Alma Mater Studiorum · Università di Bologna

SCHOOL OF SCIENCE
Degree Programme in Mathematics

Hierarchical Segmentation of the Female Pelvis: A Novel Framework for Uterus, Healthy Ovaries, and Ovarian Lesions

Master's Thesis in Mathematics

Supervisor:
Prof. Federica Gerace

Tutor:
Niccolò Tallone

Presented by:
Niccolò Chiari

Academic year 2024-2025

Contents

Introduction **i**

1 Preliminary Notions **1**
1.1 Image Segmentation 1
1.2 Fundamentals of Deep Learning 2
1.2.1 Introduction to Supervised Learning 3
1.2.2 Gradient-based learning algorithms 3
1.2.3 The Problem of Generalization 9
1.2.4 Neural Networks 10
1.2.5 Activation function 14
1.2.6 Training procedure of a Neural Network 16
1.3 Deep Neural Network Architectures 19
1.3.1 Multilayer Perceptron 20
1.3.2 Convolutional Neural Networks 21
1.3.3 U-Net 26
1.3.4 PSPNet 27
1.4 Metrics evaluation 28

2 Dataset Analysis **32**
2.1 Data Preparation for Segmentation 33
2.2 Data Preprocessing 37
2.3 Data Augmentation 40

3 A Novel Approach to Image Segmentation **46**
3.1 *Flat* Semantic Segmentation 50
3.2 Literature Review 51
3.2.1 Focal Hierarchical Loss for Semantic Segmentation 52
3.2.2 Hierarchical Semantic Segmentation Network 56

4 Numerical results **64**
4.1 *Flat* Segmentation Cross-Validation 65
4.2 Focal hierarchical Loss Cross-Validation 70
4.3 HSSN Cross-Validation 74
4.4 Evaluation of Segmentation Performance 81
4.4.1 Class-wise Evaluation Performance 81
4.4.2 Global Evaluation 88

Conclusions **91**

Bibliography **93**

List of Figures

1.1 Example of segmentation for a street-view image. 1
1.2 Comparison between semantic, instance, and panoptic segmentation of a streetview image. 2
1.3 Gradient descent minimizing $J(\theta)$. 4
1.4 Explanation of the importante of the learning rate η for gradient descent 5
1.5 Comparison between Gradient Descent and Stochastic Gradient Descent 8
1.6 Perceptron model by Rosenblatt, 1958 11
1.7 Graphical representation of a shallow neural network 14
1.8 ReLU, sigmoid, and tanh activation functions. 15
1.9 Activation function for a multiclass classification problem 15
1.10 Training workflow of a neural network with multiple hidden layers . . 16
1.11 Forward pass through the entire neural network, starting with the input data $\boldsymbol{x}_0$ to be processed layer by layer producing $\boldsymbol{x}_1 \dots \boldsymbol{x}_L$ . . . 17
1.12 Backpropagation algorithm computing gradients $\mathbf{g}_L \dots \mathbf{g}_1$ 18
1.13 Graph representation of a Multilayer Perceptron architecture characterized by $L = 3$ hidden layers. 20
1.14 Overview of a convolutional neural network 22
1.15 Convolution operation given a kernel $K \in \mathbb{R}^{3\times3}$, stride 1. 23
1.16 Example of a zero-padding operation, application of a kernel $K \in \mathbb{R}^{3\times3}$ with stride one. 24
1.17 Example of a single feature map extraction from the convolutional layer, application of a kernel $K \in \mathbb{R}^{3\times3}$ with stride one, followed by ReLu activation function 25
1.18 Illustration of pooling operation on a single feature map, $f \in \mathbb{R}^{2\times2}$, stride 1 25
1.19 Overview of a UNet architecture 27
1.20 Overview of a Pyramid scene parsing netowrk for an image segmenattion task 28
1.21 Example of a confusion matrix for a binary classification problem . . 29

2.1 General structure for the raw data 32
2.2 Examples of different ultrasound image representations in the dataset. 33
2.3 Distribution of the clinical cases among the 5 folds, for both training and validation. 34
2.4 Distribution of training and validation items across the five folds. . . 35
2.5 Distribution of annotated training and validation frames across the five folds 36

2.6 Pixel distribution across the classes for each of the five training folds 36
2.7 Visual comparison between original and preprocessed images and their corresponding masks. 38
2.8 Pixels distribution of original and preprocessed ultrasound images of training dataset. 39
2.9 Distribution of standard deviation across the training folds for the original and preprocessed datasets. 39
2.10 Cumulative distribution function of pixel intensities for the original and pre-processed training datasets. 40
2.11 Training procedure with online data augmentation. 40

3.1 Segmentation of an adnexal mass produced by the OvAi-2 segmentation algorithm. GT: ground-truth segmentation mask representing the adnexal mass. OVAI2: predicted segmentation mask generated by the OvAi-2 segmentation pipeline. 47
3.2 Segmentation of the same adnexal mass of Figure 3.1 provided by OvAi-3 segmnetation algorithm. Example of the misclassification of the mask. GT: ground-truth segmentation mask representing the adnexal mass. OVAI3: predicted segmentation mask generated by the OvAi-3 segmentation pipeline. 47
3.3 Segmentation of a healthy ovary produced by the OvAi-3 segmentation algorithm, illustrating an example of the class-reversion process. GT: ground-truth segmentation mask representing the healthy ovary. OVAI3: predicted segmentation mask generated by the OvAi-3 segmentation pipeline. 48
3.4 Class-label mapping used for the categorical representation of segmentation masks. 50
3.5 Hierarchical tree, *root* node represented as the entire ultrasound image obtain through transvaginal ultrasound imaging. 54
3.6 Representation of the branches at each level of the hierarchical tree. . 55
3.7 (a) Representation of the Positive $\mathcal{T}$-property (in red) and Negative $\mathcal{T}$-property(in blue). (b) Violation of Positive $\mathcal{T}$ constraint and Negative $\mathcal{T}$ constraint. (c) Adjustment of the score values through a loss. 59
3.8 Feature embedding map obtained from the encoder applied to a transvaginal ultrasound image. 61
3.9 Visual representation of the pixel embedding space after the application of the tree-triplet loss $\mathcal{L}^{\mathrm{TT}}$. 63

4.1 Mean training and validation F_1 score for the three best (γ, α) configurations (see Equation (4.1)) obtained from five-fold cross-validation, focal hierarchical loss approach. See Table 4.1 for the cross-validation set up. 66
4.2 Mean training and validation IoU for the three best (γ, α) configurations (see Equation (4.1)) obtained from five-fold cross-validation, focal hierarchical loss approach. See Table 4.1 for the cross-validation set up. 67

4.3 Mean training and validation F_1 score computed over the five folds of the cross-validation, *flat* segmentation approach. *Cross-validation setup*: learning rate 2×10^{-4}, batch size 48, epochs 650, $\gamma = 1.5$, $\boldsymbol{\alpha}$ weighted, data augmentation enabled. 68
4.4 Mean training and validation IoU computed over the five folds of the cross-validation, *flat* segmentation approach. *Cross-validation setup*: learning rate 2×10^{-4}, batch size 48, epochs 650, $\gamma = 1.5$, $\boldsymbol{\alpha}$ weighted, data augmentation enabled. 69
4.5 Mean training and validation F_1 score for the three best (γ, α) configurations (see Equation (4.2)) obtained from five-fold cross-validation, focal hierarchical loss approach. See Table 4.1 for the cross-validation set up 71
4.6 Mean training and validation IoU for the three best (γ, α) configurations (see Equation (4.2)) obtained from five-fold cross-validation, focal hierarchical loss approach. See Table 4.1 for the cross-validation set up. 72
4.7 Mean training and validation F_1 score computed over the five folds of the cross-validation for the focal hierarchical loss approach. *Cross validation setup*: 2×10^{-4}, batch size: 48, epochs: 650, $\gamma = 1.5$, $\boldsymbol{\alpha}$ weighted, data augmentation enabled. 73
4.8 Mean training and validation IoU computed over the five folds of the cross-validation for the focal hierarchical loss approach. *Cross validation setup*: 2×10^{-4}, batch size: 48, epochs: 650, $\gamma = 1.5$, $\boldsymbol{\alpha}$ weighted, data augmentation enabled. 73
4.9 Mean Training and validation F_1 score for the number of triplets comparison (see Equation (4.2)) obtained from five-fold cross-validation, HSSN framework. See Table 4.10 for the cross-validation setup. . . . 76
4.10 Mean Training and validation IoU for the number of triplets comparison (see Equation (4.2)) obtained from five-fold cross-validation, HSSN framework. See Table 4.10 for the cross-validation setup 76
4.11 Mean training and validation F_1 score computed over the five folds of the cross validation for the HSSN framework. *Cross validation setup*: learning rate: 8×10^{-4}, batchsize: 48, epochs: 650, $\gamma = 1.5$, $\boldsymbol{\alpha}$ weighted, data augmentation enabled. 78
4.12 Mean training and validation IoU computed over the five folds of the cross validation for the HSSN framework. *Cross validation setup*: learning rate: 8×10^{-4}, batchsize: 48, epochs: 650, $\gamma = 1.5$, $\boldsymbol{\alpha}$ weighted, data augmentation enabled. 79
4.13 Mean training and validation F_1 score computed over the five folds of the cross validation for the HSSN framework. *Cross validation setup*: learning rate: 3×10^{-4}, batchsize: 48, epochs: 650-1100, $\gamma = 1.5$, $\boldsymbol{\alpha}$ weighted, data augmentation enabled. 80
4.14 Mean training and validation IoU computed over the five folds of the cross validation for the HSSN framework. *Cross validation setup*: learning rate: 3×10^{-4}, batchsize: 48, epochs: 650-1100, $\gamma = 1.5$, $\boldsymbol{\alpha}$ weighted, data augmentation enabled. 80

4.15 Class-wise F1-score comparison among the three adopted methods. . 82
4.16 Class-wise IoU comparison among the threee adopted methods. . . . 82
4.17 True confusion matrices obtained from the evaluation on the test set. 83
4.18 Pred. confusion matrices obtained from the evalaution on the test set. 85
4.19 True confusion matrices obtained from the evaluation on the balanced test set. 87
4.20 Pred. confusion matrices obtained from the evaluation on the balanced test set. 88

List of Tables

4.1 Initial hyperparameter configuration used for the five-fold cross-validation in the *flat* segmentation approach. The experiments are performed on the standard dataset (without preprocessing), and data augmentation is disabled in order to evaluate the model performance on raw data. 65

4.2 Validation performance obtained from five-fold cross-validation for all combinations of the focal loss hyperparameters γ and $\boldsymbol{\alpha}$ in the *flat* segmentation approach. 66

4.3 Five-fold configuration adopted after selecting the optimal hyperparameters, *flat* segmentation approach. Data augmentation is enabled and the same configuration is applied to both the standard and the preprocessed datasets in order to evaluate the effect of the preprocessing. 68

4.4 Validation performance obtained from five-fold cross-validation using the selected hyperparameter configuration with data augmentation enabled, *flat* segmentation approach. The table compares results obtained using the standard dataset and the preprocessed dataset. . . . 69

4.5 Final hyperparameter configuration adopted for the *flat* segmentation approach after cross-validation and dataset comparison. 70

4.6 Validation performance obtained from five-fold cross-validation for all combinations of the hyperparameters γ and $\boldsymbol{\alpha}$ for the focal hierarchical loss. 70

4.7 Five-fold cross validation configuration adopted after selecting the optimal hyperparameters, focal hierarchical loss casee. Data augmentation is enabled and the same configuration is applied to both the standard and the preprocessed datasets in order to evaluate the effect of the preprocessing. 72

4.8 Validation performance obtained from five-fold cross-validation using the selected hyperparameter configuration in Table 4.7 with data augmentation enabled, focal hierarchical loss configuration. The table compares the standard dataset and the preprocessed dataset, both with data augmentation enabled. 74

4.9 Final hyperparameter configuration adopted for the focal hierarchical loss after cross-validation and dataset comparison. The model is trained on the preprocessed dataset with data augmentation enabled. 74

4.10 Initial hyperparameter configuration used for the five -fold cross-validation for the HSSN framework. The expriemtns are performed on the standard dataset without preprocessing, and data augmentation is disabled. 75
4.11 Estimated computational time for different values of the triplet-number hyperparameter across the five-folds of the cross-validation pipeline, HSSN framework. The computational time reported are the mean on the five-folds. 77
4.12 Validation performance obtained from five-fold cross-validation for the different number of sampled triplets, HSSN framework. 77
4.13 Five-fold cross validation configuration adopted after selecting the optimal hyperparameters for the HSSN framework. Data augmentation is enabled, and the configuration is applied only to the preprocessed dataset due to the high computational cost (training time exceeding two days). 78
4.14 Five-fold cross validation configuration adopted after the first 650 epochs for the HSSN framework. Data augmentation is enabled and the configuration is applied only to the preprocessed dataset. 79
4.15 Validation mean IoU and F_1 score over training epochs, obtained from five-fold cross-validation. Training was performed in two phases: the first 650 epochs using the hyperparameters in Table 4.13, followed by a resumed training phase as in Table 4.14 with data augmentation enabled. Results are reported for the preprocessed dataset only. . . . 81
4.16 Final hyperparameter configuration adopted for the HSSN framework after cross-validation and dataset comparison. The model is cross-validated on preprocessed dataset with data augmentation enabled. . 81
4.17 Metrics evaluation over all classes of the dataset 89
4.18 Class-wise recall comparison between Flat Segmentation and Focal Hierarchical Loss. 89
4.19 Class-wise precision comparison between Focal Loss and Focal Hierarchical Loss. 90

Introduction

Medical imaging is a tool used by doctors in the diagnosis and treatment of various diseases. Specifically, it is used to provide visual representations of the body's internal structure such as bones, muscles, organs and other parts of the body. In a clinical setting, the most commonly used methods are X-ray, magnetic resonance imaging (MRI), computed tomography (CT), and ultrasound (US). Each imaging modality has its unique advantages for different applications and diagnostic purposes, as well as their limitations.

Over the years, US imaging has gained ground due to its non-invasive nature. Unlike other imaging modalities, it does not include ionizing radiation instead uses high-frequency sound waves to create eal-time images of internal body structures. US imaging is particularly effective for visualizing soft tissues and organs such as the liver and pancreas, as well as pelvic organs including the uterus, ovaries, and prostate, making it a valuable tool for detecting tumors and cysts. Despite these advantages, clinicians still face challenges during image analysis and diagnosis. The main issues are reported below:

- Difficulties in visual interpretation of the results due to low image resulution and high presence of noise;
- Strong dependence on the operator's expertise. Unlike CT scans, where the machine automatically performs the acquisition, ultrasound examinations require direct manipulation of the probe by the physician. This may lead to diagnostic errors, such as difficulty in correctly identifying regions of interest (ROIs), including lesions.

For this reason, only a highly experienced clinician can make an appropriate diagnostic interpretation of the US images.

Ovarian tumors, primarily detected through transvaginal ultrasound (US) imaging, are often referred to as silent killers. Their symptoms are typically vague and frequently attributed to other conditions. As a result, in many cases the disease is diagnosed at an advanced stage, significantly reducing survival rates compared to those associated with early detection.

In this context, SynDiag is a company established to support physicians in the early diagnosis of ovarian tumors. To assist the diagnostic process, the company has developed a deep learning–based segmentation framework for ultrasound images, which performs frame-by-frame analysis by assigning a class label to each pixel.

However, particularly in the medical domain, strong relationships exist between

anatomical structures and their components. To date, the segmentation frameworks adopted by SynDiag do not explicitly model the hierarchical relationships among these structures. As a consequence, such approaches may not be optimal.

In this thesis, we investigate the transition from standard segmentation frameworks, where relationships between structures are not explicitly considered, to a hierarchical segmentation framework in which anatomical structures are organized according to an explicit hierarchy.

In particular, in addition to the segmentation of structures such as healthy ovaries and adnexal masses, this work introduces the segmentation of the uterus, with a specific focus on its components, namely the myometrium and the endometrium. These anatomical structures had not previously been included in the segmentation framework developed by SynDiag.

Furthermore, a comparison between non-hierarchical and hierarchical approaches is conducted on these structures. In particular, we analyze the ability of the hierarchical method to accurately predict the components of the uterus and the healthy ovary, demonstrating an improvement of approximately 10 percentage points in accuracy compared to a non-hierarchical approach of the same type as the one adopted by SynDiag. The thesis is organized as follows. Chapter 1 presents the fundamental theoretical concepts underlying this work. It begins with an introduction to the image segmentation problem, outlining its evolution from traditional computer vision techniques to deep learning–based approaches.

The chapter then introduces the core concepts of deep learning, including supervised learning and its formulation as an optimization problem, together with the main gradient descent algorithms used for training. The concept of generalization and the principal techniques to improve it are also discussed.

Finally, neural networks are presented, from early models to deep architectures, along with their training process. The main architectures used in this thesis are described, and the evaluation metrics for image segmentation are introduced to assess and compare model performance.

Chapter 2 presents the structure of the medical dataset provided by SynDiag, which is used for the image segmentation task. In particular, a comprehensive description of the data preparation process is provided, which is necessary for the subsequent use of the data within the models during the training phase.

Chapter 3 presents the segmentation problem associated with the dataset introduced in the previous chapter. In particular, two different approaches to the segmentation task are proposed: a first approach in which the relationships between the organs and their components are not considered, and a second hierarchical approach that takes into account the hierarchical structure of the problem. Regarding the hierarchical approach, two different variants are proposed.

Chapter 4 presents the numerical results obtained. In particular, the first part of the chapter describes the procedure through which the best configuration for each model is obtained. Subsequently, after identifying the hierarchical approach that achieves the best performance, a comparison with the non-hierarchical approach is performed.

Chapter 1

Preliminary Notions

1.1 Image Segmentation

Image segmentation is a computer vision technique consisting in the process of partitioning digital images into multiple image segments or objects also called sets of pixels. More precisely, it consist in assigning a label to each pixel in an image, in this way pixels with same labels share same characteristics. A visual representation follows in Figure 1.1.

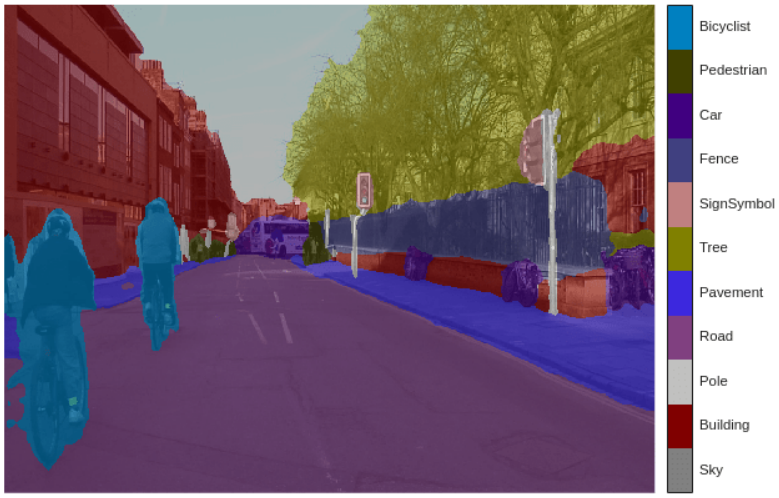


Figure 1.1: Example of segmentation for a street-view image.

Image segmentation is involved into a high number of applications, including medical image analysis (e.g., tumor detection), autonomous vehicles (e.g., pedestrian detection), satellite imagery (e.g., target identification), and more.

There are three different types of image segmentation:

- **Semantic segmentation**: Consists of assigning to every pixel in the image a semantic class. It does not make differences between instances of the same semantic class.
- **Instance segmentation**: It has the opposite approach of semantic algorithms, whereas semantic segmentation algorithms predict only semantic classes

of each pixel, instance segmentation algorithms identify instances inside the same semantic class.

- **Panoptic segmentation**: Consist in the possibility of both determine semantic classification of all pixels and differentiate each object instance in an image, combining the benefits of both semantic and instance segmentation.

The result of the three different algorithms is represented in 1.2

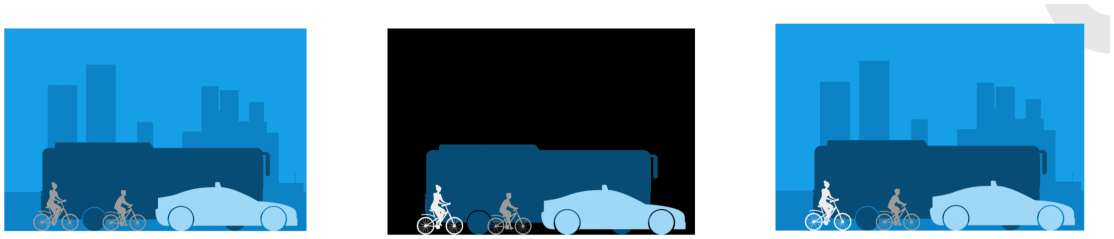


Figure 1.2: Comparison between semantic, instance, and panoptic segmentation of a streetview image.

In medical imaging, and particularly for ultrasound images, semantic segmentation is preferred over instance segmentation because it effectively identifies regions of interest (ROI), like organs or tumors, without the unnecessary complexity of distinguishing individual objects. Over the years, segmentation models have evolved from conventional computer vision techniques, like thresholding and clustering methods, to deep learning approaches, which have achieved state-of-the-art (SOTA) performance on benchmark datasets.

Deep learning, a branch of artificial intelligence, employs architectures known as deep neural networks to learn directly from input raw input data. To understand how these architectures function, it is useful to introduce fundamental concepts such as neural networks, activation functions, training algorithms, and regularization methods.

1.2 Fundamentals of Deep Learning

Deep learning is a subset of Machine Learning (ML), a branch of artificial intelligence, that enables models to learn from data how to perform specific tasks through architectures known as deep neural networks, without requiring explicit human intervention.

In this framework, models automatically extract relevant information from raw data. The mechanisms underlying this process will be discussed in the following sections.

In the machine learning field, different learning paradigms can be identified. This thesis focuses on the image segmentation problem, which is formulated as a pixel-wise classification task and is addressed within the supervised learning paradigm.

1.2.1 Introduction to Supervised Learning

Supervised learning [20] is a machine learning paradigm in which the goal is to learn a function that maps inputs to known outputs using a dataset of labeled examples. The problem can be explained in the following way:

Given a unknown function $f : \mathcal{X} \to \mathcal{Y}$ between spaces $\mathcal{X}$ and $\mathcal{Y}$, the aim is to find an approximation of f using a labelled dataset

$$\mathcal{D} = \left\{(\boldsymbol{x}^{(k)}, \boldsymbol{y}^{(k)})\right\}_{k=1}^{N},$$

where $\boldsymbol{x}^{(k)} \in \mathcal{X}$ represents the k-th input sample and $\boldsymbol{y}^{(k)} \in \mathcal{Y}$ the corresponding target output.

The supervised learning task can be resumed in these three steps:

1. Choosing a prediction function, also called model, $F : \mathcal{X}\times\times \to \mathcal{Y}$ parametrized by a space of parameters Θ.

2. Choose a loss function, also called objective function, $\mathcal{L} : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$, such that it compares the output of our model $F(\boldsymbol{x}^{(k)}, \theta)$ with $\boldsymbol{y}^{(k)}$;

3. Choose $\theta \in \times$ such that $F(\boldsymbol{x}^{(k)}, \theta)$ is the best approximation of $\boldsymbol{y}^{(k)}$.

Before digging into the learning task, it is important to distinguish between model parameters and hyperparameters. The parameters θ are the quantities that determine the behavior of the model and are adjusted during the learning process. Hyperparameters, instead, are configuration choices that are fixed prior to learning and influence the structure or the training procedure of the model.

As regards the steps introduced above, the first coincides with the choice of the model, which, in this thesis, corresponds to deep neural networks. The second and third steps can be formulated as an optimization problem: computing the value θ^* that minimizes the loss, i.e.:

$$\theta^* = \underset{\theta\in\Theta}{\operatorname{argmin}} \sum_{k=1}^{N} \frac{1}{N} \mathcal{L}\Big(F(\boldsymbol{x}^{(k)}; \theta), \boldsymbol{y}^{(k)}\Big). \tag{1.1}$$

This formulation corresponds to the empirical risk minimization (ERM) principle, it allows the model to learn from available data in order to generalize on unseen data.

1.2.2 Gradient-based learning algorithms

The empirical risk minimization problem introduced in the previous subsection is an optimization problem.

To solve this problem, we employ gradient-based learning algorithms. The simplest method in this family is gradient descent.

In this subsection, we consider $J : \Theta :\to \mathbb{R}$, $\Theta \subset \mathbb{R}^n$ be an objective function, continuos, that maps a set of input parameters to a scalar value. We denote this function by $J(\theta)$.

Our task is to solve:

$$\theta^* = \underset{\theta}{\operatorname{argmin}}\, J(\theta).$$

Gradient descent can be described by the following iterative procedure:

Algorithm 1 Gradient Descent (GD)

Require: Objective function J, initial parameter vector θ^0, learning rate η, number of steps K

Ensure: Trained parameter vector $\theta^* = \theta^K$

1: **for** $t = 0, \ldots, K-1$ **do**

2: $\quad \theta^{t+1} \leftarrow \theta^t - \eta \nabla_\theta J(\theta^t)$

3: **end for**

This algorithm is characterized by two hyperparameters, the **learning rate** $\eta \in \mathbb{R} > 0$, and the number of iterations K. A visual representation follows in Figure 1.3.

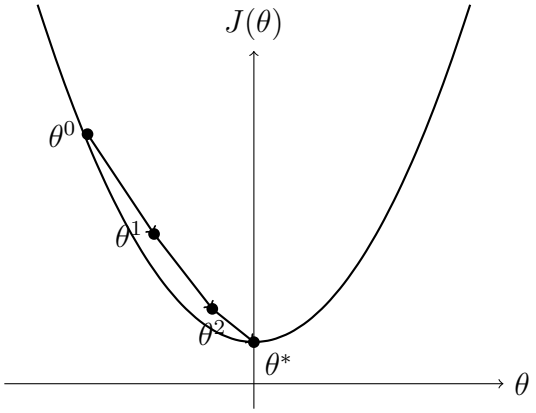


Figure 1.3: Gradient descent minimizing $J(\theta)$.

The key idea behind gradient descent is to iteratively update the parameter vector in the direction of the negative gradient of the objective function. The learning rate η constrols the magnitude of each step, and after K iterations, the algorithm produces the value $\theta^* = \theta^K$ that minimize the objective function J.

The choice of learning rate is extremely important as exemplified in 1.4.

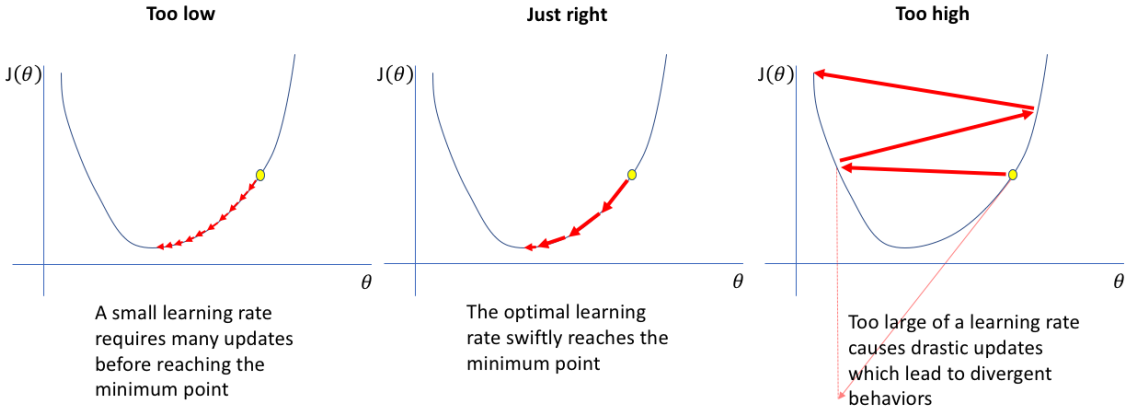


Figure 1.4: Explanation of the importante of the learning rate η for gradient descent

The gradient descent optimization algorithm is based on the computation of the gradient $\nabla_\theta J(\theta)$; therefore, the differentiability of the objective function is a fundamental assumption.

During the optimization process, gradient descent may converge to local minima, saddle points, or flat regions of the loss surface. This behavior can make it difficult to identify the global minimizer of the objective function θ^*.

In order to understand the conditions under which it is possible to find the unique global minimizer, we start introducing the following definitions

Definition 1 (First-order characterization of convexity) *Let $f : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ be a differentiable function. Then f is convex if and only if*

$$f(y) \geq f(x) + \nabla f(x)^\top (y - x), \quad \forall x, y \in \mathbb{R}^n$$

Definition 2 (Strictly convex function) *Let $f : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ be a differentiable function. The function f is said to be* strictly convex *if*

$$f(y) > f(x) + \nabla f(x)^\top (y - x), \quad \forall x, y \in \mathbb{R}^n, \ x \neq y$$

In particular, every strictly convex function is convex.

Definition 3 (Proper function) *Let $f : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$*

$$f \text{ is said proper} \iff \exists x \in \mathbb{R}^n \text{ s.t. } f(x) \neq 0$$

Definition 4 (Coercivity) *Let $f : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ continuous and proper function.*

$$f \text{ is coercive} \iff \forall x \in \mathbb{R}n \lim_{\|x\|_2 \to +\infty} f(x) = +\infty$$

The following theorem holds:

Theorem 1 *: Let $f : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ proper, continuous,coercive and strictly convex function, then f admits a unique minimizer.*

In this way, we have set the conditions under which the minimizer is unique and global.

We now study the convergence properties of the gradient descent method. To do so, we begin with the following definition.

