UCLouvain

epl

École polytechnique de Louvain

Optimizing deep learning-based vertebra identification on CT images for efficient clinical deployment

Author: **Alessandro Giansante**
Supervisor: **Benoît Macq**
Readers: **Sébastien Jodogne, Maxence Wynen, Mélanie Ghislain**
Academic year 2025-2026
Master [120] in Computer Science

ACKNOWLEDGMENTS

This thesis would not have been possible without the help of many people, and I would like to express my gratitude to everyone who contributed to it, whether directly or indirectly.

First and foremost, I would like to thank my supervisor, Professor Benoît Macq, for his valuable guidance and advice throughout this work.

I am also deeply grateful to Estelle Loyens, Maxence Wynen, and Mélanie Ghislain for their support, guidance, and insightful feedback, which were instrumental in shaping this thesis.

I would also like to thank Professor Sébastien Jodogne who has kindly accepted to review this work and be part of my jury.

My sincere thanks go to the team at Telemis as well. To Bruno Piscaglia, without whom this thesis would never have come to be. To Renaud Vander Stricht, for teaching me the intricacies of vertebral anatomy. And to Guillaume Alber, for always being available to answer my questions regarding the integration into their PACS viewer.

The training and evaluation of the models presented in this work were made possible thanks to the computational resources provided by the Consortium des Équipements de Calcul Intensif (CECI), funded by the Fonds de la Recherche Scientifique de Belgique (F.R.S.-FNRS) under Grant No. 2.5020.11 and by the Walloon Region.

Finally, I would like to thank my family and friends for their unwavering support, encouragement, and for all the moments we shared over these five years of study. It meant more than I can say.

REGARDING THE USE OF ARTIFICIAL INTELLIGENCE

During this research, artificial intelligence tools were used to discuss and compare certain approaches, as well as to better understand specific concepts from the literature.

For the writing process, I used DeepL, Gemini, and Claude AI as aids for translating, restructuring, and condensing my text. The visuals for diagrams and plots were created with the assistance of Claude AI.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE **Page**

LIST OF TABLES

ABBREVIATIONS

BCE Binary Cross-Entropy

Bi-LSTM Bidirectional Long Short-Term Memory

CECI Consortium des Équipements de Calcul Intensif

CNN Convolutional Neural Network

CPU Central Processing Unit

CT Computed Tomography

DICOM Digital Imaging and Communications in Medicine

EDT Euclidean Distance Transform

FCN Fully Convolutional Network

GPU Graphics Processing Unit

JSON JavaScript Object Notation

LSTM Long Short-Term Memory

MAE Mean Absolute Error

MICCAI Medical Image Computing and Computer Assisted Intervention

MRI Magnetic Resonance Imaging

NIfTI Neuroimaging Informatics Technology Initiative

ONNX Open Neural Network Exchange

PACS Picture Archiving and Communication System

RNN Recurrent Neural Network

SD Standard Deviation

SDK Software Development Kit

SGD Stochastic Gradient Descent

TTA Test-Time Augmentation

YOLO You Only Look Once

ABSTRACT

Identifying and numbering vertebrae on CT images is a crucial preliminary step in surgical planning and in the monitoring of spinal pathologies. However, for radiologists, this task is repetitive, time-consuming and prone to errors, particularly given the increasing workload and time pressure in clinical settings. Although numerous deep learning methods have been proposed to automate vertebra identification, the vast majority are designed in a research context and rely on architectures and pipelines that are too slow for real-world clinical deployment. This master thesis develops a deep learning model capable of automatically localizing each vertebra on CT images and assigning it its correct anatomical label, while satisfying the performance and efficiency constraints required for a future integration into a commercial PACS solution. To achieve this, we build on nnLandmark, a recent fork of the well-established nnU-Net framework, specialized in landmark detection through heatmap regression. We produced a thorough time breakdown of the nnLandmark pipeline, which revealed that the actual neural network inference represented only a small fraction of the total execution time, with preprocessing and postprocessing being the true bottlenecks. Based on this analysis, several optimizations were introduced: single-threaded execution for single-image inference, GPU-accelerated resampling, direct centroid extraction in the network space through coordinate scaling rather than resampling the full prediction volume, and model conversion to ONNX and TensorRT formats. Combined with batch size tuning and a center-of-mass-based centroid extraction, the model achieves an identification rate of 93.62%. On the efficiency side, the optimized pipeline achieves an average processing time of 1.40 seconds per image on a server-grade GPU and 1.58 seconds on a consumer-grade GPU, well within the few-second requirement for clinical deployment.

CHAPTER 1

INTRODUCTION

1.1 Motivation of automatic vertebra identification

In recent years, significant advances have been made in medical imaging thanks to the rise of deep learning. These models have been applied to various imaging modalities such as X-rays, Magnetic Resonance Imaging (MRI) and Computed Tomography (CT) [1], enabling major progress in automatic diagnosis and tumor detection. Medical professionals are thus assisted by these models in their clinical tasks, improving accuracy, reducing workload and supporting radiologists' decision-making [2].

The human anatomy is complex and many clinical tasks can benefit from automation. In the scope of this master's thesis, we focus on the spinal column, a central load-bearing structure of the human body. The study of the spine is clinically important because it plays a critical role in mobility, protects the spinal cord and it can be affected by numerous pathologies. For example, low back pain affected 7.5% of the global population in 2017 [3]. Beyond pain, the spine is subject to many other conditions, including vertebral fractures, osteoporosis, spinal stenosis, scoliosis, degenerative disc disease and vertebral tumors [4, 5, 6]. Each of these pathologies requires locating the affected level on the spinal column in order to plan treatment, which involves localizing and numbering the vertebrae. However, the spine has a particular anatomy that complicates this identification. It is composed of a succession of vertebrae stacked on top of one another that exhibit strong visual similarity on medical images. This similarity complicates differentiation, especially since the field of view of the image can vary and may be restricted to only a portion of the spinal column.

The identification and numbering of vertebrae therefore constitute an essential preliminary step in many clinical contexts. During surgical planning, the surgeon must be able to locate the vertebral level to be operated on with certainty, in order to avoid

performing surgery at the wrong level. As another example, in cases of fractures or tumors, it is sometimes necessary to monitor the patient over time, which requires comparing successive examinations based on vertebral landmarks [7].

In clinical practice, the task of identifying and numbering vertebrae on images is far from easy. It is time-consuming, as it requires the radiologist to manually scroll through all the slices of a scan, often composed of several hundred images [8]. It is a tiring task that demands sustained attention on repetitive and visually similar anatomical structures. In a hospital setting where the workload is constantly increasing, with long working days and where radiologists face time constraints, this task causes fatigue and therefore leads to mistakes [9, 10], which can have direct consequences on patient care. The number of images requiring interpretation per radiologist rose from approximately 2.9 per minute in 1999 to 16.1 per minute in 2010 [8]. It has been reported that among surveyed spine surgeons, 50% had performed surgery at the wrong vertebral level at some point during their career [11].

1.2 Aim of this master thesis

This master thesis was proposed by Telemis, a company in the medical-imaging field, specialized in Picture Archiving and Communication System (PACS) solutions. Telemis serves 463 hospitals and other treatment centers across Europe, the Middle East and Africa [12]. With the goal of relieving radiologists of the tiring and time-consuming task of manually identifying vertebrae, Telemis aims to enrich its analysis and display tools with an automatic vertebra identification feature. This thesis therefore consists of developing a deep learning model capable of automatically identifying vertebrae on CT images by predicting for each vertebra the coordinates of its centroid as well as its anatomical label.

This model must satisfy two key constraints. The first is **performance**: distinguishing vertebrae that are visually very similar across different fields of view and in the presence of potential anatomical anomalies is a complex task that requires a sufficiently powerful model. The identification and localization accuracy must be high enough to be clinically reliable. The second is **efficiency**: integrating such a model into a clinical workflow imposes strict requirements. Telemis requires the model to process an image within a few seconds without disrupting the workflow, while running on standard hardware. Vertebra identification models already exist in the literature, but their architectures often rely on large models that are demanding in terms of memory and computation time, making them difficult to deploy under real-world conditions. Finding the right balance between these two constraints is therefore a central challenge of this work.

This thesis also falls within a technology transfer effort between academic research and industry. Research in deep learning applied to medical imaging has led to the publication of vertebra identification models whose source code and pre-trained weights are made publicly available. However, these tools are designed in a research context.

Their integration into proprietary software raises several challenges: adapting input and output formats, complying with licensing constraints, ensuring memory efficiency and most importantly, optimizing the model to meet the speed requirements imposed by a clinical workflow. One of the objectives of this thesis is therefore to explore how an open-source academic solution can be adapted and optimized to become operational in an industrial context. It is important to note that this work is limited to the design, training and optimization of the model. The integration of the model into Telemis' PACS software is intended for future work.

1.3 Thesis structure

In this first chapter, we have outlined the motivation for automatic vertebra identification in an industrial context and the challenges involved. In Chapter 2, we present the background knowledge required to understand the remainder of this thesis, covering spinal anatomy, medical imaging, the identification task, deep learning architectures and evaluation metrics. Chapter 3 reviews the existing literature on vertebra identification methods and presents the nnU-Net and nnLandmark frameworks on which our approach is based. Chapter 4 describes the datasets used for training and evaluation. Chapter 5 provides a detailed analysis of the frameworks to help us choose the baseline for our improvements. Chapter 6 details the improvements made to the nnLandmark framework. Chapter 7 presents the experimental results. Finally, Chapter 8 concludes the thesis and discusses directions for future work.

CHAPTER 2

PREREQUISITE KNOWLEDGE

2.1 Spine structure

The spinal column is the central supporting structure of the human body and plays a major role in mobility and load transfer [13]. It also protects the spinal cord against injuries and shocks from physical impacts. It is composed of vertebrae connected to one another by intervertebral discs and extends from the base of the skull to the tip of the coccyx.

As shown in Figure 2.1, the spinal column is composed of 33 vertebrae divided into five regions: 7 cervical vertebrae, 12 thoracic vertebrae, 5 lumbar vertebrae, the sacrum formed by the fusion of 5 vertebrae and the coccyx formed by the fusion of 4 vertebrae [16]. Among them, the 24 cervical, thoracic and lumbar vertebrae are mobile and individually distinguishable on imaging. Vertebral labels follow a universal naming convention. Each vertebra takes the first letter of its corresponding region followed by its position from top to bottom. For example, C1 is the first cervical vertebra, followed by C2, while L5 is the last vertebra in the lumbar region.

Congenital vertebral anomalies can alter the number of vertebrae in an individual, such as the presence of a transitional vertebra L6 in the lumbar region or a 13th thoracic vertebra T13 [17, 18]. These variations represent an additional challenge for vertebra identification, as the total number of vertebrae can vary between individuals [19].

2.2 Medical imaging

2.2.1 Imaging modalities

This thesis focuses exclusively on CT imaging, though other modalities such as X-ray and MRI exist. A CT scanner emits X-rays from multiple angles as the source and detector rotate

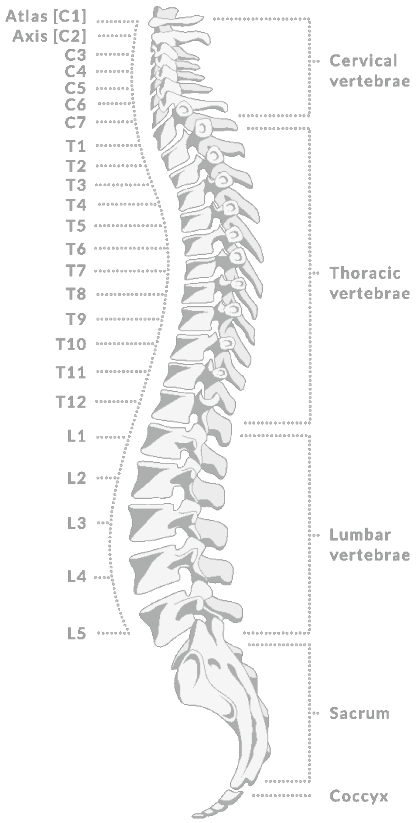


Figure 2.1: *Diagram of the spine showing the different regions and vertebral labels [14].*

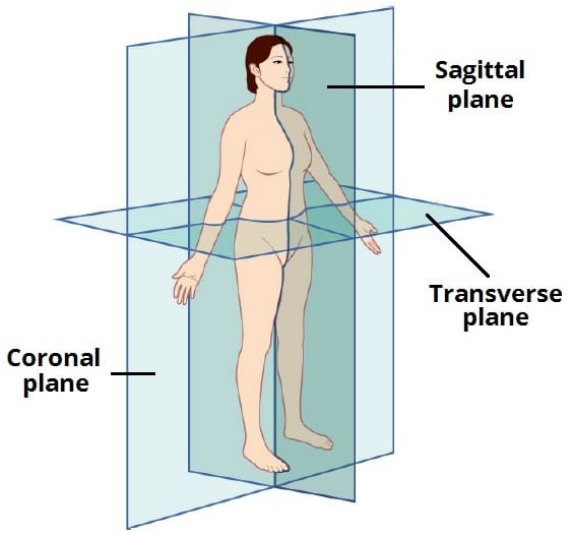


Figure 2.2: *Anatomical planes of the human body: sagittal, coronal and transverse [15].*

around the patient, and reconstruction algorithms combine these projections into a three-dimensional volume represented as a stack of two-dimensional slices [20]. Each element of this volume is called a voxel, the 3D equivalent of a pixel. Bones absorb more radiation than soft tissues, making them appear brighter on CT images, which provides clear contrast for visualizing vertebrae [13].

2.2.2 File formats

In clinical settings, CT slices are stored in the Digital Imaging and Communications in Medicine (DICOM) format, where each slice is a separate file containing both pixel data and metadata (patient information, scanner parameters, spatial position). For research, the Neuroimaging Informatics Technology Initiative (NIfTI) format is preferred because it stores the entire volume in a single file without patient metadata, making it simpler to work with. Also, conversion from DICOM to NIfTI is straightforward [21].

2.2.3 Coordinate systems

To describe positions within the human body, a standard *anatomical coordinate system* is defined using three planes (Figure 2.2): the sagittal plane (left/right), the coronal plane (anterior/posterior), and the transverse plane (superior/inferior). Their intersection defines three orthogonal axes along which any point can be described in millimeters [22].

A medical image is stored as a 3D array of intensity values indexed by (i, j, k), with no inherent notion of physical distance. To map between this *image coordinate system* and the anatomical coordinate system, three pieces of information are stored in the image header: the *voxel spacing* (physical distance between consecutive voxels), the *origin* (physical coordinates of voxel $(0,0,0)$), and an *affine matrix* that defines how image axes map onto anatomical axes (Figure 2.3) [21]. Because the affine matrix can differ between images, two images may store their voxel data in different axis orders and directions (Figure 2.4). Before processing a dataset with a neural network, the image axes must therefore be reordered into a consistent arrangement.

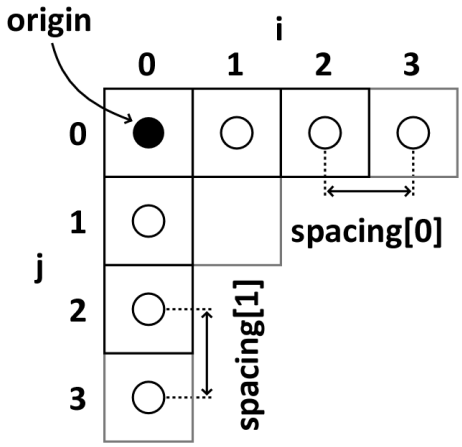


Figure 2.3: *2D example that shows the meaning of origin and spacing [23].*

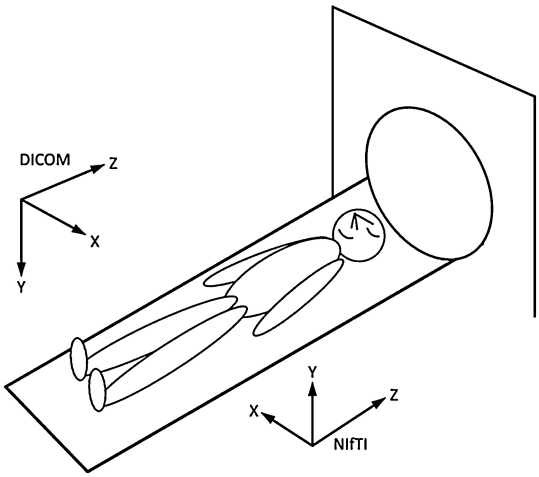


Figure 2.4: *Example of coordinate systems. The arrows of axes indicate the positive directions [21].*

2.3 Vertebra identification

When we refer to identification, the task consists of placing a landmark on the desired structure in an image and assigning it an identifier. A landmark is a single point that can be anatomically defined. In the case of vertebra identification, we locate each vertebra and assign it its anatomical label: C1 to C7 for the cervical vertebrae, T1 to T12 for the thoracic vertebrae and L1 to L5 for the lumbar vertebrae. The identification task can thus be decomposed into two sub-problems.

The first sub-problem is **localization**. It involves the detection of every vertebra present and determining their position in the image. To do this, a landmark is placed on the vertebra to represent its position. Typically, the reference point chosen is the centroid, i.e. the geometric center of the vertebral body [13]. Using a single point to represent a vertebra is a compact representation: it simplifies the entire volume of a vertebra into a single 3D coordinate while remaining interpretable for radiologists.

The second sub-problem is **labeling**. It consists of assigning the correct anatomical label to each localized vertebra. This step is particularly challenging because the field of view of the image can vary. It is therefore possible that an image does not contain the entire spinal column, meaning that the identification method lacks certain anatomical reference points.

It is important not to confuse identification with segmentation, which are two distinct tasks. Segmentation consists of delineating a region of interest by assigning a class to each voxel in a 3D image. In the case of vertebrae, segmentation produces a 3D mask for each vertebra that encompasses its entire volume thereby delineating its boundaries. Segmentation enables additional analyses beyond those provided by identification, such as computing the volume of a vertebra or analyzing its morphology.

2.4 Deep learning

2.4.1 Fundamentals

Deep learning is a subset of machine learning that uses neural networks with multiple successive hidden layers to learn representations directly from data. A network is composed of interconnected neurons organized into layers: an input layer, one or more hidden layers, and an output layer. Each neuron computes a weighted sum of its inputs, adds a bias, and applies a non-linear activation function. The weights and biases are the learnable parameters of the model [24, 25].

During training, the model iteratively adjusts these parameters to minimize a loss function that quantifies the discrepancy between predictions and ground truth labels. At each epoch, predictions are compared to the expected outputs, and the gradients of the loss with respect to all parameters are computed via backpropagation. An optimizer such as gradient descent then updates the parameters accordingly. After convergence, the trained model can be applied to new data during inference [25].

2.4.2 Convolutional neural network

Convolutional Neural Networks (CNNs) are deep learning models designed to efficiently process grid-like data structures such as images [2]. Instead of connecting every pixel to every neuron, a CNN applies small learnable filters that slide across the image, producing feature maps. Because the same filter weights are shared across all spatial positions, the number of parameters is drastically reduced compared to a fully connected network [26].

A CNN stacks multiple convolutional layers, each followed by an activation function. The early layers detect basic features such as edges and contours, while deeper layers combine them into increasingly complex representations. Pooling layers are interleaved between convolutional layers to reduce the spatial resolution of the feature maps, so that each filter captures a progressively larger context. The two most common variants are max pooling, which retains the largest value within a window, and average pooling, which computes the mean [27].

2.4.3 U-Net

U-Net is a CNN architecture with an encoder-decoder structure. Originally designed for biomedical image segmentation, it has a characteristic U-shaped architecture composed of three parts: an encoder, a bottleneck and a decoder (Figure 2.5) [28, 29].

The encoder takes an image as input and is composed of several blocks, each consisting of two convolutional layers followed by a max pooling layer. At each block, the spatial resolution decreases, allowing the network to extract increasingly abstract and global features from the original image. At the same time, while the spatial resolution is reduced, the number of feature channels increases, enabling the network to capture a greater variety of features.

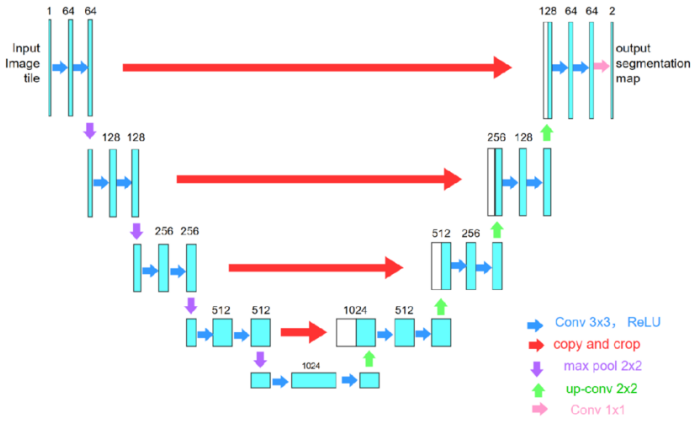


Figure 2.5: *Architecture of the original U-Net [30].*

The bottleneck forms the transition between the encoder and the decoder. It operates at the lowest spatial resolution and the highest number of feature channels.

The decoder takes the output of the bottleneck and produces the segmentation mask at the original image resolution through up-convolution operations that progressively increase the spatial resolution. Convolutional layers in the decoder combine the upsampled data with information from the encoder via skip connections. These skip connections are direct links between corresponding levels of the encoder and decoder that recover spatial details lost during downsampling. This gives the decoder's convolutions a combination of high-resolution spatial details from the encoder with coarse contextual features propagated upward from the bottleneck [29, 31]. The architecture of U-Net is notable for its ability to produce accurate results even with a small training set [28].

