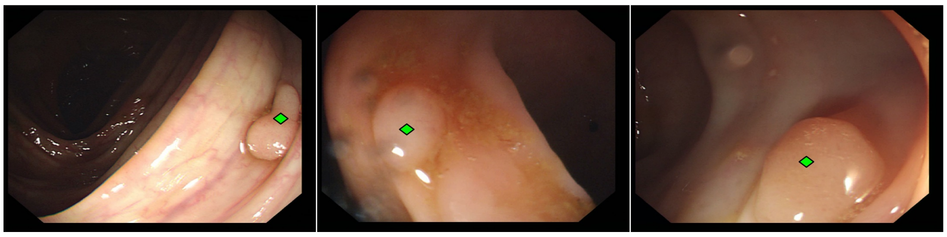


Fig. 3.6. Examples of polyps in an endoscopic still frame. Polyps locations are indicated by the green mark.

normal variability in size and shape that these structures present. In contrast with real-world images, medical imaging presents a high similarity between the structures of interest and the background. It can be attributed to the fact that the anomalies, like polyps, are variations of the normal tissue. So we can expect similarities in textures between polyps and regular colon walls.

The formulation of supervised learning methods requires annotated data. As in different medical problems, many efforts led to the creation of public datasets for polyp detection. One of the most used datasets comes from the MICCAI 2015 Polyp Detection in Colonoscopy subchallenge [10]. The subchallenge was part of the Endoscopic Vision Challenge, and presented two main problems: polyp localization and polyp detection. At this point, it is necessary to differentiate these two related problems. Polyp detection defines as a whole-image binary classification problem. This problem only tells us either a polyp is present or not in the image. The polyp localization problem cares about where the polyp is in the image. The objective is to define the polyp's position in the input image with a pixel-based coordinate or a bounding box. For the challenge, the indicator was the pixel-based coordinate of the polyp's location. The definition of the second problem can already give us the idea that a model like Faster R-CNN could deal with this problem.

The Polyp Detection subchallenge employed the CVC-ClinicDB [9, 10] the training dataset. It was provided by the Hospital Clinic and the Computer Vision Center of Barcelona, Spain, and consists of 612 colonoscopy frames, obtained from 29 video sequences. All the frames contain at least one polyp with a total of 31 unique polyps. The ETIS-Larib dataset [10] provided testing samples for the challenge. It contains 44 different polyps within 196 frames obtained from 34 videos. Both datasets present polyp masks as annotations. Outside this subchallenge, the Kvasir-SEG dataset [35] presents 1000 images together with bounding box and segmentation mask annotations.

Early polyp detection methods rely on handcrafted features based on intensity, shape, and texture descriptors [8, 27, 29]. During the MICCAI 2015 subchallenge, some methods followed a similar tendency and employed intensity-based features. Another set of models, however, incorporated ent-to-end architectures and CNNs in the detection pipeline. The last proposals achieved the best accuracy in the challenge.

A direct application of the Faster R-CNN architecture can achieve F1 score in the range around 0.67, employing the same training and testing sets as the MICCAI 2015 polyp subchallenge. The performance improves with the addition of pre and posts processing tasks. For example, the work presented in [50] employed a larger training dataset composed of 16 randomly selected sequences from the CVC-Clinic2017 dataset (MICCAI 2017 endoscopic sub-challenge). This dataset has in total 18 sequences containing 11954 frames. The authors also changed the training strategy of the original Faster R-CNN. Both localization and classification losses are optimized at the same time, taking a mini-batch as input. Testing the model on three datasets (CVC-ClinicDB, CVC-ColonDB, and CVC-EndoSceneStilll), the authors achieved an average F1 score of 0.971 in the localization task.

In a concurrent work, [71] employs the same model for polyp localization. However, some differences are noticeable with respect with [50]. The authors of this work rely on data augmentation to increase the number of training examples and post-learning methods to improve detection performance. The version of Faster R-CNN employed uses inception ResNet as backbone network, instead of VGG. Data augmentation is a common approach to increase the number of training examples. In computer vision, larger datasets of real-world images are available for model design. However, this scenario is different for medical image analysis. Medical imaging data is limited and requires experts for proper annotation. Taking the polyp detection task as an example, a certain amount of training or experience is necessary to differentiate a polyp from the rest of the colon. In fact, this is a labor that only an endoscopist should perform. It makes it complicated to generate large datasets for model training.

Image augmentation consists of applying spatial and intensity transformation to an input image to increase the number and richness of the dataset, two key-points in the training of deep models. The transformations are selected considering the problem at hand. They should bring differences from the input image but also should generate a plausible image. In the polyp detection problem, the authors of [71] experimented with different combinations of rotations, flipping, zooming, shearing, blurring, and brightness transformations to simulate the environment of endoscopic frames.

3.5 Fully Convolutional Neural Networks and Segmentation

The last section discussed classification tasks on images using supervised learning. We also introduced CNN for classification and discussed examples of its application in the polyp localization problem. In the image classification task, the model takes an image and outputs a value or vector, indicating the most likely class for the input image. We can think that CNN's work on an image-level, where the prediction describes the object contained in the image as a whole, without giving details of its spatial location in the input. In contrast, region proposal neural networks like Faster R-CNN, go into a more detailed description of the image by giving information about all the objects contained in the image and their position. We can consider that Faster R-CNN is a classifier that works on a regional level.

We can go on a higher level of detail and perform a classification at the pixel level. A criterion can be, either the pixel belongs or not to a particular object in the image. It will group the pixels according to the class they belong to, creating mutually exclusive clusters. This process is known as image segmentation (Fig. 3.7).

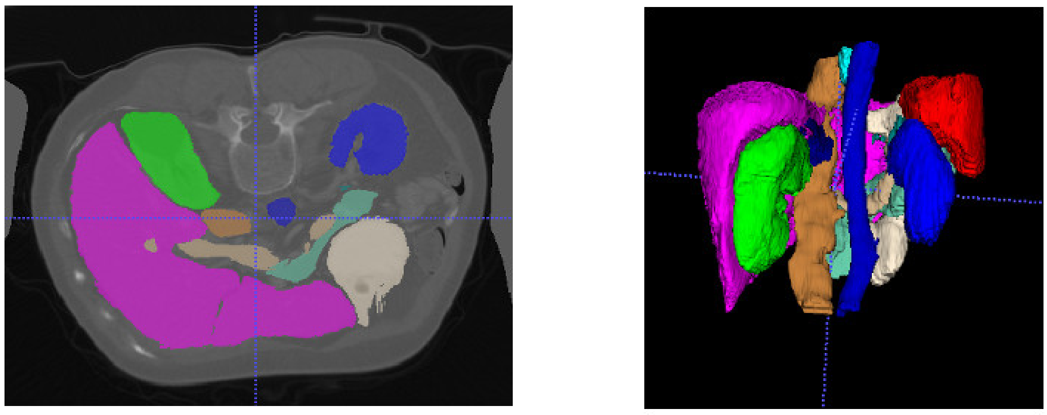


Fig. 3.7. The segmentation task can be understood as a pixel-level classification that forms clusters of pixels based on a criterion of interest. For example, the image shows a multi-organ segmentation, where each pixel is labeled according to the organ to which they belong.

Image segmentation consists in divide the pixels of an image into two or more regions. The main objective is to identify objects or regions of interest in the image using the pixels contained in these regions. It is usually a pre-processing step in image analysis. For example, a blood sample can be segmented to separate the cells. Later, a second classifier can focus on the segmentation results for cell identification. Considering an image $I \in \mathbb{R}^{R \times C}$ with $I(i,j) \in \mathbb{R}^{k_I}$ the value of the pixel at row i, column j, k_I the number of channels, we can define the segmentation process as a pixel level operator $f_p : \mathbb{R}^{k_I} \to \{0, 1 \ldots k_o\}$ with k_o the number of classes:

$$Y(i,j) = f_p(I(i,j)) \tag{3.8}$$

note that k_o indicates the number of possible objects to segment. Also, it is usual to define the label 0 for the background. The background is anything different from the regions or objects of interest. In the cell segmentation example, the background would be composed of all the regions that are not a cell. In an organ segmentation example, the background is all the tissue and image elements not belonging to the organ of interest. Similar to the image classification task, this definition corresponds to classical pre-deep-learning approaches. Before deep learning, it was usual to compute a feature representation of the pixels, considering different attributes of the region of interest. Then, these features are compared with a pre-computed model, to determine their similarity with the objects of interest, or employed to train a pixel classifier, modeling the segmentation task as a classification problem. In both cases, it is necessary to process each of the images to get the results.

With the development of deep learning, the segmentation task was also modeled as an end-to-end task. Modern CNN architectures allow image-to-image mapping, taking an image as input, and giving an array with the same dimensions as the input but indicating the classes for every pixel. Then, a CNN can compute the segmentation of a image as a function $f_I : \mathbb{R}^{R \times C} \to \{0, 1 \ldots k_o\}^{R \times C}$:

$$Y = f_I(I) \tag{3.9}$$

In this definition, the output Y of the function f_I is an image of the same dimensions as I. The difference between CNN's defined for segmentation and CNN's for classification problems is

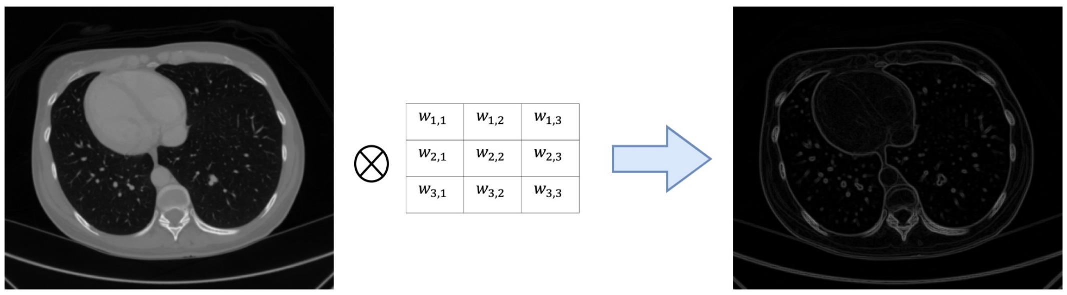

$w_{1,1}$ $w_{1,2}$ $w_{1,3}$
$w_{2,1}$ $w_{2,2}$ $w_{2,3}$
$w_{3,1}$ $w_{3,2}$ $w_{3,3}$

Fig. 3.8. Example of discontinuity-based segmentation. The example shows an input CT axial slice, a representation of a 3×3 filter, and the corresponding output (edge detection in this example). Image from the NIH pancreas dataset [16, 61, 62].

that the classification models comprise the input image into a single value or a vector, usually with dimensions equal to the number of available classes. In the segmentation models, the output is also an image. CNN's model does not process a pixel at a time, but the entire image in a single forward pass. It requires adjusting the architecture of CNN for performing dense pixel prediction.

3.5.1 Classical Image Segmentation

Before describing how CNNs adapt for segmentation purposes, we can mention the classical approach used in this task. As mentioned, initial segmentation algorithms were based on the similarity between the object of interest and the input image. Discontinuities in the image were also a source of information for segmentation [28]. It was common to compute a feature representation of the pixel or region information. Similar to the classification task, the main source of data in an image is the pixel intensities. Discontinuities in the images are used for detecting basic structures, like points, lines, and edges. The way to detected these structures is by applying filter operations on the intensities values. A filter is a small matrix (e.g., a 3×3 matrix) of real values, that is applied to the image using the convolution operation (Fig. 3.8). The filter's response is then thresholded to obtain a binary separation of the input image with the structures of interest. It is the base of different edge detectors, like Canny [14, 28].

Intensity can be applied directly to separate the image. We can consider the binary segmentation task. Similar to its classification analog, it consists in dividing the image into two groups. One of those groups is the object or region of interest, and the second usually corresponds to the background. We can take the medical problem of cell segmentation as an example. Let's consider we are interested in segmenting *Trypanosoma cruzi*, a blood parasite protozoa that causes the Chagas disease [52, 76]. A single method to segment the image is checking if the intensity values are above a given value. It will lead to segmentation function $Y(i,j) = f_p(I(x,y))$ define by the sign function:

$$Y(i,j) = f_p(i,j) = \begin{cases} 1 & \text{if } I(i,j) > \tau \\ 0 & \text{otherwise} \end{cases} \tag{3.10}$$

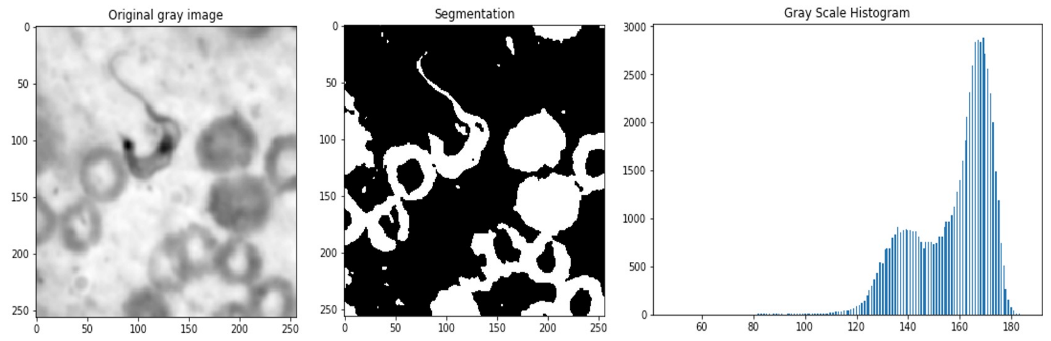


Fig. 3.9. From left to right: A gray scale image containing a *Trypanosoma cruzi* parasite; The segmented image using a threshold of $\tau = 150$; The histogram of the gray scale image

This method is known as thresholding with τ the given threshold. The threshold can be chosen base on the histogram of the image. This method is useful when the intensities of the two classes in the image are clearly distributed in different extrema of the histogram. In this situation, the histogram of the image shows two peaks, corresponding to the two dominant intensities. The histogram in the third image of Fig. 3.9 shows an example of this. If our objective is segmenting all the cells in the image (blood cells and parasite), this method can be ideal. In the results presented in the image, the background is clearly separated from the blood cells and parasite. However, there is no separation between the different instances. As a result, the parasite is almost fused with a round cell in the image. Now, let's suppose that our objective is to segment only the parasite in the image. In this case, single thresholding is not enough to separate the image components. The intensity distribution of the parasite is similar to the distribution of the blood cells, making this a challenging task for thresholding methods.

K-Means and Region Growing are other well-known segmentation [28]. K-Means is an unsupervised clusterization method. It iteratively segments the image into k clusters considering the distance of the intensity distribution of the pixels with the cluster's means. The number of clusters is user-defined and the initial means are chosen randomly between the pixels in the image. Each iteration, update these means using the pixels inside the cluster. Region growing is a graph-like algorithm that takes an initial pixel position as a seed and propagates through its neighboring (adjacent) pixels. The propagation is done only if a particular neighbor is similar to the seed pixel. The region grows through different neighborhood levels until the similarity criterion is not meet. We are not going into further details on these methods, but image 3.10 shows the results of segmenting a color blood sample with these algorithms. As can be seen, k-means can have a similar problem as thresholding, when the distribution of the different classes is similar. Region growing presents better performance but requires user-initialization. Similarly, overlapping objects with similar intensities can cause the region to grow beyond the object of interest, like the blood cell and the parasite presented in image 3.10.

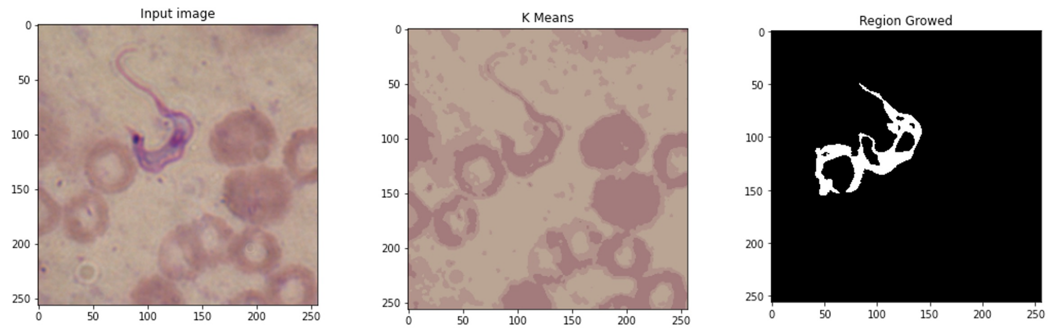


Fig. 3.10. From left to right: A color image containing a *Trypanosoma cruzi* parasite; The segmented image using k-means with $k = 3$; The results from the region growing algorithm

3.5.2 Convolutinal Networks for Image Segmentation

We have seen that one of the initial applications of CNNs is the classification problem. Like AlexNet or VGG, initial proposals include a fully connected network, like the multilayered perceptron, which performs the final classification. However, segmentation tasks require a classification at the pixel level instead of at the image level. The design of CNNs increases the effective receptive field in each convolutional layer and reduces the resolution of the features maps. It contrasts with the segmentation problem, where the output has the same or a similar resolution as the input.

In this direction, Jonathan Long [47] presented one of the first works that contributed to the extension of classification CNNs to segmentation tasks. Long presented a fully convolutional network (FCN) trained end-to-end for the semantic segmentation problem. FCNs can generate dense pixel predictions training directly on images and require any pre-processing or post-processing.

Figure 3.11 presents a graphical comparison of the FCN compared with a CNN model. Both models integrate a set of convolutional and max-pooling layers. However, while the CNN includes a fully connected layer as the final output, the FCN contains only convolutional layers. Compared with the CNN, the fully connected network removes the final softmax output and replaces the fully connected layers (blue box in Fig 3.11) with convolutional layers. This change allows the network to operate on images of arbitrary size. A model of these characteristics outputs a coarse heat map that encodes the object's position in the input. To produce a dense prediction, FCN incorporates a deconvolution layer that recovers the spatial resolution, initially lost by the effect of the convolution and max-pooling steps.

FCN models take CNNs as a building block, which is employed as a backbone network. One of the most common architectures initially employed, for example, in the pancreas segmentation problem, is the VGG 16 architecture. However, it should be possible to change the backbone network by replacing the fully connected layers of the CNN. Employing existing networks as backbone gives an additional advantage: we can pre-train the backbone network on a different dataset (even on classification) and use the pre-trained model to initialize the weights of the FCN. We can initialize almost all the weights of the segmentation model, except for the

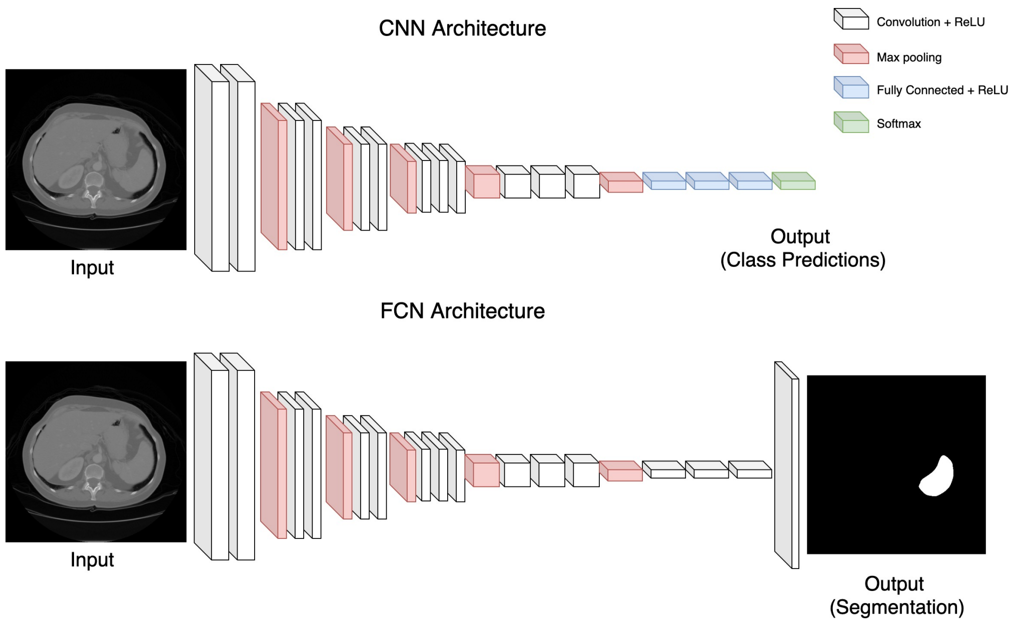


Fig. 3.11. Comparison between a (VGG-like) CNN architecture and its analogous FCN architecture. FCN replaces the last classification layer by a deconvolution filter (up-sampling) for dense pixel-wise prediction.

convolutional layers that replace the fully connected ones. This transfer learning strategy can be beneficial when working with medical images, where the amount of label data could be limited. In this sense, the backbone network can train on the large image classification datasets, like coco[4] or ImageNet[5], and fine-tuned to adapt the learned weights to the smaller segmentation datasets.

The deconvolution step of the FCN combine different feature channels from the deep layers of the backbone network. The original paper [47] tested three different versions of the FCN, integrating the max-pooling layer's output at various resolutions (FCN32, FCN16, and FCN8, see Fig. 3.12). It allows considering local and global information in the prediction of the segmentation map. FCN32 considers only the outputs after the last pooling layer, and hence, it includes information obtained at a coarse level. FCN16 and FCN8 incorporate the outputs of previous pooling layers, giving a more refined level of information, which leads to improvements in the prediction.

3.5.3 U-Shaped Networks

The proposal of Olaf Ronneberger [58] is one of the most employed architectures in medical image segmentation in different imaging modalities, including computed tomography. Ronneberger presented a fully convolutional encoder-decoder architecture for biomedical image segmentation.

