Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Engineering
Department of Mechanical and Industrial Engineering

Anders Kjelsrud

Exploring the state of the art in ultrasound image segmentation

Master's thesis in Engineering and ICT
Supervisor: Gilles Van De Vyver
Co-supervisor: Bjørn Haugen and Lasse Løvstakken
June 2024

NTNU
Norwegian University of
Science and Technology

Anders Kjelsrud

Exploring the state of the art in ultrasound image segmentation

Master's thesis in Engineering and ICT
Supervisor: Gilles Van De Vyver
Co-supervisor: Bjørn Haugen and Lasse Løvstakken
June 2024

Norwegian University of Science and Technology
Faculty of Engineering
Department of Mechanical and Industrial Engineering

NTNU
Norwegian University of
Science and Technology

Preface

This thesis marks the completion of the master´s degree program in Engineering and ICT carried out at the Norwegian University of Science and Technology. It has allowed me to apply machine learning to the fascinating field of medical image computing. First, I want to thank my supervisor Gilles van de Vyver for excellent guidance throughout the project. Another thank you goes to the Department of Circulation and Medical Imaging for letting me do this project. Finally, I want to thank family and friends for motivating and supporting me.

Trondheim, June 2024
Anders Kjelsrud

Abstract

Cardiac disease is the most common cause of death globally. Accurate segmentation of echocardiographic images from clinical examinations can enable fully automatic extraction of measurements such as ejection fraction and left ventricular volume. nnU-Net is a self-adapting framework for medical image segmentation that dynamically configures training pipelines for U-Nets based on the dataset, and is the current state-of-the-art for many domains in medical image segmentation, including ultrasound. While nnU-Net offers ease of use, the models it configures can be excessively large and complex for the task of segmenting cardiac ultrasound images. This work aims at identifying the most impactful components of the framework by the means of an ablation study, and deploy them on a significantly smaller model. The experiments evaluate the effectiveness of nnU-Nets major design choices by incrementally adding them to the smaller U-Net 1 and assessing their impact on segmentation performance. We show that a U-Net approximately 16 times smaller trained with the key elements of nnU-Net gives results which are not significantly different ($p < 0.05$) on the CAMUS dataset. Addition of the components from nnU-Net improve the average Dice score from 0.86 to 0.89 and the average Hausdorff distance from 11.68 to 8.88 from the baseline to the final model. Anatomical outliers are reduced from 42 to 6. The model's generalization ability is further investigated on the clinical HUNT4 dataset. The most important components are shown to be a robust data augmentation scheme, deep supervision, choice of loss function and connected component analysis as a postprocessing step. The smaller model can be used in real-time clinical applications, providing computational savings without sacrificing accuracy. Source code is available at: https://github.com/Anderzz/masters-thesis.

Sammendrag

Hjerte- og karsykdommer er den vanligste dødsårsaken på verdensbasis. Nøyaktig segmentering av ultralydbilder av hjertet fra kliniske undersøkelser kan muliggjøre automatisk estimering av ejeksjonsfraksjon og venstre ventrikkelvolum. nnU-Net er et selvadapterende rammeverk for segmentering av medisinske bilder som dynamisk konfigurerer U-Net-modeller basert på datasettet som skal brukes. Rammeverket produserer de mest presise modellene for mange områder innen segmentering av medisinske bilder, inkludert ultralyd. Selv om nnU-Net er enkel å ta i bruk, kan de konfigurerte modellene bli unødvendig store og komplekse for å segmentere ekkokardiografiske ultralydbilder, som er en forholdsvis enkel oppgave. Denne masteroppgaven tar sikte på å finne de viktigste komponentene i nnU-Net, for så å implementere dem i en betydelig mindre modell. Eksperimentene evaluerer innflytelsen til de viktigste designvalgene i nnU-Net ved å gradvis legge dem til den mindre U-Net 1-modellen og på den måten vurdere deres innvirkning på segmenteringsytelsen. Vi viser at et U-Net som er omtrent 16 ganger mindre, trent med nøkkelelementene i nnU-Net, gir resultater som ikke er signifikant forskjellig ($p < 0.05$) på CAMUS-datasettet. Inkludering av komponentene fra nnU-Net forbedrer gjennomsnittlig Dice-score fra 0.86 til 0.89 og gjennomsnittlig Hausdorff-avstand fra 11.68 til 8.88, sammenlignet med baseline-modellen. Videre reduseres anatomiske outliers fra 42 til 6. Modellens generaliseringsevne undersøkes videre på det kliniske HUNT4-datasettet. De viktigste komponentene viser seg å være et robust opplegg for data augmentation, deep supervision, valg av tapsfunksjon og fjerning av ikke-sammenhengende prediksjoner. Den mindre modellen gir raskere kjøretid uten å ofre nøyaktighet, slik at den kan brukes til kliniske undersøkelser i sanntid. Kildekode er tilgjengelig på: https://github.com/Anderzz/masters-thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Cardiovascular disease (CVD) is the leading cause of death worldwide, accounting for one-third of all deaths and causing 20.5 million fatalities in 2021 alone [1]. While once seen as afflictions of wealth, CVDs now predominantly impact low- and middle-income countries, accounting for over three-quarters of global CVD-related mortalities [2]. Analyzing 2D echocardiographic images is pivotal in cardiology to extract standard clinical measurements. Segmenting the anatomical structures of the images allows for extraction of measurements such as ejection fraction and left ventricular volume, which give insights on the patient's cardiovascular health. Manual or semi-automatic segmentation is today daily work due to the lacking accuracy of fully automatic methods [3]. A fully automatic method for accurate cardiac segmentation would reduce the need for manual intervention and streamline the workload for clinicians by reducing time spent on time-consuming tasks line annotation. This would allow healthcare professionals to allocate more time to patient interaction.