Definition 5 *Let $f : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ be a proper, convex, continuous and differentiable function. We say that f is a gradient-Lipschitz continuous function on $\mathbb{R}^n$ if:*

$$\exists L \geq 0 : \forall x, y \in \mathbb{R}^n \quad \|\nabla f(x) - \nabla f(y)\| \leq L \|x - y\|$$

where L is called the Lipschitz constant.

The theorem that ensures the convergence of the gradient descent method is the following

Theorem 2 (Convergence of Gradient Descent) *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a convex and differentiable function whose gradient is L -Lipschitz continuous. Let x_k be the sequence generated by the gradient descent method:*

$$x^{(k+1)} = x^{(k)} - \eta \nabla f(x^{(k)}),$$

with constant step size

$$0 < \eta < \frac{2}{L} \quad (*)$$

Then:

- *Any ∇ that satisfies $(*)$ will descrease f.*
- *the sequence of iterates converges to a global minimizer x^**

In this way, we have established the conditions under which a unique global minimizer θ^* exists and gradient descent converges to θ^*.

At this point we can introduce the concept of **momentum** [23].

Algorithm 2 Gradient Descent with Momentum

1: **Input:** objective function J, initial parameter vector θ^0, learning rate η, momentum μ, number of steps K
2: **Output:** trained parameter vector $\theta^* = \theta^K$
3: **Initialize** $v^0 = 0$
4: **for** $t = 0, \ldots, K-1$ **do**
5: $\quad v^{t+1} = \mu v^t - \eta \nabla_\theta J(\theta^t)$
6: $\quad \theta^{t+1} \leftarrow \theta^t + v^{t+1}$
7: **end for**

It is an extension to the gradient descent method, it focus on adapting the step size of the gradient, in particular the quantity:

$$v^{t+1} = \mu v^t - \eta \nabla_\theta J(\theta^t),$$

is called first moment estimate, it is an exponentially moving average of the gradients.

Gradient descent with momentum is not an adaptive learning-rate algorithm: it uses a global learning rate, while the momentum term modifies the update direction and can change the effective step size during optimization.

In general, gradient descent is not optimal from a computational complexity point of view. To understand why, consider the case in which the objective function to be minimized has the following form:

$$J\left(\left\{(\boldsymbol{x}^{(k)}, \boldsymbol{y}^{(k)})\right\}_{k=1}^{N}, \theta\right) = \frac{1}{N}\sum_{k=1}^{N} \mathcal{L}\left(F\left(\boldsymbol{x}^{(k)}; \theta\right), \boldsymbol{y}^{(k)}\right),$$

which depends on the entire dataset $\mathcal{D} = \left\{(\boldsymbol{x}^{(k)}, \boldsymbol{y}^{(k)})\right\}_{k=1}^{N}$.

Computing the gradient $\nabla_\theta J$ requires evaluating this quantity N times and then summing the resulting contributions.

Suppose now that, instead of evaluating the objective function on the entire dataset $\left\{(\boldsymbol{x}^{(k)}, \boldsymbol{y}^{(k)})\right\}_{k=1}^{N}$, we consider a subset $\left\{(\boldsymbol{x}^{(k)}, \boldsymbol{y}^{(k)})\right\}_{k=1}^{B}$, with B batch size.

Batch size is an hyperparameter which defines the number of samples taken into consideration from the entire dataset. It is a good tradeoff between accuracy and speed.

The number of gradient computations is significantly reduced compared to using the entire dataset.

This variant of gradient descent, which relies on computing the gradient over a subset of size B, is known as **stochastic gradient descent**(SGD) [23], because each iteration of descent uses a different randomly (stochastically) sampled batch of training data to estimate the gradient. It is descriped by the following:

Algorithm 3 Stochastic gradient descent (SGD).

1: **Input:** initial parameter vector θ^0, data $\{(\mathbf{x}^{(k)}, \mathbf{y}^{(k)})\}_{k=1}^{N}$, learning rate η, batch size B, number of steps K
2: **Output:** trained parameter vector $\theta^* = \theta^K$
3: **for** $t = 0, \ldots, K-1$ **do**
4: $\quad \left\{(\boldsymbol{x}^{(b)}, \boldsymbol{y}^{(b)})\right\}_{b=1}^{B} \sim \left\{(\boldsymbol{x}^{(k)}, \boldsymbol{y}^{(k)})\right\}_{k=1}^{N}$ ▷ sample batch of training data
5: $\quad \tilde{\mathbf{g}} = \frac{1}{B}\sum_{b=1}^{B} \nabla_\theta \mathcal{L}\left(F\left(\boldsymbol{x}^{(b)}; \theta\right), \boldsymbol{y}^{(b)}\right)$
6: $\quad \theta^{t+1} \leftarrow \theta^t - \eta\, \tilde{\mathbf{g}}$
7: **end for**

Furthermore, thanks to its stochastic behavior, SGD can help the model converge and avoid getting stuck in poor local minima or saddle points.

As shown in Figure 1.5 gradient descent has a smooth path towards the minimum, instead stochastic gradient descent has a more oscillatory trajectory towards the minimum.

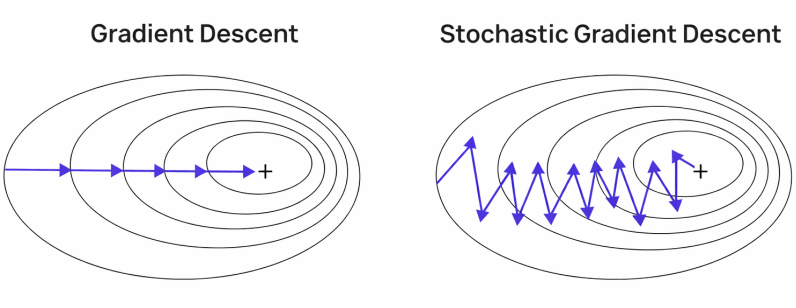


Figure 1.5: Comparison between Gradient Descent and Stochastic Gradient Descent

We now present Adaptive Moment Estimation (Adam) [9], an extension of stochastic gradient descent that incorporates the concept of momentum and introduces adaptive learning rates.

It is an adaptive learning-rate optimization algorithm based on the estimation of the first and second moments of the gradients.

- **First moment:** the exponential moving average of the gradients, which can be interpreted as an estimate of their mean.
- **Second moment:** the exponential moving average of the squared gradients, which provides an estimate of their variance.

In Adam, the first moment is represented by m, while the second moment is referred by v. Both are initialized as zero at the start of the algorithm.

Algorithm 4 Adam optimizer.

Input:

1. initial parameter vector θ^0, learning rate η, batch size B
2. dataset $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(k)}, \boldsymbol{y}^{(k)}\right)\right\}_{k=1}^{N}$, exponential decay rates β_1, $\beta_2 \in \mathbb{R}$, ϵ a constant

Output: trained parameter vector $\theta^* = \theta^K$

Initialize: $m_0 = 0$, $\nu_0 = 0$

for $t = 0, \ldots, K-1$ **do**

  $\left\{\left(\boldsymbol{x}^{(b)}, \boldsymbol{y}^{(b)}\right)\right\}_{b=1}^{B} \sim \left\{\left(\boldsymbol{x}^{(k)}, \boldsymbol{y}^{(k)}\right)\right\}_{k=1}^{N}$ ▷ sample batch of training data

  $\tilde{\mathbf{g}} = \frac{1}{B}\sum_{b=1}^{B} \nabla_\theta \mathcal{L}\left(F(\mathbf{x}^{(b)}; \theta), \mathbf{y}^{(b)}\right)$

  $\nu_{t+1} = \beta_1 \nu_t + (1-\beta_1)\tilde{\mathbf{g}}$

  $m_{t+1} = \beta_2 m_t + (1-\beta_2)(\tilde{\mathbf{g}})^2$

  $\hat{\nu}_{t+1} = \dfrac{\nu_{t+1}}{1-(\beta_1)^{t+1}}$

  $\hat{m}_{t+1} = \dfrac{m_{t+1}}{1-(\beta_2)^{t+1}}$

  $\theta^{t+1} \leftarrow \theta^t - \eta \dfrac{\hat{\nu}_{t+1}}{\sqrt{\hat{m}_{t+1} + \epsilon}}$

end for

As we can see above, in Adam algorithm we replace the gradient computation in the last optimization step with:

$$\tilde{\mathbf{g}} = \frac{\hat{\nu}_{k+1}}{\sqrt{\hat{m}_{k+1} + \epsilon}}.$$

This quantity, also called **signal to noise ratio**, represent the ratio between the first moment estimate, representative of the mean, and the square root of the second moment estimate, representative of the variance.

The numerator represent the direction in which we want our gradient to move, instead the denominator could be tought as the "noise", the variance of the gradient.

The step size η is adapted according to the ratio between these two quantities, and in particular to the level of "noise" in the gradients. If the gradients exhibit low variance, this indicates that we are moving in a consistent direction, allowing for larger update steps. Conversely, if the variance is high, this suggests that the direction is unstable or noisy, and therefore smaller step sizes are required to ensure a more controlled and reliable update.

1.2.3 The Problem of Generalization

A fundamental requirement of any supervised learning model is its ability to maintain good performance on unseen data. This property is referred to as **generalization**.

To assess and control generalization performance, the available dataset $\mathcal{D} = \left\{(\boldsymbol{x}^{(k)}, \boldsymbol{y}^{(k)})\right\}_{k=1}^{N}$ is typically divided into three disjoint subsets:

- **Training set**: The subset of data, denoted by $\mathcal{D}_{\text{train}}$, used to estimate the model parameters during the learning phase in order to minimize the empirical loss.
- **Validation set**: A disjoint subset $\mathcal{D}_{\text{val}} \subset \mathcal{D}_{\text{train}}$, obtained by the original training data, used to tune the model hyperparameters and perform model selection. It is not used for parameter estimation.
- **Test set**: A separate subset of data, denoted by $\mathcal{D}_{\text{test}}$ used exclusively to evaluate the final model and to provide an unbiased estimate of its generalization performance on unseen data.

The training set is used to learn the model parameters by minimizing the empirical loss. During training, the model performance is periodically evaluated on the validation set in order to tune hyperparameters, such as the model architecture or regularization strength.

After selecting the best configuration based on the validation performance, the final model is assessed on the test set, which provides an unbiased estimate of its generalization ability on unseen data.

However, when the available dataset is limited, the performance estimate obtained from a single partition into training and validation sets may strongly depend on the

specific split adopted. In such cases, a more robust strategy for model selection is represented by **k-fold cross-validation**. Let:

$$\mathcal{D}_{\text{train}} = \left\{(\boldsymbol{x}^{(k)}, \boldsymbol{y}^{(k)})\right\}_{k=1}^{N_{\text{train}}},$$

the entire dataset, the idea beyond this strategy consist in partitioning $\mathcal{D}_{\text{train}}$ into K fold such that:

$$\mathcal{D}_{\text{train}} = \mathcal{D}_1 \cup \cdots \cup \mathcal{D}_K.$$

The training procedure is repeated K times. At the j-th iteration:

- The fold $\mathcal{D}_j$ is used as validation set
- The remaining $K-1$ folds are used to form the training set

In this way, each fold is used once as validation and the remaining $K-1$ times for training. This allows a more robust evaluation; the overall validation performance is the mean over the K folds. K-fold cross validation is used exclusively for hyperparameter tuning; furthermore, the test set, if available, must remain completely independent and is used only once to provide an unbiased estimate of the final model's generalization ability.

The effectiveness of a supervised learning algorithm depends on two main aspects:

- A small training error;
- A small gap between training and test error.

These two factors correspond to the two central challenges in machine learning: **underfitting** and **overfitting**.

Overfitting occurs when the gap between the training error and the test error is excessively large. Underfitting occurs when the model has insufficient capacity to represent the underlying relationship between inputs and outputs, resulting in a high training error.

The K-fold cross validation procedure provides a more stable and reliable estimate of the model's generalization performance compared to a single validation split, helping also to prevent overfitting.

1.2.4 Neural Networks

Neural networks, also referred to as artificial neural networks (ANNs), are computational models composed of interconnected artificial neurons that can learn from data.

Before exploring the structure of neural networks, we first introduce the artificial neuron model proposed by McCulloch and Pitts in 1943 [14], which represents a simplified abstraction of biological neurons and was developed to mimic the 'all-or-nothing' firing mechanism. In this biological analogy, a neuron only produces an output signal if the sum of its excitatory inputs reaches a specific threshold; if the stimulus is insufficient, the neuron remains inactive. This model allows to perform the boolean logic through the choice of a threshold, manually fixed.

Building upon this foundational concept, Rosenblatt introduced the perceptron in 1958 [18], extending the artificial neuron model to a model capable of learning from data; more precisely, it is the first binary classification model.

A visual representation of the perceptron is represented in Figure 1.6.

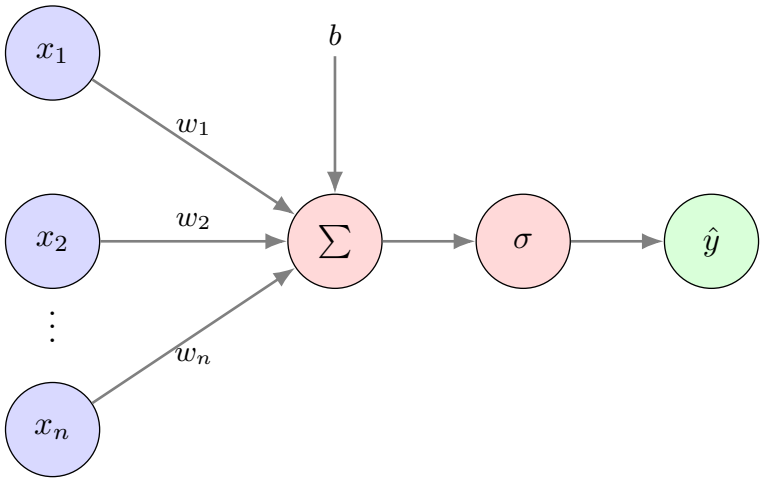


Figure 1.6: Perceptron model by Rosenblatt, 1958

The perceptron takes an arbitrary number of inputs, each associated with a weight. During the learning phase, the model first performs a forward pass, where the input data is combined with the weights and bias to generate an output. This output is then compared with the true label $y \in \{0, 1\}$, and the weights are updated accordingly when the prediction is incorrect.

During the forward pass, a linear combination of the inputs and weights is computed and a bias term is added. This value is then passed through an activation function, which introduces non-linearity into the model. The resulting value represents the output of the perceptron.

From a mathematical point of view, this process can be described as follows:

Let $\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^n$ input vector, $\boldsymbol{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} \in \mathbb{R}^n$ vector of weights, and b bias term. The forward pass to the neuron is made of the following steps.

First the following quantity, also called activation, is computed:

$$\boldsymbol{z} = \sum_{i=1}^{n} w_i x_i + b = \boldsymbol{w}^\top \boldsymbol{x} + b \in \mathbb{R}$$

Subsequently, given $\sigma : \mathbb{R} \to \mathbb{R}$ a choice of activation function, which allows to account for non-linearity in the model, the neuron output is expressed as:

$$\hat{y} = \sigma\left(\mathbf{w}^\top \boldsymbol{x} + b\right) = \sigma\left(\sum_{i=1}^{n} w_i x_i + b\right) \in \mathbb{R}$$

The choice of activation function is the Heaviside function:

$$\sigma(x) = H(x) := \begin{cases} 1 & \text{if} \quad x \geq 0 \\ 0 & \text{else} \end{cases}$$

In the perceptron case, the Heaviside function has the following form:

$$H(\boldsymbol{w}^\top \boldsymbol{x} + b) := \begin{cases} 1 & \text{if} \quad \boldsymbol{w}^\top \boldsymbol{x} \geq -b \\ 0 & \text{if} \quad \boldsymbol{w}^\top \boldsymbol{x} < -b \end{cases}$$

which is a linear classifier in $\mathbb{R}^n$ because $\boldsymbol{w}^\top \boldsymbol{x} + b = 0$ is an hyperplane in $\mathbb{R}^n$. It assigns values 0 and 1 depending on which part of the hyperplane we are on. Subsequently, during the learning phase, the predicted output $\hat{y}$ is compared with the true label $y \in \{0, 1\}$. If the prediction is incorrect, the weights and bias are updated according to the perceptron learning rule:

$$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} + \eta(y - \hat{y})\boldsymbol{x}$$

$$b^{(t+1)} \leftarrow b^{(t)} + \eta(y - \hat{y})$$

where η denotes a positive parameter called learning rate.

The perceptron update rule follows the same iterative optimization principle underlying gradient descent, as described in Algorithm **??**, since the parameters are updated in order to reduce the prediction error.

This update modifies the position of the separating hyperplane in order to reduce future misclassifications. The procedure is repeated over the training dataset until convergence, provided that the data are linearly separable.

The perceptron represents the first learning model based on error correction. The choice of the Heaviside activation function is specific to the perceptron model. In the more general case of neural networks, other activation functions are considered. A comprehensive explanation can be found at 1.2.5.

In the context of neural networks, the perceptron is the simplest computational model that could be viewed as an artificial neuron using the Heaviside step function as the activation function. The architecture representing the perceptron learning algorithm is called a single-layer perceptron.

A layer is defined as a collection of artificial neurons operating in parallel, where each neuron has its own set of weights and biases and receives the same input vector.

Starting from the single-layer perceptron, it is possible to define models obtained stacking neurons into layers. A neural network is formed by arranging layers of neurons, connected to each other through *edges*, which model the synapses in the brain. According to their role, layers are classified as input, hidden, and output layers.

A brief description of the role of each layer follows:

- **Input layer**: It is responsible for receiving the raw data and subsequently passing it to the next layer;

- **Hidden layers**: They are the set of intermediate layers between the input layer and the output layer. They are responsible for feature extraction from the data through the application of a set of weights and biases to the input data, followed by a non-linear activation function, which is discussed as a separate topic in 1.2.5.
- **Output layer**: It produces the output predictions. The number of neurons in this layer depends on the type of problem we are facing; for example, in a classification problem, we have a number of neurons equal to the number of classes. The choice of activation function depends on the type of problem; details follow in Chapter 3.

So far, we have described the function of each layer, regarding the hidden layers, several types can be employed; in this thesis, we focus in particular on dense fully connected layers and convolutional layers.

In particular convolutional layers will be introduced in section 1.3, in the description of deep neural network architectures called comvolutional neural networks (CNNs).

The transition from a single-layer perceptron to multilayer architectures is motivated by fundamental representational limitations of linear classifiers. A single-layer perceptron implements a linear decision function, meaning that it can only correctly classify datasets that are linearly separable. In other words, it defines a decision boundary in the form of a hyperplane in the input space.

However, many real-world problems are characterized by complex and highly nonlinear relationships among input variables. For such problems, no single hyperplane can correctly separate the classes. A well-known example is the XOR problem, which cannot be solved by a single-layer perceptron due to its non-linear separability.

To overcome this limitation, neurons can be arranged in multiple layers. By introducing one or more hidden layers and employing nonlinear activation functions (see 1.2.5), the network is able to construct nonlinear decision boundaries. Each hidden layer progressively transforms the input representation into a higher-level feature space, allowing the final output layer to perform classification in a space where the classes may become linearly separable.

This layered structure significantly increases the expressive power of the model. In fact, multilayer feed-forward neural networks are universal function approximators, meaning that, under mild conditions, they can approximate any continuous function on a compact domain to arbitrary precision. Therefore, stacking neurons into layers is not merely an architectural choice, but a necessary step to enable the modeling of complex nonlinear patterns.

Despite we are interested in the study of deep learning architectures, we start out with a look at shallow neural network, characterized by having few numbers fully connected hidden layer between input layer and output layer meaning that each neuron is connected to all the neurons of the input layer. As a consequence, each neuron receives the same input vector but processes it through a different set of weights and bias.

A schematic representation of the simplest shallow neural network, characterized by one hidden layer, is shown in Figure 1.7.

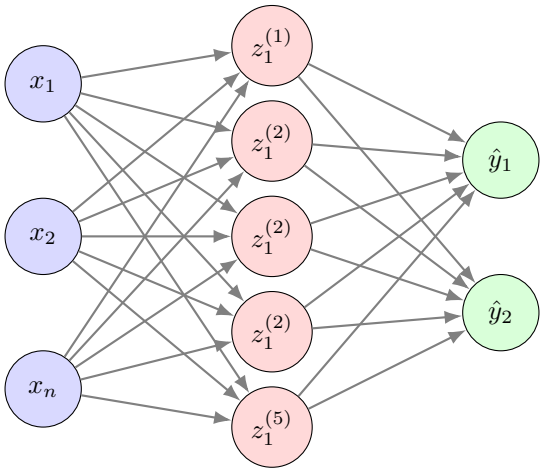


Figure 1.7: Graphical representation of a shallow neural network

The learning procedure, commonly referred to as model training, begins with a forward pass, during which the input is propagated through the layers to produce a prediction. While this step is conceptually similar to that of the perceptron, neural networks employ differentiable activation functions that enable gradient-based optimization.

Subsequently, the loss function is minimized through backpropagation, which computes the gradients of the loss with respect to the network parameters and allows their iterative update toward a minimum.

The training procedure is described in detail in subsection 1.2.6.

In particular, the learning of a shallow neural network is similar to that of the multilayer perceptron (MLP), which will be introduced in the next section. For consistency, we present the forward propagation of an artificial neural network as a special case of the forward propagation of a multilayer perceptron.

Before describing the training procedure, we first introduce the differentiable activation functions used in neural networks.

1.2.5 Activation function

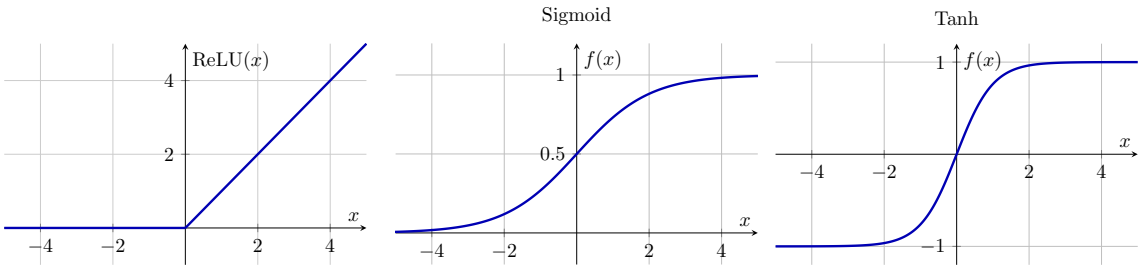
We start introducing the most important differentiable activation functions used in neural networks [20] Keeping the same notation as the previuous subsection, let $\sigma : \mathbb{R} \to \mathbb{R}$ a general activation function. Some examples are:

- **Rectified Linear Unit** (ReLU): It is the most common activation function, defined as:
$$\sigma(x) = \max\{0, x\}$$
- **Sigmoid**: It is defined as:
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$
The output values is between 0 and 1, thus it is used for binary classification problems.

- **Hyperbolic tangent**: Similar to the sigmoid, it is defined as:

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

A visual representation follows in Figure 1.8



(a) ReLu activation function (b) Sigmoid activation function (c) Tanh activation function

Figure 1.8: ReLU, sigmoid, and tanh activation functions.

These activation functions are defined as scalar functions. However, through the following abuse of notation, they can be extended to operate on a vector:

$$\sigma\left(\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}\right) = \begin{bmatrix} \sigma(x_1) \\ \sigma(x_2) \\ \vdots \\ \sigma(x_n) \end{bmatrix}$$

The last activation function we introduce is a multivariate activation function used for multi-classification problems. Let $\sigma : \mathbb{R}^n \to [0,1]^n$

- **Softmax**: It is known as normalized exponential function, its expression follows:

$$\sigma\left(\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}\right) = \frac{1}{\sum_{i=1}^{n} e^{x_i}} \begin{bmatrix} e^{x_1} \\ e^{x_2} \\ \vdots \\ e^{x_n} \end{bmatrix}$$

with n total number of classes. As shown in 1.9, this is an example for a 5 classes classification problem.

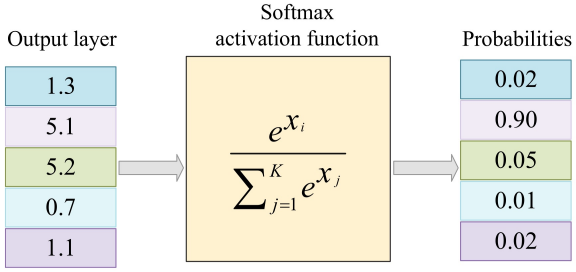


Figure 1.9: Activation function for a multiclass classification problem

The output is a probability distribution vector in which each element has values between 0 and 1, the sum of each element is up to 1.

1.2.6 Training procedure of a Neural Network

In this subsection we want to focus more on the supervised learning procedure of a neural network. The learning procedure is referred to as training procedure of a neural network.

It can be divided in three principal steps:

- **Forward propagation**;
- **Loss score computation**;
- **Backward propagation**

A schematic representation of the training procedure is presented in Figure 1.10.

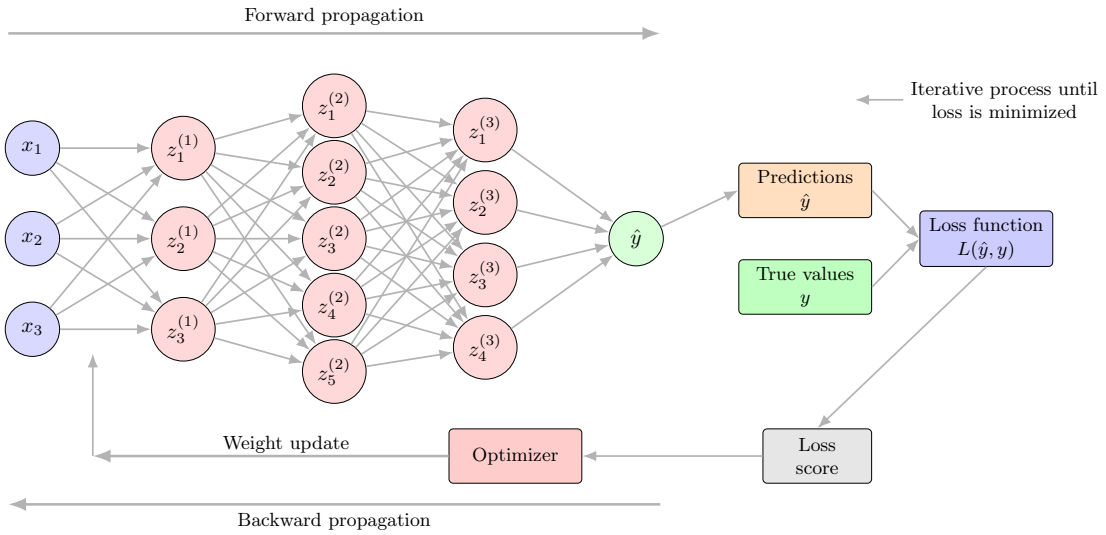


Figure 1.10: Training workflow of a neural network with multiple hidden layers

To better understand the training procedure, let's assume $\boldsymbol{x}_0 \in \mathbb{R}^n$ being the raw input data of the model, and let θ being the set of parameters of the model(weights and biases), which are initialized before the start of the training process using variance-preserving random initialization strategies in order to ensure stable signal propagation throughout the network.

Two widely adopted initialization schemes are **Xavier (Glorot)** initialization and **He (Kaiming) initialization** [11], both designed to prevent vanishing or exploding gradients in deep architectures.

Xavier initialization is typically used for networks employing symmetric activation functions such as sigmoid or tanh. In this case, the trainable weights of each layer are randomly sampled from a zero-mean distribution whose variance depends on

both the number of input connections and the number of output connections of that layer. More precisely, the variance is proportional to 2 divided by the sum of the input and output units. This choice approximately preserves the variance of activations across layers.

He initialization is specifically designed for networks using ReLU activations. Since ReLU functions set approximately half of the input values to zero, they reduce the variance of the propagated signal. To compensate for this effect, the weights are sampled from a zero-mean distribution with variance proportional to 2 divided by the number of input connections of the layer. This scaling helps maintain stable gradients during backpropagation in deep networks.

Bias terms are typically initialized to zero.

During the forward propagation, the raw input data $\boldsymbol{x}_0$ pass through each layer in order to be processed, in different ways depending on the neural network chosen. Each layer is responsible for selecting features from the input data, by applying a series of operations to the data identifying the characteristics that are most useful for solving the problem. The final output of the model is represented by $\hat{\boldsymbol{y}}$.

After the forward pass, the error between the values predicted by the model and the actual values is quantified through the loss function $\boldsymbol{\mathcal{L}}$, obtaining the loss score $J = \boldsymbol{\mathcal{L}}(\hat{\boldsymbol{y}}, \boldsymbol{y})$. The loss function is a measure of distance between the true label and the network's prediction. The objective of the training procedure is to determine the optimal parameters θ that minimize the loss function.

Backward pass consist of two steps:

- Backpropagation: computation of the gradient of the loss score J with respect to the model parameters θ, using the chain rule;
- Optimization: update of the parameters θ using a gradient-based optimization algorithm.