The U-Net integrates a set of convolutional blocks. Each block contains two consecutive convolutional layers and a max-pool layer. In the encoding part of the network, each block

[4]https://cocodataset.org/
[5]https://www.image-net.org/

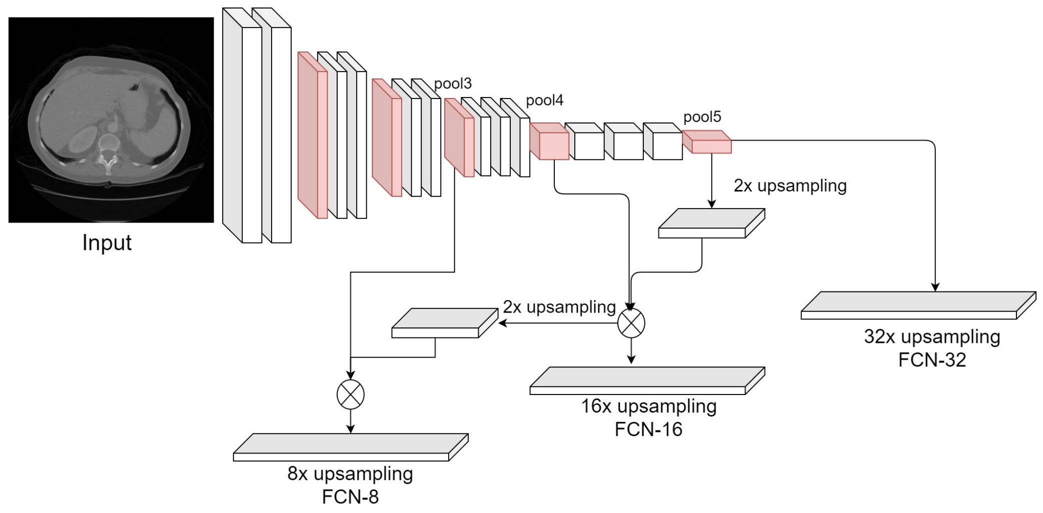


Fig. 3.12. A schematic representation of the different versions of FCN (FCN32, FCN16, and FCN8). Each version combines convolutional layers at a different resolution.

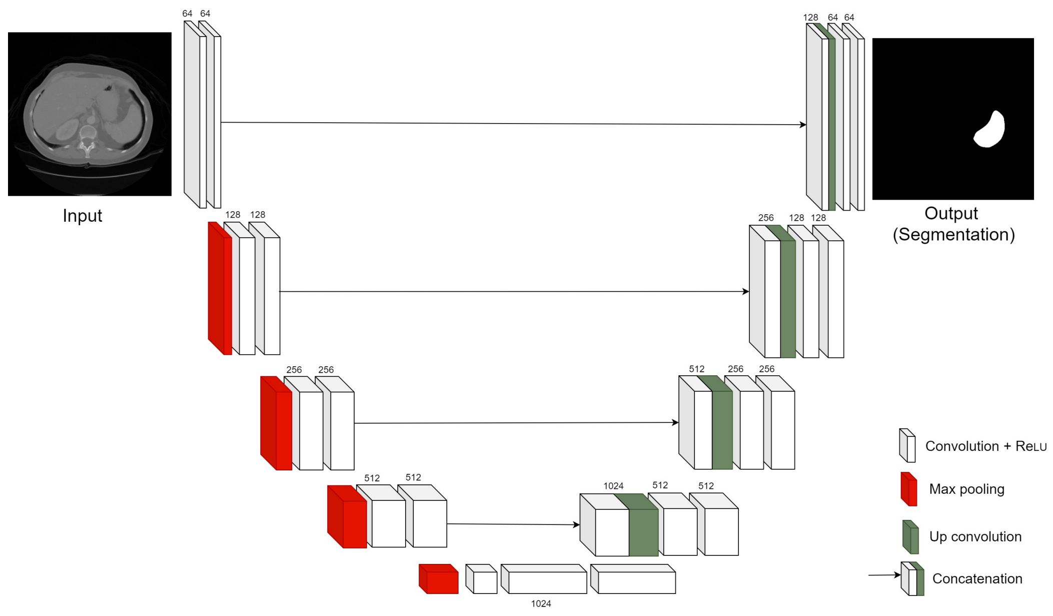


Fig. 3.13. A schematic representation of U-Net architecture.

halves the spatial dimension of the image and duplicates the number of feature channels. The expansive part of the U-Net replaces the max-pool layer for an up-convolution to recover spatial resolution. Similarly, the number of channels is duplicated in each convolutional block of the expansive part. Fig 3.13 shows the architecture of the network. The U-Net incorporates at the end a 1×1 convolution with a number of features equal to the number of segmentation classes. It integrates the feature maps from the last block of the model for pixel-wise prediction.

Another particularity of the network is the skip connections showed as arrows in Fig. 3.13. The skip connections concatenate the feature maps in the encoding part with their equivalent in the expanding path of the U-Net. In the original implementation, each convolution employs a 3×3 filter and takes the valid part of the convolution (no padding). It reduces the dimensions of the feature maps after every convolution. During the concatenation, the maps coming from the encoding part should be cropped to compensate for the loss in size. However, these are design choices that can be modified. While it is common to maintain the 3×3 convolution, it is also possible to add appropriate padding to generate an output of the exact dimensions as the input.

3.5.4 Dice Loss in Image Segmentation

The U-Net and FNC employes a pixel-wise loss function. In the case of the U-Net, it is a version of the weighted cross-entropy loss function. The general definition of the binary cross entropy loss is given by:

$$\mathcal{L}_{bce} = \sum_{i,j} w(i,j)(Y_{gt}(i,j)\log(Y_p(i,j) - (1 - Y_{gt}(i,j))\log(1 - Y_p(i,j)) \tag{3.11}$$

Considering an image I of dimensions $W \times H$ with ground truth mask $Y_{gt} \in \{0,1\}^{W \times H}$, and the predictions scores $Y_p = f(I, \theta)$, $Y_p \in [0,1]^{W \times H}$, with i and j iterating over the columns and rows of the images, respectively. In eq. 3.11, $w(i,j)$ is a weighting matrix for each pixels of the image. These loss metrics are standard in classification tasks. However, its application to segmentation can have drawbacks, especially when the amount of background pixels is large or the object of interest is small. Since these functions evaluate the error considering each pixel individually, a model can collapse into a trivial solution, predicting only background for all the inputs. One option is the careful weighting of the loss function. A more elegant solution is the formulation of an overlap metric as a loss function. In image segmentation, dice loss is one of the standard selections. It is a formulation of the dice score as a loss function. The dice score is a metric of similarity between two sets. In the case of image segmentation, it indicates the level of similarity between two binary segmentation maps I_{m1}, I_{m2}. It takes values between zero and one, and is defined as:

$$dsc = \frac{2|I_{m1}I_{m2}|}{|I_{m1}| + |I_{m2}|} \tag{3.12}$$

Where $|\cdot|$ holds for the number of non-zero pixels in the binary images. The formulation of the dice loss takes the complement of the dice score, i.e., $1 - dsc$. It is worth mentioning that

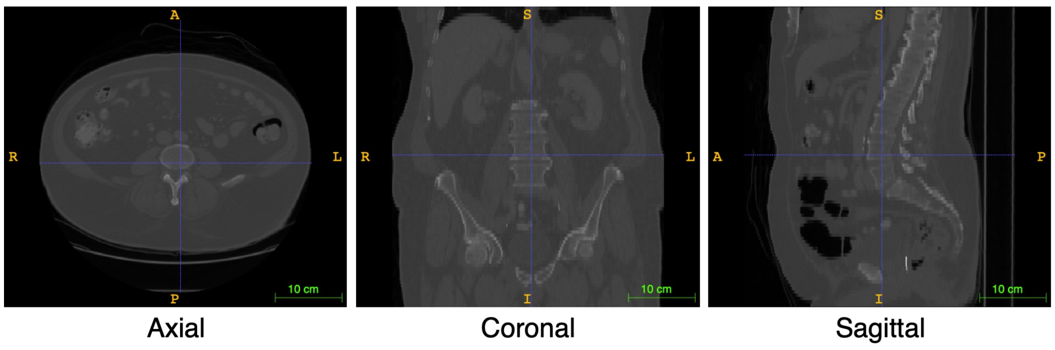


Fig. 3.14. A comparison of the three different views of a CT volume: Axial, Coronal, and Sagittal.

the dice loss employs a soft version of the dice score that compares the binary ground truth mask with the prediction scores of the CNN. The dice loss for the predictions is given by:

$$\mathcal{L}_{dsc} = 1 - \frac{2\sum_{i,j} Y_{gt}(i,j)Y_p(i,j)}{\sum_{i,j} Y_{gt}(i,j) + \sum_{i,j} Y_p(i,j)} \tag{3.13}$$

With i and j iterating over the columns and rows of the images, respectively.

3.5.5 FCN in the Medical Domain: Pancreas Segmentation

We can find examples of the application of FCN models in the pancreas segmentation problem. The pancreas is a small organ that presents high inter-patient variability in its shape compared with other organs, like the liver. It makes the segmentation of the pancreas in CT data a challenging task [22, 63]. The problem can be addressed as a 3D or 2D segmentation problem. Initial approaches employed 2D models and performed the segmentation in a slice-wise way to overcome the high GPU memory requirements of 3D models.

The initial adoption of deep learning models in pancreas segmentation followed hybrid approaches working on generated image patches [21, 60]. For example, Roth [60] proposed classifying image patches as pancreas, using a CNN. The patches were obtained based on the superpixels extracted from the input slice, using the simple linear iterative clustering algorithm [1]. The patches then feed a standard CNN for binary classification.

With the proposal of FCNs networks, researchers started to adopt this architecture in the medical segmentation problem. An application of the FCN-8 model can be found in [92]. In that work, a set of three 2D FCN-8 were trained to segment the pancreas in the different planes of the CT input (coronal, axial, and sagittal, see Fig. 3.14). The results are aggregated to produce an initial coarse segmentation of the organ, with a dice score of around 75%. The coarse segmentation is employed to generate a region of interest (ROI) around the pancreas. Finally, a second set of three FCN-8 perform an iterative segmentation inside the ROI to refine the segmentation, yielding a dice score of 82.3%. It is worth mentioning that the models in [92] train with the dice loss function.

A similar two-stage cascade approach is presented in [63], where holistically nested models are employed to generate a prediction of the pancreas' interior and boundaries over the three CT planes. The outputs create an ROI that encloses the organ. In the second stage, the score maps generated by the networks are employed to generate a superpixel over-segmentation of the ROI, that are finally aggregated using a random forest classifier.

Cascade methods can also be applied to 3D models. Two of the limitations of 3D models are the amount of data and memory required for training. One technique to deal with this issue is sub-sampling the input volume into smaller overlapping cubes. The work of [93] proposed a two-stage approach based on this strategy. The model in the first stage trains on a set of sampled sub-volumes obtained from the full-size input. This strategy allows reducing the memory requirements for training the network while producing a coarse segmentation. Again, an ROI is defined with the coarse segmentation. The second model trains to segment the organ in this lower-size input, allowing a dense number of sub-sampled volumes. The network proposed in both stages follows an encoder-decoder architecture with residual connection, similar to U-Net.

Single steps architectures have also been applied in pancreas segmentation. The work of Oktay [54] employs a single-stage method based on the 3D U-Net. Instead of employing a model to generate a coarse ROI, Oktay incorporates attention gates between the skip connections of the U-Net architecture. It allows the feature maps of the network to implicitly focus on relevant regions without explicit cropping of the input. Due to memory requirements, the inputs are downsampled to fit the hardware limitation.

3.6 Uncertainty Estimation

We have discussed the application of convolutional neural networks as predictive models. In this section, we discuss a method for uncertainty estimation of CNN. Uncertainty is inherent to probabilistic distribution. However, the current formulation of CNN follows a deterministic nature, given by the fixed model's weights. Those weights are learned in an optimization process, for example, backpropagation, with the objective to reduce the errors in the loss function. This can cause a bias towards the training set, causing the model to be overconfident about some predictions, not in the training distribution. Another consequence is the risk of overfitting the training set.

Before continuing with the discussion, we need to review some basic definitions of probability and Bayesian theory. For a discrete variable x, we define $p(x = \mathrm{x})$ as the probability of $\mathbf{x}$ being in state x. The value of $p(x = \mathrm{x})$ indicate our belief or the certainty that the state of the variable x is x [7]. A value $p(x = \mathrm{x}) = 1$ indicate a total confidence that the current state (or value) of the variable x. In contrast $p(x = \mathrm{x}) = 0$ tells us that the state of x is not x. Any other provability between 0 and 1 expresses the certainty of the given state. Considering set $\mathrm{dom}(x)$ as the set of all the possible states of x by definition we have that $0 \leq p(x_i) \leq 1$, for all $x_i \in \mathrm{dom}(x)$ and:

$$\sum_{x_i \in \mathrm{dom}(x)} p(x_i) = 1 \tag{3.14}$$

we are considering the expression $p(x_i)$ as a simplified version of $p(x = x_i)$. We call Eq. 3.14 the normalization condition and $p(x)$ the probability distribution of x.

For a pair of random variables x and y, we can express the probability of the intersection $x \cap y$ with the joint probability distribution $p(x, y) = p(y, x)$. For a pair of random variables x and y, we can express the probability of the intersection $x \cap y$ with the joint probability distribution $p(x, y) = p(y, x)$. The joint distribution expresses the probability of x being in state x at the time that y is in state y. The joint distribution also meets the normalization condition, and

$$\sum_{x_i, y_j} p(x_i, y_j) = 1 \tag{3.15}$$

with x_i and y_i iterating over dom(x) and dom(y), respectively. The extinction of the joint distribution to more than two variables is straightforward. Now that we have introduced the joint distribution, we can define two fundamental rules for the probabilities distribution [12]. The first rule is known as the sum rule or marginalization rule. Given a joint distribution $p(x, y)$, it is possible to eliminate, or integrate one of the variables by summing across their domain. The marginal distribution of $p(x, y)$ is given by:

$$p(x) = \sum_{y_j} p(x, y_j) \tag{3.16}$$

The second rule is called the product rule and relates the joint distribution with the concept of conditional probability:

$$p(x, y) = p(y|x)p(x) \tag{3.17}$$

The conditional distribution $p(y|x)$ expresses the probability of the variable y having previous knowledge about the variable x. It is read as "the probability of y given x. Note that $p(y|x)$ is a distribution over y. Setting the state of the x fix, the sum across the value of y is equal to 1. On the other hand, set y fix and integrating the values of x does not guaranty the sum will be equal to 1. If x and y are independets, then $p(x|y) = p(x)$ and $p(y|x) = p(y)$. In consequence, the joint distribution for independent random variables is given by $p(x, y) = p(x)p(y)$. Considering the product rule and the fact that $p(x, y) = p(y, x)$, we obtain the Bayes rule, which is defined as:

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)} = \frac{p(y|x)p(x)}{\sum_x p(y|x)p(x)} \tag{3.18}$$

The last formulations consider discrete variables. Continuous variables follow an analog formulation where the sums are replaced by integrals. For a continuous variable, $p(x)$ is a continuous function known as the probability density function . Similar to the discrete counterpart, the density function $p(x) \geq 0$ and

$$\int_{-\infty}^{\infty} p(x)dx = 1 \tag{3.19}$$

Similarly, for two continuous random variables x and y with joint density function $f(x, y)$, the marginal density is defined as:

$$p(x) = \int_{-\infty}^{\infty} p(x, y)dy \tag{3.20}$$

The product rule for continuous variables follow the same formulation as described by Eq. 3.17. The Bayes rule for continuous variables is then defined by:

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)} = \frac{p(y|x)p(x)}{\int_{-\infty}^{\infty} p(y|x)p(x)dx} \tag{3.21}$$

Considering the probability distribution (or density) $p(x)$ of a random variable x, we can also compute the expectation and the variance of some function $f(x)$ of the random variable with respect to its distribution. The expectation for the function $f(x)$ is defined as the average weighted by the probability distribution. For continuous random variables, it is defined as:

$$\mathbb{E}[f] = \int p(x)f(x)dx \tag{3.22}$$

for simplicity, we have omitted the range of the integral (but should cover the entire domain of x). The discrete version is equivalent, only replacing the integral with a sum. In both cases, if we consider a set of N samples of the random variable x, the expectation can be approximated with [12]:

$$\mathbb{E}[f] = \sum_{i=1}^{N} f(x_i) \tag{3.23}$$

The variance is an indicator of how much $f(x)$ varies around its mean. The variance is defined in terms of the expectation as:

$$\text{var}[f] = \mathbb{E}[f(x)^2] - \mathbb{E}[f(x)]^2 \tag{3.24}$$

Setting the identity function $f(x) = x$, we can obtain the expectation and variance of the random variable x. The variance can be also applied to pair of variables. For two random variables x and y, the covariance indicates the extent to which the variable vary together and is defined as [12]:

$$\text{cov}[x, y] = \mathbb{E}_{x,y}[xy] - \mathbb{E}[x]\mathbb{E}[y] \tag{3.25}$$

The concept of covariance can be extended to vectors composed of random variables. Given two $N-$dimensional random vectors $\mathbf{x}$ and $\mathbf{y}$, their covariance is a $N \times N$ matrix defined by:

$$\text{cov}[\mathbf{x}, \mathbf{y}] = \mathbb{E}_{x,y}[\mathbf{x}\mathbf{y}^T] - \mathbb{E}[\mathbf{x}]\mathbb{E}[\mathbf{y}^T] \tag{3.26}$$

if is common to use the notation σ^2 to represent the scalar varaince, and K or Σ for the covariance matrix.

3.6.1 Probabilistic Models

Knowing the model's uncertainty can be crucial for some applications. We can take, for example, the tasks of autonomous driving and medical applications. In those applications, the model needs to detect pedestrians or life-threatening diseases. Given that a wrong prediction can have serious consequences, it is indispensable to know when the model is unsure about its predictions.

Uncertainty is inherent to probability. In order to have a confidence estimation of a model, it is necessary to define it from a probabilistic perspective. Instead of directly taking the model's output, we can represent the uncertainty in the prediction with a random noise over the model's output. The random noise then follows a probability distribution $p(y)$ that depends on the input and model's prediction. We also need to express the model $p(f)$ in a probabilistic setting. As discussed, models employ a training set $D = (\mathbf{X}, \mathbf{Y})$ with a collection of samples $\mathbf{X}$ and corresponding labels $\mathbf{Y}$. The learning process consists in using the training set to finding a set of parameters w that minimize the loss function. We can consider a distribution overt he weights $\mathbf{w}$, that depends on the training set. Then, defined a distribution over the model's weights, we can define the predictive distribution of y for an input x as [25, 26]:

$$p(y|x, \mathbf{X}, \mathbf{Y}) = \int p(y|x, \mathbf{w}) p(\mathbf{w}|\mathbf{X}, \mathbf{Y}) d\mathbf{w} \tag{3.27}$$

the last formulation expresses the dependence of the prediction y on the input and the parameter selection. At the same time, the distribution of parameters is conditioned to the training set. The inference process in Eq. 3.27 requires computing the marginal with respect to the parameter set w. It is similar to aggregate the predictions of potentially infinite models (as defined by each parameter).

It is common to define $p(y|x, \mathbf{w})$ to be a Gaussian distribution, with mean given by the model $\mu = f(x; \mathbf{w})$ and variance σ^2. In this case, it is considering that the predictions are observed through Gaussian noise. The Gaussian distribution $\mathcal{N}(x|\mu, \sigma^2)$ is defined in terms of its mean and variance as:

$$\mathcal{N}(x|\mu, \sigma^2) = \frac{1}{(2\pi\sigma^2)^{1/2}} \exp\{-\frac{1}{2\sigma^2}(x-\mu)^2\} \tag{3.28}$$

the expectation of the Gaussian distribution is given by the μ and its variance by σ^2. In some cases, σ is expressed in terms of its inverse as β^{-1}, and called the precision parameter[12]. The vectorial version of the Gaussian distribution is.

$$\mathcal{N}\mathbf{x}|\mu, \Sigma) = \frac{1}{(2\pi)^{N/2}} \frac{1}{(|\Sigma|^{1/2}} \exp\{-\frac{1}{2}(\mathbf{x}-\mu)^T \Sigma^{-1} (\mathbf{x}-\mu)\} \tag{3.29}$$

where N is the number of elements of the input vector, Σ represents the covariance matrix $\text{cov}(\mathbf{x}, \mathbf{x})$. Putting all together, the distribution $p(y|x, \mathbf{w})$ in Eq. 3.27 can be chosen to be:

$$p(y|x, \mathbf{w}) = \mathcal{N}(x|y(x; \mathbf{w}), \beta^{-1}) \tag{3.30}$$

The distribution of the parameters $p(\mathbf{w}|\mathbf{X}, \mathbf{Y})$ in Eq. 3.27 can be obtained following the Bayes rule. It leads to the following expression:

$$p(\mathbf{w}|\mathbf{X}, \mathbf{Y}) = \frac{p(\mathbf{Y}|\mathbf{X}, \mathbf{w}) p(\mathbf{w})}{\int p(\mathbf{Y}|\mathbf{X}, \mathbf{w}) p(\mathbf{w}) d\mathbf{w}} \tag{3.31}$$

the expression $p(\mathbf{w}|\mathbf{X}, \mathbf{Y})$ is known as the posterior distribution of $\mathbf{w}$, and indicates the probability distribution of the models with parameters $\mathbf{w}$ under the evidence $D = (\mathbf{X}, \mathbf{Y})$. Similarly, $p(\mathbf{Y}|\mathbf{X}, \mathbf{w})$ is known as the likelihood, and it defines the probability for a particular model with parameters $\mathbf{w}$ to generate the training labels, considering the training inputs. The posterior distribution is intractable, specially for complex non-linear models like CNNs [12, 26]. For this reason, it is common to employ optimization and approximation methods for estimating the intractable posterior.

Variational inference is often used to approximate the posterior. It consists in employing a parametric distribution $q(w)$ with parameters Φ, and find the set of parameters that minimize a similarity metric between q and the original distribution p. The similarity between the approximation and the target distribution is given by the Kullback-Leibler diverge:

$$D_{KL}(q||p) = \int q(\mathbf{w}) \log(\frac{q(\mathbf{w})}{p(\mathbf{w}|\mathbf{X},\mathbf{Y})}) \tag{3.32}$$

Since the KL divergence also requires computing the posterior, in practice it is not explicitly optimized. Instead, the posterior is approximated by maximizing the evidence lower bound, also known as ELBO:

$$ELBO = \int q(\mathbf{w}) \log(\frac{p(\mathbf{w},\mathbf{X},\mathbf{Y})}{q(\mathbf{w})}) d\mathbf{w} = \log(p(\mathbf{X},\mathbf{Y})) - D_{KL}(q||p) \tag{3.33}$$

The distribution $\log(p(\mathbf{X},\mathbf{Y}))$ in the right of the ELBO does not depend on the prior or the parametric distribution q, and hence it is fixed in the optimization process. Therefore, the choice of the parameters of q that maximize the expression in the integral in Eq. 3.33 will also indirectly minimize the KL divergence [12, 26]. The work of Gal shows that modern CNN models, trained with dropout layers, maximize the ELBO for a particular selection of probability and parametric distributions. It indicates that a CNN can be equivalent to approximating a deep Gaussian process, and hence, brings the possibility to compute the uncertainty for deep ANNs. The process proposed by Gal [25, 26] to estimate the model's uncertainty is known as Monte Carlo Dropout.