Deep learning research has seen a remarkable surge in the last decade and is now considered a core technology in the Fourth Industrial Revolution, or Industry 4.0 [4]. Significant advancements in computational power, big data availability, and novel architectures have led to substantial progress in areas ranging from computer vision to natural language processing. Deep learning has emerged as the predominant technique in medical image segmentation within the field of medical image computing (MIC), demonstrating performance levels similar to inter-observer variability [3]. In 2015, Ronneberger et al. introduced the U-Net [5], an encoder-decoder network which has been extensively applied in MIC [6]. It still remains a highly relevant architecture, receiving over $20,000$ citations in 2023.

nnU-Net ("no-new-Net") [7] was released in 2018 and builds on the standard U-Net. The introduction marked a paradigm shift in medical image segmentation, showing

how basic, but properly configured and trained U-Nets could achieve state-of-the-art performance across multiple datasets and modalities without new architectural modifications. nnU-Net is a self-adapting framework that automatically configures several U-Net based segmentation pipelines from dataset-specific characteristics. In contrast to most deep learning models, the framework requires no manual tuning by machine learning engineers. It is frequently cited in works on medical image segmentation and used extensively in segmentation challenges [8, 9, 10, 11]. New architectures such as Transformers [12] and Mamba [13] have been adapted to the medical imaging domain, and many claim to outperform CNN-based U-Nets [14, 15, 16]. The creators of nnU-Net show that CNN-based U-Nets still give the best performance for most medical imaging datasets if tuned well, and say that further research should focus on more thorough validation of the results instead of introducing novel architectures [17].

1.2 Goal and research questions

This work seeks to find the most critical components of nnU-Net in terms of performance by the means of an ablation study. By determining which elements significantly affect segmentation outcomes, a smaller and more efficient model can be developed. To this end, nnU-Net is first picked apart and a series of experiments are run where various components are disabled. Subsequently, the most important components are added to and evaluated on a smaller U-Net. This optimized model aims to retain key features of the nnU-Net while reducing complexity and runtime drastically. A smaller model is desirable because it trains faster, offers better generalization to other datasets and gives faster inference time which is particularly useful when deployed on portable ultrasound scanners with limited compute for real-time inference. As a consequence, the thesis aims to answer the following research questions:

RQ1. What are the most important components of nnU-Net for 2D cardiac ultrasound segmentation?

RQ2. How do the components from nnU-Net affect performance on a smaller U-Net?

1.3 Thesis structure

The remaining parts of this thesis has the following structure:

Chapter 2: The chapter *Background theory* presents the relevant theory needed to understand the work carried out in this thesis. It provides a broad introduction to deep learning and neural networks before going into detail on image segmentation, relevant metrics and models. The datasets are also presented in this chapter.

Chapter 3: The chapter describing the *Method* goes into detail on the implementation of the experiments carried out in this work. It justifies design choices and gives an overview of the strategy to pick apart nnU-Net and add components to a smaller U-Net.

Chapter 4: The chapter on *Experiments and results* presents the experimental setup used and the results obtained.

Chapter 5: The *Discussion* interprets and discusses the results with regard to the research questions and puts them into context within the research area.

Chapter 6: Finally, the *Conclusion* summarizes the most important findings and gives suggestions for future work.

Chapter 2

Background theory

This chapter introduces the theory behind the methods used in this project as well as relevant models and the current state of the art. The chapter starts with the basics of neural networks, before narrowing in on models applicable to image segmentation. Next, it introduces nnU-Net and gives a detailed explanation of how it works. Covered topics include basic neural networks, CNNs, the notion of image segmentation, commonly used metrics, the models this project uses as a starting point, and finally, the datasets on which the models are trained.

2.1 Deep learning fundamentals

Deep learning, a subset of machine learning, is a methodology that employs artificial neural networks with multiple layers to learn patterns in data. The idea is to expose a network to available training data and later use it to make predictions on unseen data. Deep learning is a type of machine learning. Mitchells definition for machine learning from 1977 [18] still holds: "A computer program is said to learn from experience `E` with respect to some class of tasks `T` and performance measure `P`, if its performance at tasks in `T`, as measured by `P`, improves with experience `E`". The Universal Approximation Theorem bridges machine learning and deep learning. It asserts that a feed-forward network with a single hidden layer containing a finite number of neurons can approximate continuous functions on compact subsets of $\mathbb{R}^n$, under mild assumptions on the activation function [19]. Deep learning has revolutionized fields such as computer vision, natural language processing, and speech recognition. The "deep" in deep learning refers to the depth of layers through which data is transformed, providing a framework for learning through representations. Neural architectures are a central part of the conversation when discussing deep learning, which includes the arrangement of layers, the type of layers used, as well as the connections between these layers. This architecture is essentially a blueprint that defines how data flows through the network, how features are extracted at each layer, and how the final output is

computed. The classic training of a model is supervised, where it is provided with input-output pairs and learns to map the input to the correct output. Unsupervised and semi-supervised learning paradigms are alternatives when no or limited labelled data is available. The effectiveness of a model in deep learning is largely dependent on its architecture, the quality and quantity of data it is trained on, and the optimization strategies employed during the training process [20].