In order to understand the backpropagation algorithm, consider the following neural network as a computational graph formed by L hidden layers [23]:

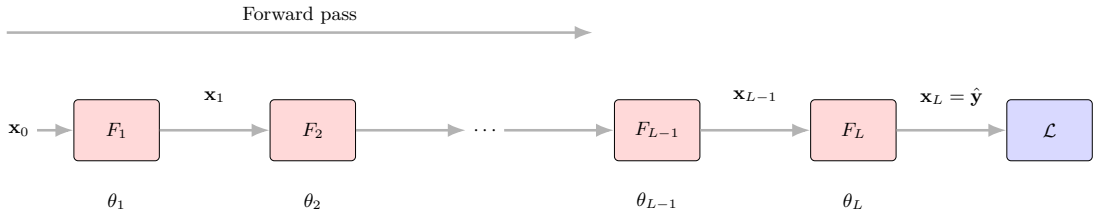


Figure 1.11: Forward pass through the entire neural network, starting with the input data $\boldsymbol{x}_0$ to be processed layer by layer producing $\boldsymbol{x}_1 \dots \boldsymbol{x}_L$

The loss take the output value $\hat{\boldsymbol{y}}$ of the last hidden layer of the network

$$\hat{\boldsymbol{y}} = F_L\left(\cdots F_2\left(F_1\left(\boldsymbol{x}_0; \theta_1\right); \theta_2\right)\cdots; \theta_L\right)$$

expressed a function of all the network's paramters We can consider the loss as a layer that takes the output value $\hat{\boldsymbol{y}}$ of the last hidden layer producing

$$J = \mathcal{L}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \mathcal{L}(F_L(\cdots F_2(F_1(\boldsymbol{x}_0; \theta_1); \theta_2) \cdots; \theta_L), \boldsymbol{y})$$

Now we need to compute the backpropagation in order to update θ to minimize the loss score J. The update of the parameters θ of the model is the optimization of the backward pass through a gradient-based learning algorithm, e.g. gradient descent

$$\theta^{(i+1)} \leftarrow \theta^{(i)} - \eta \left(\frac{\partial J}{\partial \theta} \right)$$

We need to compute $\dfrac{\partial J}{\partial \theta}$, to do so let's introduce the following vectors of partial derivatives:

- $\mathbf{g}_l = \dfrac{\partial J}{\partial \boldsymbol{x}_l}$ partial derivative of the loss score w.r.t $\boldsymbol{x}_l$ input vector of the l -th layer;
- $\mathbf{L}_l = F_l'(\boldsymbol{x}_{l-1}, \theta_l) = \dfrac{\partial \boldsymbol{x}_l}{\partial [\boldsymbol{x}_{l-1}, \theta_l]}$ jacobian of l -th layer, from which the following quantities are obtained:

$$\mathbf{L}_l^{\theta} = \frac{\partial \boldsymbol{x}_l}{\partial \theta_l} \qquad\qquad \mathbf{L}_l^{\boldsymbol{x}} = \frac{\partial \boldsymbol{x}_l}{\partial \boldsymbol{x}_{l-1}}$$

In this way, it is possible to obtain $\mathbf{g}_{l-1}$ through the chain rule as follows:

$$\mathbf{g}_{l-1} = \frac{\partial J}{\partial \boldsymbol{x}_{l-1}} = \frac{\partial J}{\partial \boldsymbol{x}_l} \frac{\partial \boldsymbol{x}_l}{\partial \boldsymbol{x}_{l-1}} = \mathbf{g}_l \mathbf{L}_l^{\boldsymbol{x}}$$

Now, the partial derivative of J with respect to θ_l parameters of the l-th layer is given by:

$$\frac{\partial J}{\partial \theta_l} = \frac{\partial J}{\partial \boldsymbol{x}_l} \frac{\partial \boldsymbol{x}_l}{\partial \theta_l} = \mathbf{g}_l \mathbf{L}_l^{\theta}$$

A visual representation of the backpropagation algorithm follows in Figure 1.12

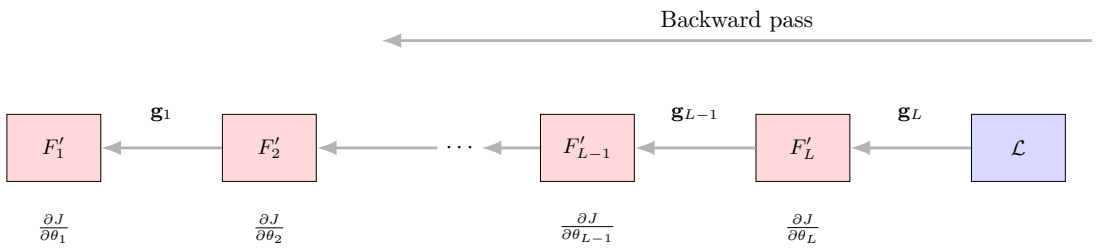


Figure 1.12: Backpropagation algorithm computing gradients $\mathbf{g}_L \dots \mathbf{g}_1$

The iteration of the full algorithm follows:

Algorithm 5 Backpropagation (for chain computation graphs).

Input: parameter vector $\theta = \{\{\theta_l\}\}_{l=1}^{L}$, layers $F_1, \ldots, F_L$, training datapoint $\{\boldsymbol{x}_0, \boldsymbol{y}\}$, loss function $\mathcal{L} : \mathbb{R}^N \to \mathbb{R}$

Output: gradient direction $\dfrac{\partial J}{\partial \theta} = \left\{ \dfrac{\partial J}{\partial \theta_l} \right\}_{l=1}^{L}$

Forward pass:

for $l = 1, \ldots, L$ **do**

    $\boldsymbol{x}_l = F_l(\boldsymbol{x}_{l-1}; \theta_l)$

end for

Backward pass:

$\mathbf{g}_L = \dfrac{\partial J}{\partial \boldsymbol{x}_L}$

for $l = L, \ldots, 1$ **do**

    $\mathbf{L}_l = F_l'(\boldsymbol{x}_{l-1}; \theta_l)$

    $\mathbf{g}_{l-1} = \mathbf{g}_l \, \mathbf{L}_l^x$

    $\dfrac{\partial J}{\partial \theta_l} = g_l \, \mathbf{L}_l^\theta$

end for

Until now, we have described the training procedure assuming that the input consists of a single vector $\boldsymbol{x}_0 \in \mathbb{R}^n$.

However, in practice neural networks are trained on the entire training dataset

$$\mathcal{D}_{\text{train}} = \left\{ \left(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)} \right) \right\}_{i=1}^{N_{\text{train}}},$$

rather than on a single input sample.

However, once we know how to compute the gradient for a single datapoint, we can easily compute the gradient for the whole dataset, due to the following identity:

$$\frac{\partial \dfrac{1}{N_{train}} \sum_{i=1}^{N_{train}} J_i(\theta)}{\partial \theta} = \frac{1}{N_{train}} \sum_{i=1}^{N_{train}} \frac{\partial J_i(\theta)}{\partial \theta}$$

where $J_i(\theta)$ is the loss score for a single $\boldsymbol{x}^{(i)} \in \mathbb{R}^n$. Therefore, to compute a gradient update for an algorithm like stochastic gradient descent, we apply backpropagation in batch mode, that is, we run it over each datapoint in our batch and then average the results.

In the following section, we present an example of the forward pass for a deep neural network architecture, restricting ourselves to the case of a single input vector for simplicity.

1.3 Deep Neural Network Architectures

In this work, we focus on deep supervised learning models. Throughout the remainder of this thesis, these models will be referred to simply as deep models. Among the

various deep learning architectures, particular attention is devoted to the Multilayer Perceptron (MLP) and the Convolutional Neural Network (CNN), which constitute the primary models investigated in this work

1.3.1 Multilayer Perceptron

A multilayer perceptron (MLP) is a feedforward deep model characterized by a number $L \in \mathbb{R}^n$ of hiddden layers, $L \geq 2$, between the input and output layers.

A graphical representation follows in Figure 1.13.

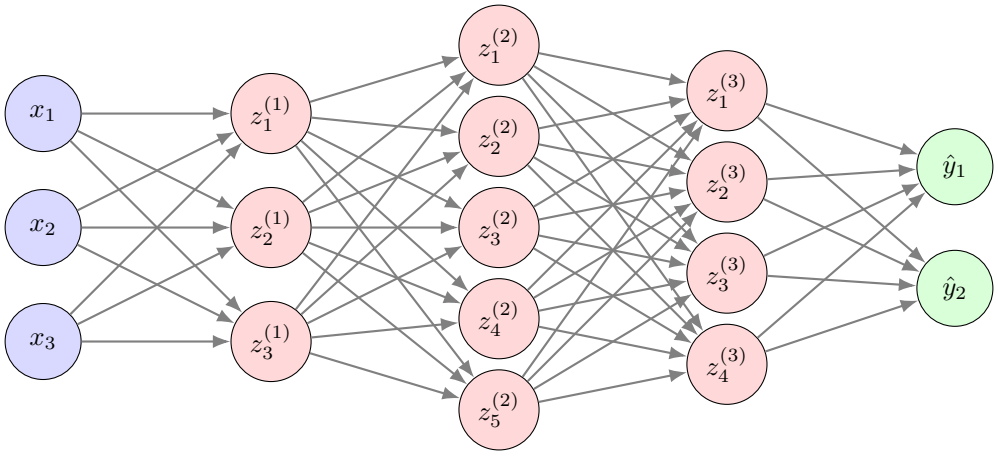


Figure 1.13: Graph representation of a Multilayer Perceptron architecture characterized by $L = 3$ hidden layers.

As we can see in the figure above, the multilayer perceptron model is fully connected, meaning that, given a certain layer, each neuron is connected to all the neurons of the following layer. Furthermore the model is characterised by having a non linear activation function between each layer, usually ReLU.

Let's introduce the first step of the training procedure for this architecture: the forward pass, during which the model computes its output based on the current parameter values.

For each layer $l = 0, \ldots, L+1$, let n_l be the number of neurons in the l-th hidden layer.

Let $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n_0} \end{bmatrix} \in \mathbb{R}^{n_0}$ denote the input vector and define:

$$a_i^{(0)} = x_i \quad i = 1, \ldots, n_0$$

For each layer $l = 1, \ldots, L$, let n_l be the number of neurons in the l-th hidden layer.

Each neuron i in layer l is associated with a weight vector $\mathbf{w}_i^{(l)} = \begin{bmatrix} w_{i,1}^{(l)} \\ w_{i,2}^{(l)} \\ \vdots \\ w_{i,n_{l-1}}^{(l)} \end{bmatrix} \in \mathbb{R}^{n_{l-1}}$ and a bias term $b_i^{(l)} \in \mathbb{R}$.

The pre-activation value of neuron i in layer l is defined as

$$z_i^{(l)} = \sum_{j=1}^{n_{l-1}} w_{i,j}^{(l)} \, a_j^{(l-1)} + b_i^{(l)}.$$

That is, $z_i^{(l)}$ is obtained as the weighted sum of the activations from the previous layer plus the corresponding bias term.

Given an activation function $\sigma : \mathbb{R} \to \mathbb{R}$, the activation of neuron i in layer l is defined as

$$a_i^{(l)} = \sigma\big(z_i^{(l)}\big).$$

In vector form, the forward propagation step for layer l can be written as

$$\mathbf{z}^{(l)} = W^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)},$$

$$\mathbf{a}^{(l)} = \sigma\big(\mathbf{z}^{(l)}\big),$$

where $W^{(l)} \in \mathbb{R}^{n_l \times n_{l-1}}$ is the weight matrix and $\mathbf{b}^{(l)} \in \mathbb{R}^{n_l}$ is the bias vector.

Finally, the output of the network is given by

$$\hat{\mathbf{y}} = \mathbf{a}^{(L)}.$$

equivalent to the activation of all the neurons in the last hidden layer.

Subsequently, the training procedure is the same as introduced in subsection 1.2.6; it involves calculating the prediction error through a loss function and utilizing the error to adjust the weights and the biases iteratively until the prediction is minimized.

A multilayer perceptron (MLP) requires the input data to be provided as a one-dimensional vector. For this reason, it is generally not well suited for structured data such as images, text, or speech.

In the case of images, the input must be flattened into a vector, which removes the spatial structure of the data. As a result, the model struggles to preserve visual features and, in particular, the relationships between neighboring pixels. These local relationships are essential for representing edges, textures, and shapes, and losing them makes learning visual patterns less efficient.

It is not the right deep model to use for a task like semantic segmentation, because of this we introduced the second deep architecture.

1.3.2 Convolutional Neural Networks

Convolutional neural networks (CNNs) [16],[29] are a type of deep neural networks used in the field of pattern recognition tasks, such as image segmentation. Their

effectiveness in image-based applications start from the structure of the input layer, which preserves the spatial dimensions (height and width) and depth, equivalent to the number of channels, of the image. Beyond the input layer, CNNs are typically composed of three main types of layers:

- **Convolutional layer**: It is the fundamental building block of CNNs, composed of a collection of convolution kernel followed by a non linear activation function, usually ReLU. The output of this operation is a feature map, also called activation map, that captures visual features from the given input. Regarding the first convolutional layer, the input corresponds to the image;
- **Pooling layer**: This layer performs downsampling along the spatial dimensionality of feature map, reducing the size and the number of parameters, while retaining important information;
- **Fully-connected layers**: They take the flattened feature maps as input and produce class scores for classification.

A visual representation of a convolutional neural network follows in Figure 1.14.

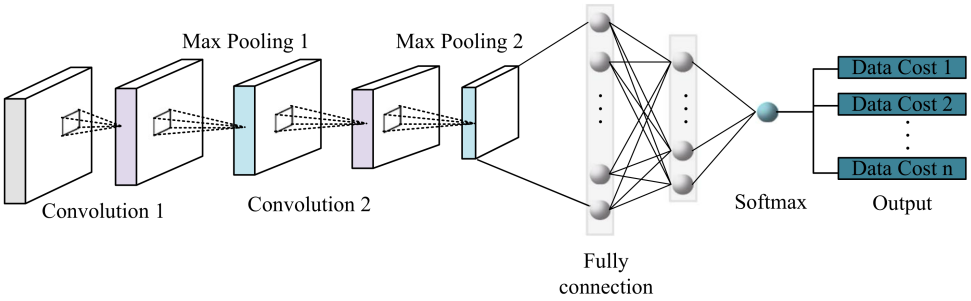


Figure 1.14: Overview of a convolutional neural network

After providing an overview of the structure, we turn to a more detailed analysis on how data inside a convolutional neural network is processed.

Starting from the convolutional layer, as introduced above, it is specifically designed to extract hierarchical visual features from the input image. It consists of a set of learnable convolutional kernels (or filters), each of which is convolved with the input to produce a corresponding feature map, followed by the application of a non-linear activation function.

The number of feature maps produced by a convolutional layer defines the number of output channels of that layer. Each convolutional kernel generates one feature map; therefore, the number of kernels determines the dimensionality of the output along the channel axis.

The depth of each convolutional kernel is equal to the number of channels of the input it operates on. For the first convolutional layer, this depth corresponds to the number of channels of the input image (e.g., 3 for an RGB image and 1 for a grayscale image). In deeper layers, the kernel depth matches the number of feature maps produced by the previous layer.

The number of channels directly influences the computational complexity of the CNN, as well as its representational capacity. Increasing the number of filters generally allows the network to learn a richer set of features, although at the cost of higher computational and memory requirements. Each filter learns to detect specific visual patterns, such as edges, textures, or more complex structures, depending on the depth of the network.

To make things simpler we restrict ourselves to square filters of small dimensions, e.g. $3 \times 3, 5 \times 5$, with depth equal to the number of channel of the input image. Given an image $I \in \mathbb{R}^{h \times w}$ and a kernel $K \in \mathbb{R}^{n \times n}$, the convolution at pixel (i, j) is defined as:

$$\big(K * I\big)_{(i,j)} = \sum_{k=-n}^{n} \sum_{l=-n}^{n} I_{(i-k)(j-l)} K_{(i,j)}$$

where n is the dimension of the filter. The convolution operation involves multiplying the kernel values by the original pixel values of the image and then summing up the results. A visual representation is reported in Figure 1.15.

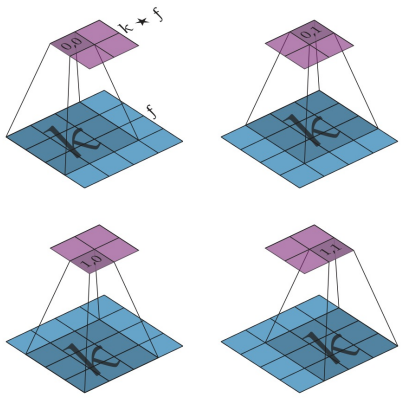


Figure 1.15: Convolution operation given a kernel $K \in \mathbb{R}^{3 \times 3}$, stride 1.

Kernels could move through the pixels of an image with steps of different size. This brings the introduction of the concept of **stride**.

Stride is the number of pixels by which a kernel moves across the input image. The output data size strongly depends on the stride dimension: while a larger stride decreases the computational cost and increases processing speed, it also reduces the spatial resolution of the feature map, potentially limiting the ability of the convolutional layer to capture high-level features. Conversely, a stride of one preserves more spatial information but requires higher computational effort.

Another important concept related to convolutional kernels is **padding**.

During the convolution operation, pixels located at the borders of the image are involved in fewer kernel applications compared to central pixels. As a consequence, spatial information near the edges may be badly represented.

To mitigate this issue, additional pixels are artificially added around the boundaries of the input image before applying the convolution. This operation is referred to as padding.

The main purpose of padding is to control the spatial size of the output feature map and to preserve information at the image borders. Depending on the chosen padding size, it is possible to maintain the same spatial dimensions as the input (commonly referred to as same padding) or to allow a reduction in size (valid padding, when no padding is applied).

Padding is typically implemented by adding zeros around the image (zero-padding), although other strategies such as reflection or replication padding can also be adopted. A visual representation is reported in Figure 1.16.

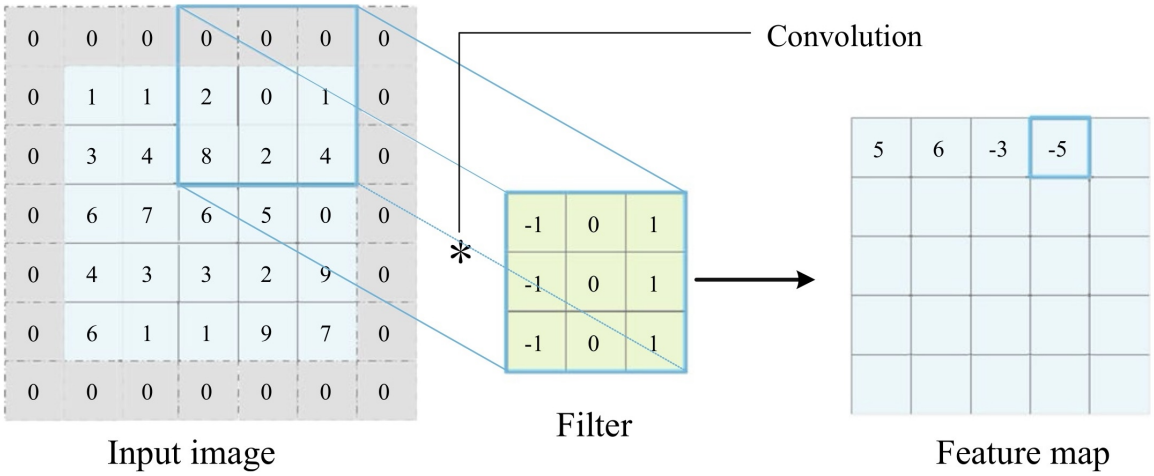


Figure 1.16: Example of a zero-padding operation, application of a kernel $K \in \mathbb{R}^{3\times3}$ with stride one.

Subsequently, as shown in Figure 1.17, a non-linear activation function, typically ReLU, is applied to the output of the convolutional layer, producing a set of feature maps, each representing a distinct type of extracted image feature.

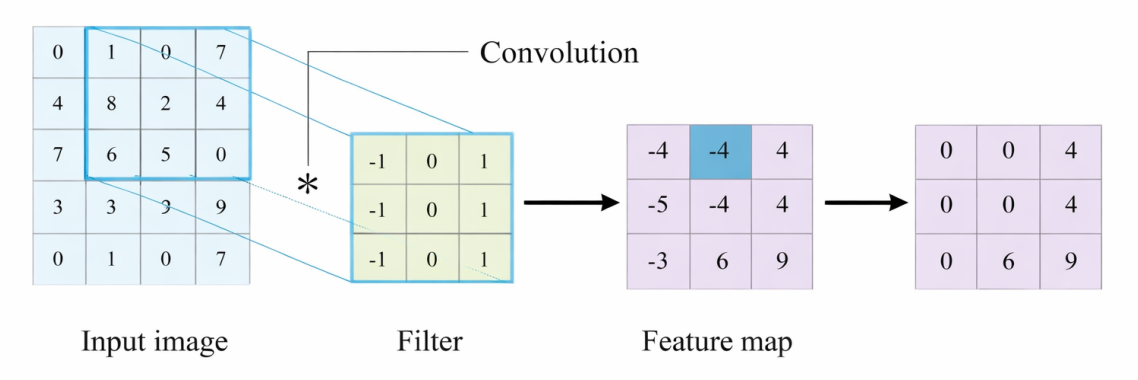


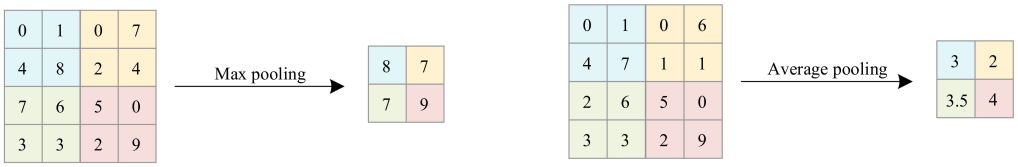
Figure 1.17: Example of a single feature map extraction from the convolutional layer, application of a kernel $K \in \mathbb{R}^{3\times3}$ with stride one, followed by ReLu activation function

After the acquisition of the feature maps, a pooling layer is added in order to reduce the dimensionality of feature maps, keeping only the important features. This reduces also the number of learnable features for the model, preventing overfitting. The pooling procedure inside this layer is similar to the convolution procedure, a small squared kernel of fixed dimension is slid over the feature map, in the same way that a convolutional kernel slide over the data. However, differently from convolution, no learnable weights are involved. In order to do that, a pooling function $f \in \mathbb{R}^{k\times k}$ acts on a local region of dimension $k \times k$ moving across the feature maps according to the stride of the pooling function.

The most common pooling functions are:

- **max pooling**: It selects the maximum value within each local region as its representative value. This operation maintain the most prominent activation in the neighborhood, emphasizing the most salient features of the feature map.
- **average pooling**: It computes the arithmetic mean of all the values within the local region, producing a smoother representation that reflects the average response of the feature map in that area.

A visual representation follows in Figure 1.18.



(a) Max pooling operation (b) Average pooling operation

Figure 1.18: Illustration of pooling operation on a single feature map, $f \in \mathbb{R}^{2\times2}$, stride 1

.

Applying the combination convolutional and pooling stages, the most relevant visual features have been extracted. The high-dimensional input image is transformed into a lower-dimensional set of feature maps, which are then flattened into a one-dimensional vector and passed to the fully connected layers. These layers perform the final classification through the application of an activation function, typically Softmax.

CNN-based architectures form the foundation of image segmentation tasks. For these tasks, the two main CNN architectures considered are U-Net and PSPNet.

1.3.3 U-Net

U-Net architecture [17], named after its characteristic U-shape form, as shown in Figure 1.19, was originally developed for biomedical image segmentation tasks.

The architecture consist of two main components, the contracting path (on the left) and the expansive path (on the right):

- **Contracting path**: It consists of repeated convolutional layers, each followed by a ReLU activation function and a max pooling operation for downsampling. The aim is to decrease the spatial dimension of the image increasing the number of channels in order to better capture visual patterns of the image.
- **Expansive path**: It consist of upsampling the feature map produced by the contracting path in order to produce the final segmentation map, where each pixel is classified individually. The upsampling process is made through the application of convolution and up-convolution, commonly known as a transposed convolution. It is a technique used in neural networks to increase the spatial dimensions of feature maps. It can be thought as the reverse process of max pooling.

The direct connection between the contracting path and the expansive path is achieved through skip connections, represented by the grey arrows. These connections allow high-resolution feature maps from each level of the contracting path to be directly transferred to the corresponding level of the expanding path, bypassing the bottleneck layers and mitigating the loss of spatial information caused by downsampling. In the original U-Net architecture, this operation is referred to as copy and crop, where the feature maps from the contracting path are cropped to match the spatial dimensions of the upsampled feature maps before concatenation.

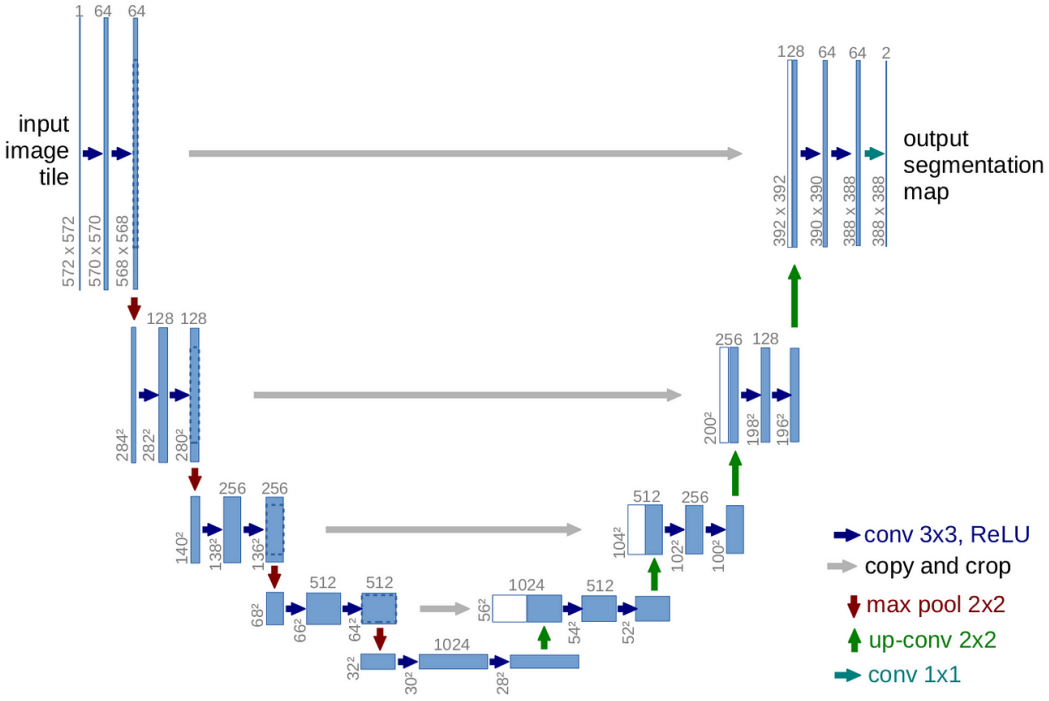


Figure 1.19: Overview of a UNet architecture

1.3.4 PSPNet

PSPNet, acronym for pyramid scene parsing network, is a deep model developed by Heng Shuang Zhao et al.[28] in 2017 for multiclass image segmentation of images.

The Principal components of PSPNet are the following:

- **CNN backbone**: It receives the input image,and through a sequence of convolutional and pooling layers, extracts hierarchical feature maps that encode both low-level and high-level visual information;
- **Pyramid Pooling Module**: It takes as input the last feature map produced by the backbone network and incorporates global contextual information through pooling operations.
- **Final classification layer**: It is responsible for the final pixel-wise classification predictions

A visual representation of the PSPNet follows in Figure 1.20.

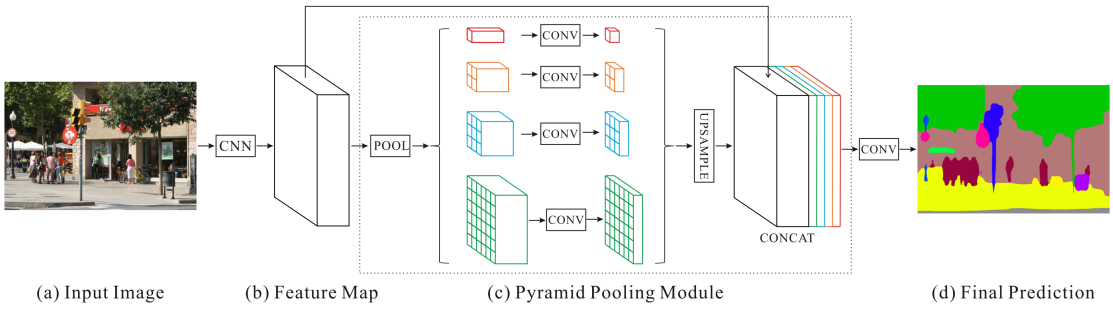


Figure 1.20: Overview of a Pyramid scene parsing netowrk for an image segmenatation task

The main innovation of this architecture is the Pyramid Pooling Module (PPM). It is designed to capture global contextual information starting from the last feature map produced by the backbone network, which contains the highest-level semantic features while having the lowest spatial resolution.

Within the Pyramid Pooling Module, four pyramid levels of different spatial sizes are generated through pooling operations. Each level produces a feature map with a different spatial dimension. A 1×1 convolutional layer is then applied to each pooled representation in order to reduce the dimensionality of the feature maps.

Subsequently, each level is upsampled to match the spatial resolution of the original backbone feature map. The upsampled features are then concatenated, forming a global contextual representation. This global representation is finally concatenated with the original high-level feature map from the backbone and passed to the final classification layer.

The motivation for introducing this second architecture, in addition to the standard U-Net, will be discussed in the following chapters.

1.4 Metrics evaluation

Up to now, we have described the aim of this thesis through the presentation of the problem, hierarchical semantic segmentation of ultrasound images, and the tools we are going to use in order to deal with it, convolutional neural networks models as U-Net and PSPNet.

In this section, we introduce the principal metrics to evaluate neural networks' performances for ultrasound image segmentation. Each metric refers to a particular aspect of the segmentation mask produced by the model itself, in particular these metrics help developers understanding weakness and strength in their models, guiding improvements and optimizations.

The metrics presented in this work are based on the fundamental principles of a machine learning task known as binary classification problem, which involves categorizing each sample in a dataset into one of two classes: Positive (P) or Negative(N). For a given dataset with its known classification, referred to as the ground-truth, there are four possible outcomes when comparing the predicted classifications to the

ground-truth labels:

- **True Positive (TP)**: The data is correctly classified as positive;
- **True Negative (TN)**: The data is correctly classified as negative;
- **False Negative (FN)**: The data is incorrectly classified as negative;
- **False Positive (FP)**: The data is incorrectly classified as positive.