3.6.2 Monte Carlo Dropout

This thesis performs the uncertainty analysis of a U-Net model using the Monte Carlo dropout strategy. We will briefly describe this method and mention some medical applications of uncertainty. Yarin Gal proposed Monte Carlo dropout in [26] as a strategy to approximate a probabilistic model with CNNs that trains with dropout layers. Considering the classification learning problem, a Gaussian process for classification can be formulated as:

$$\begin{aligned} F|X &\sim \mathcal{N}(\mathbf{0}, K(X,X)) \\ Y|F &\sim \mathcal{N}(F, I_N) \\ c_n|Y &\sim \text{Categorical}(\text{Softmax}(y_n)) \end{aligned} \tag{3.34}$$

with F representing the set of possible functions distributed normally with K the covariance function, $X \in \mathbb{R}^{N\times Q}$ is a random matrix containing a set of N Q-dimensional inputs, and $Y \in \mathbb{R}^{N\times D}$ is the random output matrix composed of N vectors y_n of dimension D, and c_n defines the observed class label. The predictive distribution is given by a Categorical distribution defined over a Softmax function evaluated on the output matrix Y. Gal defined an approximation for the covariance function given by the covariance matrix:

$$\hat{K}(x_i, x_j) = \frac{1}{K}\sum_{k=1}^{K} \sigma(w_k^T x_i + b_k)\sigma(w_k^T x_j + b_k) \tag{3.35}$$

with w_k, b_k samples of random parameters with distributions $p(w)$ and $p(b)$, respectively. This definition leads to formulating the probabilistic model as:

$$\begin{aligned} F|X, W_1, b &\sim \mathcal{N}(\mathbf{0}, \hat{K}(X,X)) \\ Y|F &\sim \mathcal{N}(F, I_N) \\ c_n|Y &\sim \text{Categorical}(\text{Softmax}(y_n)) \end{aligned} \tag{3.36}$$

with W_1 a $Q \times K$ matrix of parameters. We can obtain the predictive probability given a set of inputs by computing the marginal distribution over the variables F, W_1, Y, and b as:

$$\begin{aligned} p(c|X) &= \int p(c|Y)\Big(\int p(Y|F)p(F|W_1, b, X)p(W_1)p(b)dFdW_1db\Big)dY \\ &= \int p(c|Y)p(Y|W_1, b, X)p(W_1)p(b)dW_1dbdY \end{aligned} \tag{3.37}$$

The expression in the second line is obtained by integrating over F. Additionally, Gal introduced an auxiliary random variable $W_2 \sim \mathcal{N}(\mathbf{0}, I_K)$ of dimensions $K \times D$, yielding the following expression:

$$p(c|X) = \int p(c|Y)p(Y|W_1, W_2, b, X)p(W_1)p(b)dW_1dW_2dbdY \tag{3.38}$$

The posterior distribution of the model's parameters W_1, W_2, b is intractable. A common approach to deal with the intractable distribution is to condition the model on a finite set of random variables [25]. For it, a variational distribution q is defined, and the Kullback-Leibler (KL) divergence between the approximation and intractable posterior is minimized. Minimizing the KL divergence is equivalent to maximizing the log evidence lower bound (ELBO). Considering the distribution of eq. 3.38, the ELBO is defined as:

$$\text{ELBO} := q(\omega)\log p(c|Y)p(Y|\omega, X) - KL(q(\omega)||p(\omega)) \tag{3.39}$$

for simplicity, we wrote $\omega = \{W_1, W_2, b\}$. Gal employed the distributions

$$\begin{aligned} W_1 &= \mathbf{z_1}M_1, \text{ with } \mathbf{z_1} = \text{diag}(z_i), i = 1, \dots, Q \\ W_2 &= \mathbf{z_2}M_2, \text{ with } \mathbf{z_2} = \text{diag}(z_j), j = 1, \dots, K \\ b &= m \end{aligned} \tag{3.40}$$

with $\mathbf{z_1}$, $\mathbf{z_2}$ diagonal matrices with elements $z_i \sim$ Bernoulli$(p1)$, and $z_j \sim$ Bernoulli$(p2)$, respectively, M_1, M_2, m variational parameters, for approximating the intractable posteriors. It leads to log evidence lower bound given by (the details can be found in [25]):

$$\begin{aligned} ELBO &:= \frac{1}{N}\sum_{n=1}^{N}\log p(c_n|\hat{y_n}) - \frac{p_1}{2N}||M_1||_2^2 - \frac{p_2}{2N}||M_2||_2^2 - \frac{1}{2N}||m||_2^2 \\ \hat{y}_n &= \sqrt{\frac{1}{K}}\sigma\Big(x_n(\hat{z}_1^n M_1) + m\Big)\hat{z}_2^n M_2) \end{aligned} \tag{3.41}$$

Which is equivalent to train a two-layered neural network model with dropout applied and with regularisation over the network's weights. This results can be extended to multi-layered convolutional neural networks.

In inference time, any model trained under the last conditions can be employed to obtain an approximation of the predictive probability by using [26, 39]:

$$p(y = c|x) \approx \frac{1}{T}\sum_{t=1}^{T} \text{Softmax}(f(x, \theta_t)) \tag{3.42}$$

with $f(x, \theta)$ a CNN trained with dropout layers, and θ_t the model's weights with dropout applied during inference time. This equation is equivalent to integrate T samples from the predictive distribution (Monte Carlo integration). Once we have the approximated predictive distribution, it is possible to employ the entropy of the distribution as an indicator of uncertainty.

3.6.3 Medical Applications of Uncertainty Analysis

Knowing what the network does not know has several advantages. For example, in the semi-supervised learning problem. Semi-supervised learning defines a set of methods that employ labeled and unlabeled data for model training. The data annotation process can be complex in the medical domain, given that it requires experts in the process. It leads to creating datasets where only a portion of the samples are properly annotated for training. Given the necessity of large data availability for CNN training, it is essential to develop methods for including the unlabeled data in the training process. Yu [88] addressed the problem of semi-supervised left atrium segmentation considering the model's uncertainty. The method proposed in [88] employs as base the student-teacher strategy, where a teacher model provides the student model with pseudo-labels for the unlabeled data. The input image is augmented with noise and passes to both the student and teacher. Then, the objective is to enforce the consistency between the teacher and the student predictions. To reduce the effect of potential segmentation errors in the teacher's predictions, Yu proposed to include the uncertainty estimations in the consistency loss function to consider only the high confidence points. This strategy leads to improvements in the result, suggesting the uncertainty can bring information about the correctness of the model's predictions. A similar approach is employed in [86] for the organ segmentation problem, where a set of N models train on N different views of the data. To generate pseudo labels for a particular model i in the set, the method uses an uncertainty-aware label fusion between the outputs of the $N - 1$ remaining models.

The work of Dias [20] explores the use of model uncertainty combined with a region growing approach for the refinement of object boundaries in real-world image segmentation. Again, the method considers the confidence scores for defining potential errors in the segmentation process. It shows the potential of uncertainty analysis in confidence-driven segmentation refinement. In the medical context, the work of Nair [51] explores four uncertainty measurements computed using the MCDO approach. The measurement considered includes the predictive variance, the MC sampling variance, the predictive entropy, and the mutual information of a model trained for lesion detection and segmentation in multiple sclerosis imaging data. The results showed that excluding uncertainty predictions in the predictions improves the overall performance on the remaining points, especially for small lesions. The use of uncertainty for prediction's quality estimation for brain segmentation is explored by Roy [64]. It is showed that the uncertainty measurements obtained with MCDO correlate with the accuracy of the predicted segmentation.

The uncertainty analysis of recent CNN can give us clues about the correctness of the predictions. Similarly, the MCDO strategy allows obtaining relevant information about uncertainty without modifying or retraining the model. Inspired by these facts, this thesis proposes an uncertainty-driven refinement strategy for organ segmentation, where high confidence prediction drives the refinement process. However, we still need to describe the mechanism employed to exploit this information. For it, we rely on a recent approach used to exploit the non-euclidean relationships between the data points. In the next chapter, we describe the graph convolutional neural networks, a learning model that brings the ideas of CNN into the graph domain.

4 Deep Learning on Graph Domains

In the last chapter, we introduced deep learning as a learning model for prediction over images. The intensity information contained in an image follows a regular grid structure, and a unique euclidean coordinate can identify each pixel. Three-dimensional imaging modalities, like computed tomography, follow a similar representation, represented as a discrete sub-set of the 3D euclidean plane. We have described ANN models that work with vectorial and image data. MLP operates on the vectorial representation of the information. Before CNN's, image classification required the extraction of a feature vector representing the information contained in the image. Then, this vector can be processed by a learning model, like MLP. In contrast, CNN allows processing the information coming directly from the image, providing a trainable feature extraction that allows end-to-end training.

In this chapter, we will discuss the use of graphs as a representation of data. The graphs have the advantage of bringing information about the relationships between the data points, something that can be lost with a regular grid description. Similarly, the data follows a natural graph-like representation in some cases. For example, the users of a social network can be presented as a graph based on friendship relations. With the success of CNNs in image analysis, different works have proposed forms to employ convolution-like learnable filters on graph data. However, it brings challenges derived from the inherent differences between regular grid representations and the non-euclidean nature of the graphs. We will review the basics of graph definition and present some of the recent works on deep learning on graphs.

4.1 Graphs Structures

A graph is a data structure composed of a set of nodes grouped in a pairwise way. The term vertex is also a common name to refer to the nodes. Each pair of nodes define an edge in the graph, that represent the relationships between the nodes. Formally, we define a graph $\mathcal{G} = (X, \mathcal{E})$ with $X = \{x_1, x_2, \dots, x_N\}, x_i \in \mathbb{R}^m$ A set of N Nodes, and $\mathcal{E} \subset X^2$ The set of edges that relates the nodes with $|\mathcal{E}|$ the total number of edges. Fig. 4.1-a.1) and 4.1-b.1) present examples of the graphical representation of this structure. In the graphical representation, nodes are presented as circles connected by lines that indicate the edges. It is possible to distinguish between two kinds of graphs, considering the connections between the nodes. Row a) in Fig. 4.1 represents an undirected graph, where the edges defined between two nodes does not consider any specific direction, representing a two-way relationship between nodes. A straight line represents these connections. A social network can be a real-world example of an undirected graph. Considering each individual in a social network as a node, the friendship relations represent an undirected connection between individuals. It represents a mutual connection where if individual A has individual B as a friend implies that the second direction is also true (individual B has individual A as a fiend). On the other

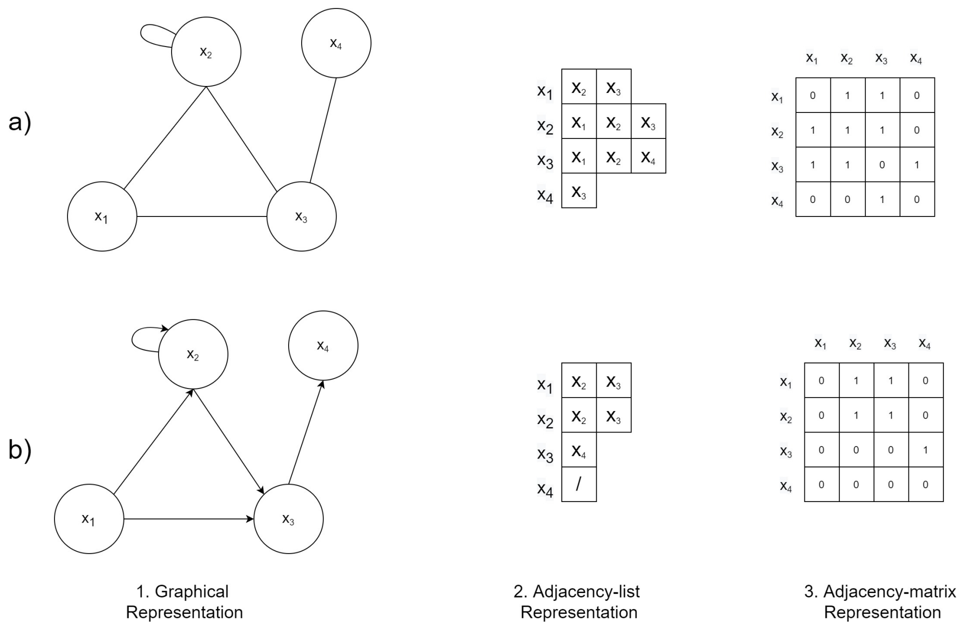

a)
b)
x1 x2 x3 x4
1. Graphical Representation
2. Adjacency-list Representation
3. Adjacency-matrix Representation

Fig. 4.1. Different representations of a) an undirected and b) a directed graph. The 1. graphical, 2. list and 3. adjacency matrix representations are presented. Note the change of the number of edges and the representations between a) and b).

hand, a directed graph is composed of directed nodes and represents a single-way relationship between their corresponding nodes (Fig. 4.1 row b)). In this case, an arrowed line represents the direction of the relations between two nodes. We can use a road network as an example of a directed graph. If we define the places in a city (e.g., houses and buildings) as the nodes of the graph, we can consider the roads and streets as the edges. Since some roads can be single-way, they define a directed graph, where the arrows indicate the valid traffic flow in the road network.

4.1.1 Adjacency List and Adjacency Matrix

Working with graphs requires a proper representation of its elements that allows the algorithmic formulation of graph-based methods and their computational implementation. In the literature [17], the nodes' relationships defined by a given graph can be described with an adjacency list (Fig. 4.1-a.2 and 4.1-b.2) or an adjacency matrix (Fig. 4.1-a.3 and 4.1-b.3).

The adjacency list representation of a graph employs an array L composed of N lists, one per node. For example, for a particular node x_i, its corresponding entry $L(i)$ will contain all the neighboring nodes of x_i. The second column of Fig. 4.1 shows examples of the adjacency list for undirected (row a) and directed (row b) graphs. Note how the different lists in the array have different lengths. Similarly, even when the graphs have the same number of nodes and connections, making the edges directed (as in Fig. 4.1 row b) modify the content of the adjacency list. In a undirected graph, if $x_j \in L(x_i)$ implies that $x_i \in L(x_j)$, while in a directed graph, this does not necessarily hold, and having $x_j \in L(x_i)$ can be interpreted as a connection from the node x_i to x_j.

The adjacency matrix A of dimensions $N \times N$ is a binary matrix where the entry $A(i, j)$ equals 1 if the edge $(x_i, x_j) \in \mathcal{E}$, and 0 otherwise. Again, the third column in Fig. 4.1 illustrates the adjacency matrix representation for an undirected (row a) and a directed (row b) graph. We can observe in the figure that the undirected graph has a symmetric adjacency matrix. The self connections (connections to the same node) are represented in the diagonal of the matrix.

Each representation has advantages and drawbacks. For example, the adjacency list can be a memory-efficient representation when working with sparse graphs, i.e., when the number of edges $|\mathcal{E}|$ is considerably lower than the number of nodes N [17]. However, knowing if particular edges exist in the graph requires performing a list search over the corresponding neighborhood list, which can be computationally inefficient. The last point is easily obtained with the adjacency matrix since it is only necessary to consult the corresponding entry for the edge. Still, a disadvantage of the adjacency matrix can be memory requirements.

4.1.2 Graphs as a Representations for Data

Deep learning (DL) and machine learning (ML) models operate on input data, where the information is represented numerically with a vector $x_i \in \mathbb{R}^m$, which belong to a more extensive collection $X \in \mathbb{R}^{N \times m}$. This mathematical definition is not much different from the formulation of nodes presented in the previous sections. The main difference between the data employed by the ML models and the graph representation is the existence of links or edges.

ANN models learn to extract meaningful information from the input features, either vector or images. However, they consider every data point individually when training. Relationships between the inputs are not considered in the standard ANNs. For example, consider the users in a social network. User information and their interactions with different websites might be available for developing a learning model. We can also assume the task of predicting either a user will like or not a product. Standard models like ANN might employ the information from individual users to train and make predictions. However, this is not the only information available. ANN models ignore the relationships between users in the social network. Relationships, like friendship, can bring additional information to the model for making better predictions. Friends and contacts in a social network might share interests. Hence, if a given user already liked the product, it is also probable that their connections would be interested in it. A single vectorial representation of the users does not include these relations. However, a graph composed of users as nodes, and friendship as edged is a natural representation of the social network. There exist different problems that can be complemented by including the relations between the individuals. Examples of it can be point cloud analysis in computer vision or protein interaction in biomedicine. Another example is the brain functional or structural networks presented by connections between corresponding brain regions in functional or structural MRI [13, 90].

Graphs brings additional information that machine learning algorithms do not consider in a standard setting. In this sense, each node in the graph can be related to a feature vector describing the data. Regarding the edges, it is possible to include complementary information about the connections between two nodes. It is usually denominated as edge weighting and can represent, for example, the importance that a determined link has for a particular node. This information can be represented in the adjacency matrix by replacing the non-zero with the corresponding value of the weight.

Given the success of the convolutional neural networks, it has been an interest in replicated the key components of CNNs in the graph domain. However, it involves different challenges mainly derived from the non-Euclidean nature of the graphs. Deep learning models operate on data that can be defined in the Euclidean domain. For example, image intensities can be assumed to be a function defined on a subset of the Euclidean plane. This definition brings a set of properties like global parametrization, a common coordinate system, or a vectorial structure [13]. These properties are not present in the graph domain, making regular DL operations, like the convolution, not well defined for graphs. The efforts for extrapolating the concepts employed in deep learning on image data into graphs has led to the definition of the graph convolutional networks as a set of learning models capable of working on graph data.

4.2 Graph Convolutional Neural Networks

The capabilities of graphs for representing relations between two data points or observations have raised the interest for implement CNN-like approaches on non-euclidean domains. Graphs can help to describe networks, interactions, or similarities between two objects [13]. On the other side, CNNs can extract local features at different image locations. By stacking different layers, the model is capable of evaluating features in a larger context, by increasing its receptive field. Apart from the localized filters, CNNs are shift-invariant and robust to translations of the input. However, the well-known convolution operator defined on images, cannon be directly applied to the graphs, given the lack of a standard coordinate system.

Let's take, for example, the graphs presented in Fig. 4.2. The image in Fig. 4.2a) shows a graph representation of a regular grid-like structure. It can define a graph representation of an image, where nodes are given by pixels in the image and edges are defined based on spatial adjacency. The blue links represent an N4 connectivity scheme, where edges are defined only between perpendicular neighbors of the graph. The edge set composed of both blue and yellow lines defines a N8 connectivity, which also includes diagonal connections. An example of a local (one-hop) neighborhood is represented with the central node in green and their neighbors in blue. This structure is regularly repeated across every node. In an N8 connectivity scheme, all nodes will have exactly 8-neighbor, which corresponds to the spatial adjacent nodes. A 3×3 convolutional kernel will operate in neighborhoods like the one presented in this graph. Changing the target node will not modify the way the convolution is applied, thanks to the regularity of the input.

Now, let's focus in the graph in Fig. 4.2b). The graph has the same number of nodes as before, but the edges no longer follow the spatial relation of the nodes, and hence, different nodes would have a different number of neighbors. For example, node1 in Fig. 4.2b) has three connections, while node 2 has only one. It makes it difficult to define localized convolutional filter operation, like in regular grid structures [18]. A possible approach to defining the convolution operation is performing the filtering in the spectral domain. The convolution

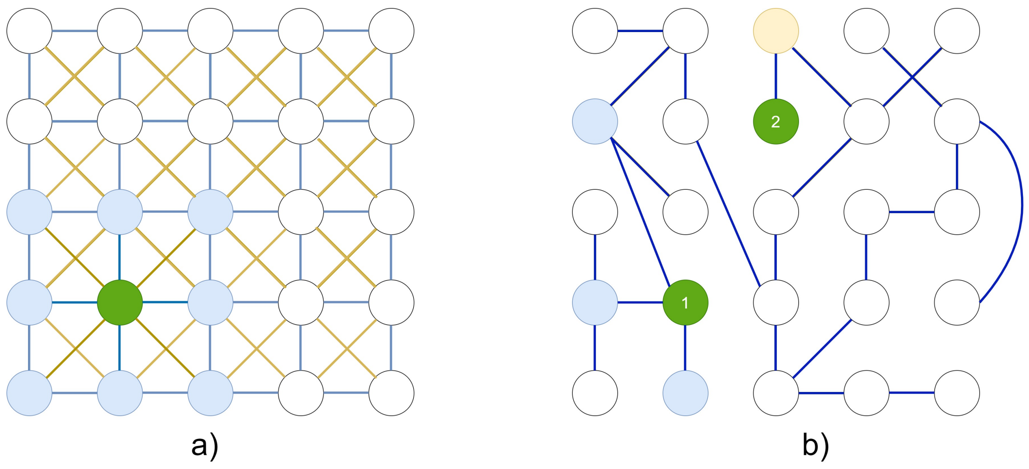


Fig. 4.2. Graph representations of a) a regular grid structure and b) an irregular representation of node relationships. A sample node (in green) is represented together with its neighborhoods (in blue and yellow). Figure a) can describe graph representation of an image with the pixels as nodes and spatial neighborhood relations as edges. The local neighborhood structure in a) is fixed, allowing the definition of kernel base operations like the convolution. The graph presented in b) is a typical structure of a real-world graph. The neighborhood is no longer regular (e.g., node 1 has three connected nodes, while node 2 has one). It makes it complicated to define a convolution operator that can work in a local neighborhood (localizer operator).

operation simplifies into an element-wise multiplication in the spectral domain. The Fourier transform of the graph is obtained through the eigenvalues of the graph laplacian.

This section focuses on a model for exporting the convolution operation into graphs, considering the Fourier transform to compute the convolution in the spectral domain. Such approaches are referred to as spectral graph convolution methods and, in general, convolutional models defined on graphs are known as graph convolutional networks (GCN). The GCN employed in this thesis was proposed by Thomas Kipf [40] for semi-supervised learning in a node classification task.