2.1.1 Neural networks

Formally, a `feed-forward` neural network configuration $\mathcal{N}$ corresponds to a directed acyclic graph (DAG) $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ [21]. The graph represents the architecture of $\mathcal{N}$ determined by a finite set of vertices or nodes $\mathcal{V}$ and a finite set of edges $\mathcal{E}$. Each $v \in \mathcal{V}$ must belong to one or more edge $e \in \mathcal{E}$. The set of nodes $\mathcal{V}$ is furthermore subdivided into three groups. Firstly, the set of input nodes $\mathcal{I}$, which comprises nodes without any incoming connections. These nodes serve as the entry points for the network's input data (independent variables). Secondly, the set of output nodes $\mathcal{O}$ contains nodes without outgoing connections. These nodes are responsible for holding the network's output (dependent variables) in response to the inputs. The third group consists of the hidden nodes, represented as $\mathcal{H} = \mathcal{V} \setminus \{\mathcal{I}, \mathcal{O}\}$. A visualization can be seen in Figure 2.1. The nodes and edges have the two following properties:

1. Each node $v \in \mathcal{V}$ that is not an input node is associated with an `activation function` $f_v : \mathbb{R} \to \mathbb{R}$ and a `bias` term $b_v \in \mathbb{R}$.
2. $\forall e \in \mathcal{E}$, there is an assigned scalar `weight` $w_e \in \mathbb{R}$.

The weights and biases are known as the `learnable parameters` and the objective when training $\mathcal{N}$ is to tune these optimally. Nodes in the network compute an activation value based on the weighted sum of the activations of all the nodes from the previous layer. The activation value for a single node is calculated as

$$a_v^{j+1} = f_v\left(\sum_{v'} w_{v,v'} a_{v'}^j + b_v^{j+1}\right) \tag{2.1}$$

where a_v^{j+1} is the activation value of node v in layer $j+1$, $w_{v,v'}$ is the weight going from node v' in layer j to node v in layer $j+1$, $a_{v'}^j$ is the activation value of node v' in layer j, b_v^{j+1} is the bias value belonging to node a_v^{j+1}, and f_v is the activation function [20]. In practice, a matrix-vector product is used to calculate the activation values for entire layers of nodes at once

$$\mathbf{a_{j+1}} = f(\mathbf{w_{j,j+1}^T a_j} + \mathbf{b_{j+1}}). \tag{2.2}$$

In order to make accurate predictions on unseen data, a network needs to tune its learnable parameters. The goal is to iteratively adjust the weights and biases to

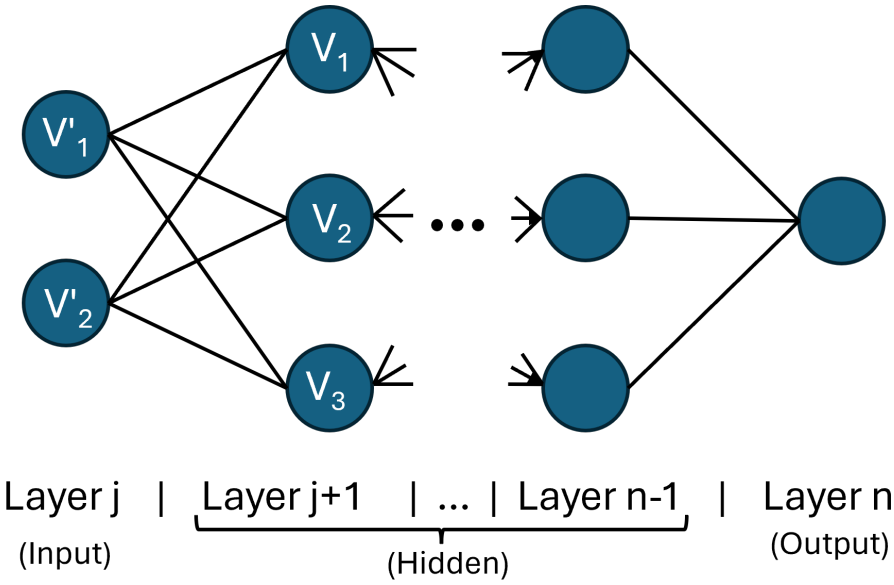


Figure 2.1: A simple neural network with 2 input nodes, an arbitrary number of hidden layers, and one output node.

minimize the cost C between prediction and ground truth, which measures the error to a perfect prediction. To tune the network, the gradient of the cost with respect to each parameter θ is used to minimize the cost [22]. The gradient, $\frac{\partial C}{\partial \theta}$, is calculated using backpropagation [23], an algorithm taking advantage of the chain rule. The choice of the cost function C largely depends on the task one wishes to solve, as well as the network architecture. Common cost functions are presented in detail in Section 2.1.3. Finally, given $\frac{\partial C}{\partial \theta}$, the parameters θ at iteration $i+1$ are updated with step length α using `gradient descent`

$$\theta_{i+1} = \theta_i - \alpha \frac{\partial C}{\partial \theta}. \tag{2.3}$$

The procedure is typically repeated for all θ until some validation metric, e.g. accuracy, starts degrading.