The four outcomes can be formulated in a 2×2 table called confusion matrix, as follows:

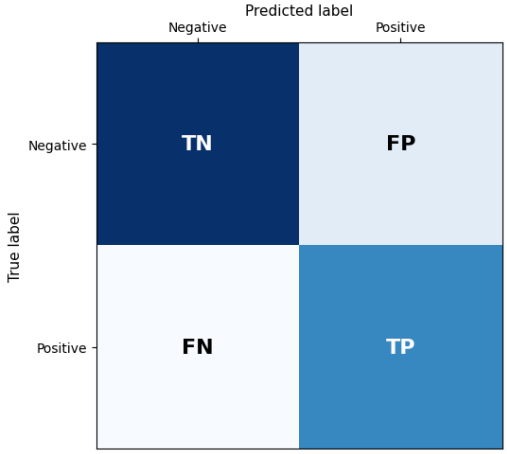


Figure 1.21: Example of a confusion matrix for a binary classification problem

In this confusion matrix all the correct predictions are located in the diagonal of the table, while prediction errors are located in the off-diagonal. Summing up the row-elements, it is possible to obtain the total number of positive and negative case of the original dataset, i.e. $\mathrm{P} = \mathrm{FN} + \mathrm{TP}$ and $\mathrm{N} = \mathrm{TN} + \mathrm{FP}$.

In the multiclass segmentation setting, the concept of the confusion matrix can be naturally extended from the binary case. Let N denote the total number of classes, and let $c \in \{1, \ldots, N\}$ represent the index of each class. Instead of a 2×2 table, the confusion matrix becomes an $N \times N$ matrix, where each row corresponds to the ground-truth class, while each column corresponds to the class predicted by the model.

An element $C_{i,j}$ of the confusion matrix represents the number of pixels whose true label belongs to class i and that are predicted as class j. Correct predictions are located along the main diagonal of the matrix, where the predicted class matches the ground-truth class, whereas misclassifications are represented by the off-diagonal elements.

For each class c, the multiclass problem can be reduced to a one vs all binary classification problem, allowing the definition of class-specific true positives $(TP)_c$, false positive $(FP)_c$, true negative $(TN)_c$, and false negative $(FN)_c$. In particular, $(TP)_c$ corresponds to pixels correctly predicted as class c, $(FP)_c$ to pixels incorrectly assigned to class c, $(FN)_c$ to pixels truly belongs to class c but predicted as different class, $(TN)_c$ to all remaining correctly classified pixels.

This formulation allows computation of these principal evaluation metrics [2]:

- **Intersection over Union (IoU)**: It is an area-based metric that measures the area between the predicted segmentation map and the ground truth overlap

$$\text{IoU}_c = \frac{TP_c}{TP_c + FN_c + FP_c} \quad \forall c \in \{1 \dots N\}$$

IoU assumes values between 0 and 1, where 0 indicates no overlap between the segmentation map and the ground truth while 1 indicates the perfect overlap between the two masks.

- **Precision**: It measures the proportion of true positive pixels among all pixel classified positive by the segmentation mask:

$$\text{Precision}_c = \frac{TP_c}{TP_c + FP_c} \quad \forall c \in \{1 \dots N\}$$

- **Recall**: It measures the proportion of true positive pixels among all true pixels in the ground truth:

$$\text{Recall}_c = \frac{TP_c}{TP_c + FN_c} \quad \forall c \in \{1 \dots N\}$$

A high precision value indicates a low number of false positive pixel predictions produced by the model, while a high recall value indicates a low number of false negative pixel predictions.

- **F_1 - score**: It combines precision and recall into a single number, more precisely it is the harmonic mean of precision and recall:

$$\text{F}_1 - \text{score}_c = 2 \times \frac{\text{Recall}_c \cdot \text{Precision}_c}{\text{Recall}_c + \text{Precision}_c} \quad \forall c \in \{1 \dots N\}$$

F_1 - score assumes values between 0 and 1, where 1 represents perfect precision and recall, while 0 represents worst performances.

Until now, the evaluation metrics presented are defined at the class level. A global measure of the model's performance is given by:

$$\begin{aligned} \text{Mean-IoU} &= \frac{1}{N}\sum_{c=1}^{N} \text{IoU}_c \\ &= \frac{1}{N}\sum_{c=1}^{N} \frac{TP_c}{TP_c + FN_c + FP_c} \end{aligned} \qquad \begin{aligned} \text{Mean-F}_1 &= \frac{1}{N}\sum_{c=1}^{N} \big(\text{F}_1\text{-score}\big)_c \\ &= \frac{1}{N}\sum_{c=1}^{N} 2 \times \frac{\text{Recall}_c \cdot \text{Precision}_c}{\text{Recall}_c + \text{Precision}_c} \end{aligned}$$

From the multiclass confusion matrix, two distinct normalized representations are derived to highlight the model's performance.

The **true confusion matrix** is obtained by applying a row-wise normalization to the confusion matrix, this means:

$$C_{i,j}^{true} = \frac{C_{i,j}}{\sum_{j=1}^{N} C_{i,j}}$$

Each element $C_{i,j}^{true}$ represents the percentage of pixels predicted as class j that actually belong to the class i. This matrix can be interpreted as a representation of the model's recall; the diagonal elements correspond to the per-class recall, while the off-diagonal elements describe how pixels of a given class are misclassified into other classes.

Conversely, the **prediction confusion matrix** is obtained through a column-wise normalization:

$$C_{i,j}^{pred} = \frac{C_{i,j}}{\sum_{i=1}^{N} C_{i,j}}$$

In this case, each element $C_{i,j}^{pred}$ represents the percentage of pixels predicted as class j that actually belong to class i. This matrix can be interpreted as a representation of the model's precision; the diagonal elements correspond to the per-class precision, while the off-diagonal elements indicate the contribution of other classes to false positive predictions.

Chapter 2

Dataset Analysis

The dataset used for the segmentation task was developed through a collaboration between Syndiag and its partner hospitals, including the Mauriziano Umberto I Hospital in Turin. It is organized hierarchically at the level of clinical cases. Ideally, each clinical case would include the complete transvaginal ultrasound diagnostic video. In practice, however, each clinical case consists of multiple items, where each item corresponds to a folder containing a specific video sequence extracted from the overall examination.

From each item, a subsequence of frames is created, resulting as a collection of ultrasound (US) images directly collected from each video. A visual representation of the dataset organization follows in Figure 2.1.

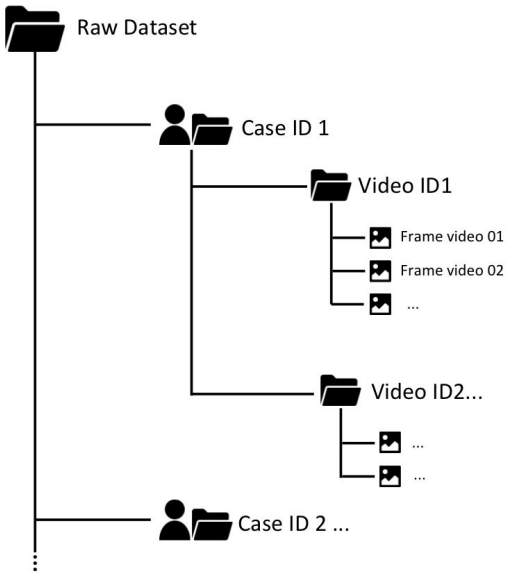
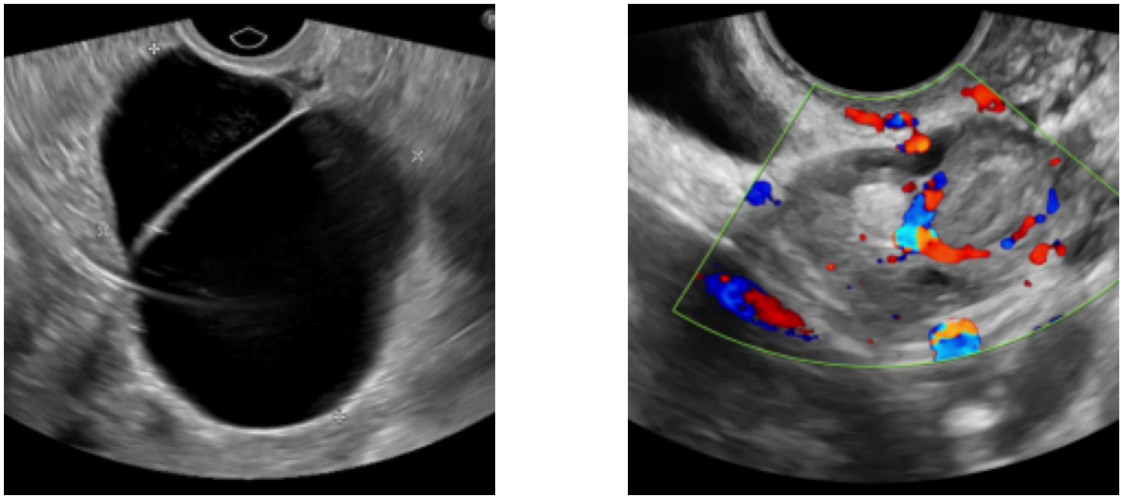


Figure 2.1: General structure for the raw data

In particular, the ultrasound (US) data are acquired using two different imaging modalities. Therefore, the video sequences contained in each item may correspond to either of the following representations:

- **B-mode:** A grayscale (black-and-white) ultrasound representation that visualizes the anatomical structures of the pelvic organs;
- **Doppler-mode:** An imaging modality that highlights blood flow within tissues, supporting the identification and characterization of solid and non-solid masses in pelvic organs.

A visual representation follows in Figure 2.2



(a) Ultrasound image acquired in B-mode.

(b) Ultrasound image acquired using the Doppler effect.

Figure 2.2: Examples of different ultrasound image representations in the dataset.

So far, we have described the overall organization of the dataset and the two different data acquisition modalities.

For completeness, we conclude by stating that the total number of clinical cases is equal to 1223.

2.1 Data Preparation for Segmentation

In this section, we describe the generation of the effective data used in the segmentation pipeline. For clarity, the segmentation task is performed at frames level, i.e., on the raw ultrasound images extracted from the video sequences. The details follows:

Train-validation-test splitting:

The entire dataset is first divided at the *clinical-case level* into two disjoint subsets: 75% of the clinical cases are allocated to the cross-validation procedure (training and validation), while the remaining 25% are reserved as an independent test set.

Within the 75% portion, a five-fold cross-validation strategy is adopted. At each iteration, four folds are used for training and one for validation. This

strategy ensures that all frames belonging to a given clinical case are assigned exclusively to either training, validation, or testing, thereby preventing data leakage.

The distribution of clinical cases, for both training and validation across the five folds, is reported in Figure 2.3.

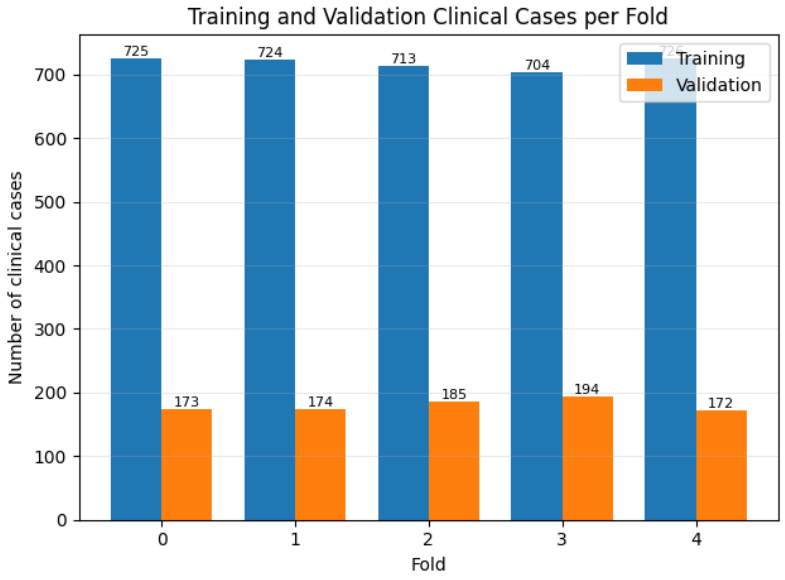


Figure 2.3: Distribution of the clinical cases among the 5 folds, for both training and validation.

Items selection

After performing the clinical-case split, all items associated with each clinical case are selected. The number of available items for a case depends on how many items were provided by the physicians. On average, across the five folds used for the cross-validation procedure, approximately 2.5 items are associated with each clinical case. This indicates that some clinical cases include a single item, whereas others contain up to three items. The distribution is reported in Figure 2.4.

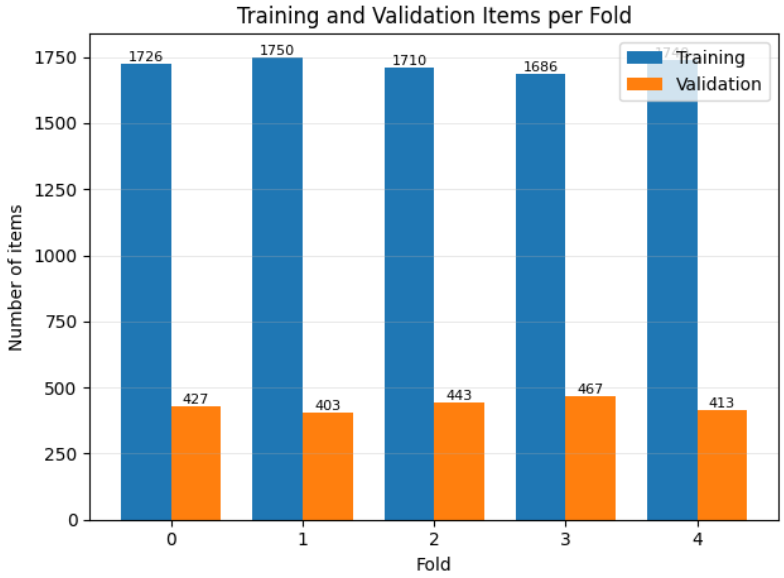


Figure 2.4: Distribution of training and validation items across the five folds.

Frames selection

Frame selection is performed at the item level. Each item is characterized by a different number of frames, depending on the video duration. This may lead to an imbalance in the dataset. To avoid causing an imbalance in the dataset, a random selection of a maximum number of frames from each item was adopted. This technique is referred to as *offline max numerosity*. In particular, for the training and validation datasets, the maximum numerosity was set to two.

For each frame selected a correspondent ground truth annotation is available, in the seemantic segmentation setting is called segmentation map. The distribution of annotated training and validation frames across the five folds is illustrated in Figure 2.5.

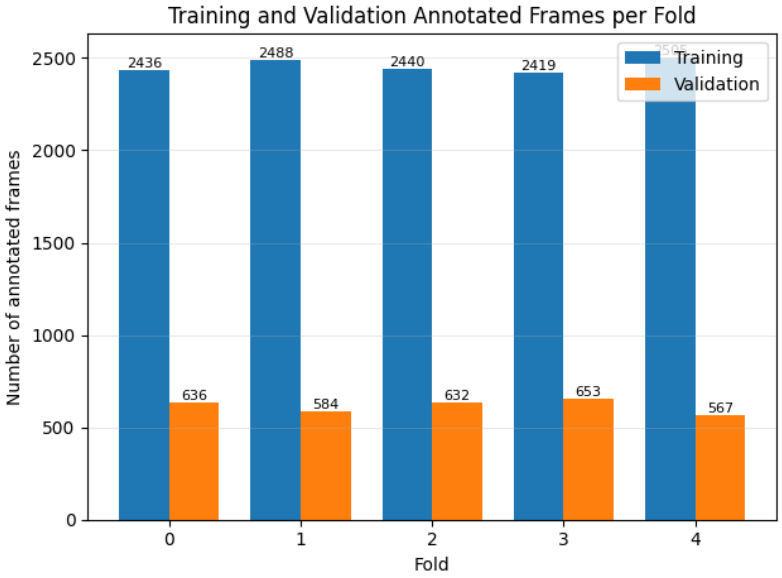


Figure 2.5: Distribution of annotated training and validation frames across the five folds

Regarding the test set, due to its smaller size compared to the dataset used for cross-validation, all available frames are used for each item..

Up to this point, we have described the effective data used for the segmentation task.

Each segmentation mask assigns a semantic label to every pixel of the image, according to the set of classes defined in Chapter 3. The pixel's distribution across the five training folds is represented in Figure 2.6.

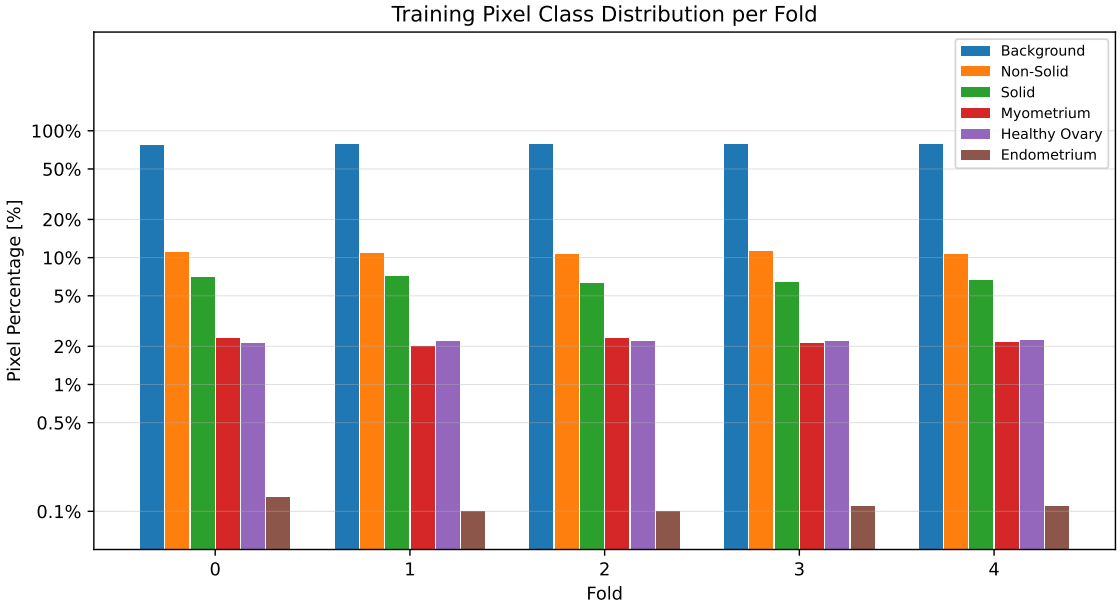


Figure 2.6: Pixel distribution across the classes for each of the five training folds

As shown in Figure 2.6, the class *endometrium* is significantly underrepresented compared to the other classes. To mitigate this class imbalance, an additional weighting term will be introduced in the loss function during model training.

In the following sections, we describe the main procedures that can be applied to the data in order to improve the model's ability to generalize.

2.2 Data Preprocessing

Data preprocessing is a fundamental step in preparing data before the main stage of model training and evaluation. Raw data are often characterized by low quality, noise, or inconsistencies, which may negatively affect the performance of deep learning models. To mitigate these issues, a series of preprocessing techniques can be applied to the dataset in order to improve data quality and ensure that the input data are suitable for analysis. This process, commonly referred to as data preprocessing, aims to enhance the reliability of the dataset and facilitate the learning process. Proper preprocessing can significantly improve the generalization capability of the model, leading to more accurate and robust results.

Regarding our case study, the main preprocessing operations applied to the dataset are the following:

- Data noise removal;
- Grayscale conversion;
- Cropping, flipping, and enhancement of pixel-wise image contrast.

A visual comparison between the original images and their preprocessed counterparts is shown in Figure 2.7.

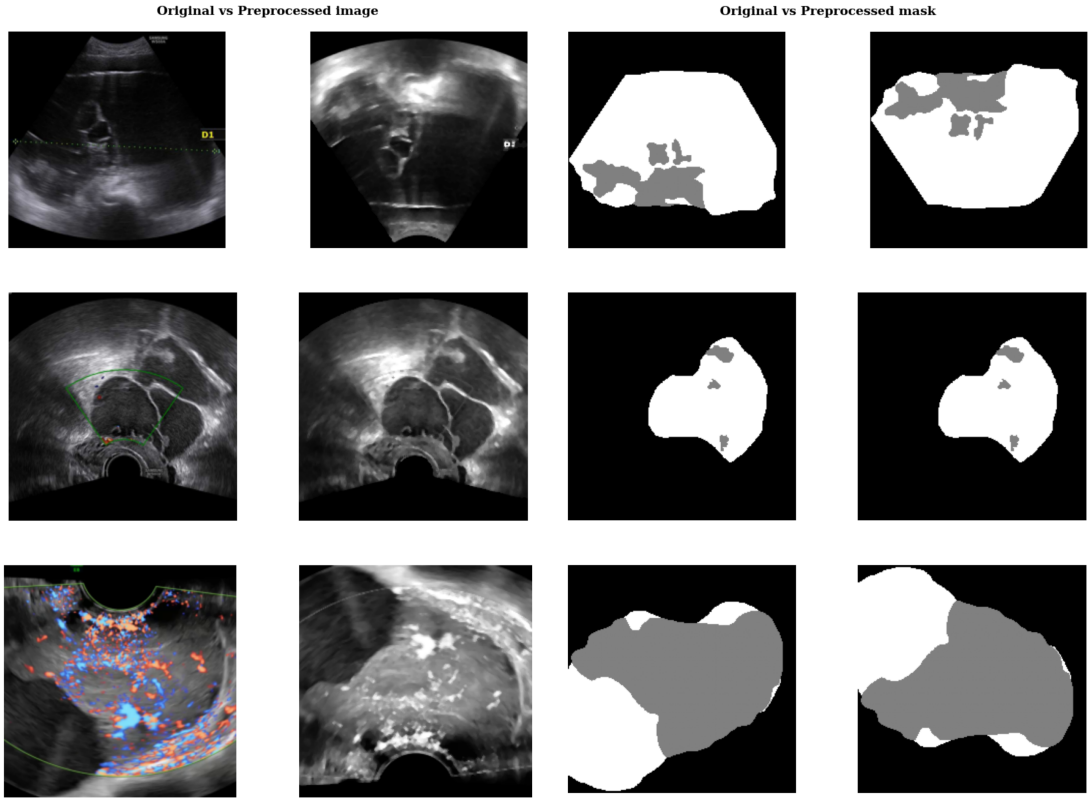


Figure 2.7: Visual comparison between original and preprocessed images and their corresponding masks.

As we can see, the preprocessing step removes noise, suppresses Doppler color through grayscale conversion, optionally flips the images, and enhances contrast.

Now, let's analyze the pixels intensity distribution of the original and preprocessed ultrasound images. As shown in Figure 2.8, the preprocessed images exhibit a stronger concentration of pixels at low intensity values. At the same time, the contrast enhancement step increases the separation between background and relevant anatomical structures, resulting in brighter pixel values for tissue regions.

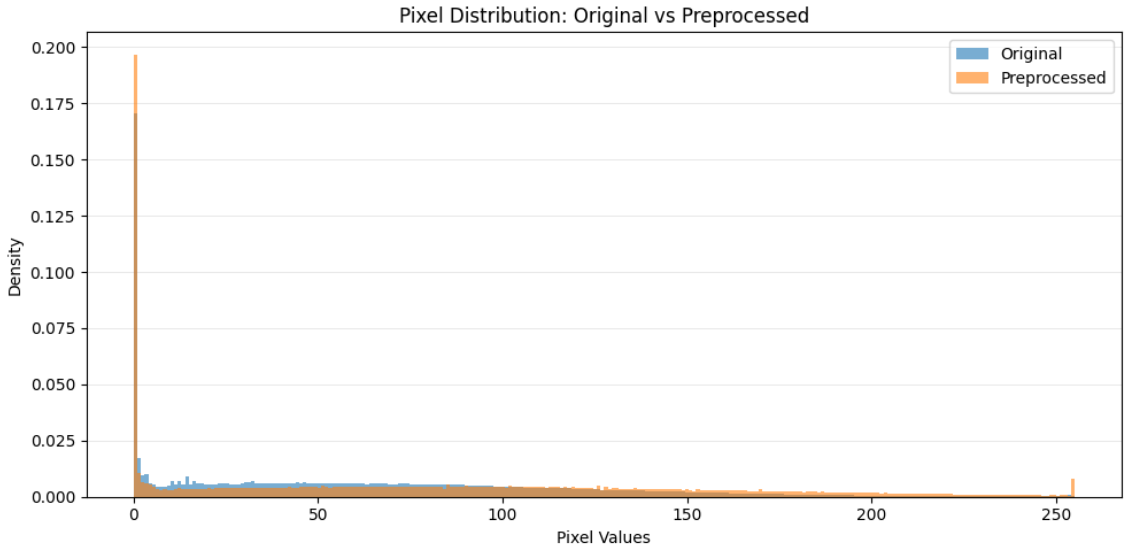


Figure 2.8: Pixels distribution of original and preprocessed ultrasound images of training dataset.

This behaviour is reflected in the representation of variability for each training fold. As shown in Figure 2.9, the preprocessed images show a higher pixel intensities compared to the original images. This increase in variability indicates a wider spread of intensity values.

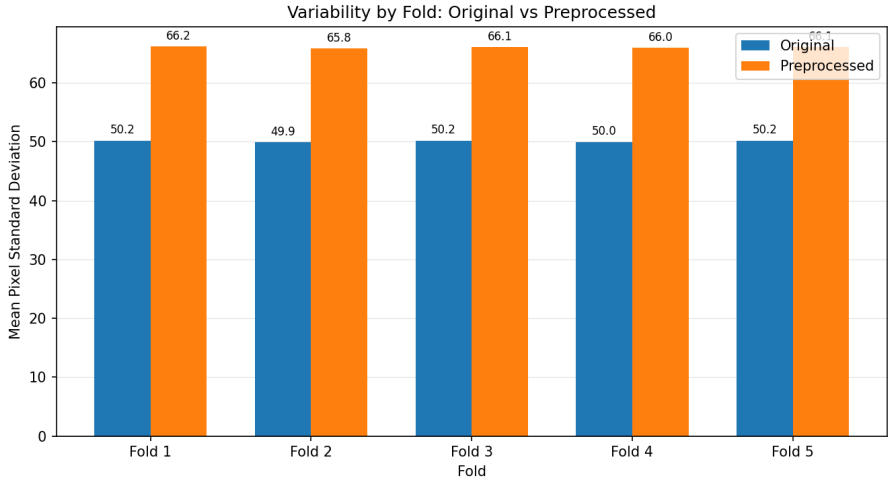


Figure 2.9: Distribution of standard deviation across the training folds for the original and preprocessed datasets.

To conclude, we present the cumulative distribution function of pixel intensities, which describes the probability that the intensity is less than a certain value, and which exhibits the enhancing effect of the preprocessing pipeline.

As shown in Figure 2.10, the orange line, representing the cumulative distribution function of the preprocessed data, lies below that of the original data, indicating that the preprocessed images exhibit higher pixel intensities overall, as a greater proportion of pixels is shifted toward larger intensity values.

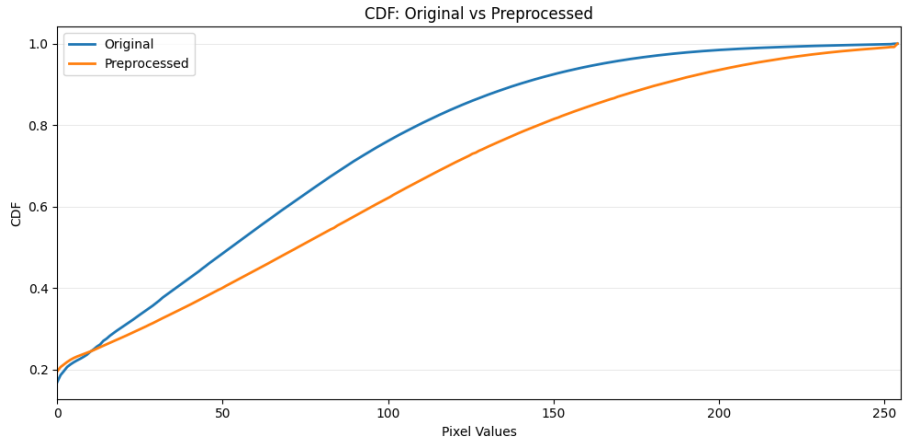


Figure 2.10: Cumulative distribution function of pixel intensities for the original and pre-processed training datasets.

2.3 Data Augmentation

Data augmentation is a widely used technique in deep learning that generates high-quality synthetic samples from the original training data. By artificially increasing the variability of the dataset, it improves the model's generalization ability and helps reduce overfitting.

Before introducing the specific strategy adopted in this work, we distinguish between two main types of data augmentation [5]. The first is offline data augmentation, where augmented samples are generated prior to the training phase and stored together with the original dataset. This approach requires additional storage space and limits augmentation to a fixed set of pre-generated samples.

The second approach is online data augmentation, where data transformations are applied dynamically during the training process. In this case, augmented samples are generated on the fly and are not permanently stored, making the method more memory-efficient. Furthermore, different augmented samples can be generated at each epoch, increasing the variability of the data observed during training.

In this work, we adopt an online data augmentation strategy. A visual representation of the workflow is shown in Figure 2.11.

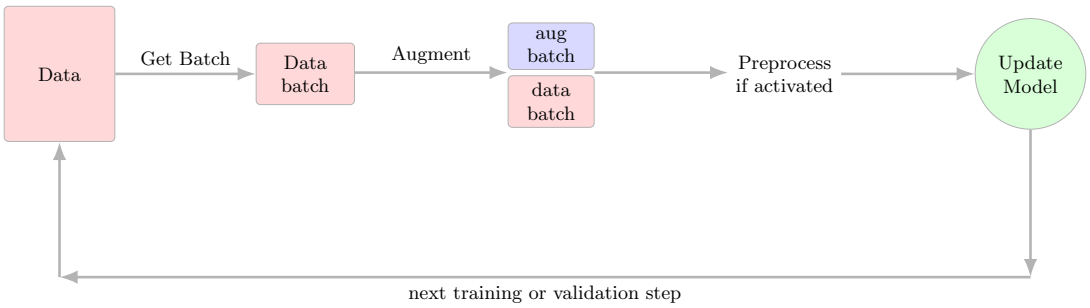


Figure 2.11: Training procedure with online data augmentation.