2.5 Evaluation metrics

In the design of a deep learning model, it is essential to evaluate its performance and efficiency in order to assess the impact of improvements and enable comparison with models from the literature.

A vertebra identification model must be able to assign the correct anatomical label to the corresponding vertebra. To measure labeling performance, we use the *identification rate* as defined in the VerSe benchmark [13].

To define this metric, consider an image of a spinal column with N ground truth annotations, one for each vertebra, corresponding to their true locations. For each vertebra i, let $\hat{x}_i$ denote the predicted location and x_i the ground truth location. The prediction for vertebra i is considered correct if x_i is the closest ground truth location to $\hat{x}_i$ among all ground truth locations $\{x_j \,\forall\, j \in 1,2,\ldots,N\}$, and the Euclidean distance between the prediction and the ground truth is less than 20 mm:

$$\|\hat{x}_i - x_i\|_2 < 20\,\text{mm}$$

For a single scan, the identification rate is the ratio of correctly identified vertebrae to the total number of vertebrae present in the scan. Across a dataset containing multiple scans, the identification rate is the mean over all scans.

Another performance metric is the mean localization distance, computed as the average Euclidean distance over a scan between each prediction $\hat{x}_i$ and its corresponding ground truth x_i:

$$d_{\text{mean}} = \frac{1}{N}\sum_{i=1}^{N} \|\hat{x}_i - x_i\|_2$$

Similarly, across a dataset containing multiple scans, the mean distance is averaged over all scans [13].

For efficiency, the chosen metric is the inference time, expressed in seconds. It corresponds to the total time required by the model to process a CT image and produce the complete set of vertebra identification predictions.

CHAPTER 3

LITERATURE REVIEW

3.1 Evolution of identification methods

Automatic vertebra identification is not a new task, and many papers have been published on this subject. In this section, we review the existing methods in the literature.

3.1.1 Traditional methods

The earliest methods for automatic vertebra identification were based on hand-crafted features, mathematical modeling, and graph theory. The first family encompasses image processing-based methods. These approaches analyze the image using classical operators, such as thresholding, edge detection, or mathematical morphology, and combine the results with deterministic anatomical rules to detect and identify each vertebra. Naegel (2007) [32] proposed such a pipeline for 3D CT scans: intervertebral discs are first detected using a black top-hat morphological operator, each vertebra is then coarsely segmented by thresholding and refined with the watershed algorithm [33]. Vertebrae are finally labeled by identifying T12, the last vertebra articulated with ribs, and counting up and down along the spine.

A second family of methods relies on deformable model-based approaches. Rather than applying generic operators guided by anatomical rules, these techniques encode prior anatomical knowledge, such as the expected shape, size and location of the target structure, into a geometric or statistical model [34]. These models are matched to the image through registration or similarity evaluation [35]. Model-based methods tend to generalize better than image processing-based approaches because the model captures general anatomical properties rather than relying on fixed heuristics [36]. Klinder et al. (2009) [35] proposed a representative method that follows a multi-step pipeline: the spine curve is first extracted by tracing tube-shaped segments from a seed point in the spinal

cord, then straightened via curved planar reformation. Vertebra candidates are located using the generalized Hough transform [37], and anatomical labels are assigned by matching each candidate against intensity-based appearance models learned from the training set. The configuration of labels yielding the highest average match identifies every vertebra.

Another traditional family is based on probabilistic graphical models. Rather than focusing on vertebra geometry alone, these methods exploit the ordered sequence of vertebrae and the relations between them, such as inter-vertebral distance. In Glocker et al. (2012) [38], a regression forest provides a coarse estimate of each centroid, then a Hidden Markov Model refines these locations by constraining inter-vertebral distances.

Despite their contributions, these traditional approaches share several fundamental limitations. They rely heavily on prior knowledge derived from the training set whether encoded as fixed heuristics, shape templates, or statistical models. Therefore, they generalize poorly to images that deviate significantly from the expected anatomy [39]. Pathological cases such as severe scoliosis, vertebral fractures, or surgical implants can violate the anatomical assumptions these methods depend on, leading to identification failures.

3.1.2 Basic deep learning

With the rise of deep learning, researchers proposed new solutions. Unlike previous methods that required manually choosing which features to extract, deep learning learns representations directly from training data through backpropagation.

One of the first applications to spinal images dates back to Suzani et al. (2015) [40], who used a deep feed-forward neural network trained on contextual intensity features extracted from CT data. For each reference voxel, the method computes the mean intensity of 500 randomly placed 3D cuboids around it, then predicts 26 distance vectors pointing to the center of each vertebra. The calculated locations act as votes to find the final landmark for each vertebra. A vertebra is considered absent if its predicted centroid falls outside the image, allowing the method to handle arbitrary fields of view. The main weakness is that each vertebra is treated independently, ignoring the anatomical continuity between adjacent vertebrae [41]. The authors report an identification rate of 96% with a runtime under 3 seconds [40], although a later study reported a Mean Absolute Error (MAE) of 18.20 ± 11.40 mm [41], reflecting the lack of global context.

3.1.3 Convolutional network-based approaches

Introduced by LeCun et al. (1998) [26], CNNs and their variants constitute the foundational backbone of vertebra detection methods, thanks to their hierarchical feature extraction capabilities. A model typically takes an image or patch as input and outputs either a classification label or a heatmap indicating probable vertebra locations [41].

To overcome the context-blindness of early CNNs, Windsor and Jamaludin (2020) [42]

introduced the Ladder algorithm, built on VGG-F [43]. Instead of treating each vertebra independently, this method reformulates spine detection as a sequential decision-making problem. Starting from S1, the algorithm iteratively predicts the bounding box of the current vertebra and regresses the location of the next vertebra above. Trained solely on lumbar scans, the model generalizes to whole-spine scans by moving up the spine with each iteration, achieving 99.4% precision and recall on a separate whole-spine dataset. The downsides are that errors in early predictions can propagate, and the fixed iteration count prevents the algorithm from adapting to anatomical variants such as the transitional L6 vertebra [42].

U-Net architectures (Ronneberger et al., 2015) [28] and their encoder–decoder designs, known for their efficient multi-scale feature aggregation, have also been widely applied to localization tasks [41]. Qin et al. (2021) [44] combined a U-Net for multi-scale feature extraction with a separate multi-label classification branch using a Bidirectional Long Short-Term Memory (Bi-LSTM) network to capture long contextual information along the spine. On the 2014 MICCAI dataset, this approach achieved a mean localization error of 2.2 mm and an identification rate of 91.1%.

Rather than performing identification directly from the raw image, some approaches leverage segmentation as an intermediate step to provide richer morphological information. Zhang et al. (2026) proposed SpineCLUE [39], a three-stage framework that performs localization, segmentation and identification sequentially. Vertebrae are first localized using a 2D detector combined with a density clustering algorithm, each vertebra is then individually segmented within a cropped region, and the masks are finally fed into an identification network pretrained with supervised contrastive learning to address inter-class similarity between neighboring vertebrae. The method achieved state-of-the-art identification rates of 98.73% and 97.87% on the public and hidden sets of the VerSe 2020 benchmark. However, the multi-stage pipeline introduces computational overhead, with runtimes from 22 to 40 seconds in average depending on the field of view, which may limit its use in time-constrained clinical settings.

3.1.4 Sequential and structural relationship modeling approaches

Convolution-based approaches have demonstrated strong performance, but they struggle to capture long-range dependencies despite postprocessing steps designed to maintain an anatomically correct sequence. The spine is an inherently sequential structure in which each vertebra occupies a fixed position from C1 to L5, and the identity of any given vertebra is strongly influenced by its position relative to the rest of the column. Accurately identifying vertebrae therefore calls for architectures designed to model sequential data [41].

Recurrent Neural Networks (RNNs) and in particular Long Short-Term Memory (LSTM) networks [45] were applied to vertebra detection. Liao et al. (2018) [46] proposed a hybrid framework combining a 3D Fully Convolutional Network (FCN) for short-range contextual

feature extraction with a multi-task bidirectional RNN that processes the sequence of patch features in both directions to capture context from the entire spine [41, 46]. On the MICCAI CSI 2014 Spine CT dataset, the method achieved a MAE of 6.50 ± 8.60 mm and an identification rate of 88.30% [41].

Although LSTMs offer improvements over RNNs, they still suffer from the vanishing gradient problem on long sequences [47]. Transformers [48] address this through self-attention, which captures distant dependencies without gradient decay. Tao et al. (2022) [49] applied this idea by using a CNN backbone to extract a 3D feature map, flattening it into a 1D token sequence, and feeding it to a Transformer that predicts a distinct feature vector for each potential vertebra. These vectors are then used to determine whether each vertebra is present and, if so, its spatial position. The authors reported an identification rate of 96.74% on the VerSe 2019 dataset.

3.1.5 Object detection-based approaches

Although sequential and structural modeling approaches improve vertebra detection by capturing global context, they come at the cost of increased computational complexity and slower inference [50], making them difficult to integrate into clinical practice. Object detection models are popular precisely because of their speed and efficiency [41]. The You Only Look Once (YOLO) family [51] processes an entire image in a single forward pass, dividing it into a grid and predicting bounding boxes and class probabilities for each cell simultaneously [52]. YOLO and its extensions have already been used for vertebrae detection [53], with versions such as YOLOv3 processing images in milliseconds on a modern GPU.

3.1.6 Foundation model-based approaches

Foundation models are large neural networks pre-trained on massive and diverse datasets, which can be adapted to specific tasks through fine-tuning [54]. Their use for vertebra identification, a fine-grained classification task, faces two main challenges: a limited ability to exploit vertebral spatial relationships and to discriminate between similar vertebral morphologies [55], and the substantial computational resources required for fine-tuning [56]. VertFound [55] addresses both by combining two foundation models, one specializing in semantic features and the other in spatial features, keeping their weights frozen and training only a lightweight fusion module. This approach achieved an identification rate of 89.05% on an in-house dataset.

3.2 The nnU-Net framework

Building a model from scratch requires extensive hyperparameter tuning (encoder depth, patch size, augmentation, learning rate, etc.), while existing codebases are often poorly maintained, dataset-specific, and difficult to adapt due to domain shift [57]. The open-

source framework no-new-Net (nnU-Net) [57] was designed to address these challenges by automating the entire configuration process.

3.2.1 Principles

nnU-Net is a deep learning framework for biomedical image segmentation whose distinctive feature is its ability to automatically self-configure. It embeds a slightly modified U-Net in a pipeline that automatically adapts preprocessing, network configuration, training and inference parameters from the dataset alone. This matters because properly configuring a model has a larger impact on performance than choosing the perfect architecture [58]. nnU-Net has achieved state-of-the-art results on 33 out of 53 segmentation tasks across several public benchmarks [57]. This framework has already been used to identify vertebrae by Wang et al. (2020) [59] who trained it to produce 3D vertebra-center activation maps. These maps are then resampled along the automatically estimated spine centerline to straighten the spine, so that variations in curvature do not affect the results. The rectified volumes are aggregated into one-dimensional activation signals and refined by an anatomically-constrained optimization enforcing vertebral ordering and inter-vertebra distances. The method achieved an identification rate of 97.4% on the SpineWeb dataset.

3.2.2 Self-Configuring Mechanism

Adapting a segmentation method to a new dataset requires choosing an appropriate preprocessing strategy, network topology, training hyperparameters, and postprocessing. These decisions are interdependent, and nnU-Net divides them into three categories: *fixed parameters* (constant across datasets), *rule-based parameters* (derived from dataset properties via heuristic rules), and *empirical parameters* (determined through cross-validation). All configuration decisions are stored in a plans file that defines how training and inference are carried out [57].

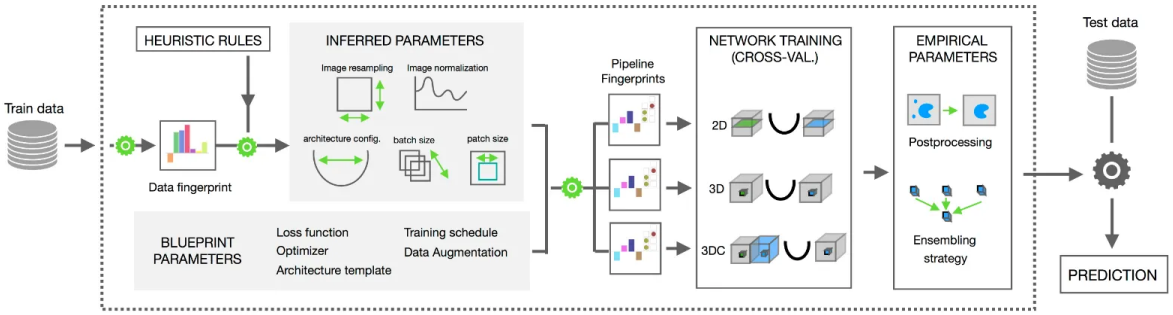


Figure 3.1: *Illustration of the self-configuration mechanism of nnU-Net [57].*

3.2.2.1 Dataset Fingerprint

nnU-Net begins by analyzing the training data to extract a *dataset fingerprint*: image sizes, voxel spacings, imaging modality, class statistics, dataset size, and intensity statistics

(mean, standard deviation, percentiles computed over foreground voxels). This fingerprint serves as input to the heuristic rules that configure the pipeline.

3.2.2.2 Fixed Parameters

Fixed parameters remain constant regardless of the dataset. The architecture follows a modified U-Net with instance normalization and leaky ReLU activations. Deep supervision is applied in the decoder to improve gradient flow. Training uses Stochastic Gradient Descent (SGD) with Nesterov momentum ($\mu = 0.99$), the "poly" learning rate schedule, and minimizes the sum of Dice loss and cross-entropy loss [60]. Each model trains for 1000 epochs of 250 iterations, with on-the-fly data augmentation (rotations, scaling, noise, blur, gamma correction, mirroring). At inference, a sliding window with half-patch overlap and Gaussian weighting is used, with optional test-time augmentation via mirroring.

3.2.2.3 Rule-Based Parameters

Rule-based parameters are derived from the dataset fingerprint. For CT images, intensity values are clipped at the 0.5th and 99.5th foreground percentiles and normalized using the global foreground mean and standard deviation. All images are resampled to a common target spacing using third-order spline interpolation to ensure consistent physical voxel sizes.

The network topology, patch size, and batch size are configured jointly under GPU memory constraints. The patch size is initialized to the median image shape and iteratively reduced until the configuration fits in memory with a batch size of at least two. The architecture depth is selected so that the receptive field covers the patch size. For datasets with very large images, where the patch covers less than 12.5% of the median image shape, nnU-Net triggers a 3D U-Net cascade: a low-resolution U-Net captures global context followed by a full-resolution U-Net that refines its output.

3.2.2.4 Empirical Parameters

After five-fold cross-validation, nnU-Net selects the best configuration in term of performance among the 2D U-Net, 3D full-resolution U-Net and (if triggered) 3D cascade, or ensembles two of them based on the average foreground Dice coefficient. It also evaluates whether connected-component-based postprocessing improves scores by removing all but the largest connected component.

3.2.3 Segmentation Pipeline

Understanding each stage of the inference pipeline and its computational cost is essential for the next chapters because our method will be based on top of it. The pipeline consists of four phases: setup, preprocessing, inference and postprocessing.

1. **Setup:** At inference time, the pipeline first reads the plans file, instantiating the data iterator (which initializes the workers that feed images to the pipeline), reconstructs

the network architecture, restoring the pretrained weights from the checkpoint file, and creating the export pool of background workers that will handle postprocessing. Since these steps run only once regardless of the number of images, their cost is amortized over the entire set.

2. **Preprocessing:** It covers all the transformations applied to each input image, the same as those applied during training, before it can be passed to the network: (1) loading the image from disk, (2) transposing the axes to a consistent orientation defined in the plan file, (3) cropping to the non-zero region to reduce computational cost, (4) applying intensity normalization using the global statistics of the training set and (5) resampling to the target spacing using third-order spline interpolation. These steps are carried out by background workers running as separate processes in parallel with inference, so the network can process one image while another is in the preprocessing stage.

3. **Inference:** It corresponds to the forward pass of the network itself, where the preprocessed volume is processed using a sliding window approach with half-patch overlap and Gaussian importance weighting to suppress stitching artifacts at window boundaries. Test-Time Augmentation (TTA) can be enabled. It's when each patch is mirrored along all combinations of spatial axes ($2^3 = 8$ versions for 3D data), and the resulting predictions are averaged. nnU-Net may also ensemble multiple model configurations by averaging their softmax probability maps voxel-wise.

4. **Postprocessing:** The network outputs logits of shape (C, D, H, W), where C is the number of classes. The prediction is resampled back to the original voxel spacing, merging into a single-channel label map via argmax, the cropping is reversed to restore the original dimensions and they transpose the axes back to their original orientation. Connected-component-based postprocessing may then remove small false-positive regions. The final segmentation mask is exported as a NIfTI file. Postprocessing is also handled by background workers, so that the overall pipeline operates as a three-stage parallel workflow: preprocessing, inference, and postprocessing run concurrently on different images.

3.2.3.1 Architecture

All nnU-Net configurations use a modified U-Net with an encoder, decoder and skip connections. The topology is adapted by the rule-based configuration (Section 3.2.2.3). The three candidate configurations differ in how they process the volume: the 2D U-Net processes slices independently, the 3D U-Net operates on 3D patches at full resolution, and the 3D cascade refines a low-resolution prediction at full resolution. Feature channels start at 32 and double at each downsampling step, up to 320 (3D) or 512 (2D) [57].

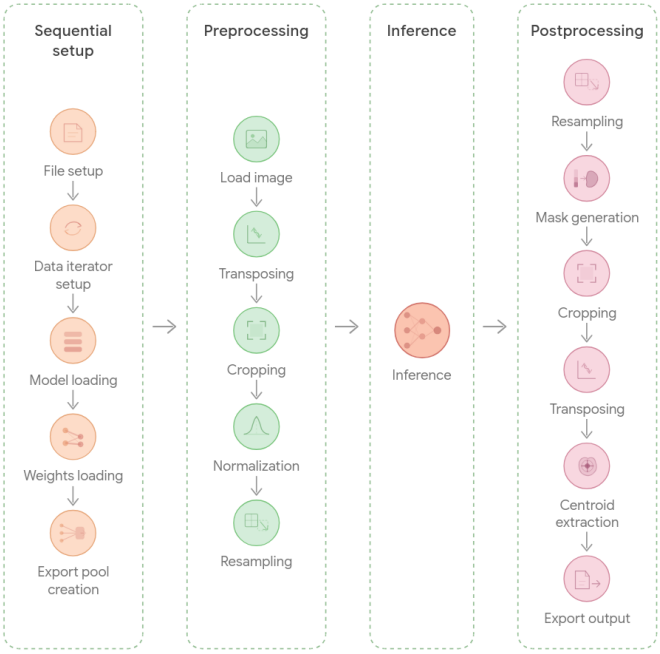


Figure 3.2: *Diagram of the nnU-Net pipeline divided into four phases.*

3.3 The nnLandmark framework

Very recently, a new framework called nnLandmark [61] appeared. It is a fork of nnU-Net and therefore shares the same automatic pipeline configuration mechanism, but instead of performing segmentation, it is specialized in 3D medical landmark detection. nnLandmark does not assign a class to every voxel, instead it predicts landmarks, that is anatomical points defined by a label and their spatial coordinates which is better suited to our identification task, because there is no need to go through the segmentation stage. Since nnLandmark is very recent, it has been applied to only a few tasks and to the best of our knowledge, it has never been tested on vertebral identification. Applying it to vertebra identification on CT images therefore represents a novel use case for this framework. Given that nnLandmark and nnU-Net share the same codebase, their pipelines are nearly identical, and the sections above apply to nnLandmark as well, with a few differences that we describe below.

3.3.1 Landmark representation

In order to be able to reuse and to remain compatible with nnU-Net's data loading pipeline, which expects segmentation inputs, nnLandmark encodes each landmark as a small $3\times3\times3$ voxel label in a multi-label segmentation map. This allows the framework to exploit the entire automatic preprocessing and self-configuration mechanism without modification. The conversion from the multi-label representation to heatmap regression happens during training in the loss computation. For each landmark, the center of mass of its label region is computed, and an Euclidean Distance Transform (EDT) with a radius of 15 voxels is gener-

ated around that point in a dedicated output channel, producing a smooth heatmap as the regression target. This on-the-fly generation avoids storing and loading large precomputed heatmaps, saving both memory and disk space [61].

3.3.2 Training adaptations

The training differs from nnU-Net on two things. First, the loss function is replaced by a Binary Cross-Entropy (BCE) TopK20 loss. The BCE is defined as :

$$\text{BCE} = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(p_i) + (1-y_i)\log(1-p_i)\right]$$

where N is the number of voxels, y_i is the actual binary label (0 or 1) of the i^{th} voxel and p_i is the predicted probability of the i^{th} voxel being in class 1. It allows to calculate the difference between the ground truth label and the predicted probabilities output by the model. TopK20 means that the voxel-wise BCE values are ranked in each patch and only the top 20% of voxels with the highest loss are kept for gradient computation. This focuses learning on the most difficult voxels and mitigates the foreground-background imbalance due to the landmark heatmaps, where the vast majority of voxels belong to the background. Second, the final layer uses a sigmoid activation instead of a softmax, constraining each output channel independently to the $[0,1]$ range [61].

3.3.3 Inference and postprocessing

At inference time, the sliding window prediction from nnU-Net is reused to produce a full-resolution heatmap for each landmark channel. The predicted coordinates are then obtained by taking the channel-wise maximum of the heatmap, i.e. the voxel with the highest intensity in each channel corresponds to the predicted landmark position [61].