2.1.2 Activation functions

In order for neural networks to learn how to be more than linear function approximators, nonlinearities are introduced through activation functions (AFs). They help networks transform non-linearly separable data into a form that is more linearly separable in higher dimensions [20]. Key properties of AFs include output range, monotonicity, smoothness and impact on gradient flow during training. Below, a selection of the most popular AFs are presented, along with remarks on their typical use cases. The functions and their derivatives are illustrated in Figure 2.2.

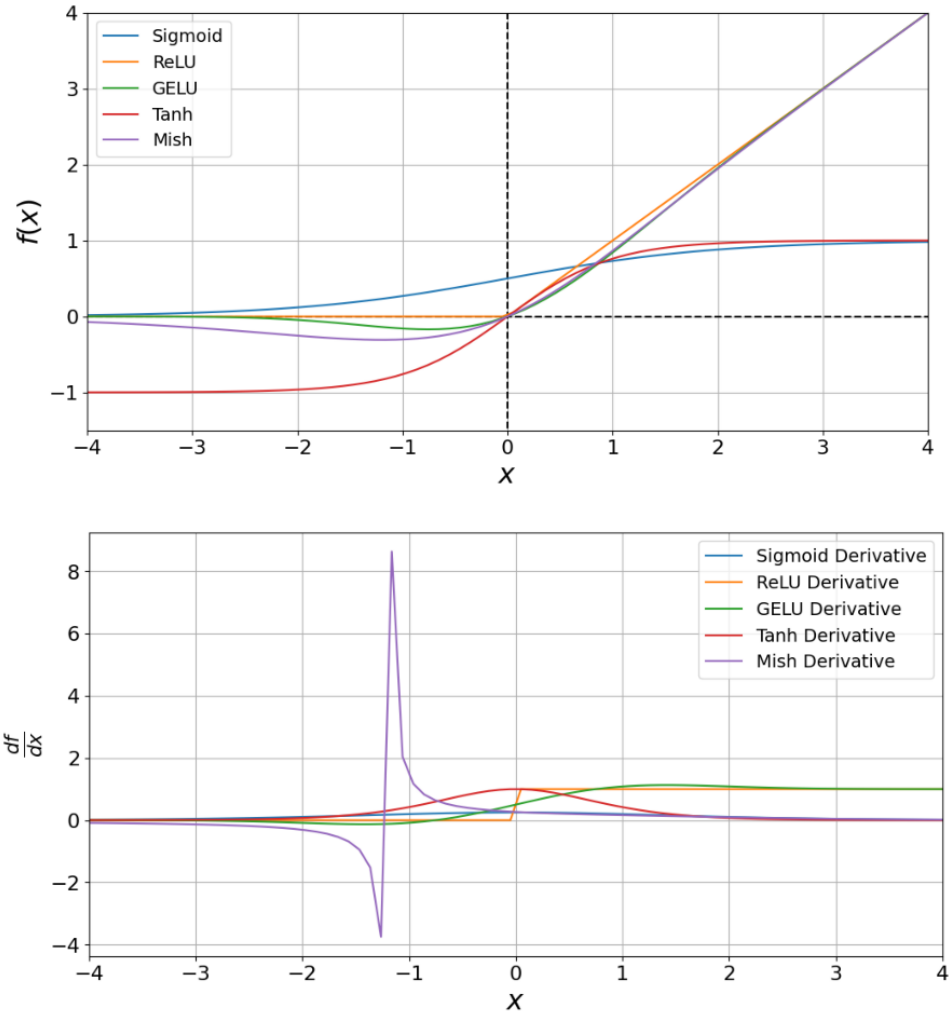


Figure 2.2: Common activation functions (top) and their derivatives (bottom).

The **Sigmoid** activation function is defined as

$$\text{Sigmoid}(x) = \frac{1}{1 + e^{-x}} \tag{2.4}$$

and compresses its output between $[0, 1]$. This function was widely used in early research on machine learning when networks were typically more shallow. It has in recent years largely been replaced by better suited ones. Deep networks using the sigmoid AF suffer from the `vanishing gradient problem` [24]. Instead, it is now most often seen in the last layer of networks performing classification, due to its suitable output range.

The **Hyperbolic Tangent**, tanh, defined as

$$\text{Tanh}(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}, \tag{2.5}$$

also suffers from the vanishing gradient problem. It has an output range of $[-1, 1]$ and can be found in Recurrent neural networks (RNNs) because it allows negative outputs [20].

The **Rectified Linear Unit** (ReLU) is unbounded and thus does not suffer from saturation of inputs. It is simple, has a very low impact on computational speed and is defined as

$$\text{ReLU}(x) = \max(0, x), \tag{2.6}$$

with a range of $[0, \infty)$. Despite being one of the most successful and widely-used AFs in deep learning as a whole, it too suffers from a form of vanishing gradient problem [25]. The issue arises when inputs to the ReLU function are less than or equal to zero, leading to a zero gradient in the backward pass. This phenomenon is often referred to as the "dying ReLU problem", and, as a result, a significant portion of the network could end up being effectively "dead". However, various solutions and modifications to ReLU have been proposed to mitigate these issues. One popular variant is the **Leaky ReLU**, which introduces a small, positive gradient when the input is less than zero, thereby allowing the flow of gradients even for negative inputs. The function is defined as:

$$\text{Leaky ReLU}(x) = \max(\alpha x, x), \tag{2.7}$$

where α is a small coefficient (e.g., 0.01).