4.2.1 Graph Spectral Convolution

The Fourier transform of the graph is defined with base on the eigendecomposition of the normalized graph Laplacian L:

$$L = I_n - D^{-1/2} A D^{-1/2} \tag{4.1}$$

With A the $n \times n$ adjacency matrix, I_n the $n \times n$ identity, and the diagonal $n \times n$ matrix D represents the degree matrix, where the diagonal elements are defined as $D_{i,i} = \sum_j A_{ij}$. The Laplacian can be decomposed in a diagonal matrix of eigenvalues Λ and eigenvectors matrix U, such that $L = U \Lambda U^T$. If we consider a single-value feature defined on every node, the set of n nodes of the graph can be represented by a vector $\mathbf{x} \in \mathbb{R}^n$. Then, the following equation defines the Fourier transform of $\mathbf{x}$:

$$\hat{f} = U^T \mathbf{x} \tag{4.2}$$

Multiplying the last expression by U returns $\mathbf{x}$ to its original domain. Considering a filter $g_\theta = diag(\theta)$ defined in the spectral domain, with $\theta \in \mathbb{R}^n$ the filter's parameters, the convolution of $\mathbf{x}$ with g is obtained by computing the Fourier transform of $\mathbf{x}$, multiplying with g and returning to the original input's domain. This process is summarized by the following expression:

$$g_\theta * \mathbf{x} = U g_\theta U^T \mathbf{x} \tag{4.3}$$

Unfortunately, computing the eigendecomposition of the Laplacian is computationally expensive for large graphs. It is usual to approximate the convolution using a polynomial expansion [40] to address this problem. Considering the filter g_θ as a function of the eigenvalues Λ we can express the filter as $g_\theta(\Lambda)$. Then, the spectral filter can be approximated by a truncated expansion of the first K^{th} order Chebyshev polynomials $T_k(\Lambda)$:

$$g_{\theta'}(\Lambda) \approx \sum_{k=0}^{K} \theta'_k T_k(\tilde{\Lambda}) \tag{4.4}$$

In this definition, θ' represents a vector in $\mathbb{R}^K$ of Chebyshev coefficients, and $\tilde{\Lambda} = \frac{2}{\lambda_{\text{max}}}\Lambda - I_n$ are the rescaled eigenvalues, where λ_{max} represents the largest eigenvalue of the Laplacian L. Finally, each Chebyshev polynomial is recursively defined as:

$$T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x), \text{with } T_0(x) = 1, T_1(x) = x \tag{4.5}$$

Kipf defined a simplified version of the polynomials expansion, we will discuss this version of the GCN in the next section.

4.2.2 Graph Convolutional Layer

Consider the approximation of the convolution given by Eq. 4.4 and the spectral convolution presented in Eq. 4.3. Combining both expressions and taking $U\lambda^k U^T = (U\lambda U^T)^k$ allow us to write the polynomials in terms of the Laplacian L, leading to the following approximation of the spectral convolution:

$$g_{\theta'} * x \approx \sum_{k=0}^{K} \theta'_k T_k(\tilde{L})x \tag{4.6}$$

Now, with $\tilde{L} = \frac{2}{\lambda_{\text{max}}}L - I_n$. Additionally, Kipf limited the polynomial expansion to $K = 1$, taking a linear function with respect to the Laplacian and approximated the largest eigenvalue $\lambda_{\text{max}} \approx 2$. Considering these two last point, and expanding the sum in Eq. 4.6 we get:

$$g_{\theta'} * x \approx \theta'_0 x + \theta'_1 \tilde{L} x = \theta'_0 x + \theta'_1 (L - I_n)x \tag{4.7}$$

Considering the definition of L given in Eq. 4.1, the convolution can be expressed in terms of the degree and adjacency matrices as:

$$g_{\theta'} * x \approx \theta'_0 x - \theta'_1 (D^{-1/2} A D^{-1/2})x = \theta(I_n + D^{-1/2} A D^{-1/2})x \tag{4.8}$$

where the expression $\theta(I_n + D^{-1/2}AD^{-1/2})x$ comes from restricting $\theta = \theta'_0 = -\theta'_0$, leading to a model defined by a single parameter. To avoid numerical instabilities during the model training, the authors introduced a renormalization that defines the adjacency matrix as $\tilde{A} = A + I_n$. and the diagonal elements of the degree matrix are now defined as $\tilde{D_{ii}} = \sum_j \tilde{A}_{ij}$. Whith this, the expresion $(I_n + D^{-1/2}AD^{-1/2})$ simplies to $(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2})$. The previous formulation assumes a single-valued feature and a single parameter, but it can be extended to features of any size, giving the following definition of a graph convolutional layer:

$$f(X;\Theta) = \sigma(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}X\Theta) \tag{4.9}$$

Where $\tilde{A}$ and $\tilde{D}$ are the renormalized adjacency and degree matrix, σ is a non-linear function, X is now a $N \times F_0$ input feature matrix modeling a set of N nodes each represented by a feature vector of dimension F_0, and Θ is a $F_0 \times F_l$ matrix of learnable parameters, employed to compute a linear transformation of the input features. For simplicity, we can write $\hat{A} = \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$, and write the convolutional layer as:

$$f(X, A; \Theta) = \sigma(\hat{A}X\Theta) \tag{4.10}$$

The adjacency matrix allows aggregating the feature representation of the one-hop (direct connections) neighborhood for a particular node, and stacking different convolutional layers, allows considering a neighborhood at larger hops. For node classification purposes, we will count with a training target vector Y with dimensions $N \times C$, with C the number of classes. Then, the last graph convolutional layer can then output a matrix with the exact dimensions as Y, which feeds a softmax non-linear output. For example, a two-layered GCN for node classification can be defined as:

$$Y' = f(X, A; \Theta_0, \Theta_1) = \text{softmax}(\hat{A}\sigma(\hat{A}X\Theta_0)\Theta_1) \tag{4.11}$$

with X a $N \times F_0$ node's feature matrix, Θ_0 and Θ_1 are $F_0 \times F_1$ and $F_1 \times C$ feature matrices. As in traditional CNNs, training a GCN requires minimizing the error between the ground truth matrix Y and the predicted output Y'. It can be done employing standard loss functions, like cross-entropy loss. Finally, notice that the same matrix $\hat{A}$ appears on both convolutional layers. It indicates that we only need to compute this matrix once during the training and evaluation process.

4.3 Medical Applications of GCNs

The graph convolutional networks are a relatively recent approach that has found applications for learning on non-euclidean real-world data. For example, Kipf tested the model presented in Eq. 4.11 with data from the Citeseer, Cora, and Pubmed datasets. These datasets contain citation networks where a set of document-related features gives the graph's nodes, and the citation links between the documents represent the edges. The addressed task is semi-supervised document classification (which traduces to node classification). The adoption of these methods in the medical community has recently started. Considering the segmentation problems, GCN models have been suggested for improving the segmentation of tubular-shaped structures.

For example, the work proposed by Selvan [69] presents a graph autoencoder for the extraction of airways. The CT scans containing airway information are pre-processed into a graph structure. The graph generated is over-complete, and the objective is to employ the graph autoencoder to recover the ground truth adjacency matrix containing the structures of interest. The autoencoder comprises a GCN for computing node embeddings that then feeds a non-linear decoder given by a radial basis function operating over pairs of node embedding.

Similarly, the work presented by Shin [70] considers the graph-like structure of vessels in the segmentation process. Shin proposed the inclusion of a GCN network for complementing the information obtained from a regular CNN architecture for segmentation in retinal and coronary artery datasets. The CNN gives an initial vessel segmentation together with a vessel probability map. This information is further processed by a two-layered GCN. The input for the GCN is a graph constructed from the initial vessel segmentation map. The final results are given by a convolutional inference module that takes the combined information from the CNN and GCN models.

Juarez [36] presents a combined architecture for the segmentation of airways in chest CT scans. The proposal is composed of a U-Net with the bottleneck replaced by a GCN. The graph's connectivity is done considering the feature map of the last convolution of the U-Net's encoding step. Each voxel of this feature map is taken as a node in the graph. The vertices are connected considering either the local 26 surrounding neighborhood of a central voxel or the 26 closest points considering the feature vector of each node.

Inspired by the works and the recent advances in uncertainty estimation, we propose a graph-based refinement approach for segmentation. Our method is described in the following chapters. We believe that our work new example of the applications of GCNs and uncertainty analysis in image segmentation and brings a novel contribution to the medical image analysis community.

Part II

Uncertainty-Driven Graph Convolutoinal Network for Segmentation Refinement

5 Uncertainty-Driven Graph Convolutional Neural Network for Segmentation Refinement

In the previous chapters, we discussed different concepts related to deep learning models, including FCN for image segmentation, uncertainty analysis on deep models, and we introduced the idea of spectral graph convolutional neural network for applying CNN ideas into non-euclidean domains, like graphs. Now, we introduce a novel segmentation refinement method for CNNs, that incorporates these ideas into a single framework.

Organ segmentation in CT data represents an essential pre-processing step in different computer-assisted intervention and diagnosis methods. As we have discussed in previous chapters, CNNs lead the state-of-art in this task. Even when deep learning approaches have led to impressive results, it is also true that the challenging environment in organ segmentation can lead to errors in the models' predictions. In this regard, we discussed in previous chapters, some works that suggest that the uncertainty analysis of CNN's can give insights into the potential errors of the network. Inspired by these works, we present a segmentation refinement method [78, 79] that employs the uncertainty levels of the predicted segmentation map to define a semi-supervised node classification problem. Then, current GCN models are applied to train a graph's node classifier considering only the high confidence predictions of the CNN network. In this section, we discuss in detail each step of our proposed algorithm.

5.1 Motivation and Overview

Deep convolutional networks, usually abbreviated as CNNs, represent a breakthrough in the computer science community, significantly impacting computer vision. Nowadays, CNNs represent the standard models for different problems, like classification, localization, and segmentation. As we discussed in previous chapters, the medical image analysis community is not indifferent to these developments. The advances in computer vision have inspired diverse architectures to process medical image data, addressing different problems, including medical image segmentation. The segmentation of anatomical structures is an initial step in many computer-aided procedures, like computer-assisted navigation and detection. However, compared to real-world images, working with medical images involves additional challenges, like the necessity of expert experience to generate training labels. It leads to a limited amount of training data for learning representative models. Additionally, in problems like organ segmentation, it is common to work with high similarity in tissues, and inter-patient variability, making it complicated to differentiate the target structure from the background. All these factors can lead to potential errors in CNN's predicted segmentation. For example, the top row

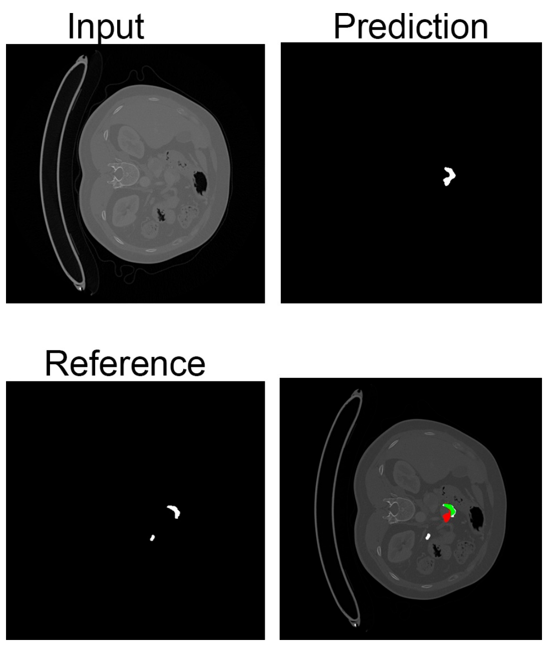


Fig. 5.1. Currently, CNN's lead state of the art in segmentation. However, different factors related to the nature of the medical images can lead to errors in the process. In the picture, we can see the input image, the pancreas segmentation output from a U-Net, the reference ground truth, and a comparison between the prediction and the ground truth. True positives, false positives, and false negatives regions are indicated in the bottom right figure in green, red, and white, respectively.

of Fig. 5.1 presents an input CT slice (left) and the predicted pancreas segmentation from a U-Net (right). The ground truth reference is given in the bottom left figure. The bottom-right illustration in Fig. 5.1 compares the prediction with the ground truth. Looking at this figure, It is clear that the model segmented correctly some regions of the organ (indicated in green). However, it also produced a group of false positives, marked in red, and missed some parts of the organ (false negatives), shown in white.

A common practice to improve the segmentation of a model is adding post-processing steps aiming at refining the results. Refinement methods are employing after the model's inference process. One example of such techniques is the conditional random fields, or CRF [41]. The applications of refinement methods are not limited to post-process the model's output. For example, [84] uses a CRF-based method to generate a set of scribbles. In combination with user-defined scribbles, the results are used to perform an image-specific fine-tune of a CNN. Similarly, semi-supervised learning methods can use refined predictions as pseudo-labels to include unlabeled data in CNN's training process [6]. The work in [46] addresses the problem of loss in spatial correlation in the task of metastasis detection in Whole-slide images due to the subdivision of the image in independent patches. The authors propose an architecture composed of a CNN that processes a group of input patches with a CRF on top. This CRF is employed to consider neighborhood information in the classification task, bringing consistency in the model's prediction.

CRF employs CNN prediction and spatial and intensity similarity between the pixels in CT slices to refine the segmentation. All this information is available from the network results and does not require retrain or modify the CNN for performing the refinement. Additional

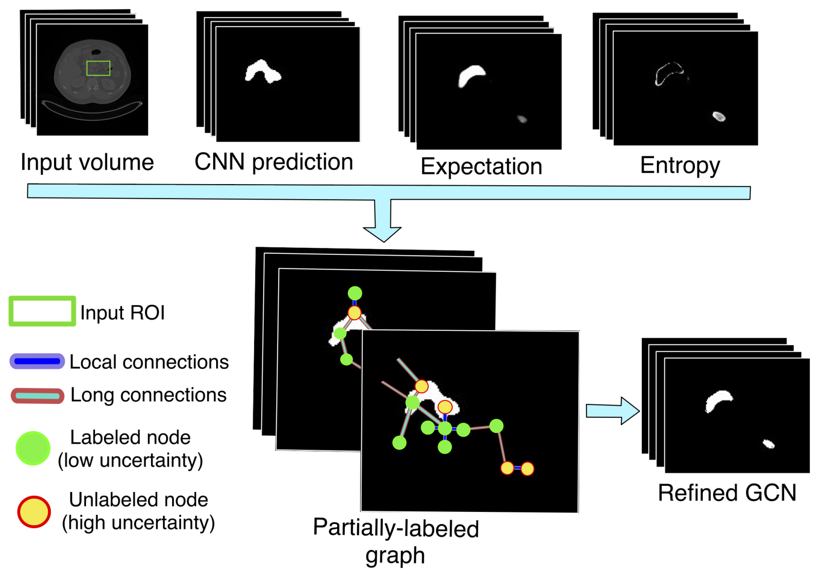


Fig. 5.2. A graphical overview of our GCN refinement strategy. Our proposal employs the input's intensities, and CNN's prediction, expectation, and entropy (uncertainty) to formulate the refinement problem as a semi-supervised graph learning problem.

information about the correctness of the prediction could bring helpful information to the process, yet it is not currently considered in the refinement process. As we discussed before, the model's uncertainty can indicate the quality or correctness of the model's output. The MCDO analysis is an option for estimated the confidence of the CNN in its predictions. It is not required to retrain or modify the model and can be applied to any architecture trained with dropout layers. It is ideal for refinement purposes, where the only available information comes from the model and its output.

Since uncertainty can bring insights regarding the potential errors in the segmentation, we still need a way to incorporate this knowledge into a refinement pipeline. In this work, we propose a method to formulate the segmentation refinement problem of CT data as a semi-supervised graph learning problem, that is solved using GCNs. Our method first apply MCDO to obtain the model's expectation and uncer- tainty, this last expressed by the model's entropy. This is used to divide the CNN output into high confidence points (background or foreground) and low confidence points. With this information, we define a semi-labeled graph that is used to train a GCN in a semi- supervised way using the high confidence predictions. The refined segmentation is obtained by re-evaluating the full graph in the trained GCN (see Fig. 5.2). To our best knowledge, this is the first time a semi-supervised GCN learning strategy is employed in the medical image segmentation task, specifically, for single organ segmentation. Also, this work presents one of the first cases of using GCN and uncertainty analysis for organ segmentation refinement. In the upcoming sections, we discuss in detail the implementation of our proposal.

5.2 Organ Segmentation and Uncertainty Analysis

Consider an input CT volume V with $V(x)$ the intensity value at the voxel position $x \in \mathbb{R}^3$; consider also, a trained CNN $g(V(x);\theta)$ with parameters θ; and a segmented volume $Y(x) = g(V(x);\theta)$ with $Y(x) \in \{0,1\}$. Our objective, is to refine the segmentation Y using a GCN trained on a graph representation of the input data. Our framework operates as a post-processing step (one volume at a time) and assumes that no information about the real segmentation (ground truth) is available.

We first look for a binary volume U_b used to highlight the potential false positives and false negatives elements of Y. The second step uses U_b, together with information coming from Y, g, and V, to refine the segmentation Y. We use uncertainty analysis to define U_b. For the second step, we solve the refinement problem using a semi-supervised GCN trained on a graph representation of our input volume.

We aim to including correct information in the refinement process. We can do this without modifying the CNN architecture with the MCDO method. We also look to define a refinement method that only depends on the information given by the CNN and its outputs. It will allow our technique to be applied in inference time and with no external knowledge about the actual ground truth. MCDO also meets this requirement since to estimate the uncertainty, it is only necessary to have the CNN model and the current input. For a particular pre-trained model $G(V(x);\theta)$ with dropout layers, following MCDO, we use the dropout layers of the network in inference time and perform T stochastic passes on the network to get the expectation of the model's prediction as:

$$\mathbb{E}(x) \approx \frac{1}{T}\sum_{t=1}^{T} g(V(x),\theta_t), \tag{5.1}$$

with θ_t the model parameters after applying dropout in the pass t. Every time we run a pass of the MCDO method, we are obtaining a sample of the approximated predictive distribution. Eq. 5.1 then collects the expected prediction over the T samples. A stable prediction will lead to an expectation close to zero or one. In contrast, an inconsistent prediction over the samples will give an expectation with a value close to 0.5, indicating random guessing. Now, we need to quantify the uncertainty of the model. We employed the entropy of the expectation for it. A stable prediction should produce low levels of entropy, and this last will increase according to the inconsistency of the MCDO samples. The model uncertainty $\mathbb{U}$ is given by the entropy, computed as.

$$\mathbb{U}(x) = H(x) = -\sum_{c=1}^{M} P(x)^c \log P(x)^c, \tag{5.2}$$

with $P(x)^c$ being the true probability of the voxel x to belong to class c, and M is the number of classes. In this work, we address the problem of binary segmentation refinement, and hence, $M = 2$ in our case. Fig. 5.3 presents a summary of the uncertainty analysis's outputs. It shows some examples of the expectation and entropy obtained for a particular input. Note that, in this figure, the uncertainty map shows higher values boundary regions and in some false positive areas.

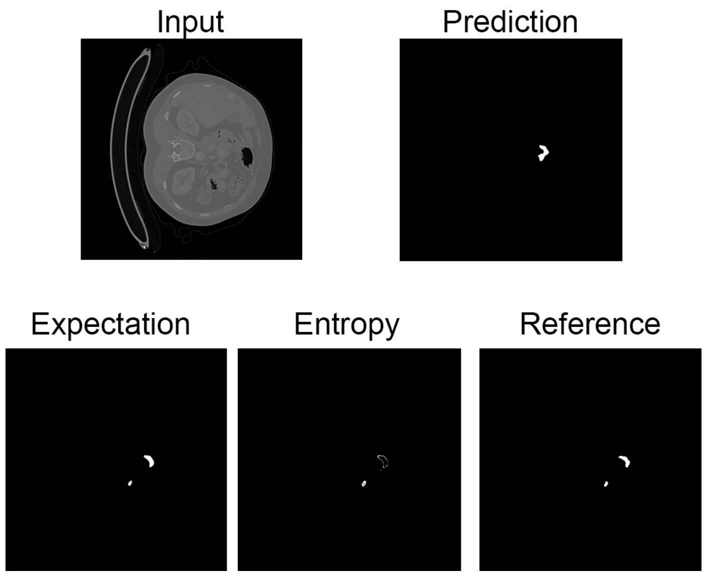


Fig. 5.3. We perform the uncertainty analysis of a CNN for a particular input. The figure shows the input images, the model's prediction (from a 2D U-Net), and the expectation and entropy obtained with the MCDO analysis. For reference, the figure also presents the ground truth segmentation.

5.2.1 Uncertain Predictions

For the final step of our analysis, we need to define the uncertain predictions. The uncertainty map takes continuous values between zero and one (if we consider log2 for computing it). To determine which elements are considered uncertain, we simply threshold the uncertainty map $\mathbb{U}(x)$, to obtain a binary uncertainty map:

$$U_b(x) = \mathbb{U}(x) > \tau, \tag{5.3}$$

The threshold controls the entropy necessary to consider a voxel x as uncertain. Thus, lower thresholds will include more voxels in the uncertain area. Conversely, higher thresholds will increase the number of confident predictions. It will have an effect on the following steps of our method, as we will discuss in the upcoming sections. Now that we have defined certain and uncertain predictions, the following step is to formulate the problem as a graph-node classification task. The next section discusses how we employ the uncertainty analysis to define a partially labeled graph.

5.3 A Graph Representation of CT Volumes

In this section, we describe the process of formulating a partially labeled graph from CT data. After the uncertainty analysis, we have a binary uncertainty mask U_b, a continuous uncertainty mask $\mathbb{U}$, and the model expectation $\mathbb{E}$. All these components are related to a particular input CT image V, and the CNN predicted segmentation map $Y = g(V(x); \theta)$. The uncertainty analysis only tells us that the model is not confident about its predictions, but it does not necessarily imply an error in the prediction. Some of the elements indicated by U_b

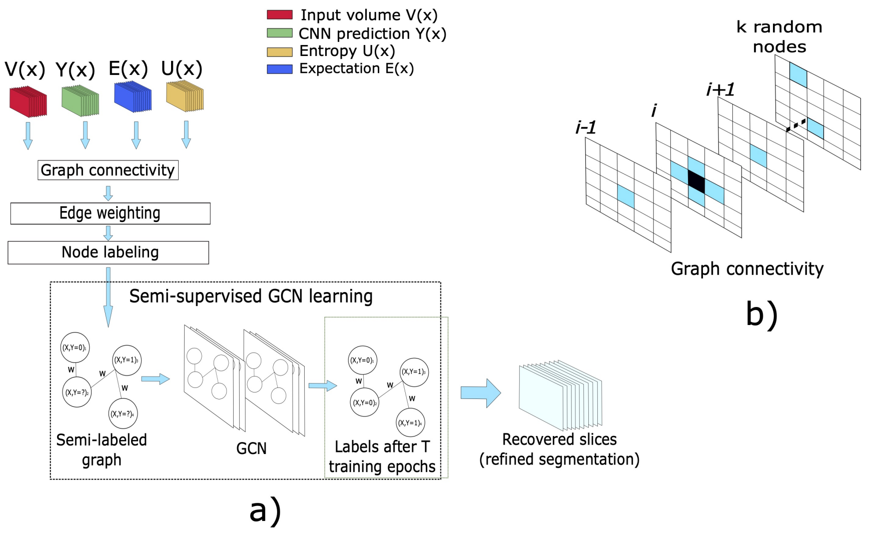


Fig. 5.4. a) The GCN refinement strategy. We construct a semi-labeled graph representation based on the uncertainty analysis of the CNN. Then, a GCN is trained to refine the segmentation. b) Connectivity. The black square is connected to six perpendicular neighbors and with $k = 16$ random voxels

could be indeed correct, and its value should not be changed. For this reason, we propose employing a learning model trained on the high confidence CNN predictions to re-classify (refine) the output of the segmentation model g. With the information given by the uncertainty analysis, we can define a partially labeled graph and describe the refinement process as a semi-supervised node classification problem.