The postprocessing also differs from nnU-Net's. Since nnLandmark predicts one heatmap per landmark instead of a segmentation mask, the classes are not mutually exclusive: a voxel can have high probabilities for multiple landmarks simultaneously. For this reason, nnLandmark applies a sigmoid function to each channel independently and keeps all channels instead of merging them into a single-channel label. The process to revert everything done during preprocessing in order to bring the probabilities back to the original space is kept, except that instead of having only one channel, we have all of them. From there, the step to extract the centroids of each vertebra is performed. For each channel, it removes voxels with a probability of exactly 1.0 to avoid degenerate plateaus, keeps only the top 0.5% of positive voxels by value, then takes the argmax within that filtered region. It returns the coordinates and the confidence probability that the vertebra is indeed present in the image. This probability is called the likelihood. nnLandmark still saves a NIfTI file, which here contains a label map with only the 27 most activated voxels per class by choice, and additionally saves the coordinates and the likelihood of each vertebra in a JSON file. There is no connected-component-based post-processing because we don't have components on coordinate prediction.

CHAPTER 4

MATERIALS

4.1 Dataset presentation

Several publicly available datasets of spinal columns exist that include vertebral annotations. Since the methods developed in this master thesis rely on CT imaging, we only consider datasets based on CT scans. To train and evaluate our models, we chose the dataset prepared and used for the Large Scale Vertebrae Segmentation Challenge (VerSe) [13], organized in conjunction with the International Conference on Medical Image Computing and Computer Assisted Intervention (MICCAI) in 2019 and 2020. Across both editions, two datasets were released, totaling 374 CT scans from 355 patients with 4505 annotated vertebrae. This dataset is widely used as a benchmark in the literature, which makes it straightforward to compare our results with those of other methods.

The data comes from 355 patients with a mean age of 59 (±17) years. It is multi-site and was acquired using multiple CT scanners from the four major manufacturers (GE, Siemens, Philips, and Toshiba). The authors aimed to reflect a realistic clinical distribution in terms of field of view, scan settings, and pathological findings. This means the dataset includes a variety of fields of view (cervical, thoraco-lumbar, and cervico-thoraco-lumbar scans), a mix of sagittal and isotropic reformations, and cases presenting vertebral fractures, metallic implants, or foreign materials [13].

Another important consideration is licensing. Some publicly available medical datasets restrict commercial use, which would be problematic for integrating the resulting model into Telemis products. The VerSe dataset is published under the CC BY-SA 2.0 license, which does not restrict commercial use.

4.2 Contents of the dataset

For this thesis, we use only the VerSe 2019 dataset, which contains 160 CT images from 141 patients, covering a total of 1725 vertebrae. The images are split into a training set (80 images), a validation set (40 images), and a test set (40 images). Each image comes with two types of annotations: the 3D coordinates of vertebral centroids and voxel-level segmentation masks. These annotations are available for all three sets, making it possible to both train the models and compare predictions against ground truth labels. In total, 25 vertebra types are considered, ranging from C1 to L5 plus the transitional vertebra L6. They are assigned integer labels from 1 to 24, with label 25 reserved for L6.

The annotations were initially generated by an automated algorithm, then manually refined by five medical students and validated by three radiologists with 30 years of experience each. All annotations were ultimately approved by a single radiologist with 19 years of experience.

The vertebrae distribution across the VerSe 2019 datasets is shown in Appendix A. The cervical region is the least represented across all 3 sets. In the training set, L6 appears in only 3 images, so the model will see very few examples of this vertebra during training, which may make its detection during evaluation on the test set difficult. Furthermore, only 2 images in the entire dataset contain the full range of vertebrae from C1 to L5, and both belong to the test set. This represents an additional challenge for the model, since it has never encountered an image covering all vertebrae during training.

CHAPTER 5

FRAMEWORK ADAPTATION AND BASELINE EVALUATION

To focus our work on the research aspect rather than on architecture design, we decided to build our identification method on top of the nnU-Net and nnLandmark frameworks presented in Chapter 3. nnLandmark directly predicts landmark coordinate through heatmap regression, which is better suited to our identification task, as it means we do not need to go through the segmentation stage. Applying nnLandmark to vertebra identification on CT images also represents a novel use case for this framework. We will do an evaluation to select the most suitable configuration.

5.1 Adapting nnU-Net to identification

As explained in Section 3.2, nnU-Net performs segmentation, but in our case, the goal is vertebral identification: finding the coordinates of the centroid of each vertebra. It is therefore necessary to extract a landmark from the multi-class segmentation mask. This section presents three approaches we considered for this conversion.

Center of mass The most straightforward approach is to compute the center of mass of each class in the segmentation mask. For a given class k, the centroid $\mathbf{c}_k$ is defined as the mean position of all voxels belonging to that class:

$$\mathbf{c}_k = \frac{1}{|V_k|} \sum_{i \in V_k} \mathbf{x}_i \tag{5.1}$$

where V_k is the set of voxels labeled as class k and $\mathbf{x}_i$ is the spatial position of voxel i.

However, this approach leads to a localization error for vertebral centroids. As illustrated in Figure 5.1, a vertebra is composed of two parts: the vertebral body and the vertebral arch. In the dataset used, the ground truth centroid of a vertebra is placed by considering only the vertebral body. The center of mass, on the other hand, takes into account

all voxels of the vertebra, including the arch, which shifts the predicted centroid away from the ground truth (Figure 5.2). The resulting distance between the predicted and the ground truth centroid can be significant. While the center of mass still allows to correctly identify which vertebra is which, this distance is a problem if we want to compare our method against others using standard localization metrics.

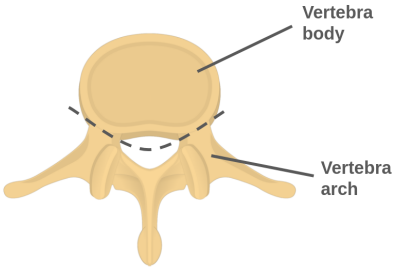


Figure 5.1: *Diagram showing the structure of a vertebra that consists of an anterior vertebral body, and a posterior vertebral arch [62].*

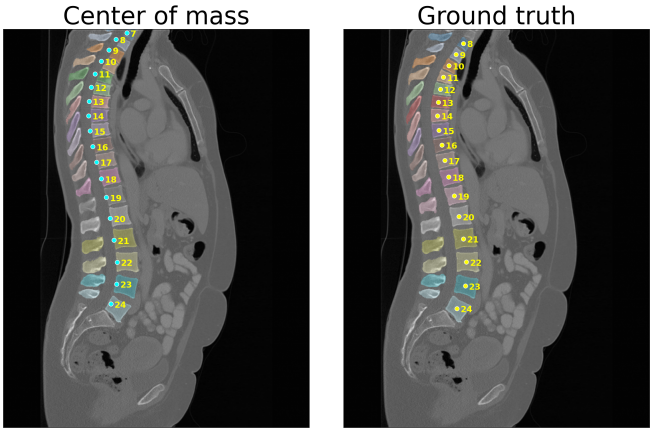


Figure 5.2: *Comparison between the centroids calculated based on the center of mass on the mask predicted by nnU-Net and the ground truth in sagittal view.*

Euclidean distance transform A second approach uses the EDT. The idea is to compute, for each foreground voxel of a given class, its distance to the nearest background voxel. The voxel with the maximum distance is the point that lies deepest inside the structure, and we take it as the centroid. This approach works well for vertebrae because the vertebral body is thicker than the vertebral arch: the point most distant from the boundary naturally falls within the vertebral body, close to where the ground truth centroid is placed. However, relying on a single maximum point can be fragile, especially when multiple voxels share the same maximum distance. To address this, a center of mass is applied on the distance map rather than on the binary mask. This way, voxels farther from the boundary receive a higher weight, so the resulting centroid is still drawn toward the interior of the structure but is computed as a smooth weighted average rather than depending on a single voxel.

Morphological opening A third approach is to apply a morphological opening to the segmentation mask before computing the center of mass. Morphological opening is a two-step operation: an erosion followed by a dilation. The erosion shrinks the mask by removing voxels at its boundary using a structuring element that eliminates thin structures such as the vertebral arch. The dilation then restores the remaining structure to approximately its original size. After opening, the mask retains mainly the vertebral body, and the center of mass computed on this cleaned mask is much closer to the ground truth centroid.

If we apply each extraction method directly on the full segmentation mask, the process becomes slow. For each vertebra, the algorithm must first identify its voxels by scanning the entire volume, including the background. This full scan is repeated for every vertebra, making the computation increasingly expensive as the number of labels grows. An optimization that applies to all three methods is to first extract a bounding box around each vertebra at the beginning. By cropping the volume to the small region where a given vertebra actually resides, the extraction method only needs to process a small part of the total voxels, which significantly reduces the computation time.

5.2 Configurations

The default models proposed by the frameworks do not directly meet our objectives in terms of performance and computational efficiency, and therefore require further improvements. Before making any improvements, we first need to choose which framework between nnU-Net and nnLandmark will serve as the basis of our work, along with the most suitable pipeline configuration. To do so, we compared both frameworks across several configurations.

For both nnU-Net and nnLandmark, the self-configuring mechanism proposed two configurations each: a 2D U-Net and a 3D full-resolution (fullres) U-Net. The 3D cascade U-Net was not proposed because the images in our dataset have a sufficiently low resolution that the patch size determined by the heuristic rules captures adequate contextual information. The configurations generated by the self-configuring mechanism are stored in the plans file. Table 5.1 summarizes the key parameters selected by each framework. Both frameworks produce identical configurations, since nnLandmark inherits the same self-configuring mechanism as nnU-Net.

Table 5.1: *Key parameters of the 2D and 3D full-resolution configurations determined by the nnU-Net and nnLandmark self-configuring mechanisms.*

Parameter	2D	3D Full Resolution
Spacing (mm)	1.0 × 1.0	1.0 × 1.0 × 1.0
Patch size	256 × 192	192 × 112 × 96
Batch size	61	2
Network depth	6 stages	6 stages
Features per stage	32–64–128–256–512–512	32–64–128–256–320–320
Strides	(1×1)–(2×2)–(2×2)–(2×2)–(2×2)–(2×2)	(1×1×1)–(2×2×2)–(2×2×2)–(2×2×2)–(2×2×2)–(2×1×1)

The self-configuring mechanism of both frameworks focuses exclusively on maximizing segmentation or detection performance, whereas our vertebra identification task requires a good trade-off with computational efficiency. To explore this trade-off, we derived new configurations from the automatically generated full-resolution plans, taking

advantage of nnU-Net's ability to inherit a base configuration and selectively override specific parameters. We refer to these variants as 3D Low Resolution (lowres) configurations.

The primary modification is an increase of the target voxel spacing. A coarser resolution reduces the number of voxels to process: the resampled volume becomes smaller, each patch covers a larger proportion of the anatomy, and the number of sliding window steps during inference decreases. Three spacings were tested for both frameworks: $1.5 \times 1.5 \times 1.5$, $2.0 \times 2.0 \times 2.0$, and $2.5 \times 2.5 \times 2.5$ mm. Alongside the spacing, the patch size was halved to $96 \times 64 \times 48$, which reduces GPU memory consumption but also limits the spatial context available for each prediction. The number of features per stage was also halved for the nnLandmark low-resolution configurations (16–32–64–128–160–160), giving a lighter model with fewer parameters, since at a coarser resolution with a smaller patch size the original number of feature maps would be unnecessarily large relative to the information present in each input patch. An additional configuration, referred to as `coarse`, was tested with a voxel spacing of $2.0 \times 2.0 \times 2.0$ mm and the same reduced patch size, but retaining the original number of features per stage, in order to evaluate whether keeping the full model capacity at a coarser resolution improves identification performance over the low-resolution variant. Together, these changes should significantly accelerate inference at the cost of a potential decrease in localization accuracy, since fine spatial details are lost during downsampling. To distinguish the configurations in the following sections, each is named after its voxel spacing. For example, the nnLandmark configuration with a spacing of $2.0 \times 2.0 \times 2.0$ mm and reduced features is referred to as `nnLandmark_3d_lowres_2.0`, while its counterpart with full features is referred to as `nnLandmark_3d_coarse_2.0`.

nnLandmark 2D was tested but had an identification rate of less than 1%, indicating an issue with the framework for 2D configurations. It will therefore not be considered among the candidates.

5.3 Measured time of pipeline parts

Since one of our objectives is to improve computational efficiency, it is important to understand where time is spent. As illustrated in Figure 3.2, we divided the pipeline into four phases: sequential setup (reading the plans file, spawning workers, loading the model), preprocessing (loading, transposing, cropping, normalizing, resampling), inference (network forward pass via sliding window), and postprocessing (resampling back to original resolution, centroid extraction, export).

The pipeline exploits parallelism through background workers (Figure 5.3): preprocessing and postprocessing run concurrently with inference on different images. However, the main thread can be blocked waiting for a preprocessed image to become available or for a postprocessing worker to accept a new result. After the last inference, it must also wait for all remaining postprocessing tasks to finish. The total time can be expressed as:

$$T_{\text{total}} \approx T_{\text{setup}} + \sum_{i=1}^{N} \left(T^{(i)}_{\text{wait\_pre}} + T^{(i)}_{\text{wait\_export}} + T^{(i)}_{\text{infer}} \right) + \sum_{i=1}^{N} T^{(i)}_{\text{wait\_post}}$$

When preprocessing and postprocessing are faster than inference, the wait times become negligible and the total time is dominated by setup and cumulative inference time. It is worth noting that Python library import times and image reorientation to a consistent coordinate system are excluded from our timing analysis. Library imports can be avoided in production by keeping the inference process running as a persistent service. Image reorientation is excluded because this operation is common to all implementations

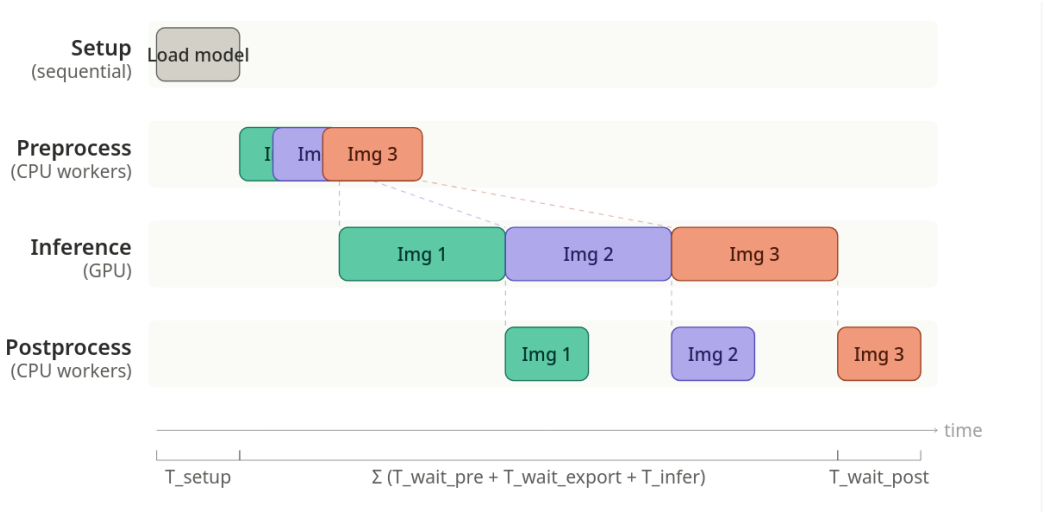


Figure 5.3: *Illustration of background workers and the main thread running in parallel.*

5.4 Selecting the best centroid extraction method from masks

5.4.1 Quantitative results

As mentioned earlier, nnU-Net outputs a segmentation mask rather than direct centroid predictions, which means centroids must be extracted from the mask in a post-processing step. In Section 5.1, we introduced three extraction methods: center of mass, EDT, and morphological opening. To determine which one to retain for the final comparison against nnLandmark, we evaluate them on three criteria: identification rate, centroid localization error, and the additional computation time introduced by the extraction step.

Table 5.2 reports the results of the three extraction methods across the three evaluation metrics. These results were obtained using the nnUnet_3dlowres_2.0 configuration, evaluated on the VerSe 2019 validation set. Each column shows the average of a given metric computed over the 40 images in the set.

Table 5.2: *Comparison of centroid extraction methods.*

Method	ID rate (%)	Distance (mm)	Extraction time (s)
Center of mass	97.06	13.04	**0.24**
EDT	**97.37**	10.04	1.10
Morphological opening	96.52	**7.53**	1.15

The difference in computation time between the three methods can be explained by the cost of the operations they rely on. The center of mass method is the most lightweight, as it only requires computing the weighted average of voxel coordinates within each label's mask, which makes it the quickest approach. The EDT method adds a distance transform before the center of mass computation, which increases the time. The morphological opening method is the most expensive, as it applies a binary opening (erosion followed by a dilation) using a large 3D structuring element. This means that for each voxel, both the erosion and the dilation must examine a large surrounding region, making the method slower. Thanks to the bounding box, the difference isn't noticeable, given the small number of voxels to be processed.

As we will see in Section 5.5, the centroid extraction step is not the bottleneck of nnU-Net in terms of computation time, so it is important to also consider the other two metrics. Among the three methods, EDT achieves the highest identification rate at 97.37%. Morphological opening, on the other hand obtains the lowest average distance to the ground truth but also the lowest identification rate, which is counterintuitive. A possible explanation is that morphological opening works well on the majority of vertebrae, successfully removing the thin structures corresponding to the vertebral arch and placing the centroid closer to the vertebral body, which explains the lowest average distance. However, when the predicted mask is inaccurate or the vertebra has an irregular shape, the erosion may fail to fully remove these structures, leaving a partially deformed mask instead. The center of mass is then computed on a distorted region, which can shift the centroid far enough to cause a misidentification. While adjusting the radius of the structuring element could help eliminate the remaining thin structures, finding an optimal value depends on the dataset, making the method harder to generalize. The other two methods do not rely on any hyperparameter, which makes them more robust to different datasets.

Based on these results, we selected EDT as the centroid extraction method for nnU-Net, as it achieves the highest identification rate, a lower average distance than center of mass, and does not depend on any hyperparameter.

5.4.2 Qualitative result

Figure 5.4 shows the centroids obtained by each of the three extraction methods on a prediction from `nnUnet_3d_lowres_2.0`, alongside the ground truth centroids. The visual observations are consistent with the quantitative results reported in Table 5.2. For all three methods, the predicted centroids appear shifted posteriorly with respect to the ground

truth, pulled toward the vertebral arch. This offset is less pronounced for morphological opening, which confirms that the erosion step successfully removes most of the arch before the center of mass is computed. Despite these shifts, the predicted centroids remain close enough to their respective vertebrae that correct identification is never visually ambiguous.

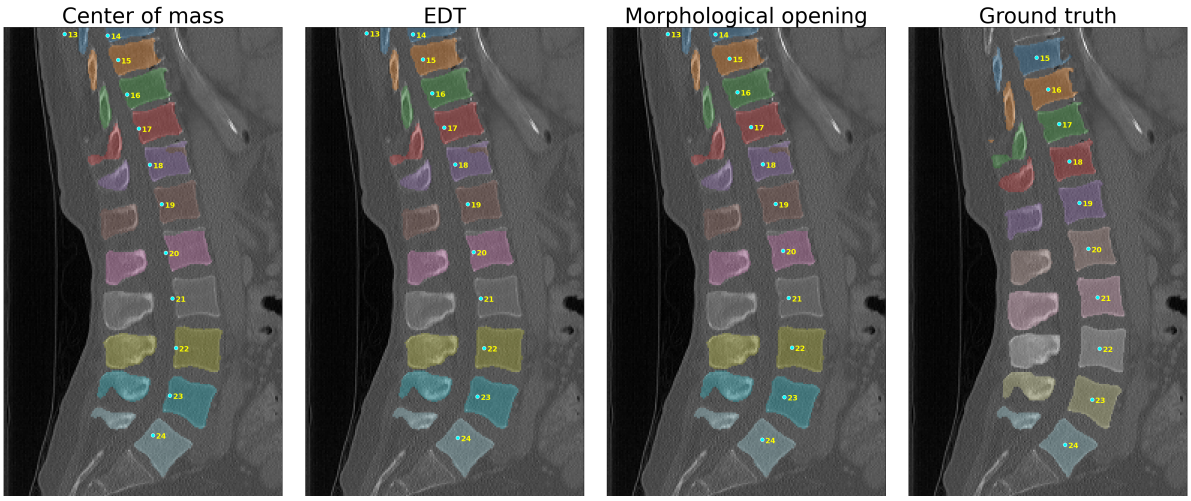


Figure 5.4: *Centroids extracted from the segmentation mask produced by* `nnUnet_3d_lowres_2.0` *using the three proposed methods, compared with the ground truth. The colored overlay on the vertebrae represents the predicted segmentation mask, with one color per label.*

5.5 Results of both frameworks

All experiments of this section were conducted on the computing clusters of the Consortium des Équipements de Calcul Intensif (CECI). Every model was trained for 500 epochs on the training set of VerSe 2019, and evaluated on the corresponding test set. Inference was performed on a compute node with an AMD EPYC 7302 16-Core Processor and an NVIDIA Tesla A100 with 40 GB of RAM, with the pipeline configured to use one preprocessing worker and one postprocessing worker running in parallel with the main inference thread. We are going to compare the configurations in terms of performance and efficiency.

5.5.1 Performance measures

To compare the configurations and identify the most promising one, two metrics were used: the identification rate and the Euclidean distance between the predicted centroid and the ground truth. For both metrics, the mean, standard deviation, and median were computed across all cases of the test set and are reported in Table 5.3. For nnLandmark, each predicted landmark is associated with a confidence score (likelihood). A landmark is

retained only if its likelihood exceeds a threshold of 0.3, which was determined empirically on the validation set.