The **Gaussian Error Linear Unit (GELU)** has gained prominence in deep learning due to its performance increase compared to ReLU across multiple domains [26]. The GELU function is defined as

$$\text{GELU}(x) = x \cdot \Phi(x), \tag{2.8}$$

where Φ represents the cumulative distribution function of a Gaussian. Mathematic-

ally, this can be approximated as:

$$\text{GELU}(x) \approx 0.5x \left(1 + \tanh\left(\sqrt{\frac{2}{\pi}}\left(x + 0.044715x^3\right)\right)\right). \tag{2.9}$$

Despite its advantages, GELU is computationally more intensive than ReLU and its variants due to the involvement of exponential and tanh functions. It is however the AF of choice in many transformer-based language models such as BERT [27] and GPT-3 [28], because it has a smoother, more continuous shape than ReLU and also because it has a non-zero gradient at $x = 0$.

The **Mish** function is a novel activation function that has gained popularity in deep learning due to its self-regularizing properties and smooth curve [29]. Mish is defined as:

$$\text{Mish}(x) = x \cdot \tanh(\ln(1 + e^x)), \tag{2.10}$$

The function multiplies the input x by the output of a non-linear function, which helps in smoothing the output across the input range. The authors report performance increases compared to ReLU on several experiments. The increased complexity stemming from several function evaluations can however be a limiting factor, especially in very deep networks. The AF is used in popular frameworks, such as the object detector YOLOv4 [30].

2.1.3 Loss functions

A cost function C measures the difference between predicted output $\hat{y}$ and the target value y and can be expressed through the average of a loss function L over n samples. Here, L is the cost for a single prediction. C is then defined as

$$C(\hat{y}, y) = \frac{1}{n}\sum_{i=1}^{n} L(\hat{y}_i, y_i). \tag{2.11}$$

Since the goal of training networks is to minimize C through gradient descent, selecting a suitable loss function is crucial. However, given the vast array of available options, selecting the best one for a specific task can pose a significant challenge for practitioners. Key properties of loss functions include convexity, differentiability, robustness and monotonicity [31].

The **Cross Entropy** loss function is popular in image segmentation due to its effectiveness in handling multi-class classification problems, where each pixel of an image needs to be classified into one of several categories. It computes the difference between the predicted probability distribution and the true distribution and has the desirable property of penalizing incorrect confident predictions more heavily than uncertain

ones. For multiple classes, it is defined for segmentation as

$$-\frac{1}{N}\sum_{i=1}^{N}\sum_{c'=1}^{C'} y_{i,c'}\log(\hat{y}_{i,c'}), \tag{2.12}$$

where N is the number of pixels, C' is the number of classes, y are the target segmentation maps, and $\hat{y}$ are the predicted segmentation maps.

Another commonly used loss function is the **Dice** loss, which is particularly effective in medical imaging because it handles class imbalance by focusing on the regions of overlap [32]. This makes it highly suitable for instances where the region of interest occupies a small part of the overall image. It is defined as

$$1 - \frac{2 \cdot |S_{gt} \cap S_p|}{|S_{gt}| + |S_p|}, \tag{2.13}$$

where S_{gt} is the ground truth segmentation and S_p is the predicted segmentation.

2.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) represent a special form of network architecture that excels in handling data organized in a grid-like structure [20]. Since images essentially are two-dimensional grids of pixels, they are particularly well-suited for CNNs. Over the last decade, these networks have achieved remarkable success in interpreting image data.

One of the key reasons for the success of CNNs is their inductive bias, which refers to the set of assumptions a model makes about the data to generalize well from a limited amount of training data. For CNNs, this bias includes assumptions about spatial hierarchies in data. The architecture of CNNs, with their convolutional layers, inherently captures local patterns such as edges and textures, which are then combined to recognize higher-level features. This hierarchical pattern recognition is crucial for learning complex structures in images.

CNNs employ **convolutional layers** which perform the mathematical operation known as convolution:

$$(I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n). \tag{2.14}$$

Here, I represents an image, while K denotes the filter that moves across I in a sliding-window fashion. A convolutional layer typically comprises multiple filters that scan the input image, computing the dot product of the filter with different sections of the image. These dot products form the output of the layer. The trainable parameters in these layers are the elements of the filter.

In addition to convolutional layers, modern neural network architectures often in-

corporate various other types of layers. For instance, Graph Convolutional Networks (GCNs) [33] are used for processing data structured as graphs. Dense, or fully connected layers, are commonly used for classification by combining features into predictions. Attention mechanisms have revolutionized the field of natural language processing through the transformer architecture and are increasingly incorporated into image-related tasks, e.g. in the Vision Transformer (ViT) [34]. Despite the emergence and success of these diverse layers, CNNs remain a cornerstone in the field of deep learning. Liu et al. demonstrates in [35] that CNNs are still among the state of the art for many applications, especially in computer vision. This work focuses on CNNs.

The output size of a convolutional layer is influenced by several parameters:

- **Stride**: This determines the filter's movement across the input grid. The stride is the number of grid spaces the filter's center skips for each calculation of the dot product. A larger stride typically results in a reduced output size.
- **Padding**: This involves adding zeros around the perimeter of the input grid. Padding allows the filter's center to reach the edges of the grid so that the spatial size of the output can be kept consistent with the input.

The size of K also impacts the output size. Common filters include 1×1, 3×3, 5×5 and 7×7. They normally have odd-numbered height and width, because then the filters have a natural center square. Mathematically, the output size of a convolutional layer is related to the input size by the formula:

$$O = \left\lfloor \frac{W - K + 2P}{S} + 1 \right\rfloor \tag{2.15}$$

where O is the output height or width, W is the input height or width, K is the size of the filter, P is the padding and S is the stride.