Formally, given a graph $\mathcal{G}$ representing our 3D volumetric data, at the inference time, we aim to obtain a refined segmentation Y^* as the results of our GCN model Γ,

$$Y^* = \Gamma(\mathcal{G}(S); \phi), \tag{5.4}$$

where the graph $\mathcal{G}$ is constructed from the set of volumes $S = \{\mathbb{E}, \mathbb{U}, V, Y\}$ (see section Fig. 5.4) and ϕ represents the GCN's parameters.

5.3.1 Node Definition

Graphs are not limited to regular grid-like structures. It allows us to employ a more convenient (but more complex) region of interest surrounding the target organ. Since most of the voxels in the volume are irrelevant for the refinement process, we define an ROI tailored to our target anatomy. We define our working region as $\text{ROI}(x) = \text{dilation}(U_b(x)) \cup \mathbb{E}_b(x)$ with $\mathbb{E}_b$ the expectation binarized by a threshold of 0.5. Since the entropy is usually high in boundary regions, including the dilated U_b ensures that the ROI is bigger enough to contain the organ. Also, this allows us to include high confidence background predictions ($Y = 0$) for training the GCN. Including the expectation in the ROI give us high confidence foreground predictions for training the model. This ROI reduces the number of nodes of the graph and, in consequence, the memory requirements.

Once defined the working area, each voxel $x \in$ ROI is determined to be a node in the graph. Each node is represented by a feature vector containing intensity $V(x)$, expectation $\mathbb{E}(x)$, and entropy $\mathbb{U}(x)$. Given that we cannot trust the high uncertainty nodes, we consider these points unlabeled nodes in the graph. Otherwise, we employ the predicted segmentation as the ground label for that node in the graph representation. Our motivation is that, even though we cannot define the correctness of the uncertainty elements, we can still consider that the low uncertainty predictions have a high chance to be correct. The label assignment for a particular node x is thus formally defined as:

$$l(x) = \begin{cases} Y(x) & \text{if } U_b(x) = 0 \\ \text{unlabeled} & \text{if } U_b(x) = 1 \end{cases} \tag{5.5}$$

5.3.2 Edge Definition

Now that we have established the nodes and labels, the edges are defined according to local and long-range neighborhoods. The most straightforward connectivity option is to consider the connectivity with adjacent voxels (n6 or n26 adjacent voxel neighborhood). However, this simple nearest neighborhood scheme may not be suitable for our problem for two reasons; First, with this scheme, every single voxel is connected with its local neighborhood but lacks global information. Considering the original volumetric representation of the data, the primary source of information is coming only from adjacent voxels. In contrast, information from the global context (not-adjacent voxels) is mainly ignored. Second, voxels with high uncertainty tend to shape contiguous clusters. Because of this, a voxel with high uncertainty will most probably be surrounded by other high uncertainty points, reducing the connection with the low uncertainty points. Hence, a simple n6 or n26 connectivity might limit information propagation from confidence to uncertain regions. A fully connected graph can take advantage of the relationships between local and long-range connections and propagate information from both certain and uncertain regions during training and inference time, but at the cost of prohibitive memory requirements.

Instead, we opted for an intermediate solution. For a particular node (or voxel) x, we create connections with its six perpendicular immediate neighbors in the volume coordinate system (N6) to consider local information. Additionally, we randomly select k additional nodes in the graph and connect these random elements and x. It defines a sparse representation that considers local and long-range connections between high and low uncertainty elements. The k random nodes can be taken from any part inside the ROI used to define the graph's nodes. We empirically found that $k = 16$ offered a balance in performance and graph size and kept this value during our experiments. This connectivity strategy is graphically represented in Fig. 5.4 b).

5.4 Edge Weighting Function

To define the weights for the edges, we use a function based on Gaussian kernels considering the intensity $V(x)$ and the 3-D position $x \in \mathbb{R}^3$ associated with the node:

$$w_1(x_i, x_j) = \lambda \text{div}(x_i, x_j) + \beta[\exp(-\frac{||V(x) - V(x_j)||^2}{2\sigma_1}) + \exp(-\frac{||x_i - x_j||^2}{2\sigma_2})] \tag{5.6}$$

where λ and β are balancing factors, div$(\cdot)$ is given by the diversity between the nodes [92], defined as div$(x_i, x_j) = \sum_{c=1}^{M}(P^c(x_i) - P^c(x_j)) \log \frac{P^c(x_i)}{P^c(x_j)}$ with $M = 2$, $P^1(x_i) = \mathbb{E}(x_i)$ and $P^2(x_i) = 1 - \mathbb{E}(x_i)$ for our binary case. We opt for an additive weighting, instead of a multiplicative one, because the GCN can take advantage of connections with both similar and dissimilar nodes (in intensity and space) in the learning process, and using a multiplicative weighting could completely cut these connections. We found out that the diversity can indirectly bring information about the similarity of two nodes, in terms of class probability.

Since the diversity does not have an upper bound, it is possible to apply a non-linear transformation in order to normalize its value to the range $(0, 1]$, leading to the following version of the diversity:

$$\text{norm\_div}(x_i, x_j) = 1 - 2^{-\text{div}(x_i, x_j)}. \tag{5.7}$$

We can integrate this into eq.(5.6) replacing the regular diversity:

$$w_2(x_i, x_j) = \lambda \text{norm\_div}(x_i, x_j) + \beta[\exp(-\frac{||V(x) - V(x_j)||^2}{2\sigma_1}) + \exp(-\frac{||x_i - x_j||^2}{2\sigma_2})]. \tag{5.8}$$

Similarly, if we only take the exponential part of eq.(5.7), we get a similarity metric between the expectation of the nodes x_i and x_j. We use this version of the diversity as a third variation of our weighting function, leading to the following expression:

$$w_3(x_i, x_j) = \lambda \text{inv\_div}(x_i, x_j) + \beta[\exp(-\frac{||V(x) - V(x_j)||^2}{2\sigma_1}) + \exp(-\frac{||x_i - x_j||^2}{2\sigma_2})], \tag{5.9}$$

where the function inv_div(x_i, x_j) is given by

$$\text{inv\_div}(x_i, x_j) = 2^{-\text{div}(x_i, x_j)}. \tag{5.10}$$

It is worth mentioning that if we set $\lambda = 0$ and keep the same value of β all the weighting functions become the same.

5.5 Segmentation Refinement as a Semi-Supervised Graph Learning Problem

This process defines a partially labeled graph constructed with the uncertainty information coming from the CNN g. To this point, it is possible to train any standard GCN using only the labeled nodes. The model proposed by Kipf [40] was originally conceived for training on partially labeled graphs. Even though we previously described this model, we repeat the main formulation in this section for clarity. A convolutional H layer is defined as:

$$H = \hat{A}XW, \text{ with } \hat{A} = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}} \tag{5.11}$$

The variable $X \in \mathbb{R}^{N \times K}$ represents the input feature matrix of the layer with N the number of nodes and K the number of input features per node. $W \in \mathbb{R}^{K \times K'}$ is the weight matrix of the current layer with K' the number of output features per node. $\hat{A}$ follows the renormalization proposed by [40] with $\tilde{A} = A + I_N$, A the adjacency matrix of $\mathcal{G}$, and the diagonal matrix $\tilde{D} = \Sigma_{\text{cols}}\tilde{A}$ that sums across the columns of $\tilde{A}$. In general, we keep the same GCN architecture, but employing a sigmoid activation function at the output layer, for binary classification. Representing our graph $\mathcal{G}$ by its adjacency matrix A and feature matrix X, this leads to the following GCN model:

$$Y^* = \Gamma(X, A; W^0, W^1) = \text{sigmoid}(\hat{A}\ \text{ReLU}(\hat{A}XW^0)W^1) \tag{5.12}$$

Finally, it is worth mentioning that, even if we validated our proposal using this GCN learning architecture, given the modular nature of our method, it is possible to employ a different semi-supervised graph learning strategy.

Experiments and Results 6

The previous chapter introduced the details of our refinement strategy, going from the uncertainty analysis to the formulation of an uncertainty-driven, partially labeled graph. Now, we discuss the implementation and results of the method on two organ segmentation datasets for the pancreas and spleen. The method we propose is independent of the CNN architecture. For simplicity and with the objective of evaluating our approach, we employed a 2D CNN architecture segmentation model. A 2D CNN operates slice-wise, and a volumetric segmentation is obtained by predicting on each slice of the 3D CT input and stacking the results together. First, we apply the refinement method to a set of input CT volumes and compare our outcomes with the state-of-art CRF refinement method [41]. Then, we evaluate the effects of variations in the graph-definition parameters, performing a sensitivity analysis with a special focus on the weighting function. All the processes run on an NVidia Titan Xp with 12 GB of graphics memory.

6.1 Implementation Details

As we mentioned previously, we selected a 2D U-Net to be our CNN model [58]. The U-Net follows a standard architecture but considering dropout layers at the end of every convolutional block of the U-Net, as indicated by the MCDO method. Furthermore, we train the U-Net to address the problem of binary organ segmentation, using the axial view of the CT volumes. The model trains for around 200 epochs keeping the overall best performing model during the entire training procedure. We employed the Adam optimizer to minimize the dice loss function. At inference time, we predicted every slice separately. Then we stacked all the predictions together to obtain a volumetric segmentation (a similar strategy was used to perform the uncertainty analysis). Finally, as a post-processing step, we compute the largest connected component in the prediction to reduce false positives. At this point, it is worth mentioning that the U-Net was used only for testing purposes, and different architectures can be used instead. It is mainly because our refinement method uses the model-independent MCDO analysis.

For the uncertainty analysis, we utilized MCDO to compute the expectation and entropy using a dropout rate of 0.3 and a total of $T = 20$ stochastic passes. To obtain volumetric uncertainty from a 2D model, we performed the uncertainty analysis on every individual slice of the input volume. Then we all the estimated uncertainty maps are stacked together to obtain the volumetric expectation and entropy. We tested different values for the uncertainty threshold τ (see section 6.3).

The GCN model is a two-layered network with 32 feature maps in the hidden layer and a single output neuron for binary node-classification. The graphical network is trained for 200 epochs with a learning rate of $1e-2$, binary entropy loss, and the Adam optimizer. We kept these same settings for all the experiments. After the refinement process, we can replace only the uncertain voxels with the GCN prediction, or we can replace the entire CNN prediction with the GCN output. We use the second approach since we found it producing better results.

6.1.1 Organ Segmentation Datasets

We train two different U-Nets each for the pancreas and spleen segmentation tasks, respectively. For the pancreas segmentation problem, we used the NIH pancreas dataset[2] [16, 61, 62]. We randomly selected 45 volumes of the NIH dataset for training the CNN model. A total of 20 volumes are reserved for testing the refinement methods.

For spleen, we employed the spleen segmentation task of the medical segmentation decathlon [75] (MSD-spleen[3]). We trained the CNN on 26 volumes and reserved nine volumes to test our framework for this problem. The MSD-spleen dataset contains more than one foreground label in the segmentation mask. We unified the non-background labels of the MSD-spleen dataset into a single foreground class since we evaluate our method for refining a binary segmentation model.

6.1.2 Statistical Significance Test

To fully validate our results, we perform a statistical significance test on the dice score performance of the refinement output. However, under consideration that with small sets, the statistical significance tests can fail [11, 81], we obtained the dice score slice-wise to increase the sample size (up with 278-1700 slices for the spleen, and pancreas respectively). Then we have run the non-parametric statistical significance test, namely Kolmogorov–Smirnov test. Finally, we perform a statistical significance analysis between the results of the GCN refinement and the initial results of the CNN. When we report the results, we will use a single start (*) to indicate a p-value < 0.05, and a double-start (**) to indicate a p-value < 0.01 concerning the original CNN prediction.

6.2 Refinement Performance

We first evaluate and compare the refinement capabilities of our proposal. We apply our method to the U-Net's output of each of the 20 NIH pancreas and 9 MSD-Spleen testing volumes. The edge weighting function for our GCN-based refinement method is given by Eq. 5.6 with $\lambda = 0.5$ and $\beta = 1$. Additionally, we apply a CRF-based refinement on the same outcomes to compare the refinement performance. We use a publicly available implementation of the CRF method presented in [41] to refine the CNN prediction.

For CRF, we employed a similar approach to [41]. The CRF method assumes dense connectivity between the voxels of the CT. It means that every voxel is connected to each other one within

[2]https://wiki.cancerimagingarchive.net/display/Public/Pancreas-CT
[3]http://medicaldecathlon.com/

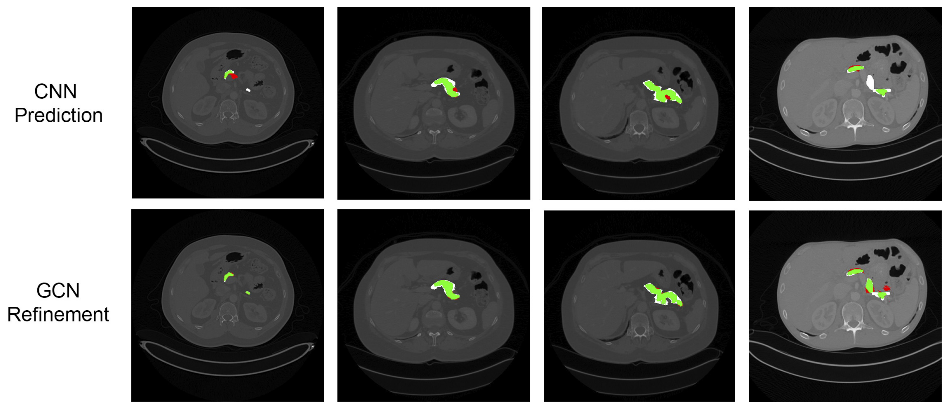


Fig. 6.1. Comparison of the CNN prediction and its corresponding GCN refinement for pancreas segmentation. Green colors indicate true positives (TP), red indicates false positives (FP), and white false negative (FN) regions. From left to right: the first column shows an FP region removed and an FN region recovered after the refinement. The second and third columns show FP regions removed. The fourth column shows an FN region recovered but also a new FP region generated.

the ROI. Also, we set one unary and two pairwise potentials for CRF. We use the prediction of CNN as the unary potential. For the two pairwise potentials, the first is composed of the voxel position in the 3D volume. The second pairwise potential is a combination of intensity and position of the voxels. For the CRF refinement, we considered the same ROI used by the GCN.

Tab. 6.1. Average dice score performance (%) of the GCN refinement compared with the CNN prediction and a CRF-based refinement of the CNN prediction. Results for pancreas and spleen are presented. Statistical significance is indicated by (*) for a p-value < 0.05, and (**) for a p-value < 0.01 with respect to the CNN prediction.

Task	CNN 2D U-Net	CRF refinement	GCN Refinement (ours)
Pancreas	76.89 ± 6.6	77.20 ± 6.5	$\mathbf{77.81 \pm 6.3}$*
Spleen	93.17 ± 2.5	93.40 ± 2.6	$\mathbf{95.07 \pm 1.3}$**

Results are presented in Table 6.1. The GCN-based refinement outperforms the base CNN model and the CRF refinement by around 1% and 0.6% respectively in the pancreas segmentation task. For spleen segmentation, our GCN refinement presented an increase in the dice score of 2% with respect to the base CNN, and 1.7% with respect to the CRF refinement. Figs. 6.1 and 6.2 show visual examples of the GCN refinement compared with the base CNN prediction.

6.2.1 Performance with a CNN Trained On Fewer Examples

We evaluate the performance of our refinement method when the base CNN trains under a low data regime. In this scenario, the uncertainty of the model should increase as a result of the limited amount of training data. For this, we randomly selected 10 out of the 45 training samples of the NIH dataset. For the spleen, we selected nine. Table 6.2 shows the results

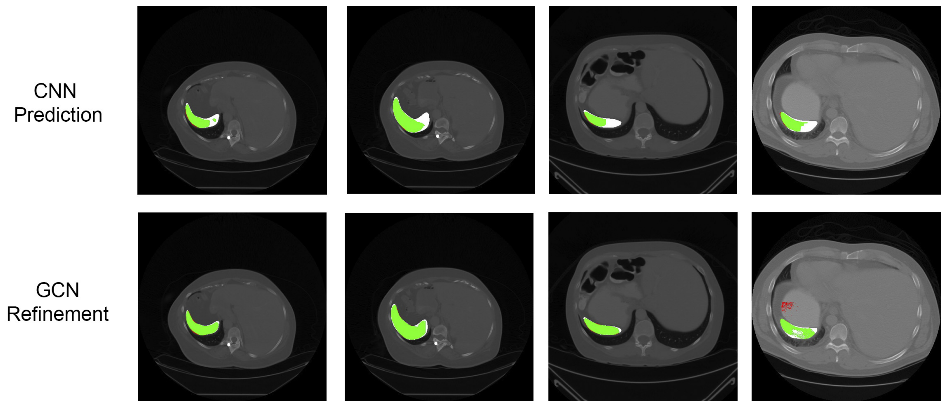


Fig. 6.2. Comparison of the CNN prediction and its corresponding GCN refinement for spleen segmentation. Green colors indicate true positives (TP), red indicates false positives (FP), and white false negative (FN) regions. From left to right: the first, second, and third columns show FN regions recovered. The fourth column shows an FN region recovered but also a new FP region generated.

for the U-Net initial segmentation and the results after applying the CRF and GCN-based refinement strategies.

Tab. 6.2. Average dice score performance (%) of the GCN refinement compared with the CNN prediction. The CNN model was trained with 10 samples for the pancreas and 9 for the spleen. Statistical significance is indicated by (*) for a p-value < 0.05, and (**) for a p-value < 0.01 with respect to the CNN prediction.

Task	CNN 2D U-Net	CRF refinement	GCN Refinement (ours)
Pancreas-10	52.14 ± 22.6	52.20 ± 22.6	$\mathbf{54.55 \pm 22.2}$*
Spleen-9	78.89 ± 28.4	78.80 ± 28.4	$\mathbf{81.15 \pm 28.9}$**

Note the increment in the standard deviation of all the models. A reason for this can be that the CNN does not generalize adequately to the testing set, due to the small number of training examples. Similar to the previous results, the increment in the dice score for the GCN refinement is about 2.4% with respect to the CNN base model for the pancreas, and improvement of 2.3% for spleen, compared with the base CNN.

6.3 Uncertainty Threshold

We perform several experiments to understand the effects of the parameters employed by our proposal. The threshold τ, used to define the uncertain elements, is one of these parameters. We evaluate the performance of our method under a wide range of values for τ. We tested the method with values of $\tau \in \{0.001, 0.3, 0.5, 0.8, 0.999\}$. In this way, we covered a wide range of conditions that define a voxel as "uncertain". We following the same configuration as in the performance comparison (Sec. 6.2). Table 6.3 compares the CNN output with the GCN refinement at different values of τ for the tasks of the pancreas and spleen segmentation.

Tab. 6.3. Average dice score performance (%) of the GCN refinement at different uncertainty thresholds τ. Pancreas-10 and Spleen-9 indicate the models trained with 10 and nine samples, respectively.

Task	GCN $\tau = 1e-3$	GCN $\tau = 0.3$	GCN $\tau = 0.5$	GCN $\tau = 0.8$	GCN $\tau = 0.999$
Pancreas	77.71 ± 6.3	77.79 ± 6.4	77.77 ± 6.3	77.81 ± 6.3	77.79 ± 6.3
Pancreas-10	54.55 ± 22.1	54.32 ± 22.1	54.15 ± 22.2	53.91 ± 22.4	53.14 ± 22.9
Spleen	95.01 ± 1.5	94.92 ± 1.4	94.98 ± 1.4	94.97 ± 1.4	95.07 ± 1.3
Spleen-9	80.91 ± 28.8	80.94 ± 28.9	80.94 ± 28.8	80.98 ± 28.9	81.15 ± 28.9

Before discussing the results, we highlight some points about τ. This parameter controls the minimum value required to consider a voxel as uncertain. Lower values lead to a higher number of uncertain elements. It follows an inverse relationship with the number of high certainty nodes in the graph representation, and hence, in the number of training examples for the GCN.

We also consider that the quality of the training examples is affected by this parameter. A high threshold relaxes the amount of uncertainty necessary to rely on the prediction of the CNN. However, it might reduce the quality of training examples since high uncertainty points that did not reach the threshold value will be considered for training the GCN. Employing a small threshold can ensure that only high confidence nodes are used for training, but at the cost of significantly reduces the number of samples available to train the GCN.

Interestingly, from the results of Table 6.3, except for pancreas-10 and spleen-9, there is no significant impact on the choice of this parameter. One reason can be that there is a clear separation between high and low uncertainty points in models trained with an adequate number of samples. Therefore, changing τ may add (remove) a few nodes that are insignificant for the learning process of the GCN.

The refinement of the pancreas model trained with ten samples (Pancreas-10) presents a progressive decrease in the dice score. Since this CNN uses fewer training examples, it is expected to have low confidence in its predictions. In this scenario, a higher uncertainty threshold increases the chance to include high-uncertainty nodes as ground truth for training the GCN, leading to lower quality in the training examples, as we discuss before. In contrast, a lower τ includes fewer points but with higher confidence. Thus, it appears to be beneficial in the pancreas segmentation model trained with fewer examples.

The opposite occurs with spleen-9, where higher τ are beneficial. This might indicate a dependency on the characteristics of the anatomies since the pancreas presents more inter-patient and inter-slice variability (see Fig. 6.3).