Table 5.3: *Landmark identification rate and distance error per configuration.*

	Identification rate (%)			Distance (mm)		
Configuration	Mean	SD	Median	Mean	SD	Median
nnLandmark_3d_fullres_1.0	91.17	17.92	100.00	**5.14**	**6.23**	**3.25**
nnLandmark_3d_coarse_2.0	92.05	15.70	100.00	5.68	5.73	4.12
nnLandmark_3d_lowres_1.5	91.27	16.80	100.00	6.70	6.25	4.69
nnLandmark_3d_lowres_2.0	92.17	15.46	100.00	5.81	5.61	4.24
nnLandmark_3d_lowres_2.5	85.23	23.57	100.00	7.51	5.89	6.08
nnUnet_2d_1.0	75.92	26.84	82.84	13.74	7.74	11.28
nnUnet_3d_fullres_1.0	**94.76**	**15.21**	100.00	10.07	4.44	9.34
nnUnet_3d_lowres_2.0	94.26	14.20	100.00	10.12	4.61	9.25

There are many interesting things to extract from this table. To start, we can see that the 2D configuration of nnU-Net performs significantly worse than the 3D versions, both in terms of mean identification rate (75.92% versus over 94%) and median (82.84% versus 100%), making it unsuitable for any real use case. This likely reflects the importance of 3D context in vertebra identification: vertebrae are large, closely interlocked with neighboring vertebrae.

Looking at the 3D configurations, nnU-Net achieves a better identification rate than nnLandmark (94.76% versus 92.17% for the best nnLandmark configuration,(`nnLandmark_3d_lowres_2.0`) while also being slightly more stable, as shown by its lower standard deviation (15.21 versus 15.46). One reason is that segmentation allows nnU-Net to learn more about the structure of each vertebra. During training, it has access to every voxel with the correct label, which allows it to learn the full contour of the vertebrae. nnLandmark, on the other hand, bases its training on a Gaussian blob placed at the center of the vertebra, which inherently provides less structural information. Additionally, since the centroid in nnU-Net is computed from the entire predicted mask, the model is robust to small segmentation errors: minor inaccuracies are smoothed out by the large number of correctly labeled voxels. nnLandmark relies on a single peak, selected as the argmax of the predicted heatmap, which makes it more sensitive to prediction noise. Voxel spacing has a limited impact on nnU-Net, with only a marginal drop from full to low resolution (94.76% to 94.26%). Among the nnLandmark configurations, the low-resolution and coarse variants at 2.0 mm spacing achieve the highest identification rates (92.17% and 92.05% respectively), both above the full-resolution configuration (91.17%). The similar performance of these two variants suggests that, at a 2.0 mm spacing, the number of features per stage has little impact on identification rate. The low-resolution configurations with reduced features hold up similarly at moderate downsampling (spacings of 1.5 and 2.0 mm, with ID rates of 91.27% and 92.17% respectively), suggesting a performance plateau around 2.0 mm spacing. Beyond this point, the resolution becomes too coarse to preserve the spatial detail needed for reliable identification, as shown by the big drop at 2.5 mm (85.23% with a standard deviation of 23.57). Choosing a spacing therefore requires

balancing accuracy against computational cost, with 2.0 mm appearing as the limit.

Despite its stronger identification rate, nnU-Net produces less accurate centroid locations, with a mean distance of 10.07 mm compared to 5.14 mm for the best nnLandmark configuration (`nnLandmark_3d_fullres_1.0`) and 5.68 mm for the coarse variant. This stems from vertebral anatomy: each vertebra consists of an anterior body and a posterior arch, and averaging over the full mask tends to place the centroid somewhere between the two rather than at a clinically meaningful location despite the EDT centroid computation. nnLandmark avoids this problem because it is trained to predict a specific anatomical point directly.

5.5.2 Efficiency measures

To evaluate the computational efficiency of each configuration, timing measurements were collected at every step of the pipeline during inference on the test set. The analysis is organized into subsections, each corresponding to one of the pipeline phases described in Section 5.3.

5.5.2.1 Setup phase

Table 5.4: *Total setup phase execution time breakdown covering both framework initialization and model loading (in seconds).*

Configuration	*File setup*	*Data iterator*	*Pool creation*	*Model loading*	*Weights loading*	***Total***
nnLandmark_3dfullres_1.0	0.46	5.60	0.00	4.95	0.45	11.47
nnLandmark_3dcoarse_2.0	0.49	6.29	0.01	6.28	0.54	13.60
nnLandmark_3dlowres_1.5	0.43	4.96	0.00	3.97	0.14	**9.51**
nnLandmark_3dlowres_2.0	0.40	5.28	0.00	4.66	0.18	10.53
nnLandmark_3dlowres_2.5	0.48	5.74	0.00	4.26	0.15	10.63
nnUnet_2d	0.57	6.05	0.01	6.01	0.56	13.19
nnUnet_3dfullres	0.50	5.50	0.00	4.78	0.69	11.47
nnUnet_3dlowres	0.39	4.93	0.00	4.75	0.73	10.80

The setup phase is dominated by two steps: data iterator initialization and model loading, which together account for most of the total time across all configurations. The remaining steps (file setup, pool creation, and weights loading) are comparatively negligible. Both frameworks show similar setup times overall, with nnU-Net configurations ranging from 10.80 to 13.19 seconds and nnLandmark from 9.51 to 13.60 seconds. The fastest setup belongs to `nnLandmark_3d_lowres_1.5` at 9.51s, thanks to a shorter model loading step (3.97s). The coarse configuration, by contrast, has the longest setup time of all configurations at 13.60s, with a model loading time of 6.28s, which is consistent with the

fact that it retains the full number of features per stage and therefore has a larger model to instantiate. The data iterator runs at comparable levels for both frameworks (4.93–6.05s for nnU-Net versus 4.96–6.29s for nnLandmark), and model loading follows the same pattern. Weights loading is the one step where a consistent difference remains, with nnU-Net requiring 0.56–0.73s compared to 0.14–0.54s for nnLandmark, the gap being driven by the low-resolution nnLandmark variants which have smaller models.

5.5.2.2 Preprocessing phase

Table 5.5: *Time spent on each step of the preprocessing per case for each configuration (in seconds).*

Configuration	Step	Mean	SD	Median
nnLandmark_3dfullres_1.0	Load	0.55	0.87	0.18
	Crop	0.34	0.53	0.10
	Normalization	0.02	0.04	0.01
	Resampling	6.48	12.48	1.49
	Total	7.42	12.95	2.85
nnLandmark_3dcoarse_2.0	Load	0.66	0.99	0.24
	Crop	0.38	0.52	0.17
	Normalization	0.02	0.03	0.01
	Resampling	2.90	3.72	1.26
	Total	3.98	5.24	1.76
nnLandmark_3dlowres_1.5	Load	0.56	0.89	0.17
	Crop	0.35	0.52	0.10
	Normalization	0.02	0.03	0.01
	Resampling	4.82	6.52	1.57
	Total	5.78	7.93	2.04
nnLandmark_3dlowres_2.0	Load	0.56	0.86	0.21
	Crop	0.35	0.52	0.12
	Normalization	0.02	0.03	0.01
	Resampling	2.77	3.72	0.95
	Total	3.72	5.12	1.25
nnLandmark_3dlowres_2.5	Load	0.60	0.94	0.23
	Crop	0.34	0.52	0.13
	Normalization	0.02	0.03	0.01
	Resampling	2.03	2.75	0.77
	Total	3.01	4.23	1.13
nnUnet_2d_1.0	Load	0.67	1.03	0.23
	Crop	0.37	0.52	0.15
	Normalization	0.02	0.03	0.01
	Resampling	6.19	12.06	2.08

Continued on next page

Table 5.5 – continued from previous page

Configuration	Step	Mean	SD	Median
	Total	7.28	12.70	2.72
nnUnet_3dfullres_1.0	Load	0.55	0.82	0.18
	Crop	0.36	0.54	0.11
	Normalization	0.02	0.03	0.01
	Resampling	6.50	12.43	2.08
	Total	7.46	12.98	2.70
nnUnet_3dlowres_2.0	Load	0.55	0.89	0.16
	Crop	0.36	0.61	0.10
	Normalization	0.02	0.04	0.01
	Resampling	3.01	4.57	0.86
	Total	3.96	6.09	1.17

In Table 5.5, resampling dominates the preprocessing phase across all configurations, while the other steps are negligible. Among the nnLandmark variants, resampling gets faster as the target voxel spacing increases: 6.48s at full resolution, 4.82s at 1.5 mm, 2.77s at 2.0 mm, and 2.03s at 2.5 mm, with diminishing gains at coarser spacings. The coarse configuration, which shares the same 2.0 mm target spacing as `nnLandmark_3d_lowres_2.0`, shows a similar resampling time (2.90s versus 2.77s), confirming that resampling cost is determined by the target voxel spacing rather than by the model. This is expected, since a coarser spacing means fewer voxels in the output grid, and therefore fewer interpolation operations to perform.

Comparing the two frameworks at the same target spacing, nnU-Net and nnLandmark show very similar preprocessing times. At full resolution, nnU-Net averages 7.46s versus 7.42s for nnLandmark, and the resampling step itself is nearly identical (6.50s versus 6.48s). At low resolution the gap remains minimal (3.01s versus 2.77s for resampling). Finally, the 2D nnU-Net configuration still requires resampling during preprocessing to match the expected slice resolution, which explains its high resampling time of 6.19s despite only processing individual slices at inference.

5.5.2.3 Inference phase

The inference phase covers the time spent by the neural network to produce a prediction from a preprocessed input volume. Table 5.6 reports the mean, standard deviation, and median inference time per case across the test set.

Across all configurations, there is a large gap between the mean and the median inference time, confirmed by the high standard deviations. This indicates that a few large input volumes disproportionately increase the average, revealing a non-uniform distribution of image sizes in the dataset. As shown in Appendix B, the inference time grows linearly with the number of voxels for all configurations ($R^2 > 0.7$). This effect has a much larger impact on configurations with slow inference, such as nnU-Net, where individual cases can take over a minute, than on nnLandmark, where even the largest images are processed in a few seconds.

Table 5.6: *Inference time per case (in seconds).*

Configuration	Mean	SD	Median
nnLandmark_3dfullres_1.0	3.27	4.13	1.13
nnLandmark_3dcoarse_2.0	0.84	0.92	0.32
nnLandmark_3dlowres_1.5	1.50	1.66	0.71
nnLandmark_3dlowres_2.0	0.77	0.85	0.29
nnLandmark_3dlowres_2.5	**0.38**	**0.43**	**0.15**
nnUnet_2d_1.0	40.32	57.83	9.42
nnUnet_3dfullres_1.0	20.53	28.82	4.16
nnUnet_3dlowres_2.0	4.32	5.67	1.35

The 2D nnU-Net configuration is by far the slowest at inference, averaging 40.32s per case. Processing each slice individually introduces far more overhead than running inference on the full 3D volume at once. The advantage of using the 2D model is that it uses less memory, but there is a significant trade-off in terms of processing time.

The 3D full-resolution nnU-Net is the second slowest at 20.53s on average, which remains too slow for clinical deployment. It also exhibits very high standard deviations (28.82s), and the large gap between its mean and median value (20.53s versus 4.16s for nnUnet_3dfullres). Beyond the effect of resolution, a clear gap exists between the two frameworks at comparable settings: at full resolution, nnLandmark infers in 3.27s versus 20.53s for nnU-Net, and at low resolution (2.0 mm), nnLandmark averages 0.77s versus 4.32s for nnU-Net. This is likely because nnU-Net must predict a segmentation map with as many output channels as there are classes, whereas nnLandmark only regresses a smaller set of heatmaps, resulting in a lighter computational load. Reducing the resolution through a coarser voxel spacing dramatically improves inference time for both frameworks, since fewer voxels need to be processed. The coarse and low-resolution nnLandmark configurations with spacings of 2.0 and 2.5 mm all average under one second (0.84s for coarse, 0.77s for lowres 2.0, and 0.38s for lowres 2.5), with the smallest standard deviations, making them the most consistent and fast enough for real-time clinical use. Notably, the coarse configuration infers in 0.84s on average, close to the low-resolution variant at the same spacing (0.77s), despite retaining the full number of features per stage. This indicates that, at a 2.0 mm spacing, the input volume size is the factor that most influences inference time, and the additional model parameters in the coarse configuration do not introduce a measurable overhead.

5.5.2.4 Postprocessing phase

Table 5.7 breaks down each of these steps to identify where the postprocessing worker spends most of its time.

Table 5.7: *Time spent on each step of the postprocessing per case for each configuration (in seconds).*

Configuration	Step	Mean	SD	Median
nnLandmark_3dfullres_1.0	Resampling	16.26	38.95	3.86
	Mask creation	4.44	7.43	1.46
	Cropping reverse	0.32	0.54	0.10
	Centroid extraction	6.16	10.70	1.81
	Export output	0.03	0.04	0.01
	Total	27.22	50.16	6.93
nnLandmark_3dcoarse_2.0	Resampling	34.87	55.38	10.31
	Mask creation	4.36	7.31	1.39
	Cropping reverse	0.32	0.53	0.10
	Centroid extraction	5.90	10.15	1.79
	Export output	0.04	0.05	0.02
	Total	45.49	73.25	13.61
nnLandmark_3dlowres_1.5	Resampling	38.45	59.24	10.66
	Mask creation	4.39	7.36	1.42
	Cropping reverse	0.32	0.54	0.10
	Centroid extraction	6.03	10.29	1.86
	Export output	0.03	0.04	0.01
	Total	49.23	76.58	14.06
nnLandmark_3dlowres_2.0	Resampling	35.85	56.53	10.80
	Mask creation	4.40	7.42	1.39
	Cropping reverse	0.32	0.54	0.10
	Centroid extraction	6.29	10.84	1.79
	Export output	0.03	0.04	0.01
	Total	46.90	75.14	14.09
nnLandmark_3dlowres_2.5	Resampling	34.87	55.62	10.37
	Mask creation	4.38	7.35	1.37
	Cropping reverse	0.32	0.54	0.10
	Centroid extraction	6.07	10.41	1.65
	Export output	0.03	0.04	0.01
	Total	45.68	73.86	13.51
nnUnet_2d_1.0	Resampling	16.37	39.95	3.45
	Mask creation	25.14	132.84	1.14
	Cropping reverse	0.02	0.02	0.01
	Centroid extraction	0.96	1.66	0.40
	Export output	0.16	0.27	0.04
	Total	42.65	170.18	6.32
nnUnet_3dfullres_1.0	Resampling	17.08	40.27	3.66
	Mask creation	25.67	136.00	1.07
	Cropping reverse	0.02	0.03	0.01
	Centroid extraction	1.02	1.89	0.30
	Export output	0.22	0.78	0.06
	Total	44.01	173.48	6.81

Continued on next page

Table 5.7 – continued from previous page

Configuration	Step	Mean	SD	Median)
nnUnet_3dlowres_2.0	Resampling	37.10	58.17	11.11
	Mask creation	24.99	133.31	0.98
	Cropping reverse	0.02	0.03	0.01
	Centroid extraction	0.85	1.65	0.29
	Export output	0.12	0.19	0.03
	Total	63.07	173.19	12.67

Resampling once again dominates during this postprocessing phase, but the pattern is now reversed compared to preprocessing: low-resolution models are slower. This is expected, since models that downsampled the input to a coarser spacing during preprocessing must now upsample the prediction back to the original resolution, producing a much larger output grid. For instance, nnLandmark at a spacing of 2.5 mm averages 34.87s for resampling, compared to only 16.26s at full resolution. The coarse configuration at 2.0 mm averages 34.87s, closely matching the low-resolution variant at the same spacing (35.85s), which again confirms that the resampling cost is driven by the voxel spacing rather than by the number of model parameters. The speed gained during inference by using a coarser spacing is therefore largely offset by this costly upsampling step in postprocessing.

Mask creation is generally fast for nnLandmark configurations (around 4.4s), while all nnU-Net configurations show highly skewed distributions, averaging around 25s but with medians of 0.98–1.14s and standard deviations above 130, indicating that a few extreme cases dominate the mean. This suggests that nnU-Net is more sensitive to the input image size during this step. By median, however, nnU-Net is slightly faster than nnLandmark (0.98–1.14s versus 1.37–1.46s), since nnU-Net applies a simple argmax over the predicted logits, whereas nnLandmark requires a sigmoid activation on each of the N output channels before the mask can be constructed.

Centroid extraction shows that it is significantly slower for nnLandmark (5.90–6.29s) than for nnU-Net (0.85–1.02s). Although finding a peak on a heatmap is cheaper per channel than computing an Euclidean distance transform on a binary mask, nnLandmark must repeat this operation across N channels, one per vertebral label whereas nnU-Net works on a single-channel segmentation mask. Combined with the fact that centroid extraction is performed on the volume at its original resolution, the per-channel cost adds up quickly for nnLandmark. The coarse configuration is no exception, requiring 5.90s on average for centroid extraction.

5.5.2.5 Total time

While the previous tables reported per case statistics, Table 5.8 measures the total time each configuration took to process all 40 images in the test set. Since both of these frameworks parallelise preprocessing and postprocessing using dedicated workers, the times per stage alone are not sufficient to determine where the pipeline is actually bottlenecking.

To identify that, the time the main thread spends waiting was recorded. The *preprocessing wait* captures how long the main thread waits before a preprocessed image becomes available, while the *export wait* captures how long it must wait for the postprocessing worker to accept the next result. These waiting times make it possible to distinguish between configurations that are fast and those where a fast inference is masked by a slow surrounding stage.

Table 5.8: *Total pipeline execution time breakdown (in seconds).*

		Processing (main thread)				
Configuration	Setup	Preproc. wait	Inference	Export wait	Postproc. wait	**Total**
nnLandmark_3dfullres_1.0	11.47	98.98	130.81	976.59	69.70	**1287.54**
nnLandmark_3dcoarse_2.0	13.60	18.82	33.44	1738.39	54.78	1859.04
nnLandmark_3dlowres_1.5	9.51	54.98	59.88	1847.77	57.06	2029.20
nnLandmark_3dlowres_2.0	10.53	17.51	30.71	1798.67	55.15	1912.58
nnLandmark_3dlowres_2.5	10.63	14.22	15.08	1761.20	53.80	1854.93
nnUnet_2d_1.0	13.19	109.44	1612.74	1156.39	93.25	2985.01
nnUnet_3dfullres_1.0	11.47	132.31	821.16	1350.31	108.19	2423.44
nnUnet_3dlowres_2.0	10.80	16.10	172.68	2263.57	96.57	2559.72

An inverse relationship emerges between inference time and export wait within each framework. When inference is fast, the main thread finishes quickly but must wait for the postprocessing worker to become available before it can send the next prediction from the neural network, leading to a high export wait. However, export wait remains substantial even for the slowest configurations: `nnUnet_2d_1.0`, despite having the longest inference (1612.74s), still shows an export wait of 1156.39s, indicating that postprocessing is a bottleneck across all configurations. This pipeline imbalance particularly affects nnLandmark. As established in earlier sections, a low-resolution or coarse model is much faster at feeding an image through the network, but will be much slower when it comes to postprocessing, where the prediction must be resampled back to the original resolution. The coarse and low-resolution nnLandmark configurations, despite being much faster at inference (15–60s versus 131s for full resolution), end up with a higher total time (1855–2029s versus 1288s). The coarse configuration at 1859.04s is among the fastest of these downsampled nnLandmark variants, close to `nnLandmark_3d_lowres_2.5` at 1854.93s. The speed gained at inference is entirely absorbed by the postprocessing bottleneck, and the main thread spends most of its time inactive, waiting to export. The current nnLandmark pipeline is not optimized to take advantage of its fast inference. The fastest configuration overall is `nnLandmark_3d_fullres_1.0` with 1287.54s, because although its inference is the longest among nnLandmark variants (130.81s), it has the lowest export wait (976.59s) since its postprocessing operates at the same resolution, avoiding the costly upsampling step. By contrast, `nnUnet_2d_1.0` is the slowest with 2985.01s, combining a very long inference time (1612.74s) with a high export wait (1156.39s). nnU-Net configurations are all slower than nnLandmark, as they combine longer inference times with high export waits, confirming that their postprocessing is also a bottleneck. Setup remains a minor contributor to the total time across all configurations because the setup time is only taken into account once for

the entire dataset, and contrary to the other two, preprocessing is not the longest according to the measurement.

5.5.3 Verdict

The previous sections have evaluated several configurations across both performance and efficiency metrics. But for now, none of them are suitable for real-time clinical deployment. Even `nnLandmark_3d_fullres_1.0` that is the fastest configuration, averages 32.19s per prediction, which is far too slow for integration into a clinical workflow. The measurements show that the main bottlenecks are resampling in both preprocessing and postprocessing, workers initialization, model loading, mask creation, and network inference. Most of these steps can be accelerated with minimal impact on identification rate, with one exception that is the inference time. Techniques such as network pruning or knowledge distillation can shorten inference time but they directly affect model performance [63]. Exploring these techniques falls outside the scope of this master thesis and is left for future work.

Since the goal is to optimize efficiency, we will focus on the configurations with the fastest inference times with a relatively good identification rate: `nnLandmark_3d_coarse_2.0` and `nnLandmark_3d_lowres_2.0`. However, `nnUNet_3d_fullres_1.0` achieves an identification rate of 94.76%, substantially higher than both nnLandmark configurations (92.05% and 92.17% respectively). The question is therefore whether this performance gap is large enough to justify the slower inference, or whether the difference could be attributable to random variation.

To answer this, I applied the Wilcoxon signed-rank test, a non-parametric pairwise test [64] suited to our data because the identification rate distribution is highly skewed (median of 1) and the test is robust to outliers. Using `nnLandmark_3d_coarse_2.0` as baseline, there is no significant difference ($p \geq 0.05$) with the nnU-Net 3D variants, despite their higher mean identification rate. The observed gap can therefore be attributed to chance rather than a better performance on this dataset.