Pooling layers are used after convolutional layers in networks in order to replace the output with a summary statistic of the nearby outputs [20]. For example, the `max pooling` operation will return the maximum value within a rectangular neighborhood of grids, as shown in Figure 2.3. The operation is commonly referred to as downsampling, as it reduces the spatial dimensions of the image representation.

Translation invariance and the inductive bias in CNNs result from both reusing weights and applying pooling operations. Pooling, particularly max pooling, is crucial for achieving translation invariance in the input's representation. It reduces the importance of the precise locations of objects within the image while preserving essential information about the objects' attributes.

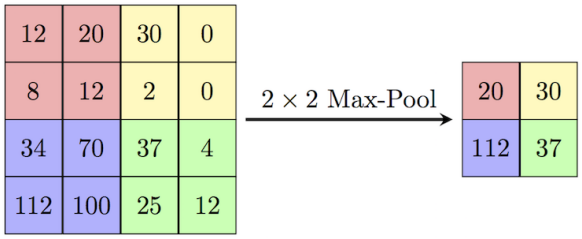


Figure 2.3: 2×2 maxpooling.

Calculating the number of learnable parameters

The total number of learnable parameters in a convolutional layer is calculated using the formula:

$$((m \cdot n \cdot d) + 1) \cdot k. \tag{2.16}$$

Here, m, n is the height and width of the filter, d is the number of filters in the previous layer and k is the number of filters in the current layer, plus 1 for the bias term. For 1×1 convolutions, this simplifies to $(d + 1) \cdot k$ because there is just one parameter to learn per filter.

2.3 Normalization techniques

Normalization layers enhance the optimization and generalization of neural networks. Among these, Batch Normalization (BN) is one of the most prominent methods [36]. BN regulates the inputs to each layer by normalizing the distribution for every mini-batch. This strategy helps stabilize the learning process by minimizing the internal covariate shift—the variability in the distribution of network activations due to changes in the parameters during training. By ensuring consistent activation distributions across mini-batches, batch normalization not only smooths the loss surface, facilitating easier optimization, but also acts as a regularizer to reduce the risk of overfitting.

Another notable normalization technique is Instance Normalization (IN), which is frequently utilized in tasks like style transfer [37]. While BN is effective in reducing internal covariate shift by normalizing data across the batch dimension, IN focuses on normalizing across each individual instance, making it particularly suitable for applications where the independence of instances is important, or where batch sizes are small.

For an input image x, BN is computed as:

$$\mathrm{BN}(x) = \gamma \left(\frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} \right) + \beta \tag{2.17}$$

where μ is the mean image intensity across the batch, σ^2 is the variance across the batch, ϵ is a small number for numerical stability and γ, β are learnable parameters.

IN is computed similarly, but the mean and variance are computed per-channel per-sample.

In addition to these, Z-score normalization is another common technique [38]. It normalizes the data to have zero mean and unit variance, which can improve convergence during training:

$$Z(x) = \frac{x - \mu}{\sigma} \tag{2.18}$$

where μ and σ are the mean and standard deviation of the data, calculated over the dataset or a specified subset of the dataset. This method ensures that the feature values are on a comparable scale.

2.4 Deep Supervision

To address the problem of vanishing gradients in very deep networks and also to improve their ability to learn discriminative features at multiple levels, Lee et al. proposed a novel technique coined Deep Supervision [39]. It aims at increasing performance by forcing the network to make meaningful predictions at intermediate levels instead of just the final layer. In encoder-decoder based segmentation models, this is achieved by attaching output layers at multiple levels of the decoder, effectively injecting gradients in some of the hidden layers and allowing said gradients to be backpropagated to earlier layers directly with no other layers in between. An illustration of this is shown in Figure 2.4. Typically, intermediate feature maps are passed through 1×1 convolutions to calculate auxiliary losses at multiple stages, which are then added to the final loss, often with a weighted scheme. This way, hidden layers are encouraged to directly predict valid segmentation maps at intermediate levels. The direct gradient flow to lower resolution features provides regularization and acts as a form of implicit data augmentation. Deep supervision has shown to be particularly effective in combination with a range of U-Net architectures, explained in detail in Section 2.8.

2.5 Overfitting

Overfitting entails a model learning the training data too intricately, including its noise and random fluctuations, and poses a significant challenge in deep learning. Overfitting results in lowered performance on new, unseen data. This section explores the concept of overfitting, its causes, consequences, and various strategies to counteract it.

Overfitting happens when a deep learning model not only learns the essential patterns of the training data but also picks up irrelevant details and noise, effectively memorizing specific data inputs. Contributing factors to overfitting include overly complex models, limited training data, and excessively long training periods. It is