In general, the results suggest that the τ parameter should be selected based on the target anatomy. Further, the influence of τ appears to be more evident in conditions of high uncertainty, e.g. when the model is trained with fewer examples. Finally, in the cases where τ

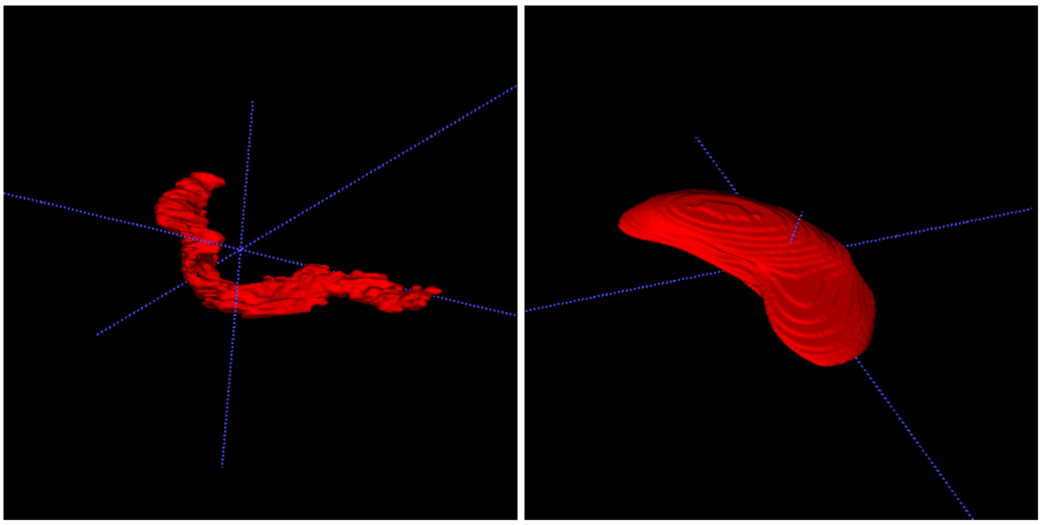


Fig. 6.3. An example of the 3D view of the pancreas and spleen. The shape of the pancreas brings more inter-patient and inter-slice variability. The shape of the spleen presents better consistency between patients and slices, leading to better performance for the CNNs. Views obtained with ITK-SNAP [89].

has no significant impact, intermediate values are preferred since they lead to a lower number of nodes and lower memory requirements.

6.4 Node Connectivity

In the next experiment, we compare the refinement performance of our method using a classical n26 neighborhood as a connectivity scheme. We repeated the experiments for different uncertainty thresholds, and the CNN model trained with different numbers of examples on the pancreas and spleen datasets. Results are presented in Tables 6.4 and 6.5 for each organ, respectively. In all tables, the weighting employed is eq. 5.6 with $\lambda = 0.5$ and $\beta = 1$.

Tab. 6.4. Average dice score performance (%) of the GCN refinement at different uncertainty thresholds τ for the **pancreas** segmentation problem. The table compares the results for the 6-surrounding + 16 random connectivity (ours) vs. a 26 surrounding connectivity (n26). cnn10 indicates results obtained with a CNN trained on ten samples.

Connectivity	GCN	GCN	GCN	GCN	GCN
	$\tau = 1e-3$	$\tau = 0.3$	$\tau = 0.5$	$\tau = 0.8$	$\tau = 0.999$
Ours	77.71 ± 6.3	77.79 ± 6.4	77.77 ± 6.3	77.81 ± 6.3	77.79 ± 6.3
n26	76.93 ± 6.1	77.16 ± 5.8	77.18 ± 5.8	77.18 ± 5.8	77.4 ± 5.9
Ours-cnn10	54.55 ± 22.1	54.32 ± 22.1	54.15 ± 22.2	53.91 ± 22.4	53.14 ± 22.9
n26-cnn10	52.50 ± 22.1	52.66 ± 22.4	52.56 ± 22.4	52.53 ± 22.5	52.43 ± 22.9

The results show a better refinement when including the random long-range connections, especially when working with the CNN trained with limited data, for both tasks. Note that

Tab. 6.5. Average dice score performance (%) of the GCN refinement at different uncertainty thresholds τ for the **spleen** segmentation problem. The table compares the results for the 6-surrounding + 16 random connectivity (ours) vs. a 26 surrounding connectivity (n26). cnn9 indicates results obtained with a CNN trained on nine samples.

Connectivity	GCN $\tau = 1e-3$	GCN $\tau = 0.3$	GCN $\tau = 0.5$	GCN $\tau = 0.8$	GCN $\tau = 0.999$
Ours	95.01 ± 1.5	94.92 ± 1.4	94.98 ± 1.4	94.97 ± 1.4	95.07 ± 1.3
n26	94.26 ± 1.6	94.11 ± 1.8	95.09 ± 1.9	94.14 ± 1.8	94.43 ± 1.8
Ours-cnn9	80.91 ± 28.8	80.94 ± 28.9	80.94 ± 28.8	80.98 ± 28.9	81.15 ± 28.9
n26-cnn9	79.63 ± 28.6	79.65 ± 28.5	79.68 ± 28.5	79.90 ± 28.5	80.38 ± 28.7

when comparing with the original CNN output presented in Tables 6.1 and 6.2, our refinement method using the n26 connectivity still having a slight improvement over the original CNN prediction.

6.5 Edge Weighting

All the experiments presented so far employed the weighting function defined by Eq. 5.6. Now, to validate and understand the effects of the weighting functions defined, we perform our refinement strategy using different variations of the functions given by equations 5.6, 5.8, and 5.9, presented in section 5.4. We explore the situations when: a) either only diversity or only the Gaussian kernels are employed (sec. 6.5.1); b) using a diversity normalized to the range [0, 1] (sec. 6.5.2); c) using similarity in expectation given by the inverse of the diversity (sec. 6.5.3); and d) we discuss deep insights regarding the three different variations of the diversity employed in the previous sections (sec. 6.5.4).

In all the experiments of this section, we keep a fixed value of $\tau = 0.5$. We will use the notation $w_{i,(\lambda,\beta)}$, $i \in \{1, 2, 3\}$ to indicate the parameters employed by the corresponding function. For example, the notation $w_{1,(0.5,1)}$ indicates that the weighting function w_1 is used with $\lambda = 0.5$ and $\beta = 1$.

6.5.1 Diversity v.s. Gaussian Similarity Kernels

Our weighting scheme is composed of the addition between the diversity function and two Gaussian kernels that indicate the similarity in intensity and space for the pair of nodes that define the edge. It is possible to use an edge weighting function that only considers diversity. In a similar way, we can weight the edges using only the Gaussian similarity kernels. In this experiment, we compare both scenarios. It is done by setting the values of λ and β to zero, accordingly. The results are presented in Table 6.6. Note that $w_{1,(0.5,1)}$ corresponds to the weighting function used in all our previous experiments.

Tab. 6.6. Average dice score performance (%) of the GCN refinement at $\tau = 0.5$. The table compares w_1 with different combinations of λ, β parameters ($w_{1,(\lambda,\beta)}$). Statistical significance is indicated by (*) for a p-value < 0.05, and (**) for a p-value < 0.01 with respect to the CNN prediction.

Task	CNN 2D U-Net	GCN $w_{1,(0.5,1)}$	GCN $w_{1,(0.5,0)}$	GCN $w_{1,(0,1)}$
Pancreas	76.89 ± 6.6	77.77 ± 6.3*	77.85 ± 6.3*	77.90 ± 6.2*
Pancreas-10	52.14 ± 22.6	54.15 ± 22.2	54.28 ± 22.2	53.03 ± 23.0
Spleen	93.17 ± 2.5	94.98 ± 1.4**	95.20 ± 1.4**	94.74 ± 1.8**
Spleen-9	78.89 ± 28.4	80.94 ± 28.8**	80.96 ± 28.9**	81.17 ± 28.9*

Different weighting functions and variations still outperform the initial CNN prediction. Now, we will focus on the slight differences between these results. As we mentioned, our weighting scheme considers diversity together with intensity and position similarity. The two later Gaussian components are commonly used in the literature and follow the intuition that two components similar in intensity and close to each other in space are likely to belong to the same class. The diversity is an additional component that allows us to include the results from the MCDO analysis in the edge weighting. The diversity has a lower bound of zero, but it does not have an upper bound in its base form in contrast with the Gaussian kernels. When two nodes have a similar expectation, the diversity between those elements will be small, and the weighting function will only rely on the Gaussian similarities. On the other hand, when the nodes have important differences in expectation (e.g 0 vs. 1), the unbounded nature of the diversity will impose over the much smaller contribution of the Gaussian kernels (the diversity can reach values of 30, while each Gaussian kernel has a maximum at 1.0). It will bias the weight to the value of diversity.

We now discuss the behavior of the refinement process on each of the test sets when we apply each weighting scheme. Starting with the pancreas models, in a high-training-data scenario, a Gaussian-kernel-only ($\lambda = 0$) scheme appears to be good enough for the refinement strategy. In the low data-regime, the GCN appears to take more advantage of diversity. Our features employ intensity, expectation, and entropy (or uncertainty). Note that the uncertainty threshold affects the labeled nodes but does not affect the connectivity. A node can be connected to any certain or uncertain neighbor, and a Gaussian-kernel-only weighting is not aware of possible inconsistencies in the node features.

The spleen segmentation problem shows different behavior. The lower inter-patient variability of the spleen could be a reason (see Figs. 6.3). In a high-training-data regime, it can bring a more stable and well-separated expectation between organ and background, making the diversity a more favorable weighting. In a low data regime, the Gaussian-kernel-only version appears to perform better. However, from Table 3, a larger τ appears to benefit the low-data spleen problem. In fact, at $\tau = 0.999$, the dice-score for $w_{1,(0.5,0)}$ and $w_{1,(0,1)}$ are 81.09, and 81.15, respectively, showing no significant difference. Note that $w_{1,(0,1)} = w_{2,(0,1)} = w_{3,(0,1)}$.

6.5.2 Using a Normalized Diversity

We previously commented that the vanilla implementation of diversity does not have a strict upper bound. However, it is possible to apply a non-linear transformation to map the values of the diversity into the range $[0, 1]$. Replacing the vanilla version with the normalized diversity leads to the weighting function w_2, described by the Eq. 5.8. We run experiments using this weighting scheme, and the results are presented in Table 6.7 where we also compare with the results when using w_1.

Tab. 6.7. Average dice score performance (%) of the GCN refinement at $\tau = 0.5$. The table shows the drop in the performance of w_2 compared with the U-Net prediction and w_1. Statistical significance is indicated by (*) for a p-value < 0.05, and (**) for a p-value < 0.01 with respect to the CNN prediction.

Task	CNN	GCN	GCN
	2D U-Net	$w_{1,(0.5,1)}$	$w_{2,(1,1)}$
Pancreas	76.89 ± 6.6	77.77 ± 6.3*	63.27 ± 9.9**
Pancreas-10	52.14 ± 22.6	54.15 ± 22.2	43.05 ± 22.1**
Spleen	93.17 ± 2.5	94.98 ± 1.4**	87.15 ± 3.1**
Spleen-9	78.89 ± 28.4	80.94 ± 28.8**	75.47 ± 26.8**

Results show that w_2 presents a negative impact on the learning process of the GCN. To explain this behavior, we would like to highlight some points from the previous section. First, the unbounded nature of the diversity makes w_1 to prefer connections with opposed expectations (and ignore the Gaussian similarity in those cases). On the other hand, the Gaussian similarity is only considered when the expectation of both nodes is basically the same, and even in those cases, the weight is small compared with the values of the diversity. In this sense, the GCN can learn mostly from examples that are different in expectation or (with a considerably lower contribution) from examples similar in intensity and position.

In contrast, normalizing the diversity into a value between 0 and 1, makes the contribution of the Gaussian kernels more representative. The normalized diversity of w_2 will no longer ignore the Gaussian similarity. It will assign the highest weights to connections between nodes that are similar in intensity and position, but at the same time different in expectation, which is counter-intuitive. Table 6.7 shows the drop in the performance using w_2 as weighting function. Given that the diversity is in range between 0 and 1, we use $\lambda = 1$ for w_2.

6.5.3 Turning the Diversity into a Similarity Metric

The diversity gives a metric of dissimilarity between two probability distributions. Withing our context, it provides information about the differences in the expectation of two nodes. We can reformulate the definition of the diversity to obtain the opposite behavior, it is, getting high values when two points have similar values in expectation. It will generate a weighting function, driven only by similarity metrics (expectation, intensity, and space).

Tab. 6.8. Average dice score performance (%) of the GCN refinement at $\tau = 0.5$. The table compares the performance of w_3 with the U-Net prediction and w_1. Statistical significance is indicated by (*) for a p-value < 0.05, and (**) for a p-value < 0.01 with respect to the CNN prediction.

Task	CNN	GCN	GCN
	2D U-Net	$w_{1,(0.5,1)}$	$w_{3,(1,1)}$
Pancreas	76.89 ± 6.6	77.77 ± 6.3*	78.19 ± 6.1*
Pancreas-10	52.14 ± 22.6	54.15 ± 22.20	52.90 ± 23.1
Spleen	93.17 ± 2.5	94.98 ± 1.4**	94.81 ± 1.9**
Spleen-9	78.89 ± 28.4	80.94 ± 28.8**	81.08 ± 28.9**

If we take only the exponential part of Eq. 5.7, we will obtain values in the range [0,1] that gives high weights to similar expectations, presenting a better agreement with the Gaussian kernels. This weighting scheme is given by w_3. The results, compared with the U-Net and the initial $w_{1,(0.5,1)}$ weighting function are presented in Table 6.8. The results show an improvement again for the GCN refinement, supporting our discussion about the inconsistencies brought by w_2.

Let's disscuss the results presented. Again, we can first focus on the results of the pancreas. The inverse of the diversity in combination with the Gaussian kernels appears to perform well with the pancreas model trained with a high number of examples. It can be because the number of training samples for the CNN is enough to obtain an adequate estimation of the expectation, in contrast with the low-data pancreas U-Net.

Moving into the spleen segmentation task, the differences appear to be not significant. In general, the results suggest that w_3 is beneficial for the full-data pancreas problem, and does work well with both spleen models. But might be sub-optimal for the low-data pancreas refinement task. A possible explanation is that w_3 will assign high weights to nodes with similar values, no matter if both have expectations of 1, 0, or 0.5. This last expectation value (0.5) represents a high uncertainty point, and it is expected to find this kind of point on an irregular organ, and with a model trained with a low number of examples, like Pancreas-10. In this sense, the inverse of the diversity in w_3 might also assign a high weight to connections between uncertainty nodes.

6.5.4 Comparing the Variations of the Diversity Function

The final experiment for the weighting function compares all the versions of the diversity. We have three versions, each one employed in the functions w_1, w_2, and w_3. We set $\beta = 0$ in the three functions to have a diversity-only weighting scheme. The results are presented in Table 6.9.

We can mention two interesting points coming from this experiment. First, comparing the results of $w_{2,(1,0)}$ in Table 6.9 with the results of $w_{2,(1,1)}$ in Table 6.7, it is clear that the only-diversity version of w_2 has better performance, supporting the idea of the inconsistent connections of $w_{2,(1,1)}$. comes when comparing the diversity of w_1 with the normalized

Tab. 6.9. Average dice score performance (%) of the GCN refinement at $\tau = 0.5$. The table compares the performance of the three variations of the diversity ($\beta = 0$ for all functions). Statistical significance is indicated by (*) for a p-value < 0.05, and (**) for a p-value < 0.01 with respect to the CNN prediction.

Task	CNN 2D U-Net	GCN $w_{1,(0.5,0)}$	GCN $w_{2,(1,0)}$	GCN $w_{3,(1,0)}$
Pancreas	76.89 ± 6.6	77.85 ± 6.3*	75.45 ± 7.1**	77.70 ± 6.2*
Pancreas-10	52.14 ± 22.6	54.28 ± 22.2	50.72 ± 23.5*	52.09 ± 23.0
Spleen	93.17 ± 2.5	95.20 ± 1.4**	94.25 ± 2.1**	94.59 ± 2.0**
Spleen-9	78.89 ± 28.4	80.96 ± 28.9**	79.84 ± 28.5	80.81 ± 28.7*

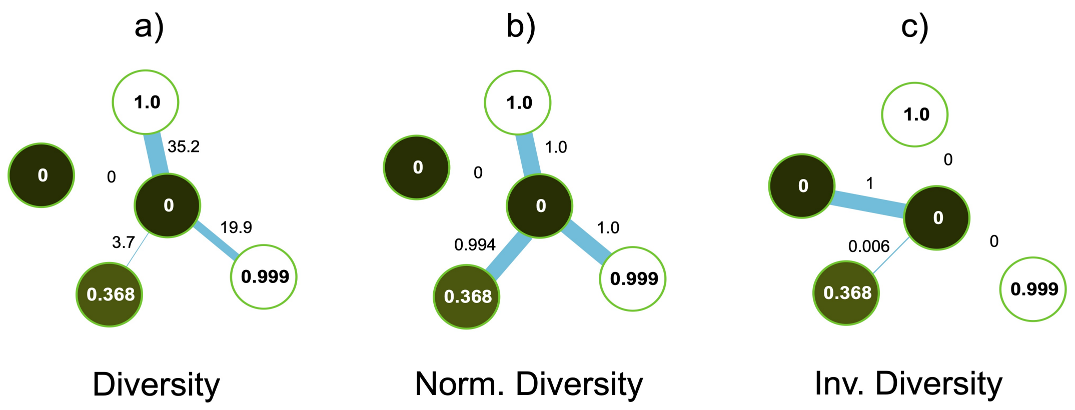


Fig. 6.4. A graphical view of the weights assigned by each variation of the diversity function (vanilla, normalized, and inverse). The three figures show the same node structures, but weighted by: a) $w_{1,(0.5,0)}$, b) $w_{2,(1,0)}$, and c) $w_{3,(1,0)}$. The value inside the node represents the expectation for that node.

diversity of w_2, both in Table 6.9. Even though these two functions are expected to behave similarly, we notice a difference in their performance. To understand the possible causes of these differences, we take a closer look at the assigned weights of these functions to a subset of nodes. Let's consider a central node with expectation close to zero, and four neighbors nodes with expectations of approximately 0.0, 1.0, 0.999, and 0.368, respectively (see Fig. 6.4). As expected, $w_{1,(0.5,0)}$ will assign high values to opposed expectations (36.2 and 19.9 in Figure 6.4.a), while the weight will be low for similar expectations, and close to zero to identical expectations (weights of 3.7 and 0 in Fig. 6.4.a). However, for the same node structure, $w_{2,(1,0)}$ will weight with a high value (close to 1) almost all the connections, except for the neighbors with the same expectation (see Fig, 6.4.b). Even though the difference between the expectations values of 1 and 0.368 is not high enough, $w_{2,(1,0)}$ will give an importance that is similar to the nodes with completely opposed expectations, leading to an inconsistent weighting scheme. In contrast, $w_{3,(1,0)}$ (which acts as a kind of similarity in expectation) assigns weights close to zero to all the nodes that do not have the same expectation, even to the pair (0, 0.368), see Fig. 6.4.c. This can suggest a better consistency for the inverse diversity, compared with diversity normalized to [0, 1]. We can also see these conclusions if we plot the values for these three variations of the diversity as a function of the difference between $p1 = 0$ and $p2 \in [0, 1]$ (see Fig. 6.5). We can see how the diversity grows exponentially when the difference between $p1$ and $p2$ is close to 1 (Fig. 6.5 Diversity). In a similar way, the inverse

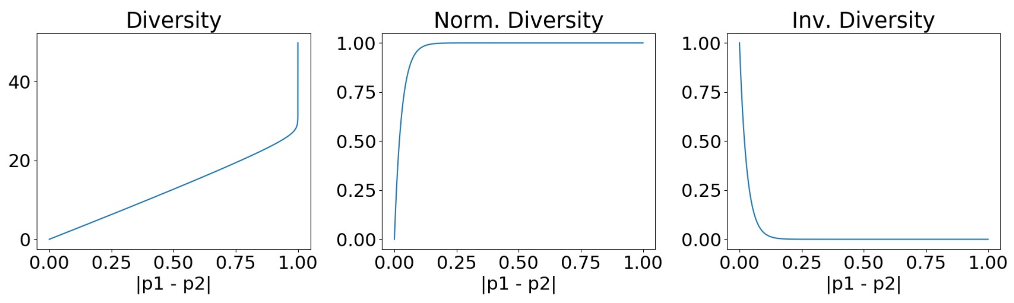


Fig. 6.5. Comparison of the weights assigned by the three variations of the diversity (vanilla, normalized, and inverse), represented as a function of the difference of the expectation $p_1 = 0$ and $p_2 \in [0, 1]$.

diversity assigns a weight of 1 to differences close to zero and decreases exponentially at the moment this difference starts to increment (Fig. 6.5 Inv. Diversity). Finally, the normalized diversity starts assigning a weight of zero to equal expectations, however, the curve grows exponentially until reaching a weight of 1 when the difference in expectations still close to 0.2 (Fig. 6.5 Norm. Diversity), leading to the inconsistencies mentioned before.

In general, a weighting function like w_2 appears to be not recommendable for the refinement with GCNs. The inverse diversity of w_3 can be a better option, however, it is also possible that under certain conditions, this weighting could introduce noisy connections. For example, when the expectation has values close to 0.5 for both nodes, the function will assign high weights, however, this might not heavily contribute to the GCN given the fact that these nodes might have high uncertainty as well. This suggests that, under high uncertainty environments, it is better to weigh based on dissimilarity in expectation. On the other hand, the inverse of diversity can take advantage of well-separated expectations.

6.6 Insights on Prediction, Expectation, and Entropy

One of the main components of our methodology is the uncertainty analysis. The graph definition is driven by the expectation and entropy obtained during this process, together with the CNN's predictions. Fig. 6.6 presents a visual overview of these three components. The CNN's prediction is employed to assign labels to the nodes, considering its confidence level. However, taking a look at Fig. 6.6, it can be noticed that the refinement obtained is similar to the expectation.

Diving deeper into this topic, we can comment that the nodes use the expectation as part of their features. Similarly, the diversity takes the expectation as its primary input. The GCN can learn how to use the CNN's expectation, together with intensity and spatial information, to reclassify the graph's nodes. However, since this component takes part in driving the refinement, it can also generate false positives if it contains artifacts. For example, Fig. 6.6 shows a region in the expectation that does not agree with the ground truth. It can also be noticed that the GCN reduced this region. It can be a positive consequence of the random

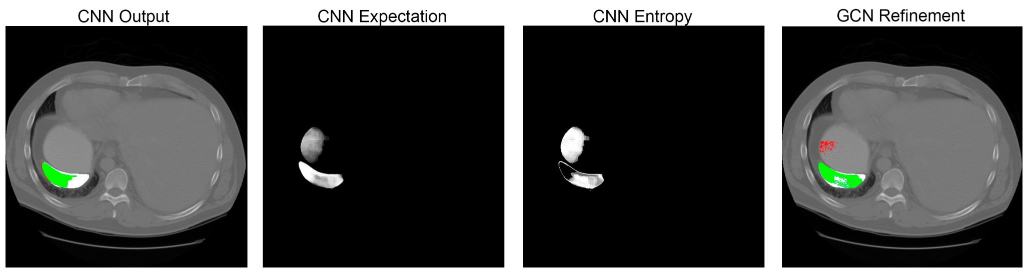


Fig. 6.6. Elements used in the graph definition. In the CNN and GCN outputs: Green colors indicate true positives, red false positives, and white false negative regions. For the expectation and entropy: brighter intensities indicate higher values.

long-range connections included in the graph definition that brings additional information beyond the local neighborhood.