We introduce data augmentation by considering a labeled dataset given by

$$\mathcal{D} = \left\{ \left(\boldsymbol{x}^{(k)}, \boldsymbol{y}^{(k)} \right) \right\}_{k=1}^{N}$$

where $\boldsymbol{x}^{(k)} \in \mathbb{R}^{H \times W \times C_{\text{image}}}$ is an image of dimension $H \times W$ with C_{image} number of channels. Each channel is a matrix of size $H \times W$ storing the intensity values of a specific colour component. Black-and-white images have a single channel, while colour images typically have three channels corresponding to the red, green, and blue components. In this work, $\boldsymbol{x}^{(k)}$ corresponds to ultrasound images.

The variable $\boldsymbol{y}^{(k)}$ denotes the ground truth annotation associated with the input image. Its precise structure depends on the specific learning task; in the segmentation setting addressed in this work, $\boldsymbol{y}^{(k)}$ correspond to the segmentation mask of the image.

Data augmentation can be formally represented by the following function

$$f_\theta : \mathcal{D} \to \tilde{\mathcal{D}}$$

where $\tilde{\mathcal{D}} = \left\{ \left(\tilde{\boldsymbol{x}}^{(k)}, \tilde{\boldsymbol{y}}^{(k)} \right) \right\}_{k=1}^{N_{\text{augmented}}}$ denotes the augmented dataset and θ is the set of hyperparameters that define the augmentation process.

In this work, the augmentation operator f_θ is stochastic. In particular, the activation of augmentation for each sample is modeled through a Bernoulli random variable

$$X \sim \text{Bernoulli}(p_{\text{aug}}),$$

where $p_{\text{aug}} \in [0, 1]$ denotes the probability of applying augmentation. In our implementation, $p_{\text{aug}} = 0.8$, meaning that each sample in $\mathcal{D}$ has an 80% probability of being augmented during training.

When augmentation is applied, two augmented samples are generated by applying the transformation f_θ with independently sampled hyperparameters.

In general, data augmentation approaches can be divided into three main groups [25]:

- **Single-instance level augmentation:** Transformations applied independently to each sample, without referring to other samples in order to generate new data;
- **Multi-instance level augmentation:** Techniques that exploit multiple samples from $\mathcal{D}$ to create new data instances;
- **Dataset-level augmentation:** Methods that operate on the entire dataset $\mathcal{D}$, generating new samples from a probability distribution.

In this thesis, we consider single-instance level augmentation, which can be further categorized as follows:

- **Value-based transformation:** Transformations applied to the feature values of samples in $\mathcal{D}$.
- **Structure-based transformation:** Transformations applied to the structural or geometric properties of the data (e.g., spatial or geometric modifications).

Of all the structure-based transformations, we want to focus on geometric transformations, capable of changing the spatial relationship between pixels without touching their effective values.

When augmentation is activated f_θ applies a geometric transformation to the input image and its corresponding ground truth.

Let

$$\Omega = \{1, \dots, H\} \times \{1, \dots, W\} \subset \mathbb{Z}^2$$

be the discrete spatial domain of the image, and let $i = (i_x, i_y)$ the coordinates of the i-th pixel.

A geometric transformation is represented as:

$$\varphi_w : \Omega \to \mathbb{R}^2 \tag{2.1}$$

which maps an input coordinate to its transformed coordinate, where w is a random variable representing the chosen geometric transformation.

Given a geometric transformation φ_w that maps each pixel coordinate $i \in \Omega$ to a new position, the inverse trasformation can be defined as:

$$\varphi_w^{-1} : \mathbb{R}^2 \to \Omega$$

Every geometric transform can be implemented as **forward mapping** or **backward mapping** [24].

Forward mapping

The forward mapping transform each pixel coordinate $i = (i_x, i_y)$ of the input image to a new coordinate $i = \left(i'_x, i'_y\right)$, and its value is assigned to that location in the output image. The main problems associated to this type of mapping are the following:

- The transformed coordinates may fall outside the bounds of the output image;
- The transformed coordinates are generally not integer values due to the applied geometric transformation.

The second problem can be addressed by assigning the neareset integer values to i'_x and i'_y. However, this may lead to holes in the output image when some output pixels remain unassigned. A different type of mapping may be considered.

Backward mapping

The backward mapping iterates over each pixel of the output image and uses the inverse transformation to determine the position in the input image from which a value must be sampled. In this case the determined positions also may not lie within the bounds of the input image and may not be integers. But the output image has no holes.

It still remains open the problem of the non integer pixel coordinate values. In particular, we apply *smooth bilinear interpolation* to images $\boldsymbol{x}^{(k)}$ and *nearest-neighbor interpolation* to ground truth annotations $\boldsymbol{y}^{(k)}$. The augmented image and label are obtained through spatial resampling:

$$\tilde{\boldsymbol{x}}(i') = \mathcal{I}_X\left(\boldsymbol{x}, \varphi_\omega^{-1}(i')\right),$$

$$\tilde{\boldsymbol{y}}(i') = \mathcal{I}_Y\left(\boldsymbol{y}, \varphi_\omega^{-1}(i')\right),$$

We now analyze the geometric transformations used in this work [10], [1]

Random Rotation

The random variable w of (2.1) is expressed as an angle θ, representing the degree of turn, distributed as follows:

$$\theta \sim \mathcal{U}(-\alpha, \alpha)$$

where α defines the maximum rotation amplitude.

The geometric transformation is defined by the rotation mapping

$$\varphi_\theta(i) = c + \mathcal{R}_\theta \ (i - c)$$

where $c \in \mathbb{R}^2$ denotes the center of the rotation and

$$\mathcal{R}_\theta = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

is the *rotation matrix* which applies the rotation counterclockwise of an angle θ to the entire image.

For each pixel coordinate $i = (i_x, i_y) \in \Omega$, the transfromation is defined as:

$$\begin{pmatrix} i'_x \\ i'_y \end{pmatrix} = c + \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} i_x - c \\ i_y - c \end{pmatrix}$$

Random Scaling

The following geometric transformation resizes the dimension of an image. There are two types of scaling:

- **Uniform:** The same scaling is applied both to width and height, preserving its aspect ration,
- **Non-uniform:** Different scaling factors are applied along horizontal and vertical directions, altering the aspect ratio.

In this thesis we consider a uniform scaling trasformation.

The random variable w of the geometric transformation (2.1) is represented by a scaling factor $s \in \mathbb{R}$. Since the scaling is uniform, we have:

$$s_x = s_y = s$$

and s is sampled as follows:

$$s \sim \mathcal{U}(1-\delta, 1+\delta)$$

where δ is the *scale limit* which defines the range for random resizing.

The scaling transformation to each pixel coordinate $i = (i_x, i_y) \in \Omega$ is defined as:

$$\begin{pmatrix} i'_x \\ i'_y \end{pmatrix} = \begin{pmatrix} s_x & 0 \\ 0 & s_y \end{pmatrix} \begin{pmatrix} i_x \\ i_y \end{pmatrix}$$

Random Translation

Geometric translation can be expressed in a matricial form in the same way as rotation and scaling using homogeneous coordinates representation.

Each pixel coordinate $i = (i_x, i_y) \in \Omega$ is represented in homogeneous coordinates as:

$$i = \begin{pmatrix} i_x \\ i_y \\ 1 \end{pmatrix}$$

Similarly to the random scaling, the random variable w of (2.1) is represenetd by a translation factor $t_x \in \mathbb{R}$, for the traslation along width, and by $t_y \in \mathbb{R}$ for translations along height.In particular t_x, t_y are sampleed as follows:

$$t_x \sim \mathcal{U}(1-\lambda_x, 1+\lambda_x) \qquad t_y \sim \mathcal{U}(1-\lambda_y, 1+\lambda_y)$$

where λ_x, λ_y are the *translation limit* for width and height.

The translation transform to each pixel coordinate $i = (i_x, i_y) \in \Omega$ is defined as

$$\begin{pmatrix} i'_x \\ i'_y \\ 1 \end{pmatrix} = \begin{pmatrix} t_x & 0 & 0 \\ 0 & t_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} i_x \\ i_y \\ 1 \end{pmatrix}$$

Random Grid distortion

The following transformation divides the image into a number of equal grid cells defined by n_{steps}, and randomly distorts the grid, producing a spatial warping effect.

The random variable w is represented by a distortion factor $d \in \mathbb{R}$ sampled as:

$$d \sim \mathcal{U}(-\mu, \mu)$$

where μ is the *distortion limit* factor, range of distortion for each cell in the grid.

The grid distortion transformation is expressed as a nonlinear spatial mapping

$$\varphi_d(i) = i + u_d(i)$$

where $u_d(i) \in \mathbb{R}^2$ is a displacement field obtained by interpolating the randomly perturbed grid control points.

Random Horizontal and Vertical Flip

These geometric transformations flip the image around the y-axis (horizontal flip) or the x-axis (vertical flip). They are self-inverse transformations, meaning that applying them twice returns the original image.

Let $p_{flip} = 0.5$ the flipping probability for a given image, and let w random variable of (2.1) respectively represented by f_H for horizontal flipping and f_V for vertical flipping. In bot cases:

$$f_H,\ f_V \sim \text{Bernoulli}\,(p_{flip})$$

The horizontal flip can be expressed through homegeneous coordinates as matricial transformation

$$\begin{pmatrix} i'_x \\ i'_y \\ 1 \end{pmatrix} = \begin{pmatrix} -1 & 0 & W-1 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} i_x \\ i_y \\ 1 \end{pmatrix}$$

where W is the image width. In a similar way the vertical flip follows

$$\begin{pmatrix} i'_x \\ i'_y \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & H-1 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} i_x \\ i_y \\ 1 \end{pmatrix}$$

where H is the image height.

Chapter 3

A Novel Approach to Image Segmentation

As mentioned in the introduction, SynDiag is a company dedicated to supporting physicians in the diagnosis of ovarian tumors. To achieve this objective, the company has developed OvAi-Focus, an artificial intelligence–based system powered by advanced deep learning models. Through the use of these architectures trained on annotated medical data, the system is able to automatically detect patterns, extract relevant features, and identify potential anomalies associated with the pelvic region.

The platform provides an accurate, data-driven assessment to assist clinicians in making more informed diagnostic decisions. The output consists of the original ultrasound image enhanced with clearly highlighted regions of interest. Through image segmentation techniques, the system identifies and emphasizes areas of concern, allowing for a more precise, consistent, and reliable interpretation of the examination.

The segmentation algorithm developed by SynDiag represents a core module within the OvAi-Focus system. Over the years, several versions of the segmentation pipeline have been implemented and progressively refined.

The first version, OvAi, was designed to identify ovarian cysts, which can be non-solid when they contain fluid, or solid when the solid component exceeds a certain percentage of the total mass. Building upon this initial model, a second version, OvAi-2, was developed to provide a more comprehensive analysis, enabling the detection and segmentation of both solid and non-solid components of an adnexal mass. Such masses may correspond to different types of ovarian lesions, including benign or malignant tumors, and their identification may support clinical evaluation and early diagnosis.

A visual representation of the segmentation results produced by OvAi-2 is shown in Figure 3.1.

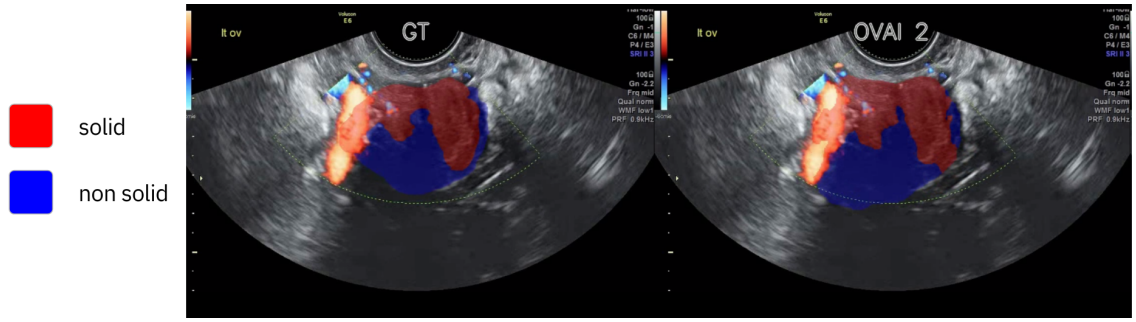


Figure 3.1: Segmentation of an adnexal mass produced by the OvAi-2 segmentation algorithm. GT: ground-truth segmentation mask representing the adnexal mass. OVAI2: predicted segmentation mask generated by the OvAi-2 segmentation pipeline.

Up to now, the last segmentation algorithm provided is the OvAi-3 in which it has been added the *healthy-ovary* class for providing its segmentation. As a preliminary qualitative validation, we provide two examples illustrating the current capabilities and limitations of OvAI-3 in handling this new class. In the first example, Figure 3.2 , the model correctly detects a structure and segments it as healthy ovary, whereas the ground truth does not annotate any structure in that region, suggesting that the model may be misclassifying a mass as healthy ovary.

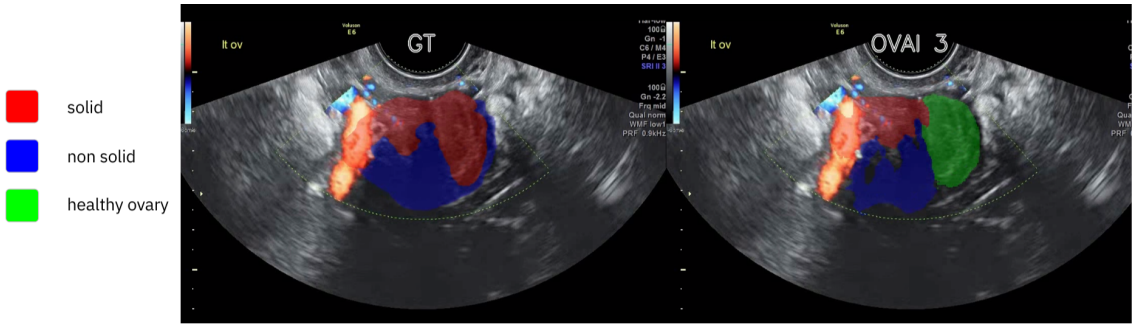


Figure 3.2: Segmentation of the same adnexal mass of Figure 3.1 provided by OvAi-3 segmnetation algorithm. Example of the misclassification of the mask. GT: ground-truth segmentation mask representing the adnexal mass. OVAI3: predicted segmentation mask generated by the OvAi-3 segmentation pipeline.

However, in the second example, Figure 3.3, the ground truth contains only a healthy ovary, yet OvAI-3 predicts additional tumor components within the same region, demonstrating the tendency of the model to revert to previously learned classes when encountering the new one.

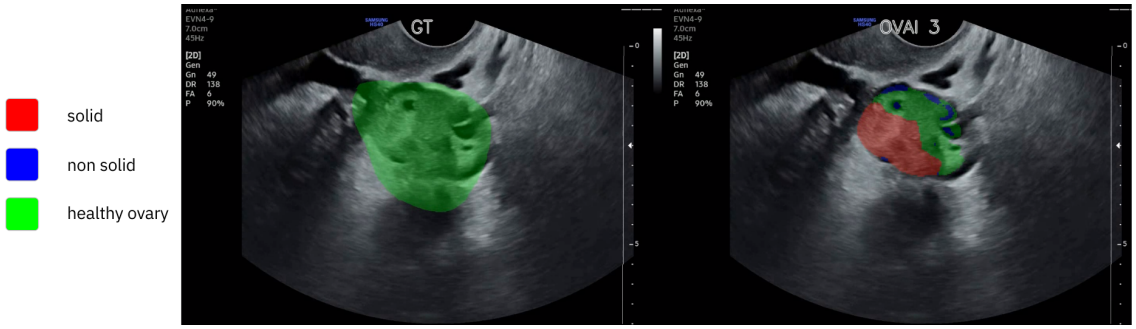


Figure 3.3: Segmentation of a healthy ovary produced by the OvAi-3 segmentation algorithm, illustrating an example of the class-reversion process. GT: ground-truth segmentation mask representing the healthy ovary. OVAI3: predicted segmentation mask generated by the OvAi-3 segmentation pipeline.

Furthermore, in the field of artificial intelligence, the introduction of new data categories gives rise to the well-known problem of *catastrophic forgetting*, in which a model tends to lose previously acquired knowledge when trained on new data. This phenomenon is reflected in OvAI-3, where the addition of the healthy ovary class has led to degraded performance on the tumor classes compared to earlier model versions, confirming that the model partially forgot what it had previously learned in order to accommodate the new class.

The addition of further classes, such as the uterus and in particular its components, the myometrium and endometrium, could lead to a deterioration in segmentation performance. However, this objective is necessary for two main reasons.

The first is to obtain a more complete and accurate segmentation algorithm capable of identifying a larger number of organs. The second is to provide additional support to physicians during the diagnostic process.

The objective of this thesis is the development of a new segmentation model, not yet developed by SynDiag, characterized by the inclusion of the uterus, more specifically its components: the myometrium and the endometrium.

To achieve this goal, an initial benchmark model is developed, based on a *flat* segmentation approach, similarly to what was done for the previous OvAi models.

Indeed, all OvAI releases adopt a *flat* pixel-wise multiclass segmentation approach, in which each pixel of a given image is independently assigned to a semantic class. All classes are considered at the same semantic level, and no hierarchical structure or relationships between classes are modeled. For this reason, such an approach is referred to as *flat* segmentation. An approach of this type, especially in the medical domain, presents several limitations:

- It ignores anatomical relationships between structures;
- It struggles to accurately classify thin or nested structures due to class imbalance;
- It may produce an errate pixel-wise classification, such as predicting a solid mass into a healthy ovary.

This benchmark model serves two main purposes. First, it allows us to highlight the limitations of the flat segmentation approach. Second, it enables us to evaluate the behavior of the segmentation model after the introduction of the new anatomical classes, namely the myometrium and the endometrium.

The analysis of these limitations motivates the adoption of a different segmentation strategy. The goal of this work is therefore to introduce a *hierarchical* approach to the segmentation task in order to preserve semantic coherence among classes and improve the correctness of pixel-wise label assignment. This is achieved through a hierarchical detection strategy that reflects the anatomical organization of the structures of interest.

As explained in Chapter 2, the labeled dataset consists of ultrasound images acquired as frames from transvaginal ultrasound examinations, together with their corresponding groundtruth segmentation masks. Such a dataset can be formally expressed in a mathematical notation as:

$$\mathcal{D} = \left\{ \left(\boldsymbol{x}^{(k)}, \boldsymbol{y}^{(k)} \right) \right\}_{k=1}^{N},$$

where N denotes the total number of samples. Each input image is defined as

$$\boldsymbol{x}^{(k)} \in \mathbb{R}^{H \times W \times C_{\text{image}}},$$

where H and W denote the spatial dimensions and C_{image} the number of channels.

The semantic classes considered in this study are:

Background *Solid* *Non-solid* *Uterus* *Myometrium* *Endometrium*

In semantic segmentation problems, the ground truth can be represented using different mask encodings. The two most common representations [7] are described below.

Categorical segmentation mask

Let $\Omega \subset \mathbb{Z}^2$ denote the discrete image domain, K the total number of classes, and $\mathcal{C} = \{0, \dots, K-1\}$ the set of integers class indices, also called class labels.

A categorical segmentation mask is defined as:

$$\mathbf{y}^{(k)} : \Omega \to \mathcal{C}, \qquad i \mapsto \mathbf{y}^{(k)}(i) = c,$$

meaning that each pixel $i \in \Omega$ is assigned to a single class label.

One-hot encoded segmentation mask

An equivalent representation can be obtained through one-hot encoding

$$\boldsymbol{y}^{(k)} : \Omega \to \{0, 1\}^{|\mathcal{C}|},$$

where each pixel is associated with a binary vector of length K, and only the component corresponding to the true class is equal to 1.

In this thesis, the categorical representation is adopted. Compared to one-hot encoding, it is more memory efficient while preserving the same semantic information.

Having defined the encoding scheme for the segmentation masks, we now introduce the class–label assignment,where each class is assigned a unique integer label.

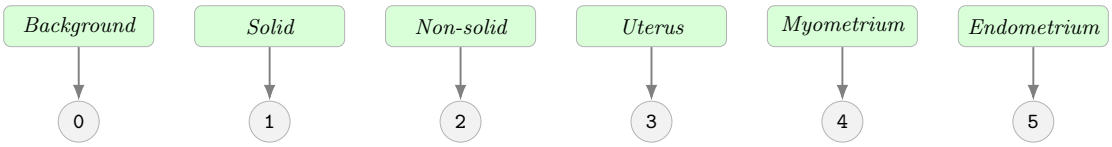


Figure 3.4: Class-label mapping used for the categorical representation of segmentation masks.

The hierarchical formulation of the problem will be presented in Section 3.2. In the following, we explain the *flat* pixel-wise multiclass segmentation approach, equipped with a class-weighting term to address the class imbalance present in the dataset.

3.1 *Flat* Semantic Segmentation

In this section, we present the architecture adopted for the *flat* semantic segmentation task, with particular emphasis on the formulation of the loss function.

The architecture adopted is a U-Net, see Subsection 1.3.3, and the gradient-descent optimization algorithm is Adam, see Algorithm 4. The output of this architecture, before the classification layer, is a pixel-wise vector of logits given by:

$$\boldsymbol{z}_i = [z_{0,i}, \dots, z_{K-1,i}]\,.$$

Subsequently, applying the non-linear activation function Softmax, we obtain the following pixel-wise probability distribution vector:

$$\mathbf{p}_i = [p_{0,i}, \dots, p_{K-1,i}]\,.$$

For each pixel i, the predicted probabilities satisfies:

$$\sum_{j=0}^{K-1} p_{j,i} = 1.$$

Furthermore, the probability associated with the c-th class at pixel i is given by

$$p_{c,i} = \frac{e^{z_{c,i}}}{\sum_{j=0}^{K-1} e^{z_{j,i}}}, \qquad c = 0, \dots, K-1.$$

Finally, the predicted class label at pixel i is obtained by selecting the class with maximum probability:

$$\hat{c}_i = \underset{c \in \{0,\dots,K-1\}}{\arg\max}\; p_{c,i}.$$

The model is trained using the sparse categorical focal loss [26], which is particularly suitable for pixel-wise multiclass segmentation tasks characterized by a strong imbalance between classes in the available data.

Sparse categorical focal loss is a modified version of sparse categorical cross entropy [22], a widely used loss function in multiclass classification problems in which class labels are given as integers rather than one-hot vectors.

For this reason, we begin by introducing the sparse categorical focal loss. For each pixel i it assumes the following form:

$$\mathcal{L}_i^{\text{SCCE}} = -\log\left(p_{c,i}\right),$$

where $p_{c,i}$ is the predicted probability assigned by the model to the ground-truth class c at pixel i.

The class imbalance problem can be addressed by introducing a class-weighting vector

$$\boldsymbol{\alpha} = [\alpha_0, \ldots, \alpha_{K-1}],$$

where each $\alpha_c > 0$ is a coefficient associated with class $c \in \mathcal{C}$. Applying this weighting to the sparse categorical cross-entropy yields its class-balanced version:

$$\mathcal{L}_i^{\text{SCCE-bal}} = -\alpha_{c,i} \log\left(p_{c,i}\right),$$

where $c = \mathbf{y}^{(k)}(i)$ denotes the ground-truth class at pixel i.

However, this balanced formulation does not account for the different contributions of misclassified and well-classified pixels. To address this issue, we introduce the modulating factor

$$(1 - p_{c,i})^{\gamma},$$

which depends on a focusing parameter $\gamma \geq 0$. The sparse categorical focal loss is defined as:

$$\mathcal{L}_i^{\text{SCFL}} = \alpha_{c,i} \left(1 - p_{c,i}\right)^{\gamma} \mathcal{L}_i^{\text{SCCE}}.$$

Setting $\gamma > 0$ reduces the loss contribution from well-classified examples while placing greater focus on hard examples. More precisely, when a pixel is misclassified, $p_{c,i}$ is small and the modulating factor is close to 1, so the loss is only slightly affected. Conversely, when $p_{c,i} \to 1$, indicating a well-classified pixel, the loss is downweighted.

Sparse categorical cross entropy is a special case of the sparse categorical focal loss, correspondent to the case when $\alpha_{c,i} = 1 \; \forall c \in \mathcal{C}$ and $\gamma = 0$.

3.2 Literature Review

In this section, we present the literature review conducted during the internship at SynDiag. Although the review initially considered a wide range of articles, we present only the most relevant one able to manage at their best the pixel-wise hierarchical approach of image segmentation.

3.2.1 Focal Hierarchical Loss for Semantic Segmentation

The first topic of this review concerns the article "A Hierarchical Loss for Semantic Segmentation" [15] by Muller et al. (2020).

As reminded above, in flat segmentation approach, classes are treated as independent categories, whereas hierarchical segmentation exploits structured relationships between them.

To understand this approach, it is necessary to introduce the concept of hierarchy among semantic structures. In the medical domain, anatomical regions are inherently organized in hierarchical relationships. In our case study, the anatomical structures follow a hierarchical organization in which larger regions contain smaller substructures.

Level 1:

The pelvic anatomical region acquired through transvaginal ultrasound imaging and the background, region outside the anatomical which primarily represent noise.

Level 2:

Primary anatomical structures and and pathological lesion:

- Uterus
- Healthy ovary
- Adnexal mass

Level 3:

Anatomical substructures of the organs listed above and internal lesion components:

- Solid tumor components, typically associated with malignant lesions;
- Non-solid components, corresponding to cystic areas or complex fluid;
- Endometrium, the mucous membrane hat lines the inner part of the uterus;
- Myometrium, the smooth muscle layer and the thickest layer of the uterine wall. Responsible for contractions during childbirth.

To describe the hierarchical structure from a mathematical perspective, we introduce the following definitions

Definition 6 (Directed Graph) *A directed graph $\mathcal{G}$ is a pair $(\mathcal{V}, \mathcal{E})$, where:*

- *$\mathcal{V}$ is a set whose elements are called vertices (or nodes),*

$$\mathcal{V} = \{v_1, \dots v_n, \}.$$

- $\mathcal{E}$ *is a set of edges representing connections between pairs of vertices,*

$$\mathcal{E} \subseteq \{(v_i, v_j) \mid v_i, v_j \in \mathcal{V},\ i \neq j\}.$$

where (v_i, v_j) denotes an ordered pair of vertices, meaning that an edge connects two vertices with a associated direction.

In a directed graph, two vertices v_i and v_j are said to be *connected* if there exists a path between them. A directed graph is said to be *connected* if every pair of vertices is connected.

Definition 7 (Simple graph) *A simple graph is a directed graph that contains no self-loops and no multiple edges between the same pair of vertices.*

A *tree* is a connected simple directed graph with no cycles. Furthermore, a tree with n vertices has exactly $n - 1$ edges.

In this thesis, we take in consideration a special type of tree called **rooted tree** characterized by a special type of node called *root* node. This graph inherently exhibits a hierarchical structure, the nodes of the tree point towards the *root* node.

In particular, given a pair of vertices (v_i, v_j), v_j is said to be *parent* node of v_i which is automatically referred to as the *child* node. Every vertex has a unique parent except the root that has no parents.

Given a vertex v in a rooted tree, the following terminology is used:

- An **ancestor** of v is any vertex that is either the parent of v or, recursively, an ancestor of the parent of v;
- A **descendant** of v is any vertex that is either a child of v or, recursively, a descendant of one of its children;
- A **leaf node** is a vertex with no children;
- An **internal node** is any vertex that is not a leaf node.

The depth of a node v is defined as the number of edges between the nodes itself and the *root* node. From this definition, the depth of a rooted tree can be defined as:

$$D_{\text{tree}} = \max_i \text{depth}\,(v_i)\,,$$

that is, the maximum depth among all vertices in the tree.

The rooted tree representing the hierarchical class structure is shown in Figure 3.5.

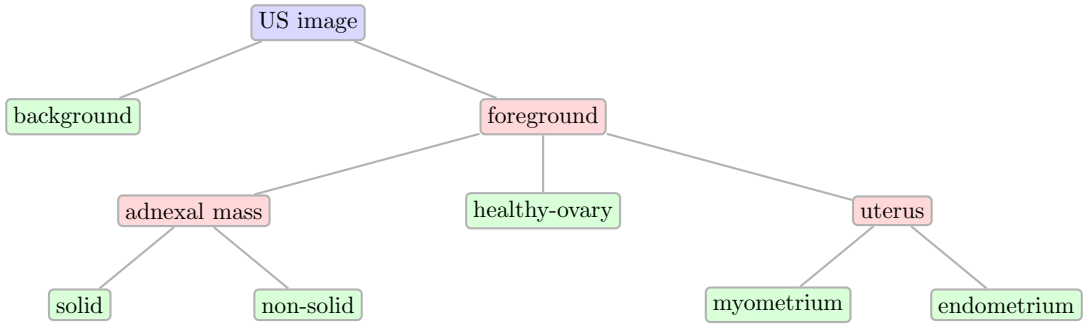


Figure 3.5: Hierarchical tree, *root* node represented as the entire ultrasound image obtain through transvaginal ultrasound imaging.

We suppose that the first m nodes of our hierarchical tree correspond to the leaf nodes, while the remaining $n - m$ correspond to the non-leaf nodes; furthermore, the leaf nodes of the tree represent the actual classes of the dataset.