We chose to proceed with `nnLandmark_3d_coarse_2.0`, which will serve as the framework and baseline configuration for the rest of the thesis. There are two reasons for this choice. Firstly, the difference in inference time is considerable (0.84s on average compared to 20.53s for nnU-Net). Secondly, nnLandmark does not rely on masks, and therefore avoids the time loss associated with this, whilst achieving relatively similar performance. Among the two nnLandmark candidates, `nnLandmark_3d_coarse_2.0` and `nnLandmark_3d_lowres_2.0`, both achieve nearly identical scores in terms of performance and efficiency. `nnLandmark_3d_lowres_2.0` has a slightly higher identification rate, while `nnLandmark_3d_coarse_2.0` produces a lower mean distance to the ground truth. `nnLandmark_3d_lowres_2.0` is also marginally faster than the coarse variant. We select `nnLandmark_3d_coarse_2.0` as the baseline for the rest of this thesis, `nnLandmark_3d_coarse_2.0` simply as a matter of preference, since the two configurations remain very similar.

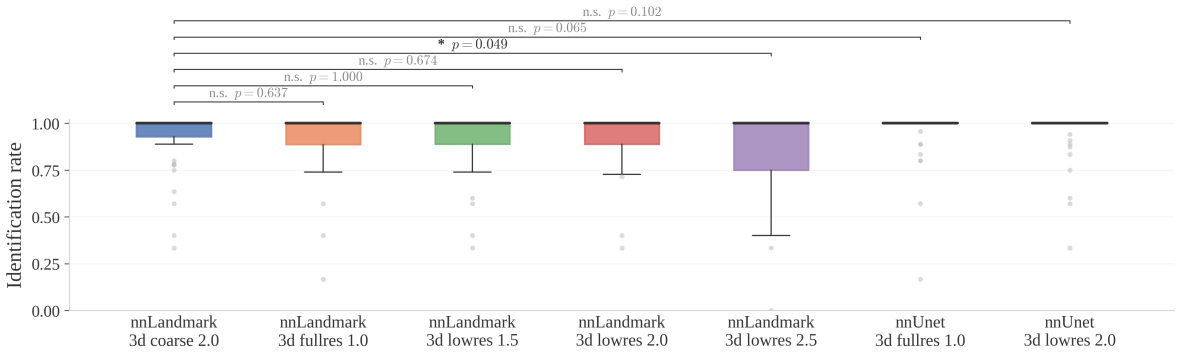


Figure 5.5: *Boxplot of identification rates across configurations. P-values from pairwise Wilcoxon signed-rank tests against `nnLandmark_3d_coarse_2.0` are shown above each comparison (n.s. = not significant).*

CHAPTER 6

IMPROVEMENTS OF NNLANDMARK

The experiments showed that nnLandmark offers the best trade-off between performance and efficiency. However, even in its best configuration, `nnLandmark_3d_coarse_2.0` is not fast enough for clinical use, with high execution times in the setup, preprocessing, and postprocessing phases. In this chapter, we walk through the framework's codebase to identify bottlenecks and introduce optimizations aimed at reducing execution time and improving performance, with the goal of reaching measurements that are suitable for a clinical workflow.

6.1 Removal of background workers

As described in Section 5.5, we observed that a portion of the total inference time was spent on the initialization and synchronization of background workers. The default nnLandmark pipeline is inherited from nnU-Net and relies on two groups of background workers that run as separate operating system processes: the preprocessing workers and the export pool. The first two subsections explain the role of each group of workers and in the following section, an improvement is presented.

6.1.1 Preprocessing workers

nnLandmark uses Python's multiprocessing library to initialize background workers. This package spawns separate operating system processes that execute the preprocessing function in parallel [65]. Each worker picks up one image, applies the full preprocessing pipeline, and returns the result to the main thread, which is responsible for running inference on the GPU. It's possible to specify the number of preprocessing workers using the flag `-npp`. This design is beneficial when predicting multiple images at once, because the workers prepare upcoming images in their own processes while the main thread

performs the forward pass on the current one, so that preprocessing and inference overlap in time.

However, in a clinical setting, it is far more common to process a single image at a time, since the radiologist typically acquires a scan of a patient to produce a study. From there, using their PACS software, they can view multiple series simultaneously and scroll through the slices. This scan is accompanied by previous scans in order to compare studies and monitor the patient's evolution. Here, we assume that the prior studies already contain, in their metadata, JSON files with labels and centroid coordinates (annotations). Therefore, only the new single study needs to be inferred. In this single-image scenario, the parallelism offers no advantage: regardless of whether one or several workers are spawned, the main thread must wait for the sole image to be preprocessed before it can begin the inference. Spawning a worker is an expensive operation because Python's spawn start method must launch a fresh Python interpreter process, re-import all required libraries, including large packages such as `torch` and `numpy`, as well as inter-process communication between the worker and the main thread. For a single image, this initialization does not benefit from parallelism, making the total preprocessing time longer than it would be if the same operations were performed sequentially on the main thread.

6.1.2 Export pool

The second group of background workers is the export pool. After the network produces a prediction for a given image, it is handed off to one of the export pool workers, which is responsible for the postprocessing steps. This allows the main thread to proceed immediately to inference on the next image without waiting for the postprocessing steps to complete.

The export pool is created at the beginning of the prediction loop with the option to specify the number of workers using the flag `-nps`. Like the preprocessing workers, each export pool worker is a full Python process that must be spawned with its own interpreter and a copy of all imported libraries. A mechanism prevents the model from submitting a new prediction if the number of queued tasks exceeds a threshold. It waits until a worker becomes available. As with the preprocessing workers, this design is effective for multiple images prediction but introduces unnecessary overhead for single-image inference. The pool must be spawned before any work can begin. So there is a cost associated with spawning and no benefit from parallelism.

6.1.3 Single-threaded pipeline

To address the efficiency issue when there's only one image, we removed the preprocessing workers and the export pool entirely. Now the preprocessing, inference, and postprocessing run sequentially on the main thread within a single process thus avoiding the need to spawn processes, which is not time-efficient when we expect only a single image.

6.2 New preprocessing

Since nnLandmark inherits from nnU-Net, it reuses its preprocessing pipeline. As explained previously, it begins by loading the image into memory. A transposition is then applied to reorder the spatial axes according to the plans file generated for the dataset. The image is then cropped to the bounding box of non-zero voxels, removing large regions of background. Next, the image is normalized, and finally, it is resampled so that its voxel spacing matches the target spacing defined for the network input. As discussed in Section 6.1, in the default pipeline these preprocessing steps are carried out by background workers running in separate processes, but in the new inference pipeline without background workers, they are performed on the main thread. Beyond this change, several of the preprocessing steps themselves can be optimized, as described in the following subsection.

When we measured the execution times across the different configurations, we observed that preprocessing accounted for a significant portion of the total inference time. This is due to several factors: loading the image into memory, initializing the background workers, cropping the image to remove empty background regions, and resampling to the target spacing. Each of these steps can be optimized, with the exception of image loading, which depends on the file size and disk speed rather than on the pipeline code itself.

6.2.1 Bounding box optimization

Before being fed to the network, each image is cropped to its non-zero region so that empty background margins are discarded and there is less data to process. In the original pipeline, this cropping step works as follows: a binary mask is built by marking every voxel whose intensity is not zero, then `scipy.ndimage.binary_fill_holes` is applied to this mask. The purpose of the hole-filling is to close small cavities inside the body that happen to have a zero intensity, such as air pockets, so that they are not mistaken for background. Without this step, these internal cavities would be excluded from the foreground mask, which means they would not contribute to the mean and standard deviation computed during intensity normalization. Once the holes are filled, a bounding box is computed around the mask and the image is cropped accordingly. The problem is that the hole-filling relies on a flood-fill algorithm, which is expensive on large 3D volumes and so it accounts for a significant proportion of the preprocessing time.

Our optimization consists in removing the hole-filling step and calculating the bounding box directly from the original mask. This approach is safe for two reasons. Firstly, hole filling only affects the voxels located inside the body. It never affects the outermost non-zero voxels along each axis, which are the only ones that determine the bounding box. The cropping is therefore identical with or without the hole filling. Secondly, removing hole filling means that a few internal voxels with a value of zero are no longer counted as part of the foreground when calculating the mean and standard deviation used for intensity normalization. It's not a problem in our dataset because we checked how many images actually contain zero-valued voxels. Out of 160 images of the VerSe 2019 dataset, only 8 do.

If we look at the number of voxels in the volumes, the average proportion of zero-valued voxels per image is 0.14%. So the normalization statistics barely change and the impact is therefore minimal.

6.2.2 GPU resampling

The most time-consuming step in preprocessing is the resampling of the cropped image to the target voxel spacing defined in the plans file. In the default nnU-Net pipeline, this is performed on the CPU using third-order spline interpolation. The interpolation must be evaluated at every voxel in the output grid, which is computationally expensive on large 3D volumes since each output voxel requires a $4 \times 4 \times 4$ neighborhood from the input grid.

To accelerate this step, we replace the CPU-based spline interpolation from Scikit-Image with GPU-accelerated trilinear interpolation from PyTorch. The image tensor is transferred to the GPU, resampled, and moved back to the CPU to continue through the rest of the pipeline. Trilinear interpolation computes each output voxel by linearly weighting the $2 \times 2 \times 2$ closest neighboring [66], which is cheaper per voxel than third-order splines, and in addition, the GPU evaluates all output voxels in parallel, so it will be faster.

This change introduces two trade-offs. The first is a small reduction in resampling accuracy. Trilinear interpolation is a first-order method that produces slightly smoother results, and unlike Scikit-Image's `resize`, PyTorch's `interpolate` does not apply a Gaussian anti-aliasing filter before downsampling, so some high-frequency details may be lost. For landmark detection, these differences are negligible since the network predicts heatmaps whose peaks correspond to vertebra centroids, and neither effect is large enough to shift these peaks, as confirmed by the results in Section 7.1. The second trade-off is the GPU memory. The GPU must hold the input volume and resampled output simultaneously, in addition to the network weights. This is not an issue in our single-threaded pipeline, where the image is resampled, moved back to the CPU, and only then is the forward pass launched, freeing the memory used for resampling. In the original worker-based pipeline, multiple workers would transfer large volumes to the GPU concurrently and compete for memory with the inference computation, which could easily exhaust the available GPU memory. GPU resampling is therefore well suited to the single-threaded pipeline but would require additional memory management in the worker-based one.

6.3 New postprocessing

The postprocessing pipeline that nnLandmark inherits from nnU-Net was described in Sections 3.2.3 and 3.3.3. Several optimizations are possible. As we have seen, nnLandmark reuses a large portion of the nnU-Net code, which means some parts are still present but no longer useful in nnLandmark. In particular, the segmentation files are saved even though they are not used, since we rely on the JSON file containing our centroids. We therefore decided to remove the code that creates the segmentation mask based on the

probabilities of the heatmap generated in each channel and saves this segmentation mask. It did not provide any information necessary for locating and identifying the centroids.

As we saw when analyzing the execution times of the nnLandmark pipeline across different configurations, the most time-consuming phase was the postprocessing. This is not solely explained by the saving of the segmentation. The most expensive step is resampling the network output containing the logits back to the original image shape. This is done via trilinear interpolation on each channel independently. It is very costly because, in addition to being performed on every channel, the interpolation operates over all voxels and runs entirely on the CPU. One way to save time would be to avoid interpolation, and this is the idea we decided to explore. Instead of resampling the full $C \times D \times H \times W$ probability volume back to the original space, we find the peak in network space and scale the coordinates back. To do this, we skip the resampling step and directly apply sigmoid to convert the logits into probabilities, then filter each channel to extract the argmax. We then have our centroids for each channel, but the coordinates are not yet in the correct space. Three operations are needed to bring them back:

1. **Scaling from network space to cropped space:** the network predicts on a grid of shape S_{pred}, whereas the image before resampling had shape S_{crop}. Each coordinate is scaled by the ratio of the two shapes along the corresponding axis:

$$c_{\text{crop}}^{(d)} = c_{\text{pred}}^{(d)} \times \frac{S_{\text{crop}}^{(d)}}{S_{\text{pred}}^{(d)}}, \qquad d \in \{z, y, x\} \tag{6.1}$$

2. **Reverting the cropping:** during preprocessing, the image was cropped to a bounding box. To recover the position in the full image, we add the smallest index along each axis:

$$c_{\text{full}}^{(d)} = c_{\text{crop}}^{(d)} + \text{bbox}_{\text{min}}^{(d)} \tag{6.2}$$

3. **Reverting the transposition:** during preprocessing, axes may have been permuted. The inverse permutation is applied to restore the original axis order:

$$c_{\text{orig}}^{(d)} = c_{\text{full}}^{(\text{transpose\_backward}[d])}, \qquad d \in \{0, 1, 2\} \tag{6.3}$$

Finally, the coordinates are reordered to match the expected output format, and saved alongside the likelihood score in the JSON file.

Once the per-channel peaks are extracted in $O(C \times N_{pred})$, recovering their original-space coordinates costs $O(C)$ (a per-landmark scaling and offset), whereas the original

pipeline recovers coordinates by resampling the full $C \times N_{pred}$ logit volume to $C \times N_{orig}$, that gives a $O(C \times N_{orig})$ interpolation cost, plus the centroid extraction in $O(C \times N_{orig})$, with N_{pred} and N_{orig} the total number of voxels in the predicted and original volumes, respectively. We thus eliminate the resampling entirely.

Beyond efficiency, this approach does not sacrifice accuracy. When the input image has a higher resolution than what the network expects, the original pipeline upsamples the predictions back to that resolution before extracting the argmax. However, trilinear interpolation computes a convex combination of neighboring values, this means the interpolated volume cannot contain a peak that does not already exist at a network grid point. The argmax of the upsampled volume will therefore map back to the same network-space voxel as our direct argmax, and both methods give the same centroid location. Conversely, when the input image has a lower resolution than the network grid, the original pipeline downsamples the predictions to match the input, which can cause the peaks to become shifted. In our approach, the centroid is extracted at the full network resolution before being scaled back, preserving the finer localization learned by the network. Our new method is therefore at least as accurate as the original in all cases, and can even be slightly more precise when the input resolution is coarser than the network resolution.

6.4 Inference acceleration through model conversion

So far, the optimizations discussed have targeted the surrounding pipeline code, but not the neural network itself. For deep learning models to be deployed in medical applications, they must run efficiently on a range of GPUs while maintaining acceptable inference time. nnLandmark uses PyTorch [67], which builds its computational graph dynamically at every forward pass. This flexibility is convenient for research, but it prevents the kind of ahead-of-time optimizations that a static graph representation would allow, resulting in slower and more memory-intensive inference [68]. A common solution is to export the trained model into a more efficient format. In our case, we first export to Open Neural Network Exchange (ONNX) [69] and then build a TensorRT [70] engine, as illustrated in Figure 6.1.

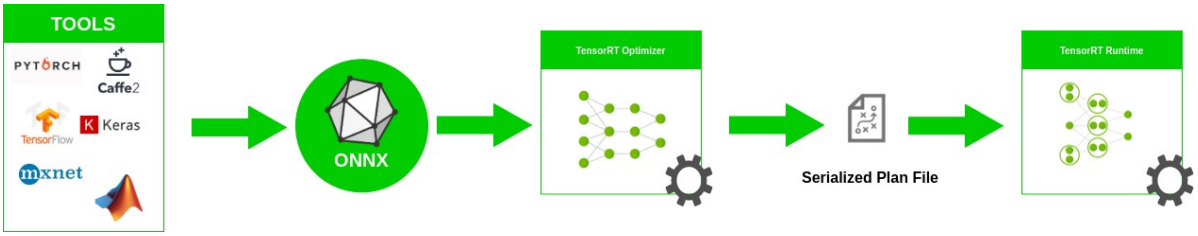


Figure 6.1: *Diagram of the ONNX to TensorRT conversion. A trained model from a framework is first exported to the ONNX format, then optimized and compiled by the TensorRT optimizer into a serialized plan file. The file is loaded by the TensorRT runtime for inference [71].*

6.4.1 ONNX

ONNX is an open-source format that represents a model's computational graph in a static form, defined entirely at export time. Its inference engine, ONNX Runtime, can analyze this graph ahead of time and apply optimizations such as operator fusion, constant folding and graph pruning, reducing the total number of operations and accelerating inference compared to PyTorch [72, 73]. We use the offline optimization mode: the ONNX model is optimized once and saved to disk, so that each inference run loads the pre-optimized graph without repeating the optimization step. It is worth noting that the performance gains depend on the model architecture, as ONNX Runtime can only optimize graph patterns for which it has matching rules.

However, nnLandmark was designed to work exclusively with PyTorch. Rather than modifying the framework to natively support ONNX, we adopted a wrapper approach: a new class inherits from a PyTorch module and overrides the `forward()` method to run inference through ONNX Runtime internally. When inference is launched, the ONNX model is loaded onto the GPU, and nnLandmark's original network is replaced by this wrapper. No code in nnLandmark's prediction pipeline needed to be modified. To avoid the overhead of still loading the original PyTorch weights before the replacement, a condition was added to skip weight loading when ONNX Runtime is used. Since nnLandmark relies on metadata stored in the weight file during initialization, we retrieve this information from the folder name instead, which encodes the configuration and trainer required.

6.4.2 TensorRT

Although ONNX Runtime provides some optimization, it does not guarantee faster inference and can still use a lot of memory. TensorRT is a Software Development Kit (SDK) made by NVIDIA that goes further by compiling the model into a binary engine specifically tuned for the target GPU [74, 72]. It applies layer and tensor fusion to eliminate intermediate memory transfers between operations, and kernel auto-tuning to select the most efficient implementation of each layer for the hardware being used. Another key optimization is quantization: we use FP16 precision, which halves the data size compared to FP32, reducing memory usage and improving speed thanks to dedicated half-precision units on the GPU. The trade-off is a small loss in numerical accuracy, but FP16 is generally considered safe for our use case [72].

To have our TensorRT engine, we followed the ONNX-to-TensorRT conversion path, which according to Zhou and Yang [74] offers the best overall performance for accelerating PyTorch inference when computational resources are limited. One drawback of TensorRT is that each engine is optimized for the specific GPU on which it was built, so running on a different GPU requires rebuilding the engine. To integrate the TensorRT engine into nnLandmark, we followed the same wrapper approach as for ONNX: a new class loads the engine and exposes a `forward()` method that nnLandmark can call as if it were a PyTorch model. The results of these new model formats are presented in Chapter 7.

6.5 Minor improvements on identification rate

6.5.1 Training batch size

In addition to the coarse resolution configurations described earlier, we also modified the batch size in the plan files. The batch size controls how many patches the network processes in one forward pass before computing the loss and updating the weights through backpropagation. Larger batch sizes provide a more accurate estimate of the gradient and are therefore preferred, but are limited by the available hardware memory. By default, nnU-Net prioritizes large patch sizes and adapts the network topology (depth, downsampling operations, feature map sizes) to fit within the GPU memory budget, with a minimum batch size of two [57]. Since our coarser resolution configurations use smaller images and patches, they require less memory during training, which leaves room to increase the batch size. To find the best value, we trained models with batch sizes of 2, 4, 6, and 8, and compared their performance. This change only affects training because the batch size influences how weights are updated but does not alter the architecture or the number of parameters, so inference remains the same.

6.5.2 Centroids extraction based on center of mass

When nnLandmark returns heatmaps with the predicted probabilities for vertebra centroid positions, the current postprocessing works as follows for each channel: voxels with a probability of exactly 1.0 are discarded, only the top 0.5% of voxels (by value) are kept, and the centroid is placed at the argmax of the remaining voxels. This means the final centroid location depends on a single voxel, which is problematic for several reasons. First, if an outlier voxel has an unusually high value, the centroid jumps straight to it, which can lead to a large positioning error that gets counted as a misidentification. Second, discarding all voxels with a probability of 1.0 removes the spots where the model is most confident, so they are completely ignored. Third, when several voxels share the same highest score, argmax always picks the first one in memory order, which means the centroid may be incorrectly positioned.

To fix these issues and make centroid extraction more reliable, we replaced the argmax with a center of mass. For each channel, we take the peak value, keep every voxel whose value is at least 50% of it, and average their positions weighted by their values. The 50% threshold corresponds to the full width at half maximum (FWHM) of the peak, which makes a balance between two extremes. Going lower would let distant, low-confidence voxels pull the centroid away from the true center, while going higher would keep too few voxels and get close to the same problems as an argmax. With a weighted averaging instead of a single voxel, the centroid naturally falls at sub-voxel positions, remains centered when several voxels share the same highest value, and outliers have a much smaller influence on the final prediction since they are averaged out by the surrounding high-probability voxels.

CHAPTER 7

RESULTS

In the previous chapter, we described the improvements made to the nnLandmark pipeline, targeting both identification performance and computational efficiency. In this chapter, we evaluate the impact of these modifications through experiments. The chapter is divided into two sections. Section 7.1 discusses the impact of the performance improvements, grouped into different configurations. We will compare their identification rates and localization distances with those of the original pipeline. Section 7.2 analyses computational efficiency by measuring the execution time of each phase of the pipeline, as well as the total time, in order to determine whether the pipeline improved meets the requirements for clinical deployment. The measure was done on a compute node from CECI with an AMD EPYC 7302 16-Core Processor and on an NVIDIA Tesla A100 with 40 GB of RAM.

7.1 Performances

We first evaluate the impact of the improvements on the identification rate and the localization distance between predicted centroids and ground truth. We begin by determining the optimal batch size, as this hyperparameter affects both training and generalization performance. Once the batch size is selected, we present the results of the incremental configurations and see whether the modifications preserve or improve the identification performance of the original pipeline.