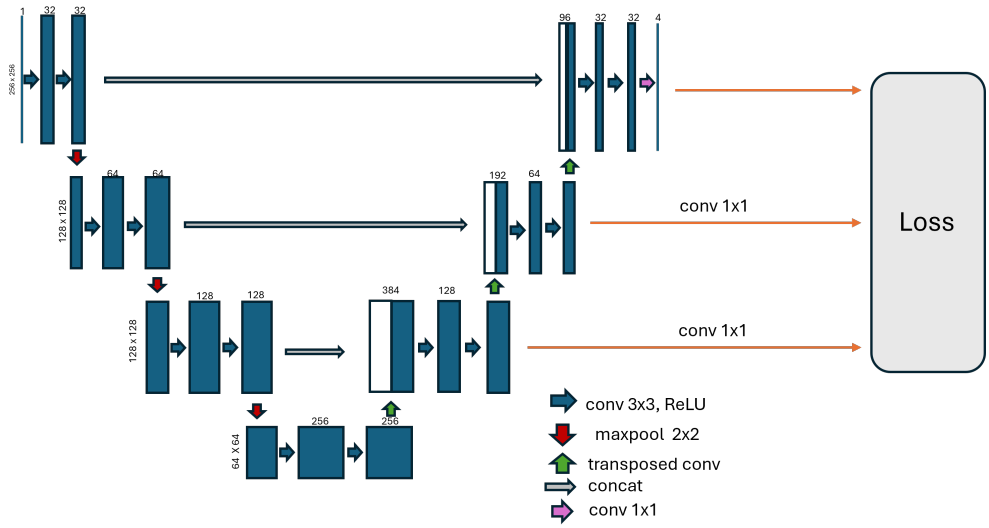


Figure 2.4: Deep Supervision on U-Net-like architecture.

often detected by a performance gap between training and validation datasets, evident through high accuracy on training data contrasted with poor validation results. Several strategies have been developed to combat overfitting, with some of the most prevalent ones discussed below.

Data Augmentation enhances the training dataset artificially. Techniques such as cropping, rotating, adding noise, and flipping images introduce variability, emulating a more extensive dataset. This added diversity aids the model in better generalizing to new data by learning to identify features under varied conditions rather than memorizing the training examples.

Regularization Techniques are techniques that aim to prevent models that are too complex for the given task. The most common regularization techniques are L1 (Lasso) and L2 (Ridge) regularization help limit the model's capacity. L1 regularization imposes penalties on the absolute values of the weights, encouraging a sparse model where some weights may be zeroed, thus highlighting more relevant features. Conversely, L2 regularization penalizes the weights' squared values, preventing any single weight from becoming too large. Both methods integrate a penalty into the loss function to avoid excessive complexity in the model.

Dropout is an approach that randomly deactivates a subset of neurons during each training cycle [40]. This technique helps prevent excessive co-adaptation of neurons and promotes the development of more robust internal representations. Dropout encourages a form of ensemble learning within the model, enhancing its generalization capabilities. In Convolutional Neural Networks (CNNs), dropout is commonly applied post-convolutional or in fully connected layers to help mitigate overfitting by intermittently turning off neurons, thus forcing the network to learn more general features.

Early Stopping is a strategy that stops the training as soon as the model's validation set performance begins to decline. Typical practices include stopping training when there has been no improvement in validation loss for a predetermined number of epochs. The number of epochs before stopping is referred to as the `patience`. This method relies on the notion that prolonged training can lead models to learn irrelevant details and noise from the training set. As illustrated by [41], early layers in neural networks are less sensitive to label noise than later ones. Progressive Early Stopping (PES) leverages this by initially training the more resilient early layers longer and gradually reducing the training time for the later, more sensitive layers, aiming to maintain an optimal balance between feature learning and noise avoidance.

Double Descent refers to a phenomenon where the performance of a model on validation data initially worsens as the model complexity increases, but then improves again after a certain point [42]. This stands in contrast to typical understanding which associates increased model complexity with a continuous decline in out-of-sample performance due to overfitting. The double descent curve describes how, after reaching a peak of poor performance (the first descent), further increasing the complexity—such as adding more layers to a neural network, can lead to a second phase of improved performance.

Double descent occurs when models are scaled beyond the interpolation threshold, which is the point where they are complex enough to perfectly fit or interpolate the training data. Beyond this threshold, models enter the overparameterized regime. Here they begin to generalize better on new, unseen data despite their capability to memorize the training data. This phenomenon suggests that very large models can use their extra capacity to regularize their predictions, effectively mitigating some forms of overfitting.

2.6 Image segmentation

Image segmentation is a technique that is used to partition an image into multiple regions based on pixel characteristics in order to simplify or change the representation of an image into something that is easier or more meaningful to analyze further. It is one of many examples of fundamental tasks in computer vision that have been revolutionized by the advent of deep learning, specifically CNNs [43]. In this context, the literature frequently refers to two main components of an image: `things` and `stuff`. `Things` refers to countable objects, like people, tumors or other specific organs. On the other hand, `stuff` represents uncountable structureless regions that do not have a specific shape or clear boundary, typically found in the background. Examples include the sky, water and walls.

There are three groups of modern AI-based image segmentation techniques: `Semantic`, `Instance` and `Panoptic`.

Semantic segmentation is concerned with associating every pixel in an image with a corresponding class [43]. Here, the `stuff` in an image is analyzed. The goal is to produce a dense pixel-wise segmentation map of an image, where each pixel is assigned to a specific class or object. For example, in an image of people walking in front of a background, all the pixels belonging to people would be assigned the same category and the pixels belonging to the background would be classified as another category. Popular models include U-Net [5], Fully Convolutional Network (FCN) [44] and SegFormer [45].

Instance segmentation deals with countable `things`. Instead of assigning the same class to all instances of an object, this technique distinguishes between them, and each unique object is assigned a distinct identifier. This means that even if there are multiple objects of the same class, such as the one with several people, instance segmentation will treat each person as a separate entity. Commonly used models include Mask R-CNN [46], YOLACT [47] and Swin Transformer [48].