To evaluate the relationship between the expectation and the GCN refinement, we compute the relative improvement between the GCN and the expectation. First, the expectation was thresholded by 0.5. Then we calculated its dice score with the ground truth. The relative improvement is computed as:

$$rel\_imp = \frac{gcn_{dsc} - expectation_{dsc}}{expectation_{dsc}} \times 100. \tag{6.1}$$

The rel_imp is computed for every input volume. Fig. 6.7 shows the results for the pancreas segmentation task, and compares the metric when the expectation was obtained from a model trained with 45 (Fig.6.7a) and 10 samples (Fig.6.7b), respectively, for pancreas segmentation.

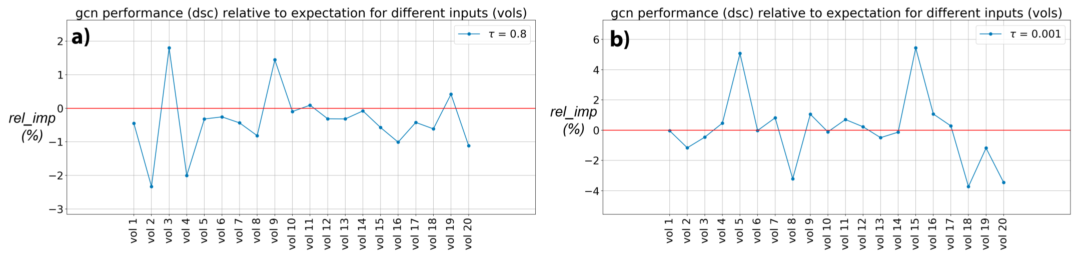


Fig. 6.7. Relative improvement (%) per input volume of different GCN configurations with respect to the expectation of a pancreas segmentation model. The red line indicates the same dsc as the expectation. a) CNN trained with 45 volumes, $\tau = 0.8$. b) CNN trained with 10 volumes, $\tau = 0.001$.

Fig. 6.7a shows that most of DICE coefficients (17/20) of the GCN refinement are either below or close to the ones of the expectation. However, three volumes show an improvement in the DICE compared to the expectation. A different scenario is presented in Fig. 6.7b. Here, (13/20) volumes show either better or similar DICE for the GCN compared to the expectation. A possible explanation is that models trained with an adequate number of examples (volumes), their expectation is good enough. In contrast, models trained with a few examples (volumes) have higher uncertainties yielding unreliable expectations. Our results suggest that our GCN refinement strategy is favorable over the expectation or uncertainty analysis in such scenarios.

7 Discussion and Conclusion

Our work presents a method to construct semi-labeled graph representation of volumetric medical data, based on the output and uncertainty analysis of a CNN model. We have also shown that graph semi-supervised learning can be used to obtain a refined segmentation. We also provided a deep analysis of the weighting function employed to construct the graph. We have shown that diversity is an adequate choice for expectation-based edge weighting. In a similar way, the inverse diversity can also be a good option under certain circumstances. The dependence of the graph to the expectation and the uncertainty analysis method employed could explain the differences in the performance when refining the prediction of two different CNNs. In this regard, alternatives to the uncertainty estimation method, and the use of calibrated uncertainties can be an interesting direction when working with different CNNs models. In this section, we discuss final remarks regarding our method.

7.1 Applicability to Other Network Architectures

As a refinement strategy, our proposed method is orthogonal to any segmentation pipeline and can be applied to different CNNs architectures equipped with the MDCO approximation. To verify this, we apply the uncertainty-GCN refinement to the predictions of QuickNat [65, 66], trained on the same two segmentation tasks. The network is trained using the same 45 volumes for the pancreas, and 26 for the spleen, under similar settings as the U-Net. The testing set is the same we used to evaluate the GCN refinement from U-Net. We employed the weighting function $w_{2,(1,1)}$ with $\tau = 0.8$ and $\tau = 0.5$ for the pancreas and spleen models, respectively. Table 7.1 presents the results for the initial QuickNat prediction and the GCN refinement.

Tab. 7.1. Average dice score performance (%) of the GCN refinement compared with the initial CNN predictions. Statistical significance is indicated by (*) for a p-value < 0.05, and (**) for a p-value < 0.01 with respect to the 2D-CNNs predictions.

Task	3D U-Net	2D U-Net	Ours	2D QuickNat	Ours
		Initial	GCN-Refinement	Initial	GCN-Refinement
Pancreas	60.14 ± 10.07	76.89 ± 6.6	78.20 ± 6.1*	61.31 ± 13.1	61.57 ± 13.0
Spleen	82.37 ± 16.8	93.17 ± 2.5	95.20 ± 1.4**	91.83 ± 6.0	92.97 ± 2.3

Reported results show an improvement over the initial CNN model but on a different level compared to the results obtained with the U-Net. While the spleen problem shows an improvement of 1% over the initial prediction, the pancreas model shows subtle changes. Such differences among different CNNs can be attributed to the epistemic uncertainty inherent to their respective models. In other words, different behaviors of the uncertainty across models can lead to different behaviors in the GCN refinement strategy. However, a deeper analysis of inter-model uncertainty and their relationship with the Graph-based refinement is necessary.

7.1.1 2D vs. 3D Architectures

In our experiments, we employed a 2D architecture since it provides us with all the necessary components to test our method. Nevertheless, our refinement strategy is orthogonal to other segmentation approaches and can be applied to any CNN that produces uncertainty measures with MCDO. This also includes 3D models. However, to our best knowledge, MCDO is most commonly employed with 2D models and its translation to 3D might require additional methodological efforts derived from working with 3D architectures together with additional requirements for data-handling due to memory constraints [44]. Further, 3D model might not necessarily provide a better initial segmentation compared to a 2D CNN as reported by [85, 91]. In this regard, we trained a standard 3D U-Net [55] subdividing the input volume into blocks of $64 \times 64 \times 32$, with a minimum feature size of 16 for the U-Net convolutions. This allows us to use batches of 8 during training on a NVidia Titan Xp of 12 GB. This lead to a dice-score of 60.14% for the pancreas segmentation problem, and 82.37% for the spleen segmentation, using the same training, and testing set as the 2D U-Nets (see Table 7.1). As can be seen, under similar conditions, the 3D model presents a lower performance compared with their 2D counterparts. This can be due to the loss of the global context derived from the subdivision of the volume. Fit the entire volume can contribute to the performance. However, this can be limited by the current GPU memory capabilities. Similarly, using a volume-level input will reduce the number of available samples, which can lead to overfitting problems. Even though we are aware of U-Net, V-Net inspired 3D architectures defined for pancreas segmentation that can reach around 80% of dice score, with the use of data augmentation and auxiliary losses for deep supervision [94], or fine-tuning with the addition of attention gates [55], we consider that the 2D U-Net gives us good-enough results on both segmentation problems with an appropriate simplicity to evaluate our framework. Nevertheless, investigating our approach using 3D models equipped with MCDO might be an interesting direction.

7.2 Future Paths: Graphs for Classification Refinement

We have introduced an application of uncertainty analysis and graph convolutional networks in the task of segmentation refinement. As discussed, graphs can capture complex relationships between data points. These relations are not captured by standard CNNs. The main motivation of this thesis was segmentation, however, it would be interesting to know if a similar idea can be applied to classification tasks. A graph representation of classification results could provide additional information not captured by standard CNN.

In a proof of concept, we defined a graph over the results of two region proposal CNNs, defined for localization of polyps and endoscopic artifacts. Instead of employing the uncertainty of the prediction to define the graph, we use the spatial relation of the predictions. Polyp detectors work in challenging environments with artifact presence. It has motivated methods for the detection and removal of endoscopic artifacts [24]. Similarly, some works evaluate their influence in the detections task and incorporate artifact information in the polyp detection model [10, 38, 83]. Similarly, the endoscopic artifact detection challenge (EAD) [2, 3, 4], has highlighted the role of artifacts in the analysis of endoscopic sequences, remarking its importance in the definition of models for endoscopy image analysis.

Following a similar idea as the graph-based refinement, it would be possible to define a graph of polyps and artifacts bounding box detections, where the nodes are defined by a feature representation of the bounding boxes. Edges can be represented by the spatial interactions of the predictions (overlap) and similarity in class or feature. This graph can then be used to learn the interactions between polyps and artifacts in a video sequence or endoscopic image collection, and then use the learned graph model as a post-processing step to refine the predictions of a model on a new set of images or frames. Note that in order to generate a graph, it is necessary to have a set of predictions. In the segmentation task, it was given by the pixels in the image. For localization purposes, we need a set of bounding box predictions. Hence, it is better to predict on a set of images, instead of individual samples. This is in order to have enough nodes to define the graph.

It is possible to employ a graph model on a training set to learn node embeddings for a node classification task. Then the model can be applied to classify the nodes of an unseen graph. This learning model is known as inductive learning. The spectral GCN employed in the segmentation refinement task is a transductive learning model. It means that it cannot be directly generalized to unseen graphs. Similarly, if a new node is included, the model should be adjusted and retrained. For the segmentation refinement task, this is not a problem, since each CT volume defines a complete graph used only for the refinement of their corresponding image. For the polyp refinement problem, it is necessary to learn a model on a training set that then can be employed on a new input sequence, making inductive models more suitable for this task. For this purpose, it is possible to employ models like GraphSAGE [30] to train an *inductive graph* based model for multi-class node classifier. GraphSAGE learns the parameters of K aggregation functions that combine neighbor information. Similarly, it employs a set of weight matrices used to combine and propagate information in the layer level (different aggregation depths). For the aggregation process, a fixed number of neighbor nodes are uniformly sampled. The results lead to a set of node embeddings that, for supervised classification problems, are learned using a standard loss function, like the categorical cross-entropy.

To define the connections of the graph, different criteria can be applied. For example, 1) we can define a connection if their bounding boxes overlap with an $IoU > 0.5$ or if one bounding box is contained inside the other (see Fig. 7.1, row 1). This is regardless of their classes (overlap criterion): 2) Two nodes can be connected if they share the same class (class-aware criterion, see Fig. 7.1, row 2); and 3) we can connect nodes if they are both polyps or if they are both artifacts, regardless of the artifact class (binary criterion, see Fig. 7.1, row 3). All these criteria are symmetric and self-connections are not allowed. We consider connections in the same frame (intra-frame) and between different frames (inter-frame), see Fig. 7.1. Criterion 1 applies only for intra-frame nodes. Criteria 2 and 3 apply for both connection types, and only one of them can be used at a time to generate the graph.

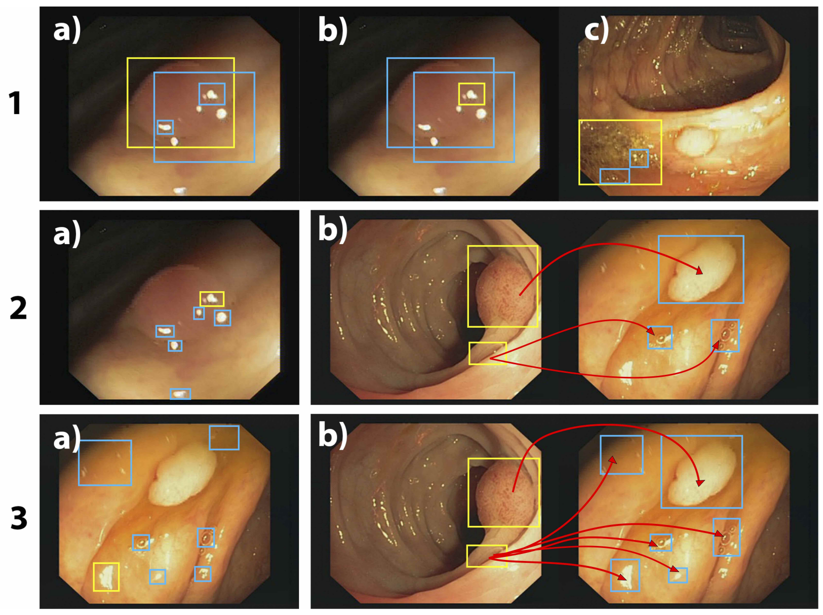


Fig. 7.1. A graph representation of region proposal network's predictions to analyse the influence of artifacts in polyp detection. Yellow bounding boxes indicate a source node with examples of its neighbors in blue. Red arrows are inter-frame connections. Row 1: connection examples of a) polyp to artifacts, b) artifact to polyp and, c) artifact to artifact. Rows 2 and 3: examples of a) intra-frame and b) inter-frame connections.

A method like this requires that both polyps and artifacts detections are in the same image. In testing or inference time, it is possible to employ a polyp and an artifact detection model to obtain the predictions. The graph then can refine the results of these models, to correct potential polyps misclassified as artifacts. However, for training purposes, it is necessary to have both boundings box ground truth in the same image. To our best knowledge, currently, there is no available dataset containing both annotations in the same image. An option can be to employ an artifact detector to extend a polyp dataset and use it for training. It is possible to train artifact detectors with publicly available datasets, like the EAD Challenge's dataset [2, 3, 4]. This dataset contains annotations for seven different artifacts, including bubbles, specular reflections, contrast, and blur.

We implemented these strategies to evaluate the performance of a model trained on a graph composed of polyps and artifacts predictions. GraphSAGE requires a feature representation for the nodes. For this, we resize the bounding boxes to a fixed size of 64×64 and compute the histogram of oriented gradients as feature. We used a publicly available implementation of GraphSAGE[1], with the mean node aggregator. We consider all the EAD artifact's classes (polyp, saturation, misc., blur, contrast, bubbles, instrument), but excluding specularities to avoid a class imbalance problem in the graph. We trained a RetinaNet polyp detector on the CVCDB_ClinicDB dataset [9]. Simlarly, we extend this dataset with artifact predictions to train the GraphSage model. Testing was done on the CVC_ColonDB dataset [8].

Initial results show an improvement on the F1 score of the graph-refinement predictions (0.73), compared with the base polyp-only RetinaNet predictions (0.71). It was employing an overlap

[1]https://github.com/williamleif/graphsage-simple/

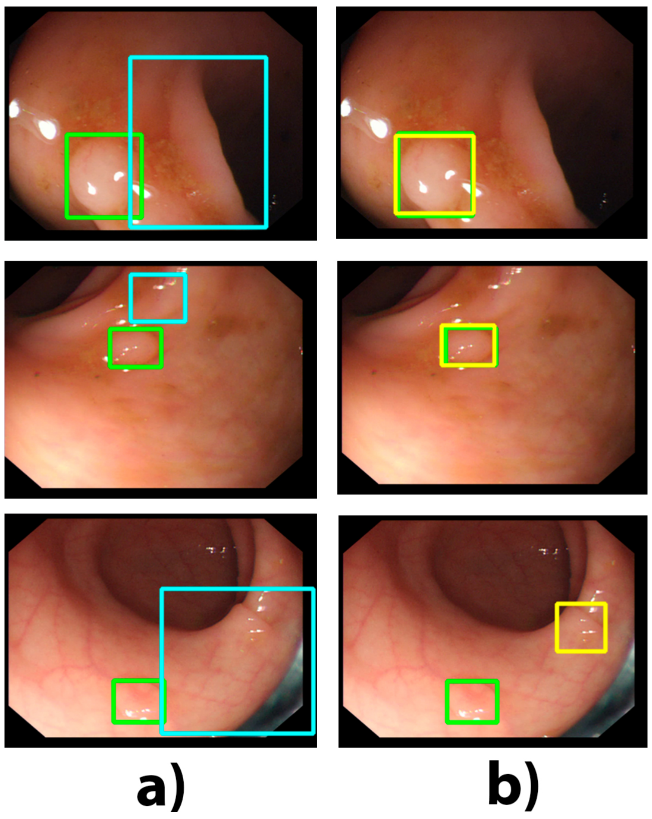


Fig. 7.2. Visual results. Polyp ground truth is presented in green. a) initial RCNN polyp predictions (blue) and b) polyp predictions after a graph post-processing method (yellow). From top to bottom, the first two rows show examples where our method removed a false positive and recovered a polyp bounding box. The third row shows an example where a false positive was removed, but a new false positive is added.

togheter with a binary connectivity criteria. Similarly, visual results show how the graph base model can correct the errors derived from artifact presence (Fig. 7.2. Additional details about this project can be consulted in [77]. Results are promising, and even though it is still necessary additional validation and experiments, it shows an interesting future approach for post-processing graph strategies, like the segmentation refinement presented in this thesis.

7.3 Concluding Remarks

We have presented a novel application of graphs and uncertainty analysis for the segmentation refinement task. Similarly, we introduced potential paths for graph-based post-processing methods on the polyp localization problem. We conclude with some remarks about our method that can lead to future research and improvements.

Regarding computational time, our defines a graph and then train a GCN model. This can require more computational requirements compared with the most efficient versions of CRF. In our experiments, the time required for training and testing the GCN in the constructed graph is around twice the time required for CRF in the fully connected version of our uncertainty graph. This is a time of around 30 sec $\sim$ 1 min for the GCN vs. 13 sec $\sim$ 30 sec for CRF. These numbers do not consider the time for uncertainty analysis and graph construction.

Similarly, in this work, we have employed MCDO [39] for uncertainty analysis, and found out the expectation could be a good choice for well-trained models, while our GCN refinement shows superior performance, compared to the expectation, in low-data regime. Nonetheless, recently proposed uncertainty measures [82], which disentangle the model's uncertainty from the one associated with the inter/intra-observer variability, might be desirable.

Regarding the graph representation, we have investigated different connectivity and weighting mechanisms in defining our graph and extracted a couple of features to represent our nodes. However, prior knowledge, like geometry, could be used to constrain the ROI and provide plausible configurations [19, 53].

We have shown that the model can be applied to different organ segmentation problems and CNN architectures. Similarly, our results suggest that the performance can depend on the characteristics of the anatomy studied. Large and stable organs like the liver can derive in performance similar to the spleen. However, further experiments are necessary to verify this. Similarly, large organs can represent a challenge for a graph-based method, since a graph constructed over voxels can lead to high memory requirements. A change on the node representation of the CT data can help in this problem, however, we leave this as future work.

Finally, in this work, we addressed a binary classification problem. For a multi-class problem, it should be possible to obtain a vectorial expectation representing each class, and the entropy can be computed considering multiple classes. Even though this brings all the elements to formulate the partially labeled graph, given the complexity in the different structures that share intensity similarities between tissues, a different weighting, connectivity, and node features might be necessary to include meaningful information about the anatomies. Similarly, the inclusion of a larger number of structures will lead to a larger number of nodes, making the efficient node representation of multi-class data an interesting future direction.

Part III

Appendix

List of Authored and Co-authored Publications

A

2020

[78] **Roger D. Soberanis-Mukul**, Nassir Navab and Shadi Albarqouni. "An Uncertainty-Driven GCN Refinement Strategy for Organ Segmentation". *MIDL20 Special Issue at the Journal of Machine Learning for Biomedical Imaging (MELBA), 2020*.

[79] **Roger D. Soberanis-Mukul**, Nassir Navab and Shadi Albarqouni. "Uncertainty-based Graph ConvolutionalNetworks for Organ Segmentation Refinement". *International Conference on Medical Imaging with Deep Learning (MIDL), 2020, Montréal, Canada*.

[3] Sharib Ali, Felix Zhou, Barbara Braden, Adam Bailey, Suhui Yang, Guanju Cheng, Pengyi Zhang, Xiaoqiong Li, Maxime Kayser, **Roger D. Soberanis-Mukul**, Shadi Albarqouni, et al. "An objective comparison of detection and segmentation algorithms for artefacts in clinical endoscopy". *Scientific Reports, 2020*.

[52] Allan Ojeda-Pat, Anabel Martin-Gonzalez, **Roger D. Soberanis-Mukul**. "Convolutional Neural Network U-Net for Trypanosoma cruzi segmentation". *International Symposium on Intelligent Computing Systems, 2020*.

2019

[37] Maxime Kayser, **Roger D. Soberanis-Mukul**, Shadi Albarqouni, Nassir Navab.. "Focal Loss for Artefact Detection in Medical Endoscopy". *Proceedings of the 2019 Challenge on Endoscopy Artefacts Detection, 2019*.

B Abstracts of Publications not Discussed in this Thesis

An objective Comparison of Detection and Segmentation Algorithms for Artefacts in Clinical Endoscopy

Sharib Ali, Felix Zhou, Barbara Braden, Adam Bailey, Suhui Yang, Guanju Cheng, Pengyi Zhang, Xiaoqiong Li, Maxime Kayser, Roger D. Soberanis-Mukul, Shadi Albarqouni, et al.

We present a comprehensive analysis of the submissions to the first edition of the Endoscopy Artefact Detection challenge (EAD). Using crowd-sourcing, this initiative is a step towards understanding the limitations of existing state-of-the-art computer vision methods applied to endoscopy and promoting the development of new approaches suitable for clinical translation. Endoscopy is a routine imaging technique for the detection, diagnosis and treatment of diseases in hollow-organs; the esophagus, stomach, colon, uterus and the bladder. However the nature of these organs prevent imaged tissues to be free of imaging artefacts such as bubbles, pixel saturation, organ specularity and debris, all of which pose substantial challenges for any quantitative analysis. Consequently, the potential for improved clinical outcomes through quantitative assessment of abnormal mucosal surface observed in endoscopy videos is presently not realized accurately. The EAD challenge promotes awareness of and addresses this key bottleneck problem by investigating methods that can accurately classify, localize and segment artefacts in endoscopy frames as critical prerequisite tasks. Using a diverse curated multi-institutional, multi-modality, multi-organ dataset of video frames, the accuracy and performance of 23 algorithms were objectively ranked for artefact detection and segmentation. The ability of methods to generalize to unseen datasets was also evaluated. The best performing methods (top 15%) propose deep learning strategies to reconcile variabilities in artefact appearance with respect to size, modality, occurrence and organ type. However, no single method outperformed across all tasks. Detailed analyses reveal the shortcomings of current training strategies and highlight the need for developing new optimal metrics to accurately quantify the clinical applicability of methods.

Scientific Reports (2020)

Convolutional Neural Network U-Net for Trypanosoma cruzi segmentation

Allan Ojeda-Pat, Anabel Martin-Gonzalez, Roger Soberanis-Mukul

Chagas disease is a mortal silent illness caused by the parasite Trypanosoma cruzi that affects many people worldwide. A blood test is one of the preferred methods to get an accurate diagnosis of the disease but takes a long time and requires too much effort from the experts to analyze blood samples in the search of the parasites presence. Therefore, it is very useful to have an automatic system to detect the parasite in blood sample microscopic images. In this paper we present a deep learning method to segment the T. cruzi parasite on blood sample images. We implemented a convolutional neural network based on the U-Net model and we trained it with different loss functions to get accurate results. We report an F2 value of 0.8013, a recall value of 0.8702, a precision value of 0.6304 and a Dice score value of 0.6825.