This approach can be regarded as a hybrid between traditional flat semantic segmentation and hierarchical segmentation. The hierarchical structure is introduced exclusively during the learning phase through the definition of a multi-level loss, while the inference architecture remains unchanged and therefore produces pixel-wise predictions only for the leaf nodes of the hierarchy.

The method is based on two fundamental principles:

- A standard *flat* segmentation architecture, U-Net, that outputs, for each pixel i, a classification over the leaf nodes of the hierarchical tree, trained Adam optimizer, see Algorithm 4.
- A hierarchical loss function defined as the sum of the losses computed at each level of the tree:

$$\mathcal{L} = \sum_{d=0}^{D_{\text{tree}}} \mathcal{L}^{(d)},$$

  where $\mathcal{L}^{(d)}$ denotes the loss associated with level d of the hierarchy. Since the root corresponds to the entire image domain, we assume $\mathcal{L}^{(0)} = 0$.

Therefore, the proposed method can ultimately be interpreted as a pixel-wise multiclass classification problem, whose optimization is guided by a hierarchical loss function.

Now, the natural question is the following:

How do we infer probabilities for the non-leaf nodes?

Probabilities for non-leaf nodes are obtained by summing the probabilities of their descendant nodes.

Let $\text{desc}(v)$ denote the set of leaf descendants of a non-leaf node v. Given the leaf probabilities $p_{c,i}$ at pixel i, the probability associated with node v is defined as

$$p_{v,i} = \sum_{c \in \text{desc}(v)} p_{c,i},$$

where each $p_{c,i} \in [0,1]$ and $\sum_{c \in \mathcal{C}} p_{c,i} = 1$. Consequently, $p_{v,i} \in [0,1]$.

Having defined how to infer probabilities for non-leaf nodes, the next step is to specify how the loss is computed at each level of the hierarchy.

The available annotations correspond only to leaf nodes, i.e., the actual classes of the dataset. Let

$$\mathcal{C} = \{0, \ldots, m-1\}$$

denote the set of leaf class labels.

For each depth d, the hierarchy induces a partition of the set $\mathcal{C}$ into disjoint groups called branches. Let

$$\mathcal{B} = \{\mathcal{B}^{(d)} \mid d = 1, \ldots, D_{\text{tree}}\}$$

denote the set of branch levels of the hierarchy, where each $\mathcal{B}^{(d)}$ is the set of branches at depth d, defined as

$$\mathcal{B}^{(d)} = \{b^{(d)}(c) \subseteq \mathcal{C} \mid c \in \mathcal{C}\}.$$

Here $b^{(d)}(c)$ denotes the branch at depth d associated with the leaf c.

A visual representation, excluding the *root* node, follows in Figure 3.6.

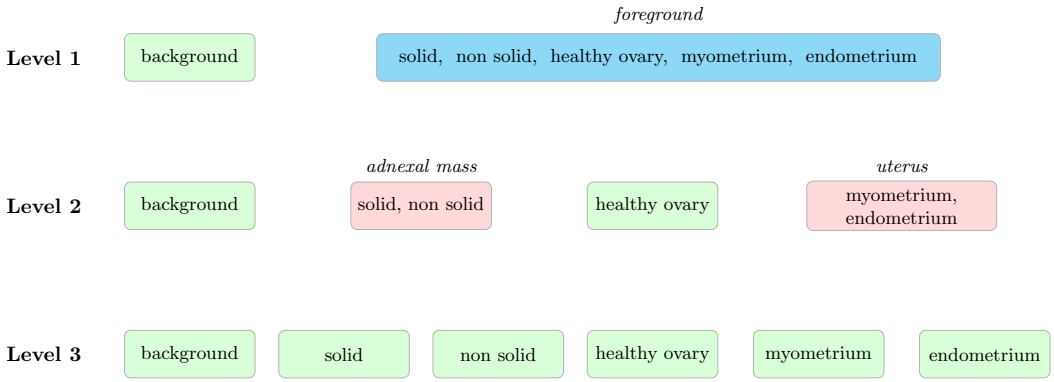


Figure 3.6: Representation of the branches at each level of the hierarchical tree.

The probability associated with leaf class c at pixel i and depth d is defined as

$$p_{c,i}^{(d)} = \begin{cases} p_{c,i} & \text{if } d = D_{\text{tree}} \\ p_{b^{(d)}(c),i}^{(d)} & \text{if } d < D_{\text{tree}} \end{cases},$$

where $p_{b^{(d)}(c),i}^{(d)}$ is defined as

$$p_{b^{(d)}(c),i}^{(d)} = \sum_{j \in b^{(d)}(c)} p_{j,i}.$$

Intuitively, $p_{b^{(d)}(c),i}^{(d)}$ corresponds to the probability of the ancestor node of the leaf node associated with class label c at depth d.

This formulation allows the use of standard loss functions while keeping the ground-truth label space unchanged.

Following the flat segmentation setting, a focal-type loss with a class-imbalance weighting factor and a modulating term is adopted. For each pixel i, the hierarchical focal loss at depth d is defined as:

$$\mathcal{L}_i^{(d)} = -\alpha_{c,i}\left(1 - p_{c,i}^{(d)}\right)^{\gamma} \log\left(p_{c,i}^{(d)}\right),$$

Under the *flat* segmentation approach, all pixel misclassifications are penalized equally; for example, predicting a tumor as a healthy ovary is penalized in the same way as predicting it as background. In constrast, the hierarchical focal loss introduces depth-dependent penalties: errors at the leaf level receive a stronger penalty, while mistakes at coarser levels of the hierarchy are penalized less severely. This behavior encourages anatomically consistent predictions and reduces semantically distant misclassifications in the final segmentation map.

3.2.2 Hierarchical Semantic Segmentation Network

The second hierarchical approach considered in this thesis for addressing the segmentation task, while explicitly considering semantic relationships among classes, is based on the work "Deep Hierarchical Learning for Semantic Segmentation" [12] (2022).

As discussed in the previous section, hierarchical semantic segmentation approaches organize semantic classes according to hierarchical tree. In this setting, classes are connected through root-to-leaf paths, e.g., foreground $\rightarrow$ adnexal mass $\rightarrow$ solid components.

In this setting, the core idea is to design a hierarchical semantic segmentation framework that explicitly incorporates the tree-structured semantic relationships among classes both at the architectural level, more precisely during the classification phase, and during the learning phase. This framework is referred to as the Hierarchical Semantic Segmentation Network (HSSN).

The HSSN framework can be regarded as an evolution of classical scene parsing approaches [21], which consist in dividing an image innto meaningful regions, and more specifically of human parsing methods [27]. Traditional scene parsing relied on graph-based representations to model relationships between classes, but did not employ neural network architectures. In contrast, human parsing introduced deep neural networks for segmentation tasks; however, these approaches did not explicitly account for hierarchical class structures.

The proposed HSSN framework combines the strengths of both paradigms. It leverages a deep neural network architecture that preserves hierarchical behavior during the learning phase while adapting the segmentation process to a hierarchical setting with only minimal architectural modifications.

In this regard, instead of the standard U-Net adopted in the two previous cases, we choose to employ PSPNet, see Subsection 1.3.4, as it is better suited to capturing the global contextual information of the image. The optimizer employed for training is Adam, see algorithm 4. The PSPNet prooperty is particularly relevant in our setting, since it enables a more effective modeling of the hierarchical relationships

among classes, which represent a core aspect of the proposed framework.

In contrast to previous segmentation approaches, characterized by a pixel-wise multiclass semantic segmentation which means that each pixel is assigned to a single class label in the classification phase of the model, the HSSN framework extends the classification process to a pixel-wise multi-label setting. This ensures that each predicted pixel belonging to a given class, it automatically belongs to his ancestor. For instance, if a pixel is classified as *solid*, it also belongs to *adnexal mass* and *foreground* according to the hierarchical structure.

The introduction of a hierarchical approach during the learning phase of HSSN framework is possible through these two hierarchical constraints:

1. A pixel belonging to a given semantic class must belong also to all his ancestors;
2. A pixel not beloging to a given class, automatically does not belong to the descendant of that class.

These hierarchical constraints brings to a learning strategy called **pixe-wise hierarchical segmentationn learning**, able to control the hierarchical coherence during the each pixel prediction. Moreover, to better enforce the hierarchy among semantic classes, we operate directly in the embedding space of each pixel, bringing embeddings that share similar semantic characteristics closer together and pushing apart those that exhibit semantically dissimilar features. This strategy is called **pixel-wise hierarchical representation learning** strategy. The entire learning procedure of HSSN will be described later.

As in the previous section, the hierarchical structure adopted to construct the HSSN framework is modeled as a rooted tree $\mathcal{T} = (\mathcal{V}, \mathcal{E})$ with the following properties:

- Each node $v \in \mathcal{V}$ is a representation of a semantic class;
- Each edge $(u, v) \in \mathcal{E}$ encodes the semantic relationship between two nodes, equivalently between two semantic classes, where the node v is *parent* node of the *child* node u;
- The root node v^r represents the most general semantic class in the hierarchy.

Furthermore, we assume that each node $v \in \mathcal{V}$ is both the parent and the child of itself, i.e. $(v, v) \in \mathcal{V}$. Although this assumption may seem unnecessary at this stage, its purpose will become clear later.

The hierarchical tree is the same represented in Figure 3.5; in this setting, the set of leaf nodes is denoted by:

$$\mathcal{V}_{\mathcal{X}} = \{\textit{background}, \textit{solid}, \textit{non-solid}, \textit{healthy-ovary}, \textit{myometrium}, \textit{endometrium}\},$$

and, furthermore, we denote by $\hat{\nu}_{\mathcal{X}} \in \mathcal{V}_{\mathcal{X}}$ the *groundtruth leaf label* associated to a leaf node. Once the hierarchical structure required to understand the HSSN framework has been introduced, we can now present the main modifications that distinguishes HSSN from previous models.

A first modification concerns the representation of the ground truth segmentation mask. In the previous approaches, the ground truth mask was categorical, i.e.,

each pixel was associated with a single class index corresponding to a leaf node. In HSSN, instead, the categorical mask is converted into a hierarchical one-hot encoded representation.

More precisely, if a pixel is labeled as solid, the corresponding one-hot vector assigns value 1 not only to the leaf class *solid*, but also to all its ancestor classes in the hierarchy, namely *tumor* and *foreground*. We are in presence of a multi-one-hot groud truth segmentation mask. This hierarchical encoding is obtained from the original categorical mask, whose indices correspond only to the leaf nodes, indicized as above, by activating all nodes along the root-to-leaf path associated with the ground truth class.

The second substantial modification concerns the choice of the activation function. In the previous pixel-wise multiclass setting, the natural choice was the Softmax function, since each pixel was assigned exactly one class label. In the present case, however, the problem becomes a pixel-wise multi-label classification task; therefore, the most appropriate activation function is the Sigmoid function, which allows multiple classes to be activated simultaneously for each pixel. Thus, as previously discussed, the score map was defined as a tensor

$$\boldsymbol{S} \in \mathbb{R}^{H \times W \times |\mathcal{V}_{\mathcal{X}}|},$$

where $|\mathcal{V}_{\mathcal{X}}|$ is the cardinality of the leaf nodes; in the proposed framework, however, the score map assumes the following form:

$$\boldsymbol{S} \in \mathbb{R}^{H \times W \times |\mathcal{V}|},$$

where $|\mathcal{V}|$ represents the total number of nodes in the hierarchical tree, including the root.

Before introducing the classification step and illustrating how it ensures consistency between pixel-wise predictions and the class hierarchy, we first provide a detailed description of the hierarchical learning phase.

Pixel-wise hierarchichal segmentation learning:

To describe how hierarchical relationships among the nodes are integrated into the learning process, ensuring that the score map satisfies the imposed hierarchical constraints, we begin by introducing the following intrinsic properties from graph theory, more specifically those related to rooted trees [3].

Definition 8 (Positive $\mathcal{T}$-property) *For each pixel i, if a class is labeled positive, all its ancestor nodes in $\mathcal{T}$ should be labeled positives.*

Definition 9 (Negative $\mathcal{T}$-property) *For each pixel i, if a class is labeled negative all its child nodes in $\mathcal{T}$ should be labeled negatives.*

As mentioned above, the goal is to enforce hierarchical consistency for each pixel-wise prediction

$$\boldsymbol{s}_i = [s_{v,i}]_{v \in \mathcal{V}} \in [0,1]^{|\mathcal{V}|}$$

For consistency, we denote by $\boldsymbol{y}_i = [y_{v,i}]_{v \in \mathcal{V}} \in \{0,1\}^{|\mathcal{V}|}$ the *ground-truth binary label vector* associated with pixel i and its corresponding score vector $\boldsymbol{s}_i$.

The two properties introduced above are applied to each component $s_{v,i}$ of the score vector as follows:

Definition 10 (Positive $\mathcal{T}$-constraint) *For each pixel i, if the class associated to the node v is labeled positive, and u is an ancestor node of v, then holds*

$$s_{v,i} \leq s_{u,i} \quad \forall\, u \in anc\,(v)\,. \tag{3.1}$$

Definition 11 (Negative $\mathcal{T}$-constraint) *For each pixel i, if the class associated to the node v is labeled negative, and u is a child node of v, then holds*

$$1 - s_{v,i} \leq 1 - s_{u,i} \quad \forall\, u \in desc\,(v)\,. \tag{3.2}$$

A visual representation of the hierarchical properties follows in Figure 3.7.

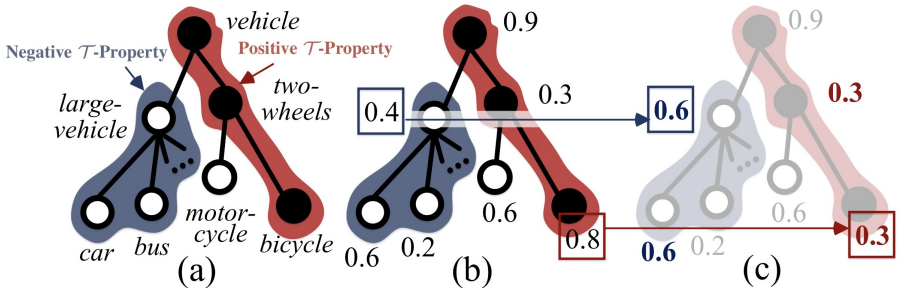


Figure 3.7: (a) Representation of the Positive $\mathcal{T}$-property (in red) and Negative $\mathcal{T}$-property(in blue). (b) Violation of Positive $\mathcal{T}$ constraint and Negative $\mathcal{T}$ constraint. (c) Adjustment of the score values through a loss.

Now, the goal is to adapt the previously introduced score map $\boldsymbol{S}$ into a new score map, denoted as $\boldsymbol{P}$, such that the positive and negative $\mathcal{T}$-constraints are satisfied. This guarantees that the resulting score map is consistent with the underlying hierarchical structure. Let:

$$\mathcal{A}_v = \{u \in \mathcal{T} \mid u \in \text{anc}\,(v)\}\,,$$

the set of all the ancestor nodes of v, and

$$\mathcal{C}_v = \{u \in \mathcal{T} \mid u \in \text{desc}\,(v)\}\,,$$

the set of all the descendant nodes of v. Note that, according to what it has been previously introduced $(v, v) \in \mathcal{V}$, follows that $v \in \mathcal{A}_v$ and $v \in \mathcal{C}_v$.

Thus, it is possible to define

$$p^+_{v,i} = \min_{u \in \mathcal{A}_v} (s_{u,i})\,,$$

which is equal to the minimum of the score map values of the ancestors of v at pixel i, if node v is labeled positive. Equivalently,

$$p^-_{v,i} = \max_{u \in \mathcal{C}_v} (s_{u,i}),$$

which corresponds to the maximum of the score map values of the descendants of v if node v is labeled negative.

Thus, for each pixel i, it is possible to define the score map vector $\boldsymbol{p}_i = [p_{v,i}]_{v \in \mathcal{V}} \in [0,1]^{|\mathcal{V}|}$ element wise as:

$$p_{v,i} = y_{v,i}\ p^+_{v,i} + (1 - y_{v,i})\ \ p^-_{v,i}.$$

This guarantees that the hierarchical constraints are satisfied for each pixel-wise prediction $\boldsymbol{p}_i$.

Once it has been explained how the pixel-wise score vector $\boldsymbol{p}_i$ satisfy the hierarchical constraints imposed by the tree, it is possible to define the **Focal Tree-Min Loss** as:

$$\begin{aligned}
\mathcal{L}^{\mathrm{FTM}}(\boldsymbol{p}_i) &= \sum_{v \in \mathcal{V}} -\alpha_{v,i}\ y_{v,i}(1 - p_{v,i})^\gamma \log(p_{v,i}) - (1 - y_{v,i})(p_{v,i})^\gamma \log(1 - p_{v,i}) \\
&= \sum_{v \in \mathcal{V}} -\alpha_{v,i}\ y_{v,i} \left(1 - \min_{u \in \mathcal{A}_v} s_{u,i}\right)^\gamma \log\left(\min_{u \in \mathcal{A}_v} s_{u,i}\right) + \\
&\quad - (1 - y_{v,i}) \left(\max_{u \in \mathcal{C}_v} s_{v,i}\right)^\gamma \log\left(1 - \max_{u \in \mathcal{C}_v} s_{u,i}\right),
\end{aligned} \tag{3.3}$$

where $\alpha_{v,i}$ is the imbalance value for the positive labelled node v at pixel i; negative samples were not reweighted, the focal term on negatives already reduces their contribution.

Pixel-wise hierarchical representation learning

Up to this point, the construction of a hierarchical score map $\boldsymbol{P}$ has been presented, where pixel-wise predictions are guaranteed to satisfy the hierarchical constraints.

However, to achieve a more accurate classification, the idea is to act directly on the pixel embedding representation space, in order to separate embeddings with dissimilar semantic features and cluster those with similar semantic features. This can be translated into the design of a specific loss function that operates directly in the embedding space.

Before introducing the proposed formulation, we briefly recall the notion of pixel-wise embeddings. The PSPNet encoder, which we denote as the function

$$f_{\mathrm{enc}} : \mathbb{R}^{H \times W \times C_{\mathrm{image}}} \to \mathbb{R}^{\frac{H}{2^d} \times \frac{W}{2^d} \times C_{\mathrm{enc}}},$$

transforms the input image I into a dense feature map $\boldsymbol{I}$, whose spatial resolution is reduced according to the encoder depth d. The value C_{enc} denotes the

number of feature maps produced by f_{enc}. Each spatial location in $\boldsymbol{I}$ is represented by a vector $\mathbf{i} \in \mathbb{R}^{C_{\text{enc}}}$, which corresponds to the semantic embedding of pixel i. A visual representation follows in Figure 3.8

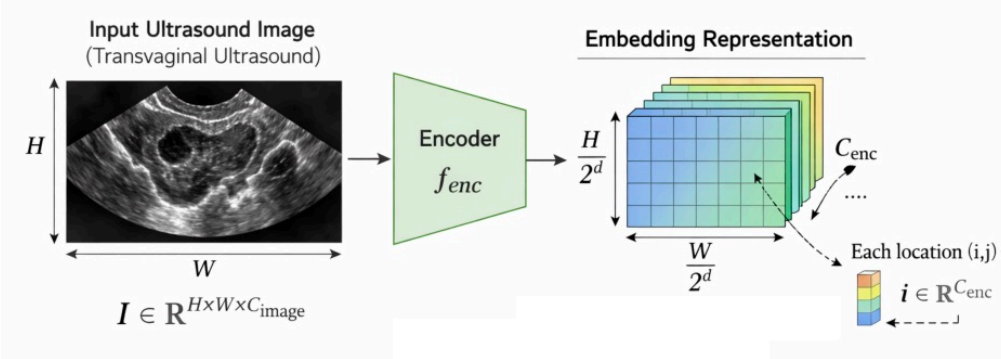


Figure 3.8: Feature embedding map obtained from the encoder applied to a transvaginal ultrasound image.

Given any hierarchichal tree structure, it is possible to introduce the following definition [4].

Definition 12 (Tree-based semantic distance) *Let* $= (\mathcal{V}, \mathcal{E})$ *be a tree representing a hierarchical label structure, where* $\mathcal{V}$ *is the set of nodes and* $\mathcal{E}$ *the set of edges. For any pair of nodes* $u, v \in V$*, their distance in the hierarchy is defined as*

$$\psi(u, v) = \textit{length of the unique shortest path between } u \textit{ and } v \textit{ in } \mathcal{T},$$

where the length is measured as the number of edges along the path.

Furthermore, the semantic distance is a metric the following properties holds:

- It is a non negative distance, and $\psi(u, v) = 0$;
- It is symmetric, $\psi(u, v) = \psi(v, u)$;
- The triangular inequality holds.

To conclude, the semantic distance $\psi(\cdot, \cdot)$ can be viewed as a metric of semantic similiarity between two classes in the heirarchy.

This fundamental property bring us to define a **tree-triplet loss** [6], also called hierarchical triplet loss, a novel approach to the standard triplet loss [19].

For each pixel i, it is possible to define a set of pixel triplet

$$\left\{i, i^+, i^-\right\},$$

referred to as the anchor pixel, the positive pixel, and the negative pixel, respectively. The ground-truth leaf labels associated with the pixel triplet are denoted by $\hat{\nu}_{\mathcal{X}}$, $\hat{\nu}_{\mathcal{X}}^+$, and $\hat{\nu}_{\mathcal{X}}^-$, such that:

$$\psi\left(\hat{\nu}_{\mathcal{X}}, \hat{\nu}_{\mathcal{X}}^+\right) < \psi\left(\hat{\nu}_{\mathcal{X}}, \hat{\nu}_{\mathcal{X}}^-\right),$$

meaning that the positive ground-truth label is semantically closer to the anchor label than the negative one.

Thus, in the tree-triplet loss, each pixel triplet is selected according to the semantic similarity relationships imposed by the hierarchical tree. This represents the main difference compared to the standard triplet loss, where the positive sample belongs to the same class as the anchor, i.e., $\hat{\nu}_{\mathcal{X}}^{+} = \hat{\nu}_{\mathcal{X}}$, while $\hat{\nu}_{\mathcal{X}} \neq \hat{\nu}_{\mathcal{X}}^{-}$.

Subsequently, for each pixel triplet $\{i, i^+, i^-\}$ it is possible consider their embedding representation obtained through the encoder of the PSPNet, obtaining a set of triplet embeddings

$$\left\{\boldsymbol{i}, \boldsymbol{i}^+, \boldsymbol{i}^-\right\},$$

where $\boldsymbol{i}, \boldsymbol{i}^+, \boldsymbol{i}^- \in \mathbb{R}^{C_{\text{enc}}}$. A projection function is then applied:

$$f_{\text{proj}} : \mathbb{R}^{C_{\text{enc}}} \to \mathbb{R}^d, \qquad f_{\text{proj}}(\mathbf{i}) = W_2\, \sigma(W_1 \mathbf{i} + \mathbf{b}_1) + \mathbf{b}_2,$$

yielding the projected pixel triplet embeddings $\left\{\boldsymbol{i}, \boldsymbol{i}^+, \boldsymbol{i}^-\right\}$ indicated as before with an abuse of notation.

Here $W_1 \in \mathbb{R}^{C' \times C_{\text{enc}}}, W_2 \in \mathbb{R}^{d \times C_{\text{enc}}}$ are two 1×1 convolutional later, $\mathbf{b}_1, \mathbf{b}_2$ are bias terms, and $\sigma(\cdot)$ is ReLu activation function. In this thesis, we set $d = 256$.

The expression of the triplet loss on the set of projected triplet embeddings $\left\{\boldsymbol{i}, \boldsymbol{i}^+, \boldsymbol{i}^-\right\}$ asssumes the following form:

$$\mathcal{L}^{TT}\left(\boldsymbol{i}, \boldsymbol{i}^+, \boldsymbol{i}^-\right) = \max\{\langle \boldsymbol{i}, \boldsymbol{i}^+ \rangle - \langle \boldsymbol{i}, \boldsymbol{i}^- \rangle + m, 0\}, \tag{3.4}$$

where m is called separator margin.

At this stage, we can highlight the second key difference with respect to the standard triplet loss. In the conventional formulation, the margin m is fixed for the entire set of input triplets. Conversely, in $\mathcal{L}^{TT}$ it is defined as

$$m = m_\epsilon + \frac{1}{2} m_{\mathcal{T}},$$

where:

$$m_\epsilon = 0.1 \quad \text{and} \quad m_{\mathcal{T}} = \frac{\psi\left(\hat{\nu}_{\mathcal{X}}, \hat{\nu}_{\mathcal{X}}^{-}\right) - \psi\left(\hat{\nu}_{\mathcal{X}}, \hat{\nu}_{\mathcal{X}}^{+}\right)}{2D_{tree}}.$$

The margin is strictly dependent on the distance between the ground-truth leaf labels in the hierarchical tree, i.e. it depends on their degree of semantic similarity or dissimilarity.

The distance $\langle \cdot, \cdot \rangle$ employed in the tree-triplet loss (3.4) is a function designed to measure the similarity between embeddings. Its formulation is given by:

Definition 13 (Cosine Similarity and Cosine Distance) *Let $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$ be two non-zero vectors. The* cosine similarity *between $\boldsymbol{x}$ and $\boldsymbol{y}$ is defined as*

$$\textit{cosine similiarity} = S_C(\boldsymbol{x}, \boldsymbol{y}) = \frac{\boldsymbol{x}^\top \boldsymbol{y}}{\|\boldsymbol{x}\|_2 \, \|\boldsymbol{y}\|_2},$$

where $\boldsymbol{x}^\top \boldsymbol{y}$ *denotes the Euclidean inner product and* $\|\cdot\|_2$ *is the* ℓ_2*-norm. The* cosine distance *is then defined as*

$$d_{\cos}(\boldsymbol{x}, \boldsymbol{y}) = 1 - S_C(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{2}\left(1 - \frac{\boldsymbol{x}^\top \boldsymbol{y}}{\|\boldsymbol{x}\|_2 \|\boldsymbol{y}\|_2}\right).$$

A visual representation of the pixel embeddings space after the tree triplet loss optimization is shown in Figure 3.9.

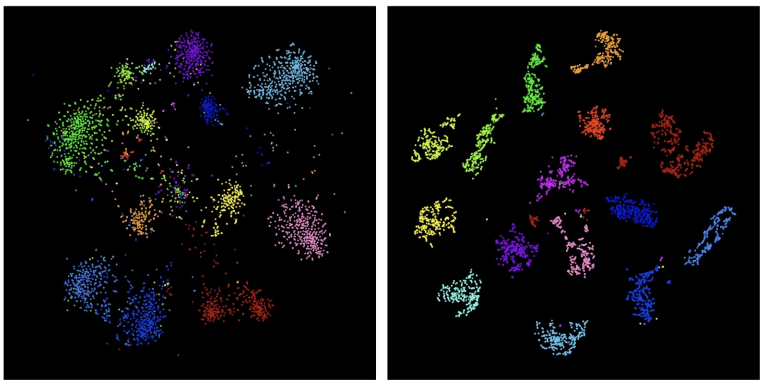


Figure 3.9: Visual representation of the pixel embedding space after the application of the tree-triplet loss $\mathcal{L}^{\mathrm{TT}}$.

To conclude, the loss function to be minimized during the learning phase takes the following form:

$$\mathcal{L} = \mathcal{L}^{\mathrm{FTM}} + \beta_{\mathrm{epoch}} \mathcal{L}^{\mathrm{TT}}, \tag{3.5}$$

where $\beta_{\mathrm{epoch}} \in [0, 0.5]$ is a parameter that decays through a cosine annealing behaviour [13] as follows:

$$\beta_{\mathrm{epoch}} = \beta_{\min} + \frac{1}{2}\left(\beta_{\max} - \beta_{\min}\right)\left(1 - \cos\left(\frac{\pi \cdot epoch}{max\ epochs}\right)\right)$$

where $\beta_{\min} = 0$, $\beta_{\max} = 0.5$, *epoch* is the current epoch of the training pipeline, and *max epochs* is the maximum number of epochs initially setted for the training procedure.

At this stage, thanks to the proposed formulation of the HSSN loss function (3.5), the hierarchical constraints are explicitly enforced during training. It is possible to introduce the classification phase. Let:

$$\mathcal{P} = \left\{v_1, \ldots, v_{|\mathcal{P}|}\right\}$$

a feasible root-to-leaf path such that $v_1 \in \mathcal{V}_{\mathcal{X}}$, $v_{|\mathcal{P}|} = v^r$ root node, and $\forall v_p, v_{p+1} \in \mathcal{P}$ we have $(v_p, v_{p+1}) \in \mathcal{E}$. For each pixel i the classification assumes the following form:

$$\left\{v_1^*, \ldots, v_{|\mathcal{P}|}^*\right\} = \arg\max_{\mathcal{P} \subseteq \mathcal{T}} \sum_{v_p \in \mathcal{P}} p_{v_p, i}$$

this ensure a hierarchical coherence beteween the predictions and the hierarchical tree.

Chapter 4

Numerical results

In this chapter, we present the process for hyperparameter tuning among the three methods previously introduced (see Chapter 3) through cross-validation. This procedure allows us to obtain a robust model and to identify the best hyperparameter configuration for evaluating the model on the test set.

As mentioned in 2, the dataset is characterized by a strong class imbalance problem, particularly with regard to the endometrium class, which is the minority class in the entire dataset. For this reason, we introduce an α vector to manage the class imbalance.

The vector used to address class imbalance is not selected as the best among several candidates evaluated through cross-validation. Instead, it was defined by directly considering the overall percentage of pixels in the training set. The resulting vector is the following:

$$\boldsymbol{\alpha} = [0.05, 0.15, 0.12, 0.2, 0.2, 0.3]$$

where α_c denotes the weight assigned to class $c \in \mathcal{C}$, following the class index assignment shown Fig. 3.4.

Through this choice, it is evident that the value α_c of the endometrium class- the minority class in the training set, accounting for only 0.1% of the pixels- is six times larger than the α_c associated with the background class, which is the majority class of the dataset with more than 70% of the pixels.