7.1.1 Impact of batch sizes

To evaluate the effect of batch size, we trained several models using the `nnLandmark_3d_coarse_2.0` configuration with batch sizes of 4, 6, and 8. The default batch size of 2 achieves a mean identification rate of 96.13% on the VerSe 2019 validation set. Results for all configurations are reported in Table 7.1.

Table 7.1: *Landmark identification rate and distance error per batch size.*

	Identification rate (%)			Distance (mm)		
Batch size	Mean	Std	Median	Mean	Std	Median
2	**96.13**	**9.23**	100.00	**5.74**	**5.19**	**4.30**
4	93.61	12.62	100.00	6.19	5.19	5.10
6	94.01	14.13	100.00	6.23	5.88	4.65
8	94.27	14.21	100.00	6.03	5.70	4.58

We can see that the identification rate and the distance error vary across batch size values, with a clear trend favoring the smallest batch size. The highest identification rate was achieved with the default batch size of 2, reaching a mean of 96.13% and with the lowest standard deviation (9.23). It has also the best mean localization distance at 5.74 mm. The other batch sizes have around 93-94% with higher standard deviations, suggesting less stable performance. The batch size is a hyperparameter, and its optimal value depends heavily on the training data and how well it represents the overall population. The question is then, which batch size should we select for the remainder of the experiments.

Previous work has studied the influence of batch size on model performance. Keskar et al. (2017) [75] showed that smaller batch sizes produce noisier gradient estimates, which steers the optimization toward flat local minima that generalize better to unseen data. Larger batch sizes give more accurate gradients but settle into sharp minima that are sensitive to distribution shifts between training and test data (see Appendix C for details). Our results are consistent with this finding. Therefore we chosen to keep the default batch size of 2 for the rest of the experiments because it achieves the best scores for identification rate and the lowest distance error, and to ensure that our data generalizes well on the test set.

7.1.2 Analysis of the improvements

In this section, we evaluate the impact of these changes on the identification rate and the distance between predicted landmarks and ground truth, reporting the mean, standard deviation, and median. All measurements were performed on an NVIDIA A100 GPU using the test set of VerSe 2019.

Each configuration builds incrementally on the previous one, as summarized in Table 7.2. The *Baseline* corresponds to the original `nnLandmark_3d_coarse_2.0` pipeline without any modification. The *Efficient* model introduces all runtime optimizations: GPU-based pre-resampling, removal of the unused segmentation export, and coordinate-space centroid extraction. The *Enhanced* model further incorporates the new centroid extraction approach. Finally, *ONNX* and *TensorRT* correspond to the *Enhanced* model exported to these respective formats.

First, we can observe that the *Baseline* and the *Efficient* model achieve very similar scores, with the identification rate dropping marginally from 92.05% to 91.94%. Looking at

Table 7.2: *Incremental model configurations.*

Model	Changes from previous
Baseline	None (original pipeline)
Efficient	+ runtime optimizations
Enhanced	+ new centroid extraction
ONNX	+ ONNX export
TensorRT	+ TensorRT compilation

Table 7.3: *Landmark identification rate and distance error per model configuration, compared with other methods evaluated on VerSe 2019 test set.*

		Identification rate (%)			Distance (mm)		
	Model	Mean	Std	Median	Mean	Std	Median
Ours	Baseline	92.05	15.70	100.00	5.68	5.73	4.12
	Efficient	91.94	15.81	100.00	5.92	5.71	4.24
	Enhanced	93.62	13.65	100.00	5.59	5.07	4.12
	ONNX	93.62	13.65	100.00	5.59	5.07	4.12
	TensorRT	93.62	13.65	100.00	5.59	5.07	4.12
Literature	Payer C. [76]	94.25	–	100.00	4.80	–	3.37
	Lessmann N. [77]	90.42	–	100.00	7.04	–	5.3
	Chen M. [13]	86.73	–	100.00	7.13	–	3.81

the per-case results, this difference corresponds to a single vertebra that was no longer correctly identified. The mean distance also increased slightly from 5.68 mm to 5.92 mm, and the median rose from 4.12 mm to 4.24 mm, indicating that the predictions are a little less precise overall. This small loss in accuracy can be explained by the GPU-based resampling introduced in the preprocessing. Trilinear interpolation is a first-order method, whereas the original CPU-based resampling used third-order spline interpolation and applied a Gaussian anti-aliasing filter before downsampling. So it gives a coarser interpolation. However, the impact remains minimal, the identification rate decreases by only 0.11%, and as we will see in the next section, this small loss is compensated by the efficiency gains it provides.

The *Enhanced* model improved the identification rate by more than one and a half percentage points over the *Efficient*, going from 91.94% to 93.62%. This gain comes from switching the centroid extraction from argmax to center of mass. The mean distance and the standard deviation and the median also decreased slightly, which shows that the change is small overall but corrects the localization in a few difficult cases.

Figure 7.1 illustrates one such case on C5 vertebra, where the left panel with the ground truth, the argmax and the center of mass on the slice. The middle panel shows the full predicted heatmap on its peak slice. It is spread over two vertebrae and its highest value falls in the intervertebral disc, below the C5 vertebra. Because the argmax keeps only this single highest voxel, it places the centroid in the disc, away from the ground truth, and the prediction ends up too far from the correct vertebra to be counted as a valid identification.

The right panel shows the same heatmap after keeping only the voxels above 50% of the peak and summing them across all slices, which is why the color scale is a cumulative probability larger than one. This threshold removes the low-value outskirts, so the region shrinks in diameter and concentrates around C5. Taking the center of mass of this region averages the whole high-confidence core instead of relying on one voxel, which brings the centroid back onto the correct vertebra and makes the prediction valid. It works better here because the useful information lies in the overall shape of the high-confidence region, which is centered on C5, rather than in the exact location of the peak. So averaging over the whole region is therefore more robust to the highest peak voxel that is on the disc, whereas the argmax depends entirely on it.

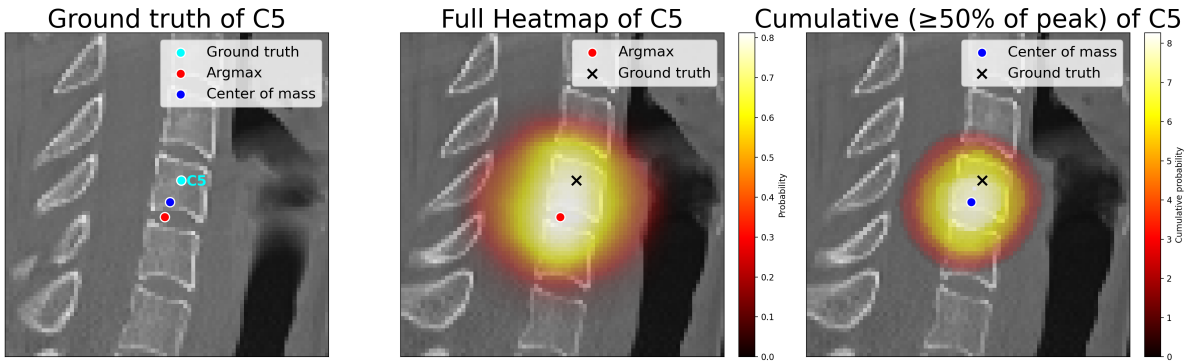


Figure 7.1: *Comparison of the argmax and center of mass centroid extraction on the predicted heatmap of vertebra C5. Left: the three positions on the slice with the ground truth (cyan), argmax (red) and center of mass (green). Middle: the full predicted heatmap on its peak slice. The heatmap spreads over two vertebrae and its peak is in the intervertebral disc, so the argmax (red) is placed below the C5 body, away from the ground truth (black cross). Right: the cumulative probability across all slices after keeping only the voxels above 50% of the peak. The region is more compact and its center of mass (blue) falls back onto the correct vertebra.*

The *ONNX* and *TensorRT* variants show no change in performance as can be seen from the identical scores with *Enhanced*, which is expected since they represent the same network exported to a different format and confirms that the conversion was carried out correctly.

To compare the performance with methods from the literature, the top three methods from the VerSe 2019 [13] challenge rankings were selected. Among them, Payer et al. (2020) [76] has the best ID rate but they report a total runtime of approximately 40 seconds per image on average with an NVIDIA TITAN V GPU. As we will see in the next section, our pipeline processes an image in only a few seconds on average, which makes it significantly more suitable for clinical deployment despite a lower identification rate.

7.2 Measured times

In this section, we evaluate the impact of the improvements described in Chapter 6 on the computational efficiency of the nnLandmark pipeline. The analysis is organized by pipeline phase presented in Section 5.3: setup, preprocessing, inference, postprocessing, and total execution time. All measurements were performed on the test set of VerSe2019 (40 cases) using an NVIDIA A100 GPU. To see the impact of each improvement on execution time, the following configurations were evaluated: *Baseline*, which corresponds to the original `nnLandmark_3d_coarse_2.0` pipeline without any modification, the *Optimized* (referred to as *Efficient* and *Enhanced* in the performance section), *ONNX* and *TensorRT*. The *Baseline* measurements were taken on one worker for preprocessing and one worker for postprocessing phase. All improved configurations use the single-threaded pipeline described in Section 6.1.3, meaning that preprocessing, inference and postprocessing run sequentially on the main thread without background workers.

7.2.1 New setup phase

Table 7.4 reports the setup phase breakdown for each configuration. There are empty entries in the table, this means the corresponding step is no longer used in the configuration and so the time is 0 seconds. The measurements shown in the table correspond to the time taken to set up the framework and the model. This phase takes place before preprocessing captures the first image. The setup phase does not depend on the number of images, as the model is run only once for all 40 images. The time shown is therefore the total time.

Table 7.4: *Total setup phase execution time breakdown covering both framework initialization and model loading (in seconds).*

Configuration	*Wrapper*	*File setup*	*Data iterator*	*Pool creation*	*Model loading*	*Weights loading*	***Total***
Baseline	–	0.49	6.29	0.01	6.28	0.54	13.60
Optimized	–	–	–	–	5.04	0.48	5.53
ONNX	1.10	–	–	–	0.00	0.00	1.10
TensorRT	0.11	–	–	–	0.00	0.00	**0.11**

By comparing the *Baseline* with the *Optimized*, we can immediately see the effect of removing the background workers. In the *Baseline*, the file setup (0.49s), the data iterator initialization (6.29s) and the export pool creation (0.01s) together add up to 6.79s. These three steps have been removed in the *Optimized* because the single-threaded pipeline does not require spawning background workers. The data iterator alone consumed 6.29s in the *Baseline*, which corresponds to the time needed by Python's multiprocessing library to spawn a fresh interpreter, import all required libraries and establish the inter-process communication. There is also a difference in model loading and weights loading, with the

Optimized configuration being faster on average for both. Since no modification was made to these steps, we attribute this to normal variability in the measurements.

The *ONNX* configuration further reduces the setup time to 1.10s. The Wrapper column in Table 7.4 corresponds to the time required to load the *ONNX* or *TensorRT* model through the wrapper class. As explained before, the wrapper replaces the PyTorch model with an ONNX Runtime session or a TensorRT engine, and the class loads the model stored in the corresponding format, including the model and the weights, so everything is loaded in one go. There is no need to initialize the PyTorch network architecture (model loading = 0s) and to load the PyTorch checkpoint file (weights loading = 0s). Instead, the only cost is the wrapper initialization (1.10s), which loads the *ONNX* model file into ONNX Runtime and allocates the GPU resources. There is a 92% time saving compared to *Baseline*.

The *TensorRT* setup is the fastest, taking only 0.11 seconds. It has the same advantages as *ONNX* in that it does not load the PyTorch model and weights. The difference is in the wrapper time. As mentioned earlier, it works by using a serialized file with a simplified computational graph thanks to several node optimizations. In addition, it uses quantization to reduce memory consumption.

7.2.2 New preprocessing

Table 7.5 reports the sequential preprocessing time breakdown where we can see each step involved in this phase. The biggest improvement is in the resampling step, which was identified in Section 5.5 as the most expensive stage of preprocessing. In the *Baseline*, resampling averages 2.90s per image with a high standard deviation of 3.72s and a median of 1.26s. In the *Optimized* configuration, this drops to 0.01s with a standard deviation of 0.02s, effectively making resampling instantaneous. This speedup is a direct consequence of replacing the CPU-based third-order spline interpolation from Scikit-Image with GPU-accelerated trilinear interpolation from PyTorch. The massive parallelism of the GPU allows all output voxels to be computed simultaneously.

The cropping step also benefits significantly from optimization. In the *Baseline*, cropping averages 0.38s with a standard deviation of 0.52s. In the *Optimized* configuration, it drops to 0.01s with a standard deviation of 0.02s. This improvement comes from the bounding box optimization in which we no longer perform the flood-fill operation.

Image loading remains very similar between the two configurations (0.66s versus 0.55s), which is expected since we didn't touch it because it depends on file size and disk speed rather than on pipeline optimizations. Normalization shows a very small increase (0.02s to 0.05s), which is negligible and likely due to measurement variability. Overall, the total preprocessing time per case decreases from a mean of 3.98s to 0.63s, a reduction of 84%. This confirms the effectiveness of the combined improvements. We did not compare them against the *ONNX* and *TensorRT* configurations because they share the exact same preprocessing pipeline as *Optimized*, meaning the processing times would be identical.

Table 7.5: *Time spent on each step of the preprocessing steps per case for each configuration (in seconds).*

Configuration	Step	Mean	Std	Median
Baseline	Load	0.66	0.99	0.24
	Crop	0.38	0.52	0.17
	Normalization	0.02	0.03	0.01
	Resampling	2.90	3.72	1.26
	Total	3.98	5.24	1.76
Optimized	Load	0.55	0.86	0.19
	Crop	0.01	0.02	0.00
	Normalization	0.05	0.09	0.02
	Resampling	0.01	0.02	0.00
	Total	0.63	0.98	0.22

7.2.3 Inference optimization

Table 7.6 reports the inference time per case.

Table 7.6: *Inference time per case (in seconds).*

Configuration	Mean	Std	Median
Baseline	0.84	0.92	0.32
Optimized	0.50	0.57	0.17
ONNX	0.45	0.59	0.10
TensorRT	0.24	0.32	0.06

We can see that *Optimized* reduces the mean, standard deviation, and median compared to the *Baseline*, although no modifications were made to the neural network's topology. One explanation for this speedup is that the entire new pipeline runs on a single thread, which means the tensor is already on the main thread and can be fed directly into the network. In the *Baseline*, background workers indirectly influence the inference time because there is an inter-process communication where the background worker and the main thread exchange data, along with context switches, so they must be added to the model inference time.

The *ONNX* configuration averages 0.45s per case, which is slightly faster than the *Optimized* (0.50s). Although ONNX Runtime can apply improvements to the computational graph such as operator fusion or constant folding, these optimizations can only be applied if the model contains operations that ONNX Runtime knows how to optimize. Since our network is based on a simple U-Net variant with standard convolutional layers, the opportunities for fusion are limited. Nevertheless, ONNX is consistently faster, both in mean (0.45s versus 0.50s) and median (0.10s versus 0.17s), suggesting that even moderate graph-level optimizations provide a small gain.

The *TensorRT* configuration achieves the best results across all three metrics, with a 71% reduction in mean inference time compared to the *Baseline*. This confirms that the

optimizations offered by TensorRT, such as kernel auto-tuning and FP16 quantization, have made our original model more efficient. The absolute time difference remains small when compared on the order of a second, since our original model is already lightweight and the measurements were performed on a high-end GPU. It is therefore more meaningful to consider the relative speedup, which is expected to generalize across different model and hardware configurations.

7.2.4 New postprocessing

Table 7.7 reports the sequential postprocessing time breakdown per case, showing each step involved in this phase for both the *Baseline* and the *Optimized* configuration. Since *ONNX* and *TensorRT* share the same pipeline as *Optimized*, the execution times for the postprocessing phase will be identical, so there is no need to include them in the table.

Table 7.7: *Postprocessing time breakdown per case (in seconds).*

Configuration	Step	Mean	Std	Median
Baseline	Resampling	34.87	55.38	10.31
	Mask creation	4.36	7.31	1.39
	Cropping reverse	0.32	0.53	0.10
	Centroid extraction	5.90	10.15	1.79
	Export output	0.04	0.05	0.02
	Total	45.49	73.25	13.61
Optimized	Resampling	0.00	0.00	0.00
	Mask creation	0.00	0.00	0.00
	Cropping reverse	0.00	0.00	0.00
	Centroid extraction	0.45	0.57	0.14
	Export output	0.00	0.00	0.00
	Total	0.45	0.58	0.14

The postprocessing phase is the part of the pipeline where efficiency has improved the most, as can be seen from the measured times. The *Baseline* averages 45.49s per case, making it by far the most time-consuming phase of the original pipeline, as already identified in Section 5.5.2.4. In the *Optimized* configuration, this drops to 0.45s, a reduction of 99%. This observation is a direct consequence of the new postprocessing approach described in Section 6.3.

The mask creation and export output times both drop to 0s. This is because we no longer compute the segmentation mask nor save the NIfTI segmentation file, since neither is needed for centroid extraction. The mask provided no additional information in our case because the network already outputs one heatmap per channel, from which the centroid can be extracted directly. The only file saved is a lightweight JSON containing the centroid coordinates and their likelihoods, which takes a negligible amount of time to write.

In the *Baseline*, the network output had to be resampled back to the original image resolution before any centroid could be extracted. This resampling step alone averaged

34.87s per case and accounted for the vast majority of the postprocessing time. In the *Optimized* configuration, the resampling and cropping reverse steps are entirely eliminated because the centroids are now extracted directly in network space and their coordinates are mapped back to the original image geometry through a simple scaling and offset. These operations are performed in constant time per landmark, which explains why the time for these two steps is very close to zero second. The only remaining cost is the centroid extraction itself, which drops from 5.90s to 0.45s. The center of mass is computed on the heatmap at network resolution rather than on the full-resolution resampled volume, meaning far fewer voxels need to be processed.

There is always a significant gap between the mean and the median, whether for the *Optimized* or the *Baseline* model, in the centroid extraction results, indicating that certain images cause a significant increase in processing time. However, the standard deviation is much lower relative to the mean for the *Optimized* model, suggesting that the number of cases having an impact has been reduced. Based on our measurements, the new approach is therefore more robust to variations in input image size.

7.2.5 Total time improvement

We were able to analyze the various stages of the pipeline. Now Table 7.8 and Table 7.9 report the total pipeline execution time for each configuration across all 40 cases in the test set. For the *Baseline*, the columns *Preproc. wait*, *Export wait* and *Postproc. wait* correspond to the time the main thread spent waiting for the background workers, as described in Section 5.3. For the *Optimized*, *ONNX* and *TensorRT* configurations, the pipeline runs on a single thread without background workers, so it corresponds to the actual durations of the total preprocessing and postprocessing times.

Table 7.8: *Total pipeline execution time breakdown for Baseline (in seconds).*

		Processing (main thread)				
Configuration	Setup	Preproc. wait	Inference	Export wait	Postproc. wait	**Total**
Baseline	13.60	18.82	33.44	1738.39	54.78	1859.04

Table 7.9: *Total sequential pipeline execution time breakdown (in seconds).*

Configuration	Setup	Preprocessing	Inference	Postprocessing	**Total**
Optimized	5.53	25.20	19.87	18.00	68.60
ONNX	1.10	26.61	17.84	18.68	64.23
TensorRT	0.11	27.43	9.77	18.80	**56.11**

The difference in execution time between the original nnLandmark pipeline on the `nnLandmark_3d_coarse_2.0` configuration and the *Optimized* version is striking. The total time drops from 1859.04s for the *Baseline* to 68.60s for the *Optimized* configuration, a 96% reduction, while preserving the same identification performance. This confirms that

the improvements introduced throughout Chapter 6 have successfully made the framework efficient without sacrificing accuracy. In the *Baseline*, the pipeline was dominated by the export wait (1738.39s), which corresponds to the time the main thread spent waiting for the postprocessing worker to finish resampling the predictions back to the original resolution and to find the centroids. This bottleneck meant that the fast inference time of nnLandmark was entirely wasted, as the main thread spent most of the time waiting. By replacing the costly resampling step with the scaling and offset coordinate mapping in *Optimized* configuration, the bottleneck is entirely removed.

The *ONNX* configuration achieves a comparable total time of 64.23s. As discussed earlier in the analysis of the setup phase, the slight difference with *Optimized* comes from the setup phase (1.10s versus 5.53s), where loading an *ONNX* model through the wrapper is faster than instantiating the full PyTorch architecture and loading its weights. The preprocessing and postprocessing times are nearly identical since both configurations share the same pipeline code for these phases. Inference is also slightly faster on *ONNX* than on *Optimized*.

The *TensorRT* configuration is the fastest overall at 56.11s, driven primarily by its lower inference time (9.77s versus 19.87s for the *Optimized*). The setup phase is also the fastest at 0.11s, since loading a precompiled TensorRT engine requires minimal initialization compared to building a network from a plan file. On a per-case basis, *TensorRT* processes an image in an average of 1.40s, which is within the range required for clinical deployment. In addition, it's much faster than the base model's average of 46.48s.

7.3 Efficiency measurements in different conditions

7.3.1 Context

All the timing measurements presented throughout this chapter were obtained on the computing clusters of the CECI with an NVIDIA A100 GPU. Images were processed consecutively within a single run, so the setup phase was amortized over the entire set. These conditions were well suited to benchmarking, as they allowed us to compare configurations and isolate the impact of each improvement, but they do not capture all conditions under which the model may be deployed in practice. Two factors in particular influence the total execution time: the available hardware and the way images are submitted to the pipeline.