International Symposium on Intelligent Computing Systems (2020)

Focal Loss for Artefact Detection in Medical Endoscopy

Maxime Kayser, Roger D Soberanis-Mukul, Shadi Albarqouni, Nassir Navab

Endoscopic video frames tend to be corrupted by various artefacts impairing their visibility. Automated detection of these artefacts will foster advances in computer-assisted diagnosis, post-examination procedures and frame restoration software. In this work, we propose an ensemble of deep learning object detectors to automate multi-class artefact detection in video endoscopy. Our approach achieved a mean average precision (mAP) of 0.3087 and an average intersection-over-union (IoU) of 0.3997 on the EAD2019 test set. This resulted in a final score of 0.3451 and the 3rd rank in the EAD 2019 object detection sub-challenge leaderboard.

Proceedings of the 2019 Challenge on Endoscopy Artefacts Detection (2019)

Bibliography

[1] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk. "SLIC Superpixels Compared to State-of-the-art Superpixel Methods". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2012) (cit. on p. 41).

[2] S. Ali, F. Zhou, A. Bailey, et al. "A deep learning framework for quality assessment and restoration in video endoscopy". In: *arXiv preprint arXiv:1904.07073* (2019) (cit. on pp. 87, 88).

[3] S. Ali, F. Zhou, B. Braden, et al. "An objective comparison of detection and segmentation algorithms for artefacts in clinical endoscopy". In: *Scientific Reports* (2020) (cit. on pp. 87, 88, 93).

[4] S. Ali, F. Zhou, C. Daul, B. Braden, A. Bailey, et al. "Endoscopy artifact detection (EAD 2019) challenge dataset". In: *CoRR* (2019). arXiv: `1905.03209` (cit. on pp. 87, 88).

[5] S. Ameling, S. Wirth, D. Paulus, G. Lacey, and F. Vilarino. "Texture-based Polyp Detection in Colonoscopy". In: *Bildverarbeitung für die Medizin 2009: Algorithmen - Systeme - Anwendungen*. 2009 (cit. on p. 20).

[6] W. Bai, O. Oktay, M. Sinclair, et al. "Semi-supervised Learning for Network-Based Cardiac MR Image Segmentation". In: *Medical Image Computing and Computer Assisted Intervention (MICCAI)*. 2017 (cit. on p. 62).

[7] D. Barber. *Bayesian Reasoning and Machine Learning*. cambridge university press, 2012 (cit. on p. 42).

[8] J. Bernal, J. Sánchez, and F. Vilariño. "Towards automatic polyp detection with a polyp appearance model". In: *Pattern Recognition*. 2012 (cit. on pp. 31, 32, 88).

[9] J. Bernal, F. J. Sánchez, G. Fernández-Esparrach, D. Gil, C. Rodríguez, and F. Vilariño. "WM-DOVA maps for accurate polyp highlighting in colonoscopy: Validation vs. saliency maps from physicians". In: *Computerized Medical Imaging and Graphics* 43 (2015), pp. 99 –111 (cit. on pp. 32, 88).

[10] J. Bernal, N. Tajkbaksh, F. J. Sánchez, et al. "Comparative validation of polyp detection methods in video colonoscopy: results from the MICCAI 2015 endoscopic vision challenge". In: *IEEE transactions on medical imaging* 36.6 (2017), pp. 1231–1249 (cit. on pp. 32, 87).

[11] D. J. Biau, S. Kernéis, and R. Porcher. "Statistics in brief: the importance of sample size in the planning and interpretation of medical research". In: *Clinical orthopaedics and related research* (2008) (cit. on p. 72).

[12] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006 (cit. on pp. 9, 13, 16, 22, 43–46).

[13] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. "Geometric Deep Learning: Going beyond Euclidean data". In: *IEEE Signal Processing Magazine* 34.4 (2017), pp. 18–42 (cit. on pp. 53, 54).

[14] J. Canny. "A Computational Approach to Edge Detection". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 1986 (cit. on p. 35).

[15] F. Chadebecq, F. Tilmart, and C. Bartoli. “How big is this neoplasia? llive colonoscopic size measurement using the infocus-breakpoint”. In: *Medical Image Analysis* (2015) (cit. on p. 31).

[16] K. Clark, B. Vendt, K. Smith, et al. “The Cancer Imaging Archive (TCIA): Maintaining and Operating a Public Information Repository”. In: *Journal of Digital Imaging* (2013) (cit. on pp. 35, 72).

[17] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. Third Edition. The MIT Press, 2009 (cit. on pp. 52, 53).

[18] M. Defferrard, X. Bresson, and P. Vandergheynst. “Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering”. In: *NeurIPS*. 2016 (cit. on p. 54).

[19] M. A. Degel, N. Navab, and S. Albarqouni. “Domain and geometry agnostic CNNs for left atrium segmentation in 3D ultrasound”. In: *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer. 2018, pp. 630–637 (cit. on p. 90).

[20] P. A. Dias and H. Medeiros. “Semantic Segmentation Refinement by Monte Carlo Region Growing of High Confidence Detections”. In: *Asian Conference on Computer Vision (ACCV)*. 2019 (cit. on pp. 5, 48).

[21] A. Farag, L. Lu, H. R. Roth, J. Liu, E. Turkbey, and R. M. Summers. “A Bottom-up Approach for Pancreas Segmentation using Cascaded Superpixels and (Deep) Image Patch Labeling”. In: *IEEE Transactions on Image Processing IEEE Transactions on Image Processing IEEE Transactions on Image Processing IEEE Transactions on Image Processing IEEE Transactions on Image Processing* (2016) (cit. on p. 41).

[22] A. Farag, L. Lu, E. Turkbey, J. Liu, and R. M. Summers. “A Bottom-Up Approach for Automatic Pancreas Segmentation in Abdominal CT Scans”. In: *International MICCAI Workshop on Computational and Clinical Challenges in Abdominal Imaging*. 2014 (cit. on p. 41).

[23] K. Fukushima. “Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position”. In: *Biological Cybernetics* (1980) (cit. on p. 22).

[24] I. Funke, S. Bodenstedt, C. Riediger, J. Weitz, and S. Speidel. “Generative adversarial networks for specular highlight removal in endoscopic images”. In: *Medical Imaging 2018: Image-Guided Procedures, Robotic Interventions, and Modeling*. 2018 (cit. on p. 87).

[25] Y. Gal and Z. Ghahramani. “Dropout as a Bayesian Approximation: Appendix”. In: *International Conference on Machine Learning*. 2016 (cit. on pp. 45–47).

[26] Y. Gal and Z. Ghahramani. “Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning”. In: *International Conference on Machine Learning*. 2016 (cit. on pp. 5, 45, 46, 48).

[27] M. Ganz, X. Yang, and G. Slabaugh. “Automatic segmentation of polyps in colonoscopic narrow-band imaging data”. In: *IEEE Transactions on Biomedical Engineering* 59.8 (2012), pp. 2144–2151 (cit. on p. 32).

[28] R. C. Gonzales and R. E. Woods. *Digital Image Processing*. Prentice Hall, 2002 (cit. on pp. 35, 36).

[29] S. Gross, T. Stehle, A. Behrens, et al. “A comparison of blood vessel features and local binary patterns for colorectal polyp classification”. In: *Medical Imaging 2009: Computer-Aided Diagnosis*. Vol. 7260. International Society for Optics and Photonics. 2009, 72602Q (cit. on p. 32).

[30] W. L. Hamilton, R. Ying, and J. Leskovec. “Inductive Representation Learning on Large Graphs”. In: *31st Conference on Neural Information Processing Systems (NIPS 2017)*. 2017 (cit. on p. 87).

[31] K. He, X. Zhang, S. Ren, and J. Sun. “Deep Residual Learning for Image Recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016 (cit. on pp. 29, 30).

[32] D. K. Iakovidis, D. E. Maroulis, S. A. Karkanis, and A. Brokos. “A Comparative Study of Texture Features for the Discrimination of Gastric Polyps in Endoscopic Video”. In: *Proceedings of the 18th IEEE Symposium on Computer-Based Medical Systems*. 2005 (cit. on p. 20).

[33] S. Ioffe and C. Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *Proceedings of the 32nd International Conference on International Conference on Machine Learning*. 2015 (cit. on pp. 24, 27).

[34] Y. Iwahori, A. Hattori, Y. Adachi, M. K. Bhuyan, R. J. Woodham, and K. Kasugai. “Automatic Detection of Polyp Using Hessian Filter and HOG Features”. In: *19th International Conference in Knowledge Based and Intelligent Information and Engineering Systems*. 2015 (cit. on p. 20).

[35] D. Jha, P. H. Smedsrud, M. A. Riegler, et al. “Kvasir-seg: A segmented polyp dataset”. In: *International Conference on Multimedia Modeling*. Springer. 2020, pp. 451–462 (cit. on p. 32).

[36] A. G.-U. Juarez, R. Selvan, Z. Saghir, and M. de Bruijne. “A joint 3D UNet-Graph Neural Network-based method for Airway Segmentation from chest CTs”. In: *Machine Learning in Medical Imaging*. Springer. 2019 (cit. on p. 58).

[37] M. Kayser, R. D. Soberanis-Mukul, S. Albarqouni, and N. Navab. “Focal Loss for Artefact Detection in Medical Endoscopy”. In: *Proceedings of the 2019 Challenge on Endoscopy Artefacts Detection*. 2019 (cit. on p. 93).

[38] M. Kayser, R. D. Soberanis-Mukul, A.-M. Zvereva, P. Klare, N. Navab, and S. Albarqouni. “Understanding the effects of artifacts on automated polyp detection and incorporating that knowledge via learning without forgetting”. arXiv preprint arXiv:2002.02883v3. 2020 (cit. on p. 87).

[39] A. Kendall and Y. Gal. “What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision?” In: *NeurIPS*. 2017 (cit. on pp. 48, 90).

[40] T. N. Kipf and M. Welling. “Semi-Supervised Classification with Graph Convolutional Networks”. In: *International Conference on Learning Representations (ICLR)*. 2017 (cit. on pp. 5, 55, 56, 69).

[41] P. Krähenbühl and V. Koltun. “Efficient inference in fully connected CRFs with Gaussian edge potentials”. In: *Proceedings of the 24th International Conference on Neural Information Processing Systems (NIPS)*. 2011 (cit. on pp. 4, 62, 71, 72).

[42] A. Krizhevsky, I. Sutskever, and G. E. Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *NeurIPS* (2012) (cit. on pp. 4, 14, 24, 25).

[43] V. Kumar and X. Wu, eds. *The Top Ten Algorithms in Data Mining*. CRC Press, 2009 (cit. on p. 9).

[44] T. LaBonte, C. Martinez, and S. Roberts. “We Know Where We Don’t Know: 3D Bayesian CNNs for Credible Geometric Uncertainty”. arXiv:1910.10793. 2019 (cit. on p. 86).

[45] Y. LeCun, B. Boser, J. S. D. D. Henderson, R. Howard, W.Hubbard, and L. D. Jackel. “Backpropagation Applied to Handwritten Zip Code Recognition”. In: *Neural Computation* (1989) (cit. on p. 23).

[46] Y. Li and W. Ping. “Cancer Metastasis Detection With Neural Conditional Random Field”. In: *1st Conference on Medical Imaging with Deep Learning (MIDL 2018)*. 2018 (cit. on p. 62).

[47] J. Long, E. Shelhamer, and T. Darrell. “Fully Convolutional Networks for Semantic Segmentation”. In: *Conference on Computer Vision and Pattern Recognition*. 2015 (cit. on pp. 4, 37, 38).

[48] A. L. Maas, A. Y. Hannun, and A. Y. Ng. “Rectifier Nonlinearities Improve Neural Network Acoustic Models”. In: *ICML*. 2013 (cit. on p. 14).

[49] T. M. Mitchell. *Machine Learning*. McGraw-Hill Science/Engineering/Math, 1997 (cit. on pp. 7–9, 13, 16, 17).

[50] X. Mo, K. Tao, Q. Wang, and G. Wang. “An Efficient Approach for Polyps Detection in Endoscopic Videos Based on Faster R-CNN”. In: *International Conference on Pattern Recognition*. 2018 (cit. on pp. 4, 33).

[51] T. Nair, D. Precup, D. L. Arnold, and T. Arbel. “Exploring Uncertainty Measures in Deep Networks for Multiple Sclerosis Lesion Detection and Segmentation”. In: *Medical Image Computing and Computer Assisted Intervention (MICCAI)*. 2018 (cit. on pp. 5, 48).

[52] A. Ojeda-Pat, A. Martin-Gonzalez, and R. Soberanis-Mukul. “Convolutional neural network U-Net for Trypanosoma cruzi segmentation”. In: *International Symposium on Intelligent Computing Systems* (2020) (cit. on pp. 35, 93).

[53] O. Oktay, E. Ferrante, K. Kamnitsas, et al. “Anatomically constrained neural networks (ACNNs): application to cardiac image enhancement and segmentation”. In: *IEEE transactions on medical imaging* 37.2 (2017), pp. 384–395 (cit. on p. 90).

[54] O. Oktay, J. Schlemper, L. L. Folgoc, et al. “Attention U-Net: Learning Where to Look for the Pancreas”. In: *MIDL*. 2018 (cit. on p. 42).

[55] O. Oktay, J. Schlemper, L. L. Folgoc, et al. “Attention U-Net: Learning Where to Look for the Pancreas”. In: *1st Conference on Medical Imaging with Deep Learning (MIDL 2018)*. 2018 (cit. on p. 86).

[56] D. L. Poole and A. K. Mackworth. *Artificial Intelligence Foundations of Computational Agents*. Cambridge University Press, 2010 (cit. on p. 8).

[57] S. Ren, K. He, R. Girshick, and J. Sun. “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”. In: *NeurIPS* (2015) (cit. on p. 30).

[58] O. Ronneberger, P. Fischer, and T. Brox. “U-Net: Convolutional Networks for Biomedical Image Segmentation”. In: *Medical Image Computing and Computer Assisted Intervention (MICCAI)*. 2015 (cit. on pp. 38, 71).

[59] L. Rosado, J. M. C. da Costa, D. Elias, and J. S. Cardoso. “A Review of Automatic Malaria Parasites Detection and Segmentation in Microscopic Images”. In: *Anti-Infective Agents* (2016) (cit. on p. 20).

[60] H. R. Roth, A. Farag, L. Lu, E. B. Turkbey, and R. M. Summers. “Deep convolutional networks for pancreas segmentation in CT imaging”. In: *SPIE Medical Imaging* (2015) (cit. on p. 41).

[61] H. R. Roth, A. Farag, E. B. Turkbey, L. Lu, J. Liu, and R. M. Summers. “Data From Pancreas-CT. The Cancer Imaging Archive”. http://doi.org/10.7937/K9/TCIA.2016.tNB1kqBU. 2016 (cit. on pp. 35, 72).

[62] H. R. Roth, L. Lu, A. Farag, et al. “DeepOrgan: Multi-level Deep Convolutional Networks for Automated Pancreas Segmentation”. In: *Medical Image Computing and Computer Assisted Intervention (MICCAI)*. 2015 (cit. on pp. 35, 72).

[63] H. R. Roth, L. Lu, N. Lay, et al. “Spatial Aggregation of Holistically-Nested Convolutional Neural Networks for Automated Pancreas Localization and Segmentation”. In: *Medical Image Analysis* (2018) (cit. on pp. 41, 42).

[64] A. G. Roy, S. Conjeti, N. Navab, and C. Wachinger. “Inherent Brain Segmentation Quality Control from Fully ConvNet Monte Carlo Sampling”. In: *Medical Image Computing and Computer Assisted Intervention (MICCAI)*. 2018 (cit. on p. 49).

[65] A. G. Roy, S. Conjeti, N. Navab, C. Wachinger, et al. “Bayesian QuickNAT: Model uncertainty in deep whole-brain segmentation for structure-wise quality control”. In: *NeuroImage* 195 (2019), pp. 11–22 (cit. on p. 85).

[66] A. G. Roy, S. Conjeti, N. Navab, C. Wachinger, et al. “QuickNAT: A fully convolutional network for quick and accurate segmentation of neuroanatomy”. In: *NeuroImage* 186 (2019), pp. 713–727 (cit. on p. 85).

[67] S. Russell and P. Norvig. *Artificial Intelligence a Modern Approach*. second. Prentice Hall, 2003 (cit. on p. 8).

[68] D. Scherer, A. Müller, and S. Behnke. “Evaluation of Pooling Operations in Convolutional Architectures for ObjectRecognition”. In: *20th International Conference on Artificial Neural Networks*. 2010 (cit. on p. 25).

[69] R. Selvan, T. Kipf, M. Welling, J. H. Pedersen, J. Petersen, and M. de Bruijne. “Extraction of Airways using Graph Neural Networks”. In: *1st Conference on Medical Imaging with Deep Learning (MIDL 2018)*. 2018 (cit. on p. 58).

[70] S. Y. Shin, S. Lee, I. D. Yun, and K. M. Lee. “Deep Vessel Segmentation By Learning Graphical Connectivity”. In: *Medical Image Analysis* (2019) (cit. on p. 58).

[71] Y. Shin, H. A. Qadir, L. Aabakken, J. Bergsland, and I. Balasingham. “Automatic colon polyp detection using region based deep cnn and post learning approaches”. In: *IEEE Access* (2018) (cit. on p. 33).

[72] R. L. Siegel, K. D. Miller, S. A. Fedewa, et al. “Colorectal cancer statistics, 2017”. In: *A Cancer Journal for Clinicians* (2017) (cit. on p. 31).

[73] R. L. Siegel, K. D. Miller, and A. Jemal. “Cancer statistics, 2018”. In: *CA: a cancer journal for clinicians* 68.1 (2018), pp. 7–30 (cit. on p. 31).

[74] K. Simonyan and A. Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: *ICLR 2015* (2015) (cit. on pp. 4, 29).

[75] A. L. Simpson, M. Antonelli, S. Bakas, et al. “A large annotated medical image dataset for the development and evaluation of segmentation algorithms”. arXiv:1902.09063. 2019 (cit. on p. 72).

[76] R. Soberanis-Mukul, V. Uc-Cetina, C. Brito-Loeza, and H. Ruiz-Piña. “An automatic algorithm for the detection of Trypanosoma cruzi parasites in blood sample images”. In: *Computer methods and programs in biomedicine* (2013) (cit. on p. 35).

[77] R. D. Soberanis-Mukul, S. Albarqouni, and N. Navab. “Polyp-artifact relationship analysis using graph inductive learned representations”. In: *arXiv preprint arXiv:2009.07109* (2020) (cit. on p. 89).

[78] R. D. Soberanis-Mukul, N. Navab, and S. Albarqouni. “An Uncertainty-Driven GCN Refinement Strategy for Organ Segmentation”. In: *MIDL20 Special Issue at the Journal of Machine Learning for Biomedical Imaging (MELBA)*. 2020 (cit. on pp. 61, 93).

[79] R. D. Soberanis-Mukul, N. Navab, and S. Albarqouni. “Uncertainty-based Graph Convolutional Networks for Organ Segmentation Refinement”. In: *International Conference on Medical Imaging with Deep Learning (MIDL 2020)*. 2020 (cit. on pp. 61, 93).

[80] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. “Dropout: A Simple Way to Prevent Neural Networks fromOverfitting”. In: *Journal of Machine Learning Research* (2014) (cit. on pp. 24, 26).

[81] D. Szucs and J. P. A. Ioannidis. “When Null Hypothesis Significance Testing Is Unsuitable for Research: A Reassessment”. In: *Frontiers in Human Neuroscience* (2017) (cit. on p. 72).

[82] A. Tomczack, N. Navab, and S. Albarqouni. “Learn to estimate labels uncertainty for quality assurance”. In: *arXiv preprint arXiv:1909.08058* (2019) (cit. on p. 90).

[83] D. Vázquez, J. Bernal, F. J. Sánchez, et al. “A benchmark for endoluminal scene segmentation of colonoscopy images”. In: *Journal of healthcare engineering* (2017) (cit. on p. 87).

[84] G. Wang, W. Li, M. A. Zuluaga, et al. “Interactive medical image segmentation using deep learning with image-specific fine tuning”. In: *EEE Transactions on Medical Imaging* (2018) (cit. on p. 62).

[85] Y. Wang, Y. Zhou, W. Shen, S. Park, E. K. Fishman, and A. L. Yuille. “Abdominal multi-organ segmentation with organ-attention networks and statistical fusion”. In: *Medical image analysis* 55 (2019) (cit. on p. 86).

[86] Y. Xia, F. Liu, D. Yang, et al. “3D Semi-Supervised Learning with Uncertainty-Aware Multi-View Co-Training”. arXiv:1811.12506. 2018 (cit. on p. 48).

[87] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. “How transferable are features in deep neural networks?” In: *NeurIPS* (2014) (cit. on p. 28).

[88] L. Yu, S. Wang, X. Li, C.-W. Fu, and P.-A. Heng. "Uncertainty-aware Self-ensembling Model for Semi-supervised 3D Left Atrium Segmentation". In: *Medical Image Computing and Computer Assisted Intervention (MICCAI)*. 2019 (cit. on p. 48).

[89] P. A. Yushkevich, J. Piven, H. C. Hazlett, et al. "User-Guided 3D Active Contour Segmentation of Anatomical Structures: Significantly Improved Efficiency and Reliability". In: *Neuroimage* 31.3 (2006), pp. 1116–1128 (cit. on p. 76).

[90] L. Zhang, L. Wang, and D. Zhu1. "Recovering Brain Structural Connectivity from Functional Connectivity via Multi-GCN Based Generative Adversarial Network". In: *MICCAI*. 2020 (cit. on p. 53).

[91] Y. Zhou, Z. Li, S. Bai, et al. "Prior-Aware Neural Network for Partially-Supervised Multi-Organ Segmentation". In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)* (2019), pp. 10671–10680 (cit. on p. 86).

[92] Y. Zhou, L. Xie, W. Shen, Y. Wang, E. K. Fishman, and A. L. Yuille. "A Fixed-Point Model for Pancreas Segmentation in Abdominal CT Scans". In: *MICCAI*. 2017 (cit. on pp. 41, 68).

[93] Z. Zhu, Y. Xia, W. Shen, E. K. Fishman, and A. L. Yuille. "A 3D Coarse-to-Fine Framework for Volumetric Medical Image Segmentation". In: *International Conference on 3D Vision*. 2018 (cit. on p. 42).

[94] Z. Zhu, Y. Xia, W. Shen, E. K. Fishman, and A. L. Yuille. "A 3D Coarse-to-Fine Framework for Volumetric Medical Image Segmentation". In: *2018 International Conference on 3D Vision (3DV)*. 2018 (cit. on p. 86).

List of Figures

List of Tables