In this way, the idea is to penalize prediction errors on the endometrium class more strongly than errors on the background class. Similarly, the same criterion has been adopted for the remaining classes, assigning smaller α_c values to classes characterized by a larger pixel distribution.

In addition to this weighted configuration, we also consider the case in which $\alpha_c = 1$ for all classes. In this setting, no class weighting is applied and the loss treats all classes equally.

We start analyzing the benchmark method, correspondent to the *flat* segmentation approach.

4.1 *Flat* Segmentation Cross-Validation

As mentioned in Section 3.1, the flat segmentation approach is based on a sparse categorical focal loss. The $\boldsymbol{\alpha}$ weighting vector used in the cross-validation procedure have been introduced above, from now on we are call it as $\boldsymbol{\alpha}$ weighted for pratical reasons. Regarding the parameter γ, see Section 3.1, the values considered are $\gamma = 0, 1.5$, and 3.5. Thus, we are considering the follwoing combinations:

- Case $\gamma = 0$ and $\boldsymbol{\alpha} = \mathbf{1}$: sparse categorical cross entropy;
- Case $\gamma = 0$ and $\boldsymbol{\alpha}$ weighted: sparse categorical cross entropy weighted on classes;
- Case $\gamma > 0$ and $\boldsymbol{\alpha}$ weighted: focal loss weighted on classes;
- Case $\gamma > 0$ and $\boldsymbol{\alpha} = \mathbf{1}$: focal loss not weighted.

We conduct a five-fold cross-validation by evaluating all combinations of α and γ hyperparameters in order to determine the optimal configuration for the *flat* segmentation task.

At this stage, data augmentation is not applied to the training dataset, in order to observe the model performance without the contribution of synthetically generated data. Similarly, no preprocessing is applied to either the training or the validation data, in order to evaluate the model performance on raw data.

The hyperparameters fixed at the beginning of the cross-validation pipeline are reported in Table 4.1.

Learning Rate	**Batch Size**	**Epochs**	**Data**	**Data Augmentation**
1×10^{-3}	32	150	standard	no

Table 4.1: Initial hyperparameter configuration used for the five-fold cross-validation in the *flat* segmentation approach. The experiments are performed on the standard dataset (without preprocessing), and data augmentation is disabled in order to evaluate the model performance on raw data.

The validation results obtained at the end of the five-fold cross-validation for each combination of α and γ are reported in the following tables in terms of mean IoU and mean F1 score.

γ	$\boldsymbol{\alpha}$	**Mean IoU**	**Mean F1**
0	**1**	**0.61698 ± 0.01190**	**0.74740 ± 0.00942**
0	weighted	0.60874 ± 0.02340	0.74124 ± 0.01880
1.5	**1**	0.61235 ± 0.02340	0.74191 ± 0.01450
1.5	weighted	**0.61604 ± 0.01870**	**0.74703 ± 0.01560**
3.5	**1**	0.59340 ± 0.22200	0.70932 ± 0.02710
3.5	weighted	**0.60482 ± 0.01523**	**0.73802 ± 0.01250**

Table 4.2: Validation performance obtained from five-fold cross-validation for all combinations of the focal loss hyperparameters γ and $\boldsymbol{\alpha}$ in the *flat* segmentation approach.

Among the configurations reported in the tables above, the three best ones correspond to the following pairs of hyperparameters:

$$(\gamma, \boldsymbol{\alpha}) = (0, \mathbf{1}), \quad (1.5, \boldsymbol{\alpha}_{\text{weighted}}), \quad (3.5, \boldsymbol{\alpha}_{\text{weighted}}) \tag{4.1}$$

At this stage, for this configuration, we report the mean training and validation F_1 score across the five folds cross-validation pipeline in Figure 4.1.

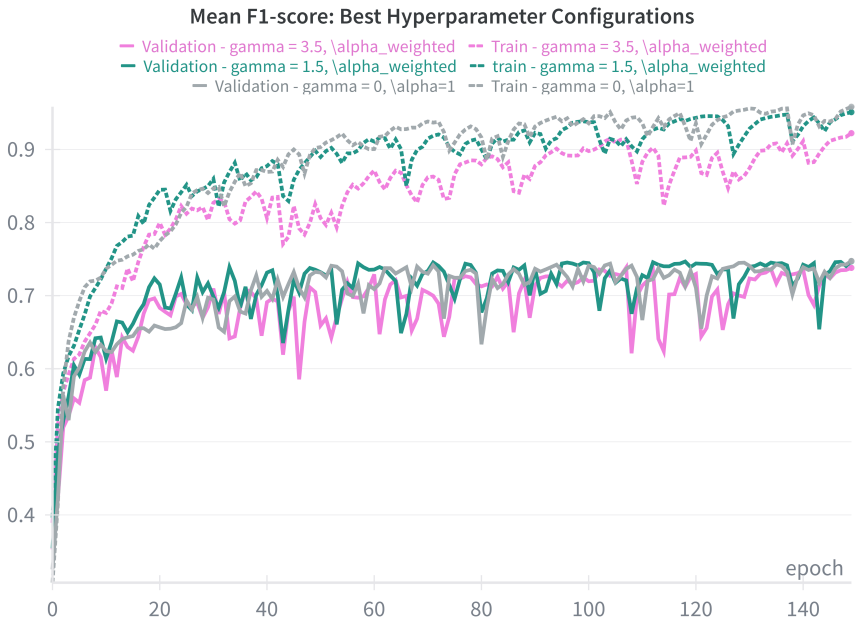


Figure 4.1: Mean training and validation F_1 score for the three best (γ, α) configurations (see Equation (4.1)) obtained from five-fold cross-validation, focal hierarchical loss approach. See Table 4.1 for the cross-validation set up.

We also report the mean training and validation IoU in Figure 4.2.

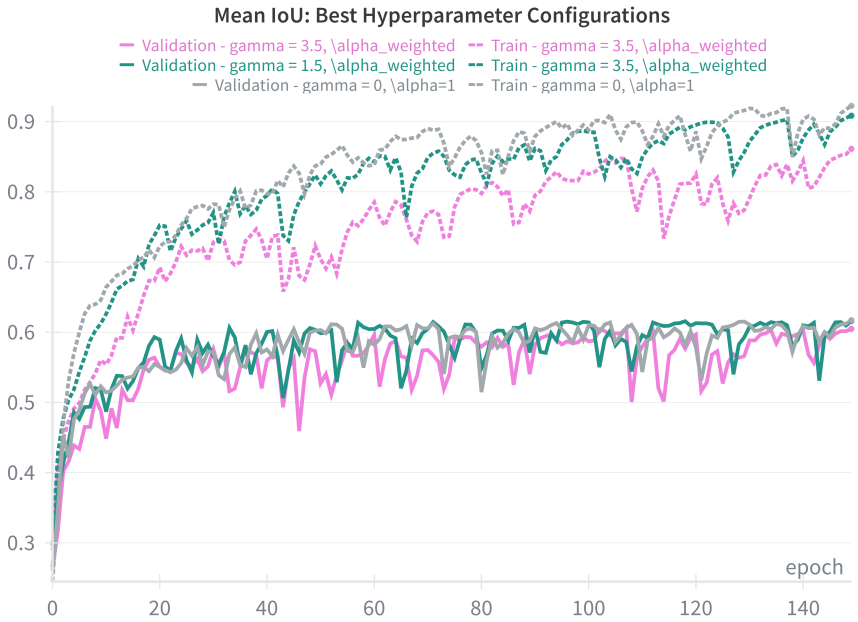


Figure 4.2: Mean training and validation IoU for the three best (γ, α) configurations (see Equation (4.1)) obtained from five-fold cross-validation, focal hierarchical loss approach. See Table 4.1 for the cross-validation set up.

According to the cross-validation results in Table 4.1, the best configuration corresponds to $(\gamma, \boldsymbol{\alpha}) = (0, \mathbf{1})$. However, we selected the configuration with $\gamma = 1.5$ and $\boldsymbol{\alpha} = [0.05, 0.15, 0.12, 0.2, 0.2, 0.3]$ due to its ability to weight classes, corresponding to the green configuration. The values reported at each epoch already represent the mean across the five folds.

At this point, once the optimal hyperparameter configuration has been identified, data augmentation is applied to the selected configuration. Two experimental settings are then considered in order to perform a fair comparison:

- Data augmentation applied to the standard dataset;
- Data augmentation applied to the preprocessed dataset.

This allows the evaluation of the impact of the preprocessing pipeline under the same augmentation conditions.

Furthermore, due to the introduction of data augmentation on both the standard and the preprocessed datasets, the batch size was increased to 48 to improve data handling, as the number of frames increases during cross-validation. At the same time, the learning rate was reduced to $2e-4$ and the number of epochs increased to 650 because of the slowness of the procedure. The table of the hyperparamters can be updated as follows:

Learning Rate	Batch Size	Epochs	γ	$\boldsymbol{\alpha}$	Data	Data Augmentation
2×10^{-4}	48	650	1.5	weighted	standard / preprocessed	yes

Table 4.3: Five-fold configuration adopted after selecting the optimal hyperparameters, *flat* segmentation approach. Data augmentation is enabled and the same configuration is applied to both the standard and the preprocessed datasets in order to evaluate the effect of the preprocessing.

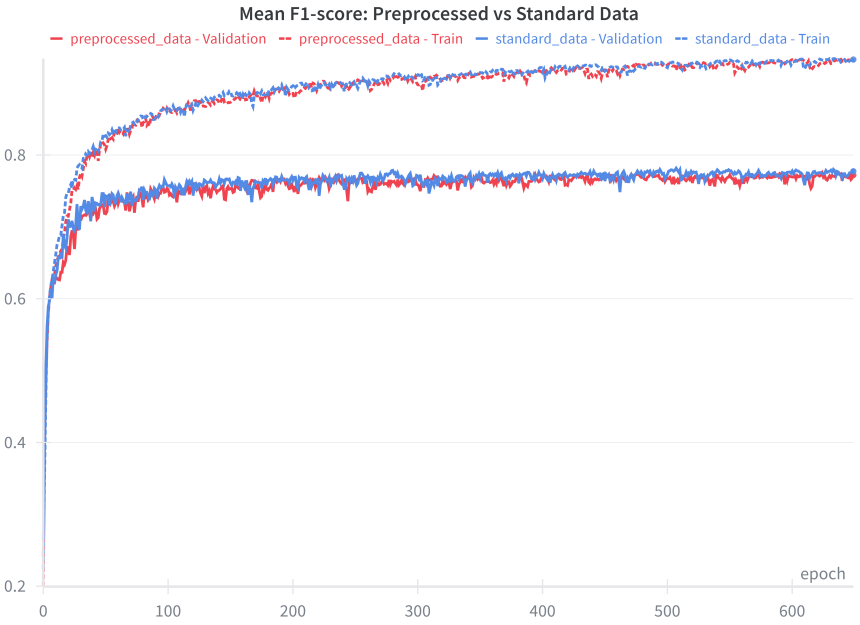


Figure 4.3: Mean training and validation F_1 score computed over the five folds of the cross-validation, *flat* segmentation approach.
Cross-validation setup: learning rate 2×10^{-4}, batch size 48, epochs 650, $\gamma = 1.5$, $\boldsymbol{\alpha}$ weighted, data augmentation enabled.

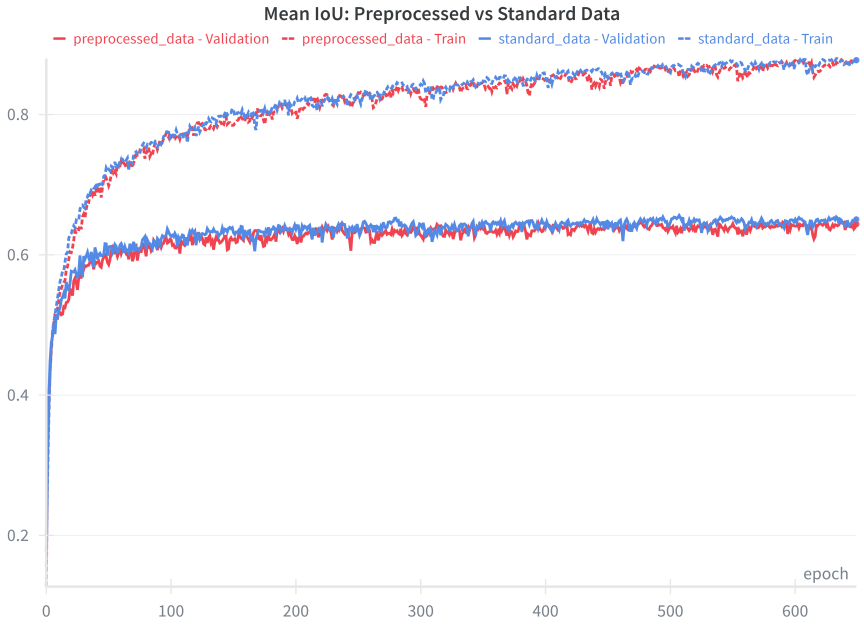


Figure 4.4: Mean training and validation IoU computed over the five folds of the cross-validation, *flat* segmentation approach.
Cross-validation setup: learning rate 2×10^{-4}, batch size 48, epochs 650, $\gamma = 1.5$, $\boldsymbol{\alpha}$ weighted, data augmentation enabled.

The Table 4.4 reports the final validation performance obtained through five-folds cross validation for the tested hyperparameter configuration. Data augmentation is applied in both settings, while the comparison evaluates the impact of the preprocessing pipeline.

Data	**Mean IoU**	**Mean F1**
standard	0.65043 ± 0.13120	0.77734 ± 0.09900
preprocessed	**0.64370 ± 0.00860**	**0.77213 ± 0.00762**

Table 4.4: Validation performance obtained from five-fold cross-validation using the selected hyperparameter configuration with data augmentation enabled, *flat* segmentation approach. The table compares results obtained using the standard dataset and the preprocessed dataset.

As can be observed, the table shows slightly better performance on the standard dataset, represented in red, compared to the preprocessed one, represented in violet. Nevertheless, the configuration based on the preprocessed dataset was selected as the final one. This choice is motivated by the improved data quality obtained through preprocessing, which produces images with higher contrast and reduced noise, as explained in Section 2.2. The optimal hyperparameter configuration obtained through the five-fold cross-validation for the focal hierarchical loss follows in Table 4.5

Learning Rate	Batch Size	Epochs	γ	$\boldsymbol{\alpha}$	Data	Data Augmentation
2×10^{-4}	48	650	1.5	weighted	preprocessed	yes

Table 4.5: Final hyperparameter configuration adopted for the *flat* segmentation approach after cross-validation and dataset comparison.

From the two graphs above, an overfitting behavior can be observed starting from epoch 400. For this reason, the model was retrained on the combined training and validation dataset for only 400 epochs, corresponding to the point at which the best validation results are achieved before the onset of overfitting. The model weights obtained at this stage were then saved and used for the testing phase.

4.2 Focal hierarchical Loss Cross-Validation

The cross-validation procedure for the hierarchical method equipped with the focal hierarchical loss is analogous to that performed for the *flat* segmentation approach. Indeed, as explained in Subsection 3.2.1, the focal hierarchical loss can be interpreted as a sum of focal losses applied at the different levels of the generated hierarchical tree.

The choice of $\boldsymbol{\alpha}$, the class weighting factor, was set at the beginning, while the values for the parameter γ are the same used for the previous *flat* approach.

The validation results obtained at the end of the five-fold cross-validation for each combination of $\boldsymbol{\alpha}$ and γ are reported in the following tables in terms of mean IoU and mean F1 score.

γ	$\boldsymbol{\alpha}$	Mean IoU	Mean F1
0	**1**	0.60696 ± 0.04600	0.73578 ± 0.04400
0	weighted	**0.61219 ± 0.02150**	**0.73375 ± 0.18300**
1.5	**1**	0.59785 ± 0.01760	0.72760 ± 0.16600
1.5	weighted	**0.61302 ± 0.01690**	**0.74478 ± 0.01410**
3.5	**1**	0.52525 ± 0.10500	0.64067 ± 0.11700
3.5	weighted	**0.56050 ± 0.07230**	**0.69395 ± 0.07230**

Table 4.6: Validation performance obtained from five-fold cross-validation for all combinations of the hyperparameters γ and $\boldsymbol{\alpha}$ for the focal hierarchical loss.

Among the configurations reported in the tables above, the three best ones correspond to the following pairs of hyperparameters:

$$(\gamma, \alpha) \quad = \quad (0, \boldsymbol{\alpha}_{\text{weighted}}), \quad (1.5, \boldsymbol{\alpha}_{\text{weighted}}), \quad (3.5, \boldsymbol{\alpha}_{\text{weighted}})$$

As done for the *flat* segmentation approach, we report the mean training and validation F_1 score during the cross-validation pipeline in Figure 4.5

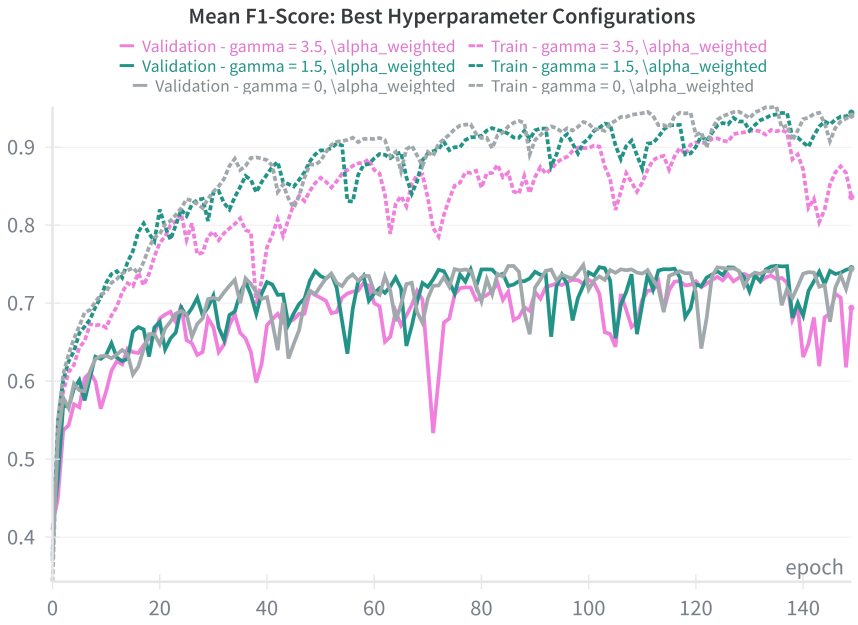


Figure 4.5: Mean training and validation F_1 score for the three best (γ, α) configurations (see Equation (4.2)) obtained from five-fold cross-validation, focal hierarchical loss approach. See Table 4.1 for the cross-validation set up

We also report the mean training and validation IoU in Figure 4.6

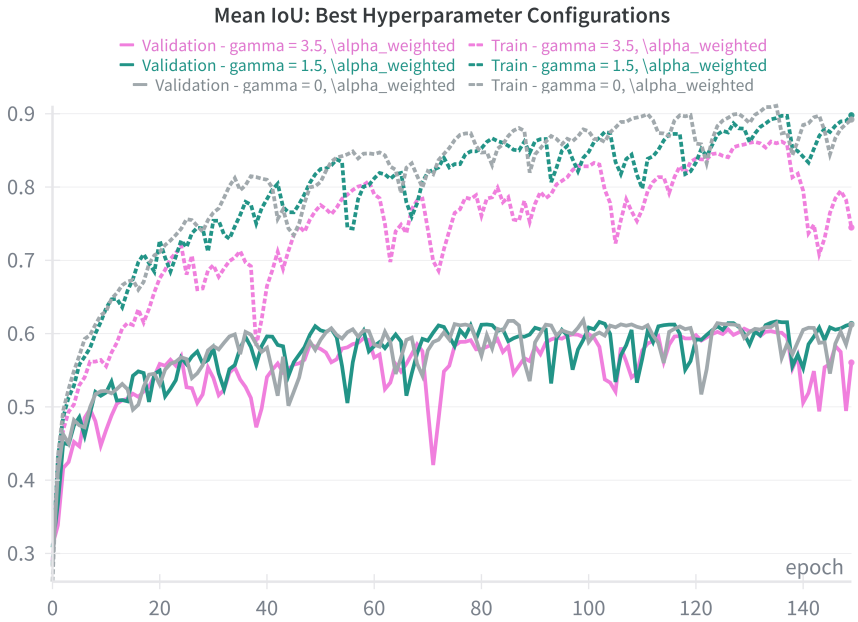


Figure 4.6: Mean training and validation IoU for the three best (γ, α) configurations (see Equation (4.2)) obtained from five-fold cross-validation, focal hierarchical loss approach. See Table 4.1 for the cross-validation set up.

The optimal configuration is represented by the green one, correspondent to $\gamma = 1.5$ and $\boldsymbol{\alpha} = [0.05, 0.15, 0.12, 0.2, 0.2, 0.3]$.

Proceeding as in the *flat* segmentation approach, we apply data augmentation, increase the number of epochs to 650, and decrease the learning rate.

The Table of the hyperparameters follows:

Learning Rate	Batch Size	Epochs	γ	$\boldsymbol{\alpha}$	Data	Data Augmentation
2×10^{-4}	48	650	1.5	weighted	standard / preprocessed	yes

Table 4.7: Five-fold cross validation configuration adopted after selecting the optimal hyperparameters, focal hierarchical loss casee. Data augmentation is enabled and the same configuration is applied to both the standard and the preprocessed datasets in order to evaluate the effect of the preprocessing.

We report the mean training and validation F_1 score and IoU across the five folds of the cross-validation for the standard and preprocessed datasets in Figures 4.7 and 4.8, respectively.

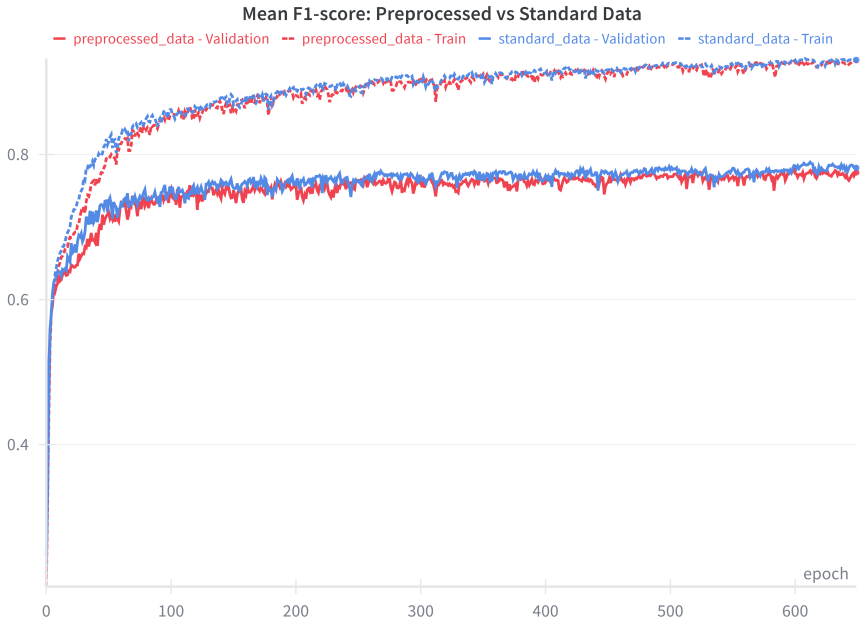


Figure 4.7: Mean training and validation F_1 score computed over the five folds of the cross-validation for the focal hierarchical loss approach.
Cross validation setup: 2×10^{-4}, batch size: 48, epochs: 650, $\gamma = 1.5$, $\boldsymbol{\alpha}$ weighted, data augmentation enabled.

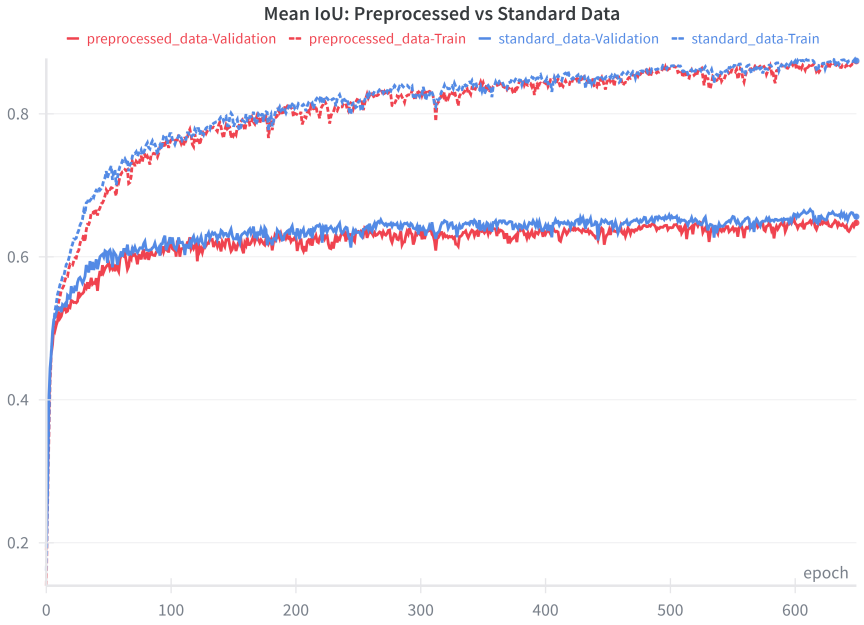


Figure 4.8: Mean training and validation IoU computed over the five folds of the cross-validation for the focal hierarchical loss approach.
Cross validation setup: 2×10^{-4}, batch size: 48, epochs: 650, $\gamma = 1.5$, $\boldsymbol{\alpha}$ weighted, data augmentation enabled.

Similarly to the flat segmentation approach, we report the table containing the mean IoU and F1 score values on the validation set for both the standard and the preprocessed datasets at the last epoch, equal to 650.

Data	Mean IoU	Mean F1
standard	0.65621 ± 0.01550	0.78157 ± 0.0137
preprocessed	**0.64755 ± 0.00896**	**0.77527 ± 0.00832**

Table 4.8: Validation performance obtained from five-fold cross-validation using the selected hyperparameter configuration in Table 4.7 with data augmentation enabled, focal hierarchical loss configuration. The table compares the standard dataset and the preprocessed dataset, both with data augmentation enabled.

Analogously to the previous case, the configuration obtained using the preprocessed data is selected as the best one.

The optimal hyperparameter configuration obtained through the five-fold cross-validation for the focal hierarchical loss follows in Table 4.9

Learning Rate	Batch Size	Epochs	γ	$\boldsymbol{\alpha}$	Data	Data Augmentation
2×10^{-4}	48	650	1.5	weighted	preprocessed	yes

Table 4.9: Final hyperparameter configuration adopted for the focal hierarchical loss after cross-validation and dataset comparison. The model is trained on the preprocessed dataset with data augmentation enabled.

Subsequently the model is trained again on the entire training dataset, obtained merging the training and validation split of a fold used for the cross-validation of the model. The model is then trained for 400 epochs and the resulting configuration is used for the final evaluation on the test set.

4.3 HSSN Cross-Validation

As explained in Subsection 3.2.2, the second hierarchical method is composed of a loss defined as the sum of two terms:

- **Focal Tree-Min Loss**, which is a hierarchy-aware focal loss equipped with a focal modulation for both positive and negative predictions;
- **Tree Triplet Loss**, a loss defined in the embedding space that encourages embeddings sharing semantic similarities to cluster together while pushing apart semantically distant ones.

The overall loss function is defined as follows:

$$\mathcal{L} = \mathcal{L}^{FTM} + \beta \mathcal{L}^{TT}.$$

After this brief recap of the HSSN loss formulation, we can identify the parameters on which the five-fold cross-validation procedure will be performed.

Since the Focal Tree-Min Loss represents a hierarchical formulation of the focal loss, the focusing parameter γ must be adjusted accordingly. In the previous experiments, a value of $\gamma = 1.5$ was adopted. However, in the Focal Tree-Min formulation the focal modulation is applied to both positive and negative terms of the loss. For this reason, using the same value would excessively amplify the focusing effect. Therefore, a smaller value $\gamma = 0.7$ is adopted to balance the penalization of positive and negative predictions.

Furthermore, since HSSN strongly relies on the semantic relationships between classes, the class-balancing parameter is set to $\boldsymbol{\alpha} = [0.05, 0.15, 0.12, 0.2, 0.2, 0.6]$. In particular, the weight associated with the *Endometrium* class is increased to 0.6 (instead of 0.3 used in previous settings). This choice was motivated by the fact that the penalization introduced by the Focal Tree-Min Loss prevented the model from properly learning this class, which represents the most underrepresented category in the dataset. Increasing its weight allows the loss function to place stronger emphasis on this class, improving its validation performance.

The Tree-Triplet Loss relies on the random sampling of embedding triplets $\{\mathbf{i}, \mathbf{i}^+, \mathbf{i}^-\}$, where the number of generated triplets is controlled by the hyperparameter $n_{triplets}$. At this stage we perform a five fold cross validation on the hyperparamter number of triplets, in particular we analyze the computational time needed by the model on each fold with different values of $n_{triplets}$. In particular, we explore three cases:

$$n_{triplets} = 50, \quad 100, \quad 300 \tag{4.2}$$

The table with the fixed hyperparameters for the cross valdiation pipeline follows:

Learning Rate	**Batch Size**	**Epochs**	γ	α	**Data**	**Data Augmentation**
8×10^{-4}	32	250	0.7	weighted	**standard**	no

Table 4.10: Initial hyperparameter configuration used for the five -fold cross-validation for the HSSN framework. The expriemtns are performed on the standard dataset without preprocessing, and data augmentation is disabled.

We report the mean training and validation F_1 score across the cross-validation pipeline in Figure 4.9.