The first factor is the hardware. The model could be deployed on a centralized server with a powerful GPU, whether in a hospital or in the cloud, or directly on the radiologist's local workstation, which is far less powerful and not dedicated to AI inference. To represent these two cases, we use the NVIDIA A100 as the server-class GPU and the NVIDIA RTX 3060 Ti as the workstation GPU. As Telemis recommends an RTX 3060 or RTX 4060 as the minimum GPU for its PACS solution, the RTX 3060 Ti is slightly more powerful but is still

worth trying out and can be a representative choice for a typical clinical workstation.

The second factor is the way images are submitted to the pipeline. In the multi-case scenario, all images are placed in a folder and processed sequentially in a single run, with the model loaded once and kept in memory throughout. This reflects situations where several scans are analyzed together or previous studies are reprocessed, as well as a dedicated server handling a queue of images without reloading the model. In the single-case scenario, the pipeline is restarted for each image: the model is loaded onto the GPU, the image is processed to produce the predictions, and the pipeline shuts down once it is done. This better matches the typical clinical workflow, where a radiologist opens a new study and expects the identification within a few seconds. It is also relevant for servers, since releasing the model when the GPU is idle avoids needless memory use and power consumption. The single-case mode is therefore the most demanding condition, as the full setup overhead is counted in the time for every image.

Combining these two factors gives four scenarios: server with multi-case, server with single-case, workstation with multi-case, and workstation with single-case. By comparing the two processing modes on the same GPU, we can measure how much the setup overhead affects the total time. This overhead includes not only loading the model but also the GPU initialization that occurs the first time a process performs a computation on the device. By comparing the two GPUs on the same processing configuration, we can see how much the hardware matters. Together, these four scenarios cover a range of conditions, giving a clear view of how the pipeline performs in different cases.

7.3.2 Measurements

To evaluate the robustness of our improvements, we therefore measured the performance of each configuration under different combinations of these factors. Table 7.10 reports the end-to-end mean time per case across both GPUs and both scenarios. The *Baseline* configuration is excluded because its execution time, as shown in the previous sections, is too slow to be relevant for clinical deployment. Also in multi-case mode, the per-image time excludes the setup phase, since this scenario represents a dedicated server where the model is loaded once at startup and remains in memory across requests. In single-case mode, setup is included because each image starts a new pipeline. It is worth noting that the library import time takes a few seconds for loading packages such as PyTorch, NumPy, ONNX Runtime, TensorRT but this is not included in these measurements. In a production deployment, this cost can be avoided entirely by keeping the inference process running as a persistent service, so that the libraries are loaded once at startup and remain in memory. The exact strategy depends on how the model is integrated into the PACS software and falls outside the scope of this thesis.

In the multi-case scenario on the A100, all three configurations achieve comparable times between 1.40s and 1.72s per case, confirming that when the setup cost is amortized, the differences between the formats are small. *TensorRT* is the fastest at 1.40s, benefiting from its compiled engine and FP16 quantization. On the RTX 3060 Ti, the times increase to

Table 7.10: *Mean inference time per image for each combination of GPU and processing mode (in seconds).*

	A100		RTX 3060 Ti	
Configuration	Multi-case	Single case	Multi-case	Single case
Optimized	1.72	2.39	2.01	2.50
ONNX	1.60	2.84	2.09	2.30
TensorRT	1.40	1.76	1.58	1.67

between 1.60s and 2.09s, but all configurations remain under 2.1s per case, which is well within an acceptable range for clinical use.

The single-case scenario reveals more significant differences between configurations, since the setup overhead is no longer amortized over multiple images. On the A100, the *Optimized* configuration takes 2.39s per case, an increase of 0.67s compared to multi-case. *ONNX* is slower at 2.84s, because initializing an ONNX Runtime session for every image introduces an overhead that does not exist in the multi-case scenario where the session is created once. *TensorRT*, on the other hand, remains the fastest at 1.76s, with only a 0.36s increase compared to multi-case. This is consistent with the setup phase measurements reported in Section 7.2.1, where loading a precompiled TensorRT engine was shown to be a very lightweight operation.

A surprising result is that *ONNX* in single-case mode runs slower on A100 (2.84s) than on the RTX 3060 Ti (2.30s). Since this observation was not seen in multi-case mode, the cause must be related to the setup and depend on the hardware on which the model is running. This hardware factor also complicates *Optimized* and *ONNX*. The difference between the two is very small, and depending on the scenario, one may be more efficient than the other. Overall, *TensorRT* is the most consistent configuration across all conditions, achieving the lowest times in every scenario and showing minimal sensitivity to both the hardware and the type of image submissions. All configurations remain under 3s per case even in the least favorable condition (*ONNX*, single-case, A100 at 2.84s), which falls within the few-second requirement set by Telemis for clinical integration. TensorRT has therefore clearly proven that it is adapted for deployment, whether the model is integrated as a persistent server-side service or as an on-demand tool on the radiologist's workstation. Using TensorRT offers real benefits, but the other configurations while they take longer on average to process images, also meet deployment requirements in terms of efficiency by taking just a few seconds on average. This would not have been possible without our neural network, which was specifically designed for rapid inference. Furthermore, without the improvements to the framework to reduce the time required for preprocessing and post-processing, TensorRT alone would not have been able to have acceptable results for clinical deployment.

CHAPTER 8

CONCLUSION

8.1 Summary of key findings

This master's thesis focused on developing a deep learning model for automatic vertebra identification on CT images that satisfies two strict constraints for clinical deployment: high identification accuracy and fast execution time. The research evaluated two open-source frameworks, nnU-Net and nnLandmark, comparing them across multiple configurations to select the most promising candidate for optimization with a view to integrating it into a medical application.

Our experiments revealed a clear trade-off between performance and efficiency. The nnU-Net framework achieved the highest identification rate at 94.76%, but its inference was far too slow for real-world clinical use, averaging 20.53s per case at full resolution. Conversely, the nnLandmark model inferred much faster, averaging under 1 second for the coarser configuration but had a lower identification rate of 92.05%. A detailed analysis of the nnLandmark pipeline revealed that the primary bottleneck was the postprocessing phase, where resampling heatmaps back to the original resolution consumed an average of 35 seconds per case. The actual neural network inference represented only a small fraction of the total execution time, which was heavily dominated by preprocessing and postprocessing steps.

After applying a series of architectural optimizations to the nnLandmark pipeline, the total processing time for the 40 images test set was drastically reduced from 1859.04s to 68.60s, and further reduced to 56.11s using a TensorRT engine. This translates to an average execution time of 1.58s (*Optimized*) and 1.40s (*TensorRT*) on an NVIDIA A100 GPU. Even on consumer-grade hardware (NVIDIA RTX 3060 Ti), the average execution time remained around 2 seconds, satisfying the time requirements for clinical workflow

integration. Furthermore, identification performance improved from 92.05% to 93.62% by replacing the argmax extraction with a center-of-mass calculation.

8.2 Contributions to the field

This research provides several practical and methodological contributions in the field of medical imaging. To the best of our knowledge, this is the first time that the newly released nnLandmark framework has been applied to vertebra identification on CT scans. We demonstrated that it generalizes well to this task, delivering competitive accuracy with significantly lower computational cost than segmentation-based approaches like nnU-Net. Additionally, we provided a comparison of both pipelines by breaking down execution times across setup, preprocessing, inference and postprocessing phases rather than just reporting a single total inference time. By publishing this detailed analysis, we are establishing a baseline for researchers, and we have highlighted that pre- and post-processing steps often dominate the total execution time.

Furthermore, we developed an open-source pipeline designed specifically for efficient clinical use (source code available at `https://github.com/Akialess/Efficient-nnLandmark`. The key improvements are the single-threaded execution for single image inference, GPU-accelerated resampling, and extracting centroids via center-of-mass directly in the prediction space by mapping back to the original coordinate system using a simple scaling and offset. We also demonstrated that integrating ONNX and TensorRT (with FP16 quantization) via custom wrappers effectively accelerates inference without modifying the model and the original codebase. Since these optimizations target pipeline-level inefficiencies rather than task-specific design choices, they are highly transferable to other projects built on nnU-Net or nnLandmark. Finally, to address the practical challenges associated with technology transfer, our solution separates the deep learning framework from the clinical software. The pipeline generates a lightweight JSON file containing vertebra labels, their coordinates, and confidence scores, enabling an easy integration with any PACS viewer.

8.3 Limitations and future work

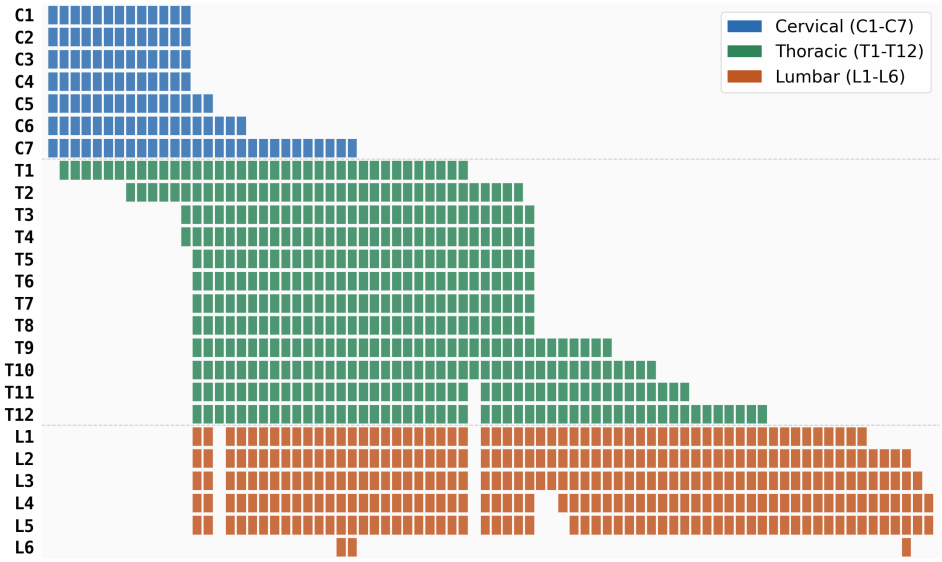
As a feasibility study designed to be completed within a single academic year, this thesis successfully proved that accurate and rapid vertebra identification is achievable for clinical use. However, several limitations remain before production deployment. The main limitation concerns performance. An identification rate of 93.62% is a reasonable result but it is insufficient for real world clinical application and the mean localization distance of approximately 6 mm is still imprecise. To improve the performance in a future work, we can train the model on more diverse datasets, like VerSe 2020, CTSpine1K and BioMedIA to improve generalization. This is especially important given that the VerSe 2019 dataset lacks full-spine images and completely omits the T13 transitional vertebra.

To correct network misclassifications, we can also integrate postprocessing anatomical constraints that enforce vertebral ordering, position and spacing. For example, Bürgin et al. [78] propose a Graph Neural Network that refines the local maxima extracted from U-Net heatmaps.

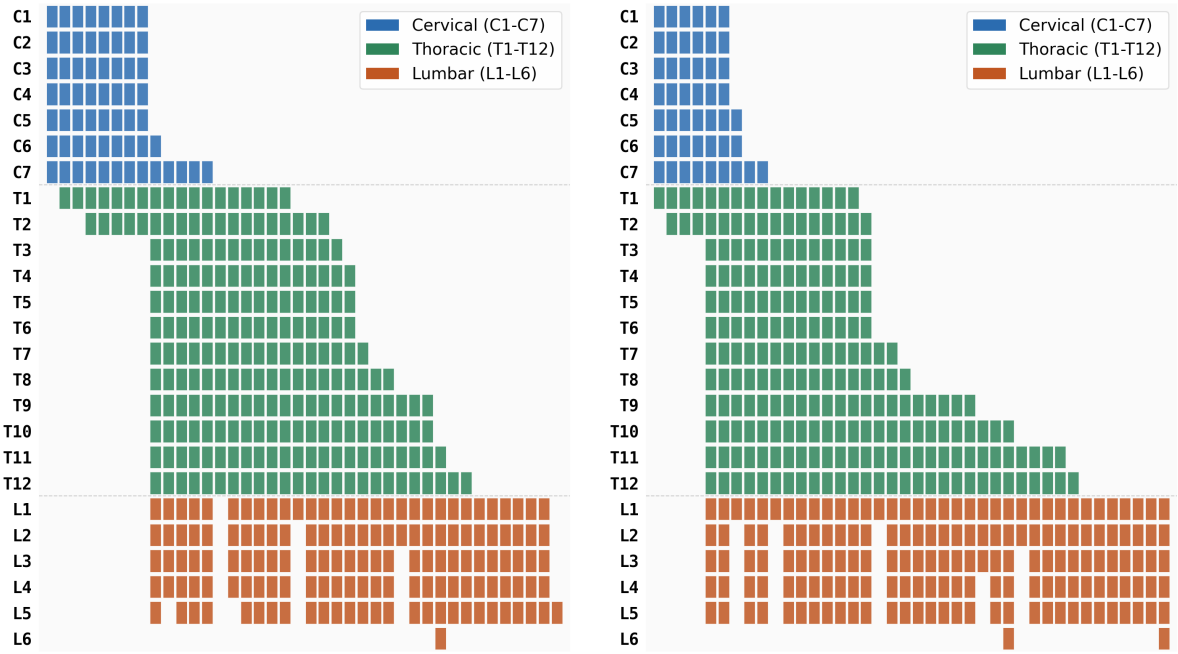
Beyond performance, hardware limits and clinical integration present additional challenges. While the pipeline performs well on an RTX 3060 Ti, it should be benchmarked on lower-end GPUs or CPU-only setups to test extreme hardware limits. If time requirements are not met on weaker hardware, model compression techniques like pruning and knowledge distillation should be explored. This work only covers the deep-learning pipeline with design, training and optimization. Deploying a model in a clinical environment raises additional challenges. For instance, the pipeline will require adding support for the DICOM format, as the pipeline currently relies on NIfTI volumes. In addition, a deployment strategy would have to be defined: running inference on a dedicated server offers scalability but adds infrastructure cost, while executing directly on the radiologist's workstation is cheaper to set up but depends on the available hardware. This hardware dependency is compounded by TensorRT, as its engines must be explicitly rebuilt for each specific GPU architecture. Once these questions are resolved, the next step would be to integrate the model into a PACS viewer to conduct a study with radiologists to see if the speed gains observed in this thesis translate into a real reading time reduction.

APPENDIX A

DISTRIBUTION OF VERTEBRAE ACROSS VERSE 2019



(a) *Training set.*



(b) *Validation set.* **(c)** *Test set.*

Figure A.1: *Overview of vertebrae distribution. Each column represents a CT image. Each row represents the presence of one of the 25 vertebrae, from C1 to L6.*

APPENDIX B

INFERENCE TIMES SCALING WITH IMAGE SIZE

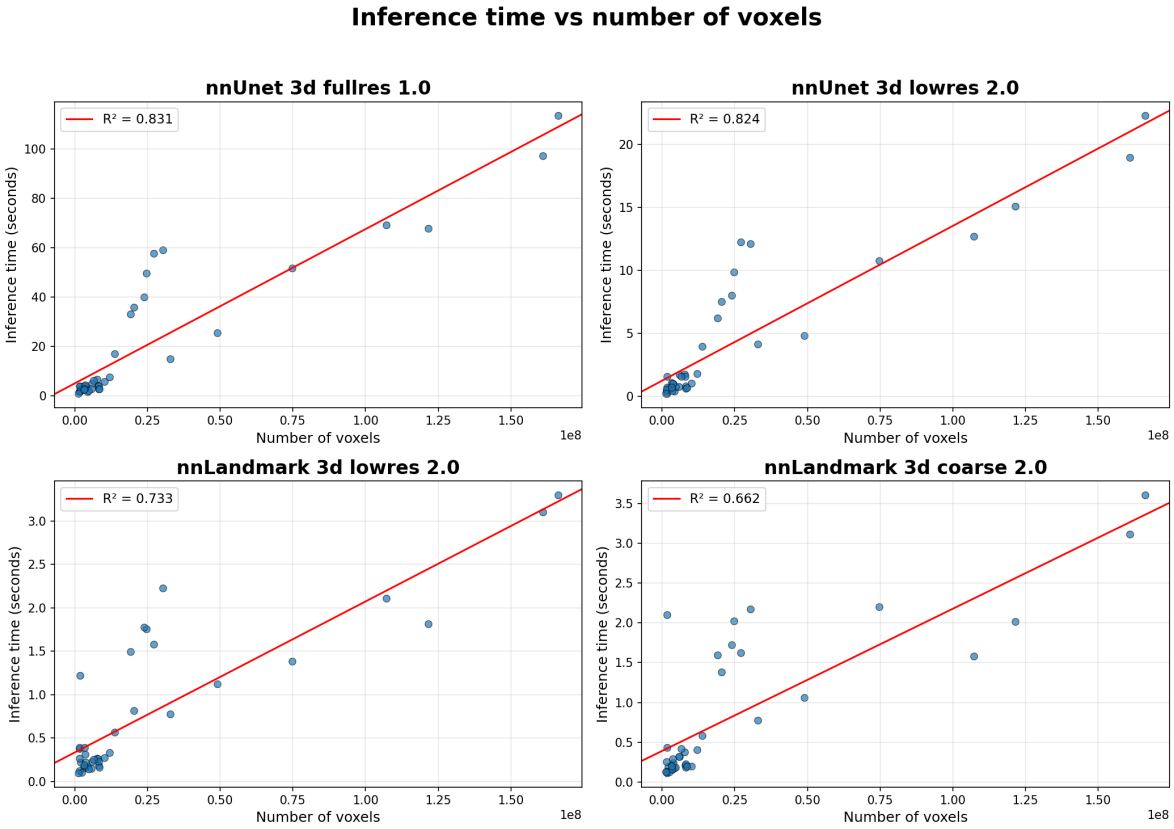


Figure B.1: *Inference time versus number of voxels for each configuration. Each point is a single image. Here the y-axis range is the same for every subplot so the linear trend is visible even for nnLandmark, where inference takes only a few seconds*

For all configurations, the inference time scales linearly with the number of voxels in the input image, as shown by the fitted regression lines. The R^2 values range from 0.662 to 0.831, confirming that image size is the primary factor driving inference time. Figure B.1 uses an adapted y-axis per subplot to reveal the linear trend even for the nnLandmark configurations, where inference times remain under a few seconds. Figure B.2 uses a shared y-axis across all subplots, which makes clear the large absolute difference between nnU-Net and nnLandmark.

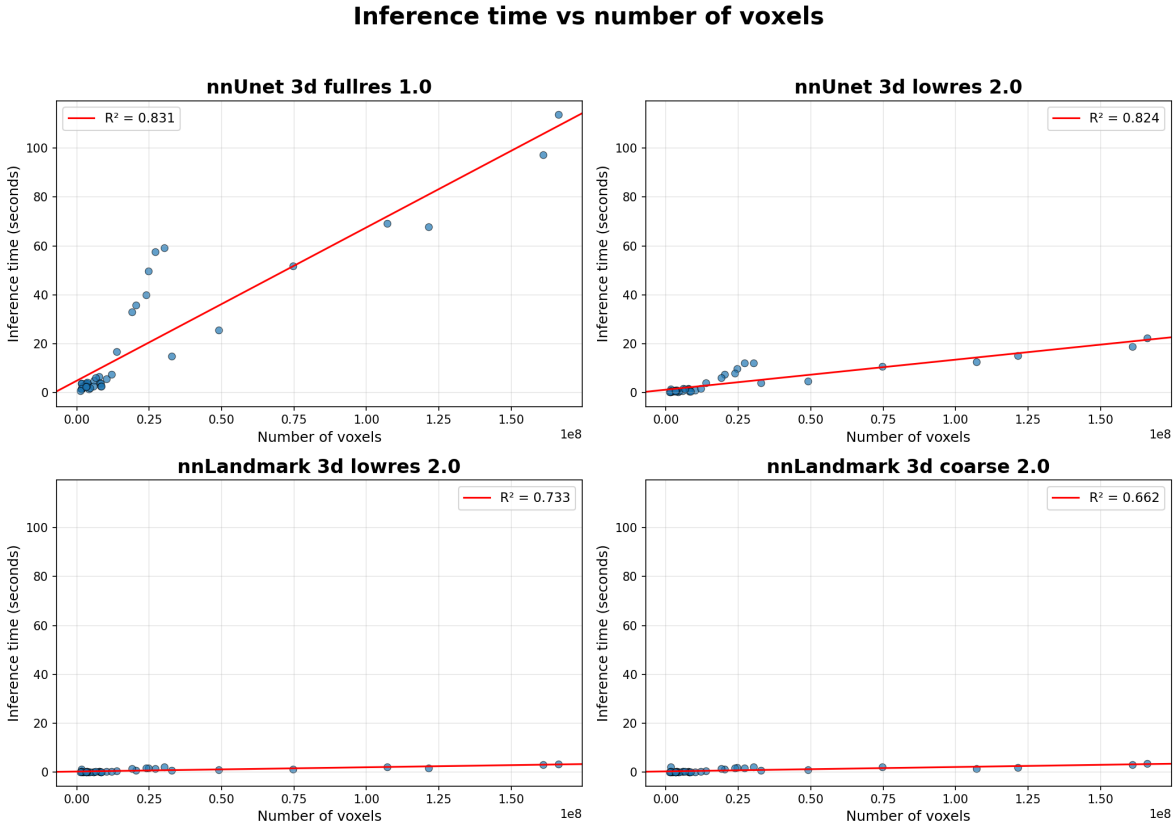


Figure B.2: *Same data as Figure B.1, but with a shared y-axis to show how much faster nnLandmark is compared to nnU-Net in absolute terms.*

APPENDIX C

INFLUENCE OF BATCH SIZE ON GENERALIZATION

Keskar et al. (2017) [75] showed that smaller batch sizes produce noisier gradient estimation, since the model sees fewer data points at each step. Counterintuitively, this noise is beneficial: it steers the optimization toward flat local minima, which tend to generalize better to unseen data. In contrast, larger batch sizes yield more accurate gradient estimates with less noise, which causes the optimizer to follow the steepest descent path and settle into sharp local minima. While these sharp minima fit the training data well, they are highly sensitive to small changes in the data distribution, leading to poorer generalization on new samples.