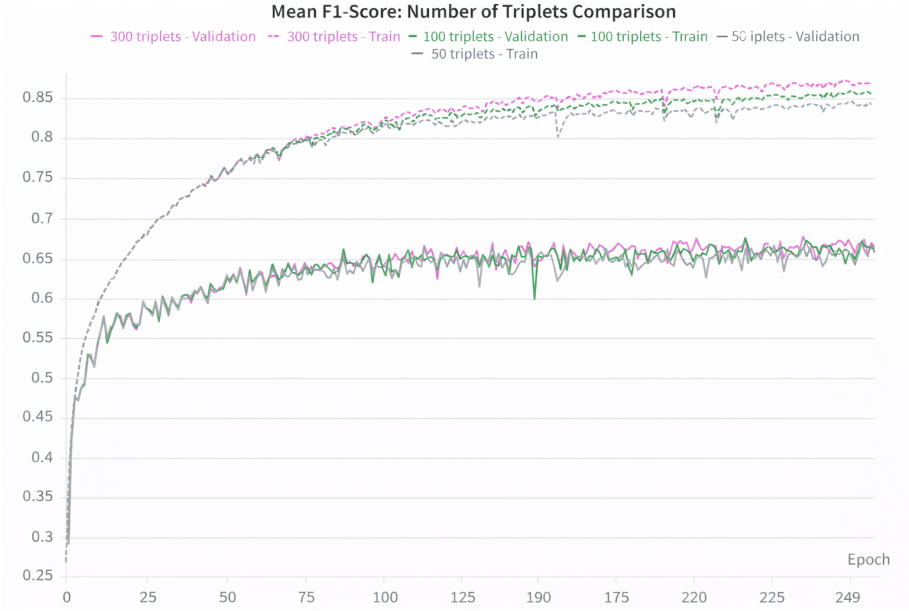


Figure 4.9: Mean Training and validation F_1 score for the number of triplets comparison (see Equation (4.2)) obtained from five-fold cross-validation, HSSN framework. See Table 4.10 for the cross-validation setup.

We also report the mean training and validation IoU in Figure 4.10.

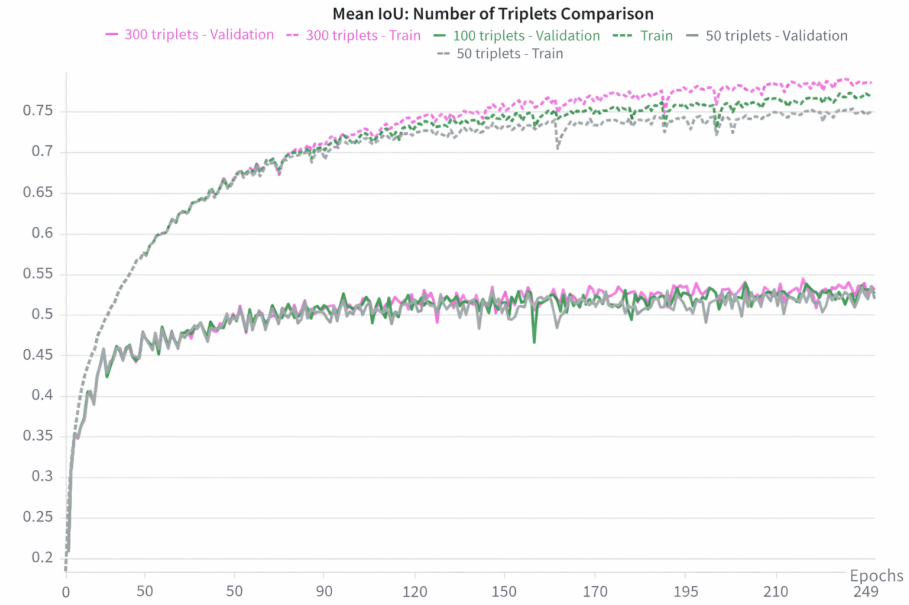


Figure 4.10: Mean Training and validation IoU for the number of triplets comparison (see Equation (4.2)) obtained from five-fold cross-validation, HSSN framework. See Table 4.10 for the cross-validation setup

The results in terms of computational time follows in Table 4.11.

Time	**50 triplets**	**100 triplets**	**300 triplets**
Computational time	2h	**2h 30min**	3h 30min

Table 4.11: Estimated computational time for different values of the triplet-number hyperparameter across the five-folds of the cross-validation pipeline, HSSN framework. The computational time reported are the mean on the five-folds.

The validation results obtained at the end of the five-fold cross-validation for $n_{triplets}$ are reported in Table 4.12 in terms of mean IoU and mean F1 score.

Triplets	**Mean IoU**	**Mean F1**
50	0.51903 ± 0.02100	0.65754 ± 0.01910
100	**0.52593 ± 0.01600**	**0.66323 ± 0.01760**
300	0.52997 ± 0.01460	0.66644 ± 0.01560

Table 4.12: Validation performance obtained from five-fold cross-validation for the different number of sampled triplets, HSSN framework.

Validation performance obtained using five-fold cross-validation for different numbers of training triplets in the HSSN framework.

From the following tables, we can conclude that the best configuration corresponds to n_{triplets} equal to 100, considering both the validation performance and the computational time. The configuration with $n_{\text{triplets}} = 50$ is discarded since, although the validation results are similar to those obtained with $n_{\text{triplets}} = 100$, the latter allows the generation of twice as many triplets with only about 30 additional minutes of computational time. On the other hand, the configuration with $n_{\text{triplets}} = 300$ is considered too computationally expensive.

However, it can be observed that the validation values are lower compared to those obtained with the previous models, resulting in poor performance on validation set, while requiring a relatively high training time of approximately 12 hours for the full cross-validation process.

For this reason, preprocessing and data augmentation are introduced to improve the validation performance. In addition, the number of training epochs is increased to 650 and the batch size to 48. Due to the high computational cost of the cross-validation procedure, the experiments are conducted only on the preprocessed dataset. The table of hyperparamters follows:

Learning Rate	Batch Size	Epochs	N. Triplets	γ	α	Data	Data Augmentation
8×10^{-4}	48	650	100	0.7	weighted	preprocessed	yes

Table 4.13: Five-fold cross validation configuration adopted after selecting the optimal hyperparameters for the HSSN framework. Data augmentation is enabled, and the configuration is applied only to the preprocessed dataset due to the high computational cost (training time exceeding two days).

The training and validation performance obtained using the preprocessed dataset are reported below. Results are presented in terms of mean F_1 score and IoU, computed via five-fold cross-validation.

Figure 4.11: Mean training and validation F_1 score computed over the five folds of the cross validation for the HSSN framework.
Cross validation setup: learning rate: 8×10^{-4}, batchsize: 48, epochs: 650, $\gamma = 1.5$, $\boldsymbol{\alpha}$ weighted, data augmentation enabled.

Figure 4.12: Mean training and validation IoU computed over the five folds of the cross validation for the HSSN framework.
Cross validation setup: learning rate: 8×10^{-4}, batchsize: 48, epochs: 650, $\gamma = 1.5$, $\boldsymbol{\alpha}$ weighted, data augmentation enabled.

The model checkpoints were saved after the cross-validation procedure, and training was resumed for an additional 450 epochs with a reduced learning rate of 3×10^{-4} in order to further improve validation performance, while keeping the other hyperparameters unchanged. The resumed training configuration follows in Table 4.14.

Phase	Learning Rate	Batch Size	Epochs	N. Triplets	γ	α	Data	Data Augmentation
resumed training	3×10^{-4}	48	450	100	0.7	weighted	preprocessed	yes

Table 4.14: Five-fold cross validation configuration adopted after the first 650 epochs for the HSSN framework. Data augmentation is enabled and the configuration is applied only to the preprocessed dataset.

Figure 4.13: Mean training and validation F_1 score computed over the five folds of the cross validation for the HSSN framework.
Cross validation setup: learning rate: 3×10^{-4}, batchsize: 48, epochs: 650-1100, γ = 1.5, $\boldsymbol{\alpha}$ weighted, data augmentation enabled.

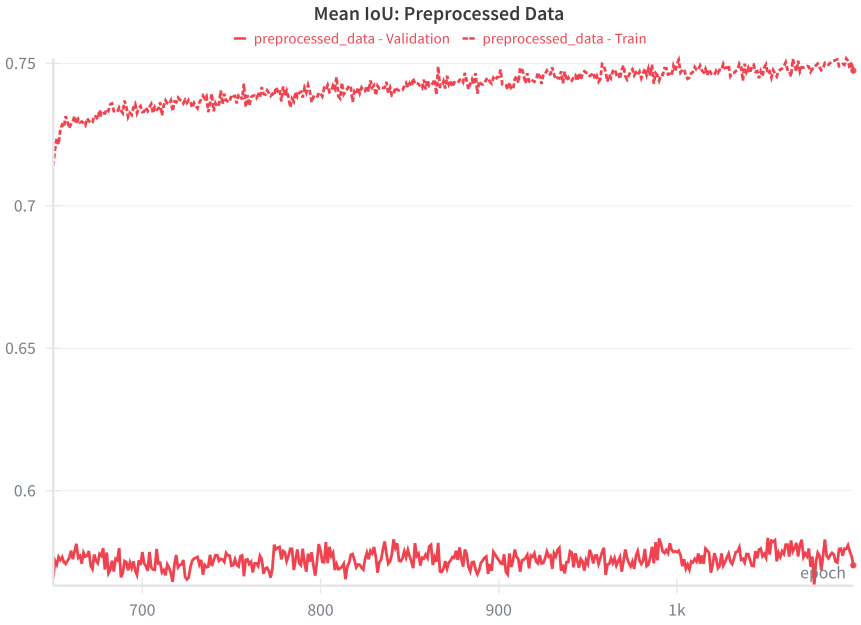


Figure 4.14: Mean training and validation IoU computed over the five folds of the cross validation for the HSSN framework.
Cross validation setup: learning rate: 3×10^{-4}, batchsize: 48, epochs: 650-1100, γ = 1.5, $\boldsymbol{\alpha}$ weighted, data augmentation enabled.

Data	Mean IoU	Mean F1
preprocessed	0.57383 ± 0.01320	0.71190 ± 0.01180

Table 4.15: Validation mean IoU and F_1 score over training epochs, obtained from five-fold cross-validation. Training was performed in two phases: the first 650 epochs using the hyperparameters in Table 4.13, followed by a resumed training phase as in Table 4.14 with data augmentation enabled. Results are reported for the preprocessed dataset only.

Thus, the optimal hyperparameter configuration obtained through the five-fold cross validation for the HSSN framework follows in Figure 4.16

Phase	Learning Rate	Batch Size	Epochs	γ	α	Data	Data Augmentation
Training	8×10^{-4}	48	650	0.7	weighted	preprocessed	yes
resumed training	3×10^{-4}		450				

Table 4.16: Final hyperparameter configuration adopted for the HSSN framework after cross-validation and dataset comparison. The model is cross-validated on preprocessed dataset with data augmentation enabled.

Subsequently, the model is then retrained on the entire training dataset for a total of 1050 epochs using the selected configuration.

4.4 Evaluation of Segmentation Performance

In this section, we want to present the evaluation results of the segmentation task. The aim is showing better performances, in terms of metrics and generalization of the model, of the hierarchical semantic segmentation approach with respect to the *flat* segmentation approach, also known as benchmark approach.

As introduced in Section 1.2.3, the final evaluation of the model is applied on the test set, yet untouched and used to simulate the hypotetical real-word situation in which we have to use the model on the unseen data.

4.4.1 Class-wise Evaluation Performance

We begin by presenting the class-wise evaluation metrics. Our aim is to identify the most effective hierarchical approach for the segmentation task by analyzing the performance for each class, with particular attention to the healthy ovary and uterus class. The achieved performances are reported in the following histograms.

F1-score

As shown in Figure 4.15, the segmentation method characterized by the focal hierarchical loss (H Loss) achieves better (or equal) F1 score (i.e. the harmonic

mean between recall and precision) on each class compared to the benchmark approach and the HSSN.

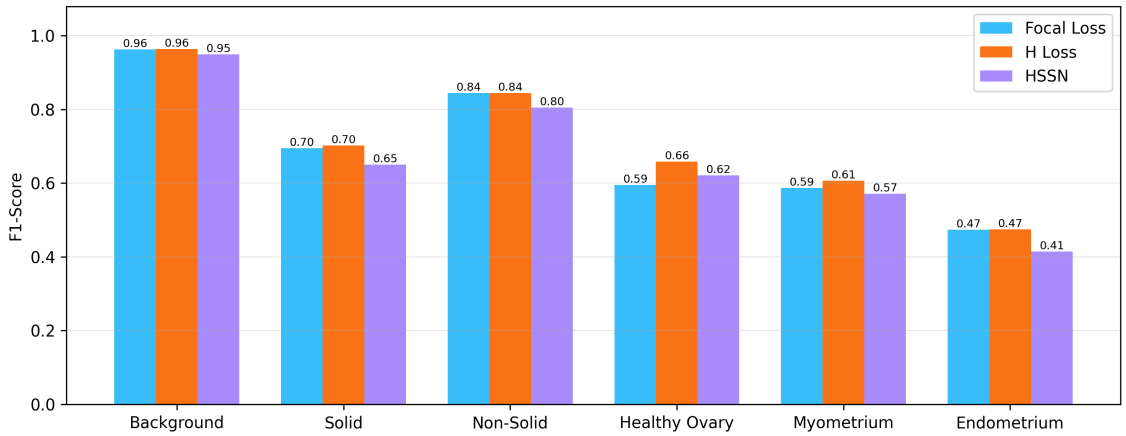


Figure 4.15: Class-wise F1-score comparison among the three adopted methods.

Intersection Over Union (IoU)

As shown in Figure 4.16, the considerations regarding the IoU are similar to those previously made for the F1-score. The hierarchical method with hierarchical focal loss achieves better (or equal) results compared to the other two methods for each class.

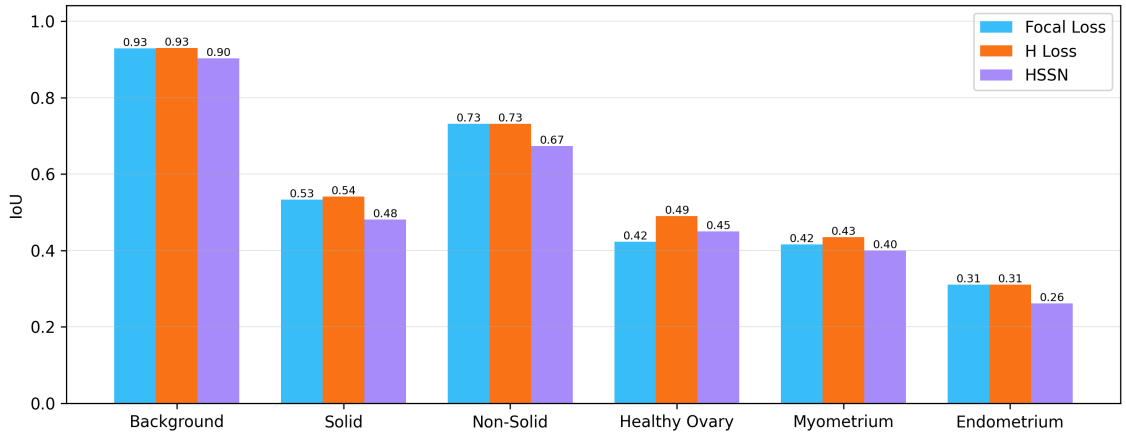


Figure 4.16: Class-wise IoU comparison among the threee adopted methods.

Focal loss and focal hierarchical loss (H Loss) achieve the same results in terms of F1 score and IoU for the endometrium class, which, as previously mentioned, is the minority class in terms of pixel count. However, the focal hierarchical loss shows better performance on both the healthy ovary and myometrium classes. Therefore, we can conclude that the focal hierarchical loss represents the better method overall.

Recall and Precision

The class-wise representation of recall and precision is presented using a different visualization strategy. Instead of adopting a histogram representation, the

results are shown through *true confusion matrix*, and *pred. confusion matrix*, see Section 1.4.

These representations allows both the class-wise recall and precision of the model to be easily analyzed. Moreover, it provides additional insight into the distribution of errors, highlighting the percentage of class-wise false negatives as well as misclassified predictions, which correspond to false positives for the predicted classes.

The *true confusion matrix*, whose main diagonal represents the model's recall, follows in Figure 4.17.

Figure 4.17: True confusion matrices obtained from the evaluation on the test set.

In all the evaluated methods, a strong presence of false negatives associated with the background class can be observed in the recall of the remaining classes. In particular, a significant portion of pixels belonging to anatomical structures or tumoral masses are misclassified as background. This behaviour is expected given the intrinsic characteristics of ultrasound (US) images, characterized by noise.

Regarding the healthy ovary class, the HSSN method achieves the lowest recall, equal to 69%. In this case, a notable portion of pixels is misclassified as non-solid tumoral masses, equal to 15%, while only 5% are misclassified as solid tumoral masses.

Considering instead the focal hierarchical loss, of all the healthy ovary pixels, 11% are wrongly predicted as non-solid tumoral masses, while the proportion of pixels misclassified as solid tumoral masses is approximately equal to that observed for the HSSN method.

The results for the *flat* segmentation approach are similar to the ones of the focal hierarchical loss.

The recall for the endometrium is the lowest across all three methods, which is consistent with its limited pixel representation in the dataset. The endometrium is mainly confused with the myometrium class, which is reasonable since both structures are parts of the uterus. Furthermore, across all three methods, approximately 12–15% of the pixels are detected as solid tumoral masses, representing additional false negatives for the endometrium class.

The *pred. confusion matrix*, whose main diagonal represents the model's precision, follows in Figure 4.17.

Figure 4.18: Pred. confusion matrices obtained from the evalaution on the test set.

Similarly to what was observed for the recall, a large proportion of predictions for several classes correspond to pixels that actually belong to the background class, indicating a significant amount of background misclassification.

The HSSN method achieves the lowest prediction performance among the two hierarchical approaches. Regarding the healthy ovary class, it is correctly predicted in 57% of the cases, while among pixels predicted as healthy ovary 5% are actually solid and 5% are actually non-solid.

Considering instead the first hierarchical method, a stronger presence of false positives predicted as solid tumor masses can be observed, accounting for 11%. In contrast, the false positives predicted as non-solid masses are approximately similar to those obtained with the HSSN method.

The flat segmentation approach is the method with the highest number of false positives for the healthy ovary class. Among pixels predicted as healthy ovary, 16% are actually solid and 10% are actually non-solid.

The uterus classes, endometrium and myometrium, are well predicted by the

Focal Hierarchical Loss. In particular, for the endometrium class, most of the pixels predicted as endometrium are correctly classified. Excluding the background class, the only relevant false positives correspond to the myometrium class, which is coherent with the strong anatomical similarity between these tissues and with the relatively small pixel distribution of the endometrium class.

Similar results are obtained for the HSSN method, although the precision of the endometrium class is lower compared to the focal hierarchical loss.

To conclude, we present the same confusion matrices on the balanced test set. As introduced above, the balanced test set is characterized by a limited number of frames for each item, indeed it has been applied a max numerosity frame selection equal to five. The usage of the balanced test set ensures the effective performances of a given model: there is no frames imbalance in the dataset. This prevents specific clinical cases from dominating the evaluation, ensuring that the confusion matrices reflect the model's performance across different cases rather than across highly similar frames from the same video.

Figure 4.19: True confusion matrices obtained from the evaluation on the balanced test set.

Figure 4.20: Pred. confusion matrices obtained from the evaluation on the balanced test set.

The estimation of the recall and precision is more rubust than the one obtained from the evaluation on the test set.

4.4.2 Global Evaluation

Now, we can introduce the table reporting the evaluation metric results as a mean among all the classes. In this way, we can compare the two hierarchical methods on global evalaution metrics and determine which of them performs better for our segmentation task. The results follows in Table 4.17.

Method	F1-score	IoU	Recall	Precision
Flat segmentation	0.6931	0.5570	0.7100	0.6850
Focal Hierarchical Loss	0.7081	0.5730	0.6950	0.7383
HSSN	0.6682	0.5280	0.6850	0.6630

Table 4.17: Metrics evaluation over all classes of the dataset

The HSSN has the lowest values in terms of metric evaluation. The F1 score, defined as the harmonic mean between precision and recall, assumes a value of 0.6682, which is lower than the F1 score obtained by the hierarchical approach characterized by the focal hierarchical loss, equal to 0.7081. In particular, in our case the HSSN performs worse both in terms of recall and precision compared to the focal hierarchical loss.

To complete the section concerning the segmentation results, we present two tables, in which we compare the benchmark method, corresponding to a *flat* segmentation approach, with the hierarchical method characterized by the focal hierarchical loss. The comparison is performed in terms of class-wise metrics, namely recall and precision.

In particular, Table 4.18 reports the recall values obtained for each class, highlighting possible improvements or deteriorations introduced by the hierarchical approach with respect to the *flat* method in detecting the different anatomical structures.

Class	Flat Segmentation recall	Focal Hierarchical Loss recall	Δ Recall
Background	96%	96%	—
Solid mass	70%	**73%**	↑ 3%
Non-solid mass	85%	85%	—
Healthy ovary	74%	**73%**	↓ 1%
Myometrium	56%	**51%**	↓ 5%
Endometrium	45%	**39%**	↓ 6%

Table 4.18: Class-wise recall comparison between Flat Segmentation and Focal Hierarchical Loss.

The hierarchical approach shows a better behaviour in detecting the effective solid masses compared to the *flat* approach, gaining a recall improvement of 3%. However, it exhibits a slight decrease in the identification of healthy ovaries, with a reduction of 1% in recall. Regarding the uterus components, the flat approach achieve better performances in detecting these anatomical parts, the hierarchical

approach loses 5% and 6% percentage points for myometrium and endometrium, respectively.

In Table 4.19, it is possible to observe the class-wise precision values obtained by the two approaches. The results highlight the contribution of the hierarchical strategy to the semantic segmentation task in terms of prediction reliability.

Class	Flat Segmentation precision	Focal Hierarchical Loss precision	Δ Precision
Background	97%	97%	—
Solid mass	69%	**67%**	↓ 2%
Non-solid mass	84%	84%	—
Healthy ovary	50%	**60%**	↑ 10%
Myometrium	61%	**74%**	↑ 13%
Endometrium	50%	**61%**	↑ 11%

Table 4.19: Class-wise precision comparison between Focal Loss and Focal Hierarchical Loss.

The usage of hierarchy between semantic classes has led to an improvement of 10% in model's precision for the healthy ovary class. This means that the model's prediction on this class is more accurate, suggesting a higher level of correctness. However, the detection of the healthy ovary class is slightly reduced due to a minor decrease in recall. Similar improvements are observed for the uterus components. In particular, myometrium and endometrium gain 13% and 11% percentage points in precision, respectively, compared to the *flat* approach.

Conversely, for the solid mass class, the hierarchical approach shows a slight decrease in precision, losing 2% percentage points with respect to the benchmark method.

Conclusions

In this thesis, we initially introduced the problem of image segmentation in general terms, in order to provide a clearer understanding of the topic. In particular, the need to address this problem using deep neural networks was highlighted, and a comprehensive explanation of their functioning was provided, presenting the main architectures employed in this work.

Subsequently, the objective of this thesis was defined, namely the segmentation of pelvic ultrasound images. In particular, the segmentation of the uterus was addressed for the first time, with specific reference to the myometrium and endometrium, as well as the segmentation of healthy ovaries and adnexal masses. The medical dataset was entirely provided by the company Syndiag, thanks to its active collaboration with several hospitals, including Ospedale Mauriziano Umberto I.

A further contribution of this work consists in the proposal of a hierarchical segmentation framework capable of modeling the relationships among different anatomical structures. This approach is compared with a classical *flat* segmentation framework, in which such relationships are not explicitly considered and which is used as a benchmark method for comparison.

From a methodological perspective, a *flat* segmentation approach was first developed, based on a U-Net architecture and the use of a focal loss, capable of assigning a specific class to each pixel of the image.

Subsequently, for the implementation of hierarchical methods, a hierarchical tree was defined to model the relationships among the different anatomical structures. To this end, two different approaches were proposed. The first method is also based on a U-Net architecture and is inspired by the work *A Hierarchical Loss for Semantic Segmentation* [15]; its main innovation lies in the loss function, referred to as focal hierarchical loss, designed to penalize the different levels of the hierarchy, incorporating hierarchical relationships only during the optimization process, while still producing a single class prediction for each pixel in the final output. This method can be considered as a hybrid solution between a *flat* and a fully hierarchical approach.

The second method, instead, is based on an approach inspired by the work *Deep Hierarchical Learning for Semantic Segmentation* [12], in which the hierarchy is explicitly modeled within the network architecture. In this case, segmentation is performed by directly considering multiple hierarchical levels, allowing the model to learn more structured and consistent representations of the relationships among different classes. This framework is referred to as HSSN.

The obtained results show that the hybrid method outperforms the HSSN framework. Furthermore, the comparison with the *flat* benchmark approach highlights

how the introduction of hierarchical modeling leads to a significant improvement in segmentation performance. In particular, the hybrid method achieved an improvement of over 10% in prediction performance for structures such as healthy ovaries and the uterus, confirming the effectiveness of incorporating hierarchical relationships into the segmentation process.

Bibliography

[1] Albumentations Team. *Albumentations Documentation*. https://albumentations.ai/docs/api-reference/albumentations/augmentations/geometric/transforms/. 2026.

[2] Mohammed Yusuf Ansari et al. "Advancements in deep learning for B-mode ultrasound segmentation: a comprehensive review". In: *IEEE Transactions on emerging topics in computational intelligence* 8.3 (2024), pp. 2126–2149.

[3] Wei Bi and James T Kwok. "Multi-label classification on tree-and dag-structured hierarchies". In: *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*. 2011, pp. 17–24.

[4] Kent Munthe Caspersen et al. "A hierarchical tree distance measure for classification". In: *International Conference on Pattern Recognition Applications and Methods*. Vol. 2. SciTePress. 2017, pp. 502–509.

[5] Vitor Cerqueira et al. *Online Data Augmentation for Forecasting with Deep Learning*. 2025. arXiv: 2404.16918 [cs.LG]. URL: https://arxiv.org/abs/2404.16918.

[6] Weifeng Ge. "Deep metric learning with hierarchical triplet loss". In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 269–285.

[7] GeeksforGeeks. *One Hot Encoding vs Label Encoding*. Last updated: 22 Jan 2026. 2026. URL: https://www.geeksforgeeks.org/machine-learning/one-hot-encoding-vs-label-encoding/?utm_source=chatgpt.com.

[8] IBM Editorial Staff. *What are Vector Embeddings?* Accessed: 2026-02-25. IBM. 2024. URL: https://www.ibm.com/think/topics/vector-embedding.

[9] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[10] Kris Kitani. *10.0 2D Transforms*. Lecture slides, Carnegie Mellon University. Accessed: 2026-03-03. 2017. URL: https://www.cs.cmu.edu/~16385/s17/Slides/10.0_2D_Transforms.pdf.

[11] Siddharth Krishna Kumar. "On weight initialization in deep neural networks". In: *CoRR* abs/1704.08863 (2017). arXiv: 1704.08863. URL: http://arxiv.org/abs/1704.08863.

[12] Liulei Li et al. "Deep hierarchical semantic segmentation". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2022, pp. 1246–1257.

[13] Ilya Loshchilov and Frank Hutter. "Sgdr: Stochastic gradient descent with warm restarts". In: *arXiv preprint arXiv:1608.03983* (2016).

[14] Warren S McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133.

[15] Bruce R Muller and William AP Smith. "A Hierarchical Loss for Semantic Segmentation." In: *VISIGRAPP (4: VISAPP)*. 2020, pp. 260–267.

[16] Keiron O'shea and Ryan Nash. "An introduction to convolutional neural networks". In: *arXiv preprint arXiv:1511.08458* (2015).

[17] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation". In: *International Conference on Medical image computing and computer-assisted intervention*. Springer. 2015, pp. 234–241.

[18] Frank Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6 (1958), p. 386.

[19] Florian Schroff, Dmitry Kalenichenko, and James Philbin. "Facenet: A unified embedding for face recognition and clustering". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 815–823.

[20] Bart Smets. "Mathematics of neural networks (lecture notes graduate course)". In: *arXiv preprint arXiv:2403.04807* (2024).

[21] Erik B Sudderth et al. "Learning hierarchical models of scenes, objects, and parts". In: *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*. Vol. 2. IEEE. 2005, pp. 1331–1338.

[22] Juan Terven et al. "A comprehensive survey of loss functions and metrics in deep learning". In: *Artificial Intelligence Review* 58.7 (Apr. 2025). ISSN: 1573-7462. DOI: 10.1007/s10462-025-11198-7. URL: http://dx.doi.org/10.1007/s10462-025-11198-7.

[23] Antonio Torralba, Phillip Isola, and William T. Freeman. *Foundations of Computer Vision*. Adaptive Computation and Machine Learning series. Cambridge, MA: MIT Press, 2024. URL: https://visionbook.mit.edu/.

[24] University of Auckland, School of Computer Science. *Geometric Operations*. https://www.cs.auckland.ac.nz/courses/compsci773s1c/lectures/ImageProcessing-html/topic2.htm. Lecture notes – Image Processing. n.d.

[25] Zaitian Wang et al. *A Comprehensive Survey on Data Augmentation*. 2025. arXiv: 2405.09591 [cs.LG]. URL: https://arxiv.org/abs/2405.09591.

[26] Michael Yeung et al. "Unified focal loss: Generalising dice and cross entropy-based losses to handle class imbalanced medical image segmentation". In: *Computerized Medical Imaging and Graphics* 95 (2022), p. 102026.

[27] Xiaomei Zhang et al. *Deep Learning for Human Parsing: A Survey*. 2023. arXiv: 2301.12416 [cs.CV]. URL: https://arxiv.org/abs/2301.12416.

[28] Hengshuang Zhao et al. "Pyramid scene parsing network". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 2881–2890.

[29] Xia Zhao et al. "A review of convolutional neural networks in computer vision". In: *Artificial Intelligence Review* 57.4 (2024), p. 99.