This trade-off is illustrated in Figure C.1. The black curve represents the training loss function, while the dashed red curve represents the testing loss function. At the flat minimum on the left, both curves reach similarly low values, meaning the model performs well on both training and test data. At the sharp minimum on the right, the training loss is low but the testing loss is considerably higher, because even a slight shift in the data distribution causes the loss to increase.

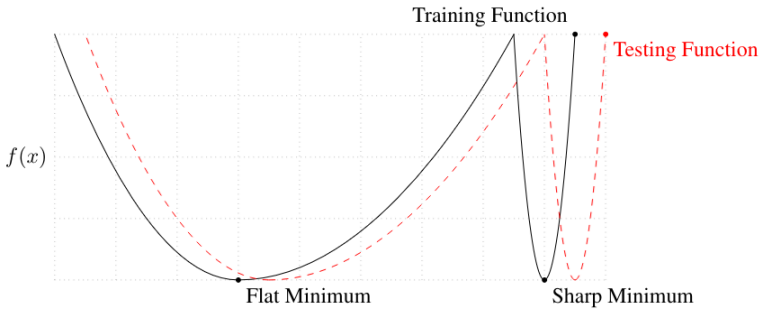


Figure C.1: *Example of a flat and sharp minima. The Y-axis indicates value of the loss function and the X-axis the variables (parameters) [75]*

BIBLIOGRAPHY

[1] Geert Litjens et al. “A survey on deep learning in medical image analysis.” In: *Medical Image Analysis* 42 (2017), pp. 60–88. ISSN: 1361-8415. DOI: https://doi.org/10.1016/j.media.2017.07.005. URL: https://www.sciencedirect.com/science/article/pii/S1361841517301135.

[2] Ibomoiye Domor Mienye et al. “Deep Convolutional Neural Networks in Medical Image Analysis: A Review.” In: *Information* 16.3 (2025). ISSN: 2078-2489. DOI: 10.3390/info16030195. URL: https://www.mdpi.com/2078-2489/16/3/195.

[3] A. Wu et al. “Global low back pain prevalence and years lived with disability from 1990 to 2017: estimates from the Global Burden of Disease Study 2017.” In: *Annals of Translational Medicine* 8.6 (Mar. 2020), p. 299. DOI: 10.21037/atm.2020.02.175.

[4] F. Ruiz Santiago et al. “The role of radiography in the study of spinal disorders.” In: *Quantitative Imaging in Medicine and Surgery* 10.12 (Dec. 2020), pp. 2322–2355. DOI: 10.21037/qims-20-1014.

[5] C. J. Donnally 3rd et al. “The 100 most influential spine fracture publications.” In: *Journal of Spine Surgery* 5.1 (Mar. 2019), pp. 97–109. DOI: 10.21037/jss.2019.01.03.

[6] M. Ciftdemir et al. “Tumors of the spine.” In: *World Journal of Orthopedics* 7.2 (Feb. 2016), pp. 109–116. DOI: 10.5312/wjo.v7.i2.109.

[7] Dong Yang et al. “Automatic Vertebra Labeling in Large-Scale 3D CT Using Deep Image-to-Image Network with Message Passing and Sparsity Regularization.” In: *Information Processing in Medical Imaging*. Ed. by Marc Niethammer et al. Cham: Springer International Publishing, 2017, pp. 633–644.

[8] Robert J. McDonald et al. “The Effects of Changes in Utilization and Technological Advancements of Cross-Sectional Imaging on Radiologist Workload.” In: *Academic Radiology* 22.9 (2015), pp. 1191–1198. ISSN: 1076-6332. DOI: https://doi.org/10.1016/j.acra.2015.05.007. URL: https://www.sciencedirect.com/science/article/pii/S1076633215002457.

[9] Elizabeth A. Krupinski et al. “Long Radiology Workdays Reduce Detection and Accommodation Accuracy.” In: *Journal of the American College of Radiology* 7.9

(2010), pp. 698–704. ISSN: 1546-1440. DOI: https://doi.org/10.1016/j.jacr.2010.03.004. URL: https://www.sciencedirect.com/science/article/pii/S1546144010001341.

[10] Stephen Waite et al. "Interpretive Error in Radiology." In: *American Journal of Roentgenology* 208.4 (2017), pp. 739–749. DOI: 10.2214/AJR.16.16963. eprint: https://doi.org/10.2214/AJR.16.16963. URL: https://doi.org/10.2214/AJR.16.16963.

[11] M. G. Mody et al. "The prevalence of wrong level surgery among spine surgeons." In: *Spine (Phila Pa 1976)* 33.2 (Jan. 2008), pp. 194–198. DOI: 10.1097/BRS.0b013e31816043d1.

[12] Telemis. *Company information*. URL: https://www.telemis.com/fr/societe.

[13] Anjany Sekuboyina et al. "VerSe: A Vertebrae labelling and segmentation benchmark for multi-detector CT images." In: *Medical Image Analysis* 73 (2021), p. 102166. ISSN: 1361-8415. DOI: https://doi.org/10.1016/j.media.2021.102166. URL: https://www.sciencedirect.com/science/article/pii/S1361841521002127.

[14] Brain spine clinic. *Spine anatomy*. URL: http://www.brainandspineclinic.com/spine/spine-anatomy.

[15] OpenStax. *Anatomy and Physiology*. URL: https://anatomytool.org/content/openstax-anatphys-fig114-planes-body-english-labels.

[16] Vishy Mahadevan. "Anatomy of the vertebral column." In: *Surgery (Oxford)* 36.7 (2018), pp. 327–332. ISSN: 0263-9319. DOI: https://doi.org/10.1016/j.mpsur.2018.05.006. URL: https://www.sciencedirect.com/science/article/pii/S0263931918300978.

[17] G.P. Konin and D.M. Walz. "Lumbosacral Transitional Vertebrae: Classification, Imaging Findings, and Clinical Relevance." In: *American Journal of Neuroradiology* 31.10 (2010), pp. 1778–1786. ISSN: 0195-6108. DOI: 10.3174/ajnr.A2036. eprint: https://www.ajnr.org/content/31/10/1778.full.pdf. URL: https://www.ajnr.org/content/31/10/1778.

[18] C. K. Chiu et al. "Variations in the Number of Vertebrae, Prevalence of Lumbosacral Transitional Vertebra and Prevalence of Cervical Rib Among Surgical Patients With Adolescent Idiopathic Scoliosis: An Analysis of 998 Radiographs." In: *Spine (Phila Pa 1976)* 49.1 (Jan. 2024), pp. 64–70. DOI: 10.1097/BRS.0000000000004711.

[19] Y. Z. Yan et al. "Rate of presence of 11 thoracic vertebrae and 6 lumbar vertebrae in asymptomatic Chinese adult volunteers." In: *Journal of Orthopaedic Surgery and Research* 13.1 (May 2018), p. 124. DOI: 10.1186/s13018-018-0835-9.

[20] Fnu Neha et al. “An analytics-driven review of U-Net for medical image segmentation.” In: *Healthcare Analytics* 8 (2025), p. 100416. ISSN: 2772-4425. DOI: https://doi.org/10.1016/j.health.2025.100416. URL: https://www.sciencedirect.com/science/article/pii/S2772442525000358.

[21] Xiangrui Li et al. “The first step for neuroimaging data analysis: DICOM to NIfTI conversion.” In: *Journal of Neuroscience Methods* 264 (2016), pp. 47–56. ISSN: 0165-0270. DOI: https://doi.org/10.1016/j.jneumeth.2016.03.001. URL: https://www.sciencedirect.com/science/article/pii/S0165027016300073.

[22] Rebecca Jordan. *Anatomical Planes*. URL: https://geekymedics.com/anatomical-planes/.

[23] 3D Slicer. *Coordinate systems*. URL: https://slicer.readthedocs.io/en/latest/user_guide/coordinate_systems.html.

[24] Ibomoiye Domor Mienye and Theo G. Swart. “A Comprehensive Review of Deep Learning: Architectures, Recent Advances, and Applications.” In: *Information* 15.12 (2024). ISSN: 2078-2489. DOI: 10.3390/info15120755. URL: https://www.mdpi.com/2078-2489/15/12/755.

[25] Databricks Staff. *What is Deep Learning?* URL: https://www.databricks.com/blog/what-is-deep-learning.

[26] Y. Lecun et al. “Gradient-based learning applied to document recognition.” In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[27] Matthew D. Zeiler and Rob Fergus. “Visualizing and Understanding Convolutional Networks.” In: *CoRR* abs/1311.2901 (2013). arXiv: 1311.2901. URL: http://arxiv.org/abs/1311.2901.

[28] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “U-Net: Convolutional Networks for Biomedical Image Segmentation.” In: *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*. Ed. by Nassir Navab et al. Cham: Springer International Publishing, 2015, pp. 234–241.

[29] Benlahmar. *Architecture U-Net: Une explication détaillée*. URL: https://datasciencetoday.net/index.php/en-us/deep-learning/228-unet.

[30] Ruochen Feng. “Key Technologies Analysis of Deep Learning-based Medical Image Segmentation.” In: *Applied and Computational Engineering* 160 (June 2025), pp. 175–182. DOI: 10.54254/2755-2721/2025.TJ23590.

[31] Amith Kamath et al. “The impact of U-Net architecture choices and skip connections on the robustness of segmentation across texture variations.” In: *Computers in Biology and Medicine* 197 (2025), p. 111056. ISSN: 0010-4825. DOI:

https://doi.org/10.1016/j.compbiomed.2025.111056. URL: https://www.sciencedirect.com/science/article/pii/S0010482525014088.

[32] Benoît Naegel. "Using mathematical morphology for the anatomical labeling of vertebrae from 3D CT-scan images." In: *Computerized Medical Imaging and Graphics* 31.3 (2007), pp. 141–156. ISSN: 0895-6111. DOI: https://doi.org/10.1016/j.compmedimag.2006.12.001. URL: https://www.sciencedirect.com/science/article/pii/S0895611106001492.

[33] D. L. Pham, C. Xu, and J. L. Prince. "Current methods in medical image segmentation." In: *Annual Review of Biomedical Engineering* 2 (2000), pp. 315–337. DOI: 10.1146/annurev.bioeng.2.1.315.

[34] Tobias Heimann and Hans-Peter Meinzer. "Statistical shape models for 3D medical image segmentation: A review." In: *Medical Image Analysis* 13.4 (2009), pp. 543–563. ISSN: 1361-8415. DOI: https://doi.org/10.1016/j.media.2009.05.004. URL: https://www.sciencedirect.com/science/article/pii/S1361841509000425.

[35] Tobias Klinder et al. "Automated model-based vertebra detection, identification, and segmentation in CT images." In: *Medical Image Analysis* 13.3 (2009), pp. 471–482. ISSN: 1361-8415. DOI: https://doi.org/10.1016/j.media.2009.02.004. URL: https://www.sciencedirect.com/science/article/pii/S1361841509000085.

[36] H. Dang, J. H. Siewerdsen, and J. W. Stayman. "Prospective regularization design in prior-image-based reconstruction." In: *Physics in Medicine & Biology* 60.24 (Dec. 2015), pp. 9515–9536. DOI: 10.1088/0031-9155/60/24/9515.

[37] D.H. Ballard. "Generalizing the Hough transform to detect arbitrary shapes." In: *Pattern Recognition* 13.2 (1981), pp. 111–122. ISSN: 0031-3203. DOI: https://doi.org/10.1016/0031-3203(81)90009-1. URL: https://www.sciencedirect.com/science/article/pii/0031320381900091.

[38] Ben Glocker et al. "Automatic Localization and Identification of Vertebrae in Arbitrary Field-of-View CT Scans." In: *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2012*. Ed. by Nicholas Ayache et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 590–598.

[39] Sheng Zhang et al. "SpineCLUE: Automatic vertebrae identification using contrastive learning and uncertainty estimation." In: *Artificial Intelligence in Medicine* 171 (2026), p. 103285. ISSN: 0933-3657. DOI: https://doi.org/10.1016/j.artmed.2025.103285. URL: https://www.sciencedirect.com/science/article/pii/S0933365725002209.

[40] Amin Suzani et al. "Fast Automatic Vertebrae Detection and Localization in Pathological CT Scans - A Deep Learning Approach." In: *Medical Image Computing*

and Computer-Assisted Intervention – MICCAI 2015. Ed. by Nassir Navab et al. Cham: Springer International Publishing, 2015, pp. 678–686.

[41] Zhuofan Xie et al. "Deep learning for automatic vertebra analysis: A methodological survey of recent advances." In: *Computerized Medical Imaging and Graphics* 125 (2025), p. 102652. ISSN: 0895-6111. DOI: https://doi.org/10.1016/j.compmedimag.2025.102652. URL: https://www.sciencedirect.com/science/article/pii/S0895611125001612.

[42] Rhydian Windsor and Amir Jamaludin. "The Ladder Algorithm: Finding Repetitive Structures in Medical Images by Induction." In: *2020 IEEE 17th International Symposium on Biomedical Imaging (ISBI)*. 2020, pp. 1729–1733. DOI: 10.1109/ISBI45749.2020.9098469.

[43] Ken Chatfield et al. "Return of the Devil in the Details: Delving Deep into Convolutional Nets." In: *CoRR* abs/1405.3531 (2014). arXiv: 1405.3531. URL: http://arxiv.org/abs/1405.3531.

[44] Chunli Qin et al. "Vertebrae labeling via end-to-end integral regression localization and multi-label classification network." In: *IEEE Transactions on Neural Networks and Learning Systems* 33.6 (2021), pp. 2726–2736.

[45] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory." In: *Neural Computation* 9.8 (1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735.

[46] Haofu Liao, Addisu Mesfin, and Jiebo Luo. "Joint Vertebrae Identification and Localization in Spinal CT Images by Combining Short- and Long-Range Contextual Information." In: *IEEE Transactions on Medical Imaging* 37.5 (2018), pp. 1266–1275. DOI: 10.1109/TMI.2018.2798293.

[47] Chuan Qin et al. "Long short-term memory with activation on gradient." In: *Neural Networks* 164 (2023), pp. 135–145. ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2023.04.026. URL: https://www.sciencedirect.com/science/article/pii/S0893608023002125.

[48] Ashish Vaswani et al. "Attention is all you need." In: *Advances in neural information processing systems* 30 (2017).

[49] Rong Tao, Wenyong Liu, and Guoyan Zheng. "Spine-transformers: Vertebra labeling and segmentation in arbitrary field-of-view spine CTs via 3D transformers." In: *Medical Image Analysis* 75 (2022), p. 102258. ISSN: 1361-8415. DOI: https://doi.org/10.1016/j.media.2021.102258. URL: https://www.sciencedirect.com/science/article/pii/S1361841521003030.

[50] Alaa El Ichi and Khalide Jbilou. *A Computationally Efficient Multidimensional Vision Transformer*. 2026. arXiv: 2602.19982 [cs.LG]. URL: https://arxiv.org/abs/2602.19982.

[51] Joseph Redmon et al. "You only look once: Unified, real-time object detection." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 779–788.

[52] Zoumana Keita. *YOLO Object Detection Explained*. URL: https://www.datacamp.com/blog/yolo-object-detection-explained.

[53] Ruyi Zhang et al. "VDVM: An automatic vertebrae detection and vertebral segment matching framework for C-arm X-ray image identification." In: *Journal of X-ray Science and Technology* 31.5 (2023), pp. 935–949.

[54] Abdussalam Elhanashi et al. "Generative AI and the Foundation Model Era: A Comprehensive Review." In: *Big Data and Cognitive Computing* 10.3 (2026). ISSN: 2504-2289. DOI: 10.3390/bdcc10030094. URL: https://www.mdpi.com/2504-2289/10/3/94.

[55] Yinhao Wu et al. "VertFound: Synergizing Semantic and Spatial Understanding for Fine-Grained Vertebrae Classification via Foundation Models." In: *Medical Image Computing and Computer Assisted Intervention – MICCAI 2024*. Ed. by Marius George Linguraru et al. Cham: Springer Nature Switzerland, 2024, pp. 763–772.

[56] Rishi Bommasani et al. "On the Opportunities and Risks of Foundation Models." In: *CoRR* abs/2108.07258 (2021). arXiv: 2108.07258. URL: https://arxiv.org/abs/2108.07258.

[57] F. Isensee et al. "nnU-Net: a self-configuring method for deep learning-based biomedical image segmentation." In: *Nature Methods* 18.2 (2021), pp. 203–211.

[58] Geert Litjens et al. "A survey on deep learning in medical image analysis." In: *Medical Image Analysis* 42 (2017), pp. 60–88. ISSN: 1361-8415. DOI: https://doi.org/10.1016/j.media.2017.07.005. URL: https://www.sciencedirect.com/science/article/pii/S1361841517301135.

[59] Fakai Wang et al. "Automatic Vertebra Localization and Identification in CT by Spine Rectification and Anatomically-constrained Optimization." In: *CoRR* abs/2012.07947 (2020). arXiv: 2012.07947. URL: https://arxiv.org/abs/2012.07947.

[60] Fabian Isensee et al. "nnU-Net: Self-adapting Framework for U-Net-Based Medical Image Segmentation." In: *CoRR* abs/1809.10486 (2018). arXiv: 1809.10486. URL: http://arxiv.org/abs/1809.10486.

[61] Alexandra Ertl et al. *nnLandmark: A Self-Configuring Method for 3D Medical Landmark Detection*. 2026. arXiv: 2504.06742 [cs.CV]. URL: https://arxiv.org/abs/2504.06742.

[62] Scott A. Sheffield MS. *Lumbar Vertebrae Anatomy*. URL: https://www.getbodysmart.com/vertebral-column/lumbar-vertebrae/.

[63] Cuong Manh Nguyen and Truong-Son Hy. *Efficient Deep Learning for Medical Imaging: Bridging the Gap Between High-Performance AI and Clinical Deployment*. 2026. arXiv: 2602.00910 [cs.LG]. URL: https://arxiv.org/abs/2602.00910.

[64] Janez Demšar. "Statistical Comparisons of Classifiers over Multiple Data Sets." In: *J. Mach. Learn. Res.* 7 (Dec. 2006), pp. 1–30. ISSN: 1532-4435.

[65] Python documentation. *multiprocessing — Process-based parallelism*. URL: https://docs.python.org/3/library/multiprocessing.html.

[66] A. P. Mahmoudzadeh and N. H. Kashou. "Evaluation of interpolation effects on upsampling and accuracy of cost functions-based optimized automatic image registration." In: *International Journal of Biomedical Imaging* 2013 (Aug. 2013), p. 395915. DOI: 10.1155/2013/395915.

[67] *PyTorch official documentation*. URL: https://docs.pytorch.org/docs/2.12/index.html.

[68] StackGPU. *Understanding PyTorch's Dynamic Computational Graphs*. URL: https://medium.com/@StackGpu/understanding-pytorchs-dynamic-computational-graphs-92c42f41e334.

[69] *Open Neural Network Exchange official website*. URL: https://onnx.ai/.

[70] *NVIDIA TensorRT official website*. URL: https://developer.nvidia.com/tensorrt.

[71] Houman Abbasian, Yu-Te Cheng, and Josh Park. *Speeding Up Deep Learning Inference Using TensorFlow, ONNX, and NVIDIA TensorRT*. July 2021. URL: https://developer.nvidia.com/blog/speeding-up-deep-learning-inference-using-tensorflow-onnx-and-tensorrt/.

[72] Ishrak Jahan Ratul, Yuxiao Zhou, and Kecheng Yang. "Accelerating Deep Learning Inference: A Comparative Analysis of Modern Acceleration Frameworks." In: *Electronics* 14.15 (2025). ISSN: 2079-9292. DOI: 10.3390/electronics14152977. URL: https://www.mdpi.com/2079-9292/14/15/2977.

[73] *ONNX Runtime official documentation (graph optimizations page)*. URL: https://onnxruntime.ai/docs/performance/model-optimizations/graph-optimizations.html.

[74] Yuxiao Zhou and Kecheng Yang. “Exploring TensorRT to Improve Real-Time Inference for Deep Learning.” In: *2022 IEEE 24th Int Conf on High Performance Computing Communications; 8th Int Conf on Data Science Systems; 20th Int Conf on Smart City; 8th Int Conf on Dependability in Sensor, Cloud Big Data Systems Application (HPCC/DSS/SmartCity/DependSys)*. 2022, pp. 2011–2018. DOI: 10.1109/HPCC-DSS-SmartCity-DependSys57074.2022.00299.

[75] Nitish Shirish Keskar et al. “On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima.” In: *CoRR* abs/1609.04836 (2016). arXiv: 1609.04836. URL: http://arxiv.org/abs/1609.04836.

[76] Christian Payer et al. “Coarse to Fine Vertebrae Localization and Segmentation with SpatialConfiguration-Net and U-Net.” In: *Proceedings of the 15th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - Volume 5: VISAPP*. Vol. 5. 2020, pp. 124–133. DOI: 10.5220/0008975201240133.

[77] Nikolas Lessmann et al. “Iterative fully convolutional neural networks for automatic vertebra segmentation and identification.” en. In: *Med. Image Anal.* 53 (Apr. 2019), pp. 142–155.

[78] Vincent Bürgin, Raphael Prevost, and Marijn F. Stollenga. “Robust Vertebra Identification Using Simultaneous Node and Edge Predicting Graph Neural Networks.” In: *Medical Image Computing and Computer Assisted Intervention – MICCAI 2023*. Ed. by Hayit Greenspan et al. Cham: Springer Nature Switzerland, 2023, pp. 483–493. ISBN: 978-3-031-43996-4.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**