Panoptic segmentation combines the previous two in order to coherently identify both the `stuff` and the `things` in a scene [49]. `Things` (countable, distinct objects), are segmented individually, while for classes belonging to `stuff` (uncountable, amorphous regions) all pixels belonging to the same class are grouped together without distinction between different instances. Popular models include Mask DINO [50] and different variants of Mask R-CNN.

This work focuses on semantic segmentation.

2.7 Metrics

A metric provides an objective basis to assess the effectiveness of a model, enabling fair comparison between various configurations. An effective metric for semantic segmentation accurately quantifies the overlap between the predicted segmentation and the ground truth. The following sections will introduce some of the most commonly employed metrics, outlining their advantages and limitations.

For the purpose of notation, it is useful to first understand the concept of the confusion matrix. For simplicity, we will look at the matrix for a binary segmentation task, where each pixel belongs to either the foreground (positive) or background (negative). Four cases then arise:

- True Positives (TP): Pixels correctly identified as foreground.
- False Positives (FP): Pixels incorrectly identified as foreground.
- True Negatives (TN): Pixels correctly identified as background.
- False Negatives (FN): Pixels incorrectly identified as background.

These will serve as the basis for constructing most of the following metrics [51]. It is trivial to extend the matrix to multiple classes.

2.7.1 Accuracy

A first, naive approach to evaluating a model is to calculate the pixel-wise accuracy, which is defined as the proportion of correctly identified pixels out of all pixels in the image. It can be expressed using the above measures as

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{TN} + \text{FN}}. \tag{2.19}$$

Accuracy is poorly suited for evaluating segmentation models, because it does not consider class imbalance. Often when segmenting images, the majority of pixels are made up of the background, implying that a model could achieve a high accuracy by simply predicting that all pixels belong to the background.

2.7.2 Precision and Recall

$$\text{Precision} = \frac{\text{TP}}{\text{TP+FP}} \tag{2.20}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP+FN}} \tag{2.21}$$

Precision, also known as positive predictive value, indicates the proportion of positive predictions that were actually correct. It answers the question: "Of all the instances the model labeled as positive, how many are actually positive?". Recall, or true positive rate, addresses the question: "How many of the actual positives does the model successfully capture?". Precision and recall are not very useful in isolation, since it is possible to achieve near perfect precision by only selecting a few very likely pixels. Similarly, to have a perfect recall, a model could simply select all pixels. Thus, it is apparent that balancing these two are of great importance. Which measure to deem most important depends on the use-case. For example, in medical diagnosis, a false positive test can lead to unnecessary treatment and expenses. In this situation, it is useful to value precision over recall. In other applications, for example email spam filtering, the cost of a false negative is relatively high.

2.7.3 Dice

Dice score, also known as the F_1 score is the harmonic mean of precision and recall [51] and is illustrated in Figure 2.5. It is defined as

$$\text{Dice} = \frac{2\text{TP}}{2\text{TP} + \text{FP} + \text{FN}}. \tag{2.22}$$

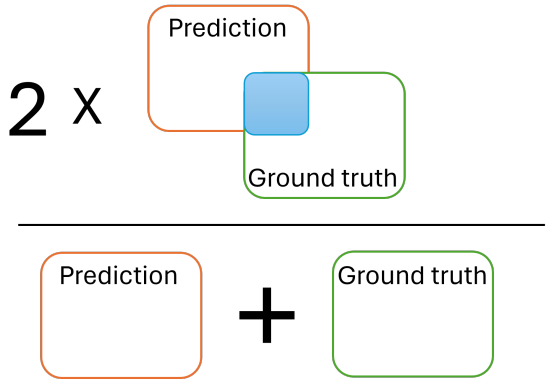


Figure 2.5: Dice score

It can also be expressed using set notation. Let S_{gt}, S_{p} be the ground truth and predicted segmentation masks respectively. The Dice score is then given as

$$\text{Dice} = \frac{2|S_{\text{gt}} \cap S_{\text{p}}|}{|S_{\text{gt}}| + |S_{\text{p}}|}, \tag{2.23}$$

where $|\cdot|$ denotes cardinality. A score of 0 means no overlap, while a score of 1 means the two sets have full overlap.

2.7.4 IoU

Intersection over union, commonly referred to as the Jaccard index or the Jaccard similarity coefficient [52] is another popular metric and is defined as

$$\text{IoU} = \frac{\text{TP}}{\text{TP} + \text{FP} + \text{FN}}, \tag{2.24}$$

or, using set notation,

$$\text{IoU} = \frac{|S_{\text{gt}} \cap S_{\text{p}}|}{|S_{\text{gt}} \cup S_{\text{p}}|}. \tag{2.25}$$

It is closely related to the dice score:

$$\text{IoU} = \frac{\text{Dice}}{\text{2-Dice}}, \tag{2.26}$$

$$\text{Dice} = \frac{2\text{IoU}}{1+\text{IoU}}. \tag{2.27}$$

It is worth noting that for any single ground truth segmentation map, if model 1 is better than model 2 under one metric, it is also `always` better under the other, implying that the two metrics are positively correlated. Because they measure the same system ranking, selecting both as validation metrics does not provide additional information compared to just using one of them [51]. However when averaging over

at set of samples, they are distinguished by the way they quantify how much better model 1 is than 2. IoU is said to have a squaring effect on errors relative to the Dice score, which indicates that IoU punishes large mistakes more. This means that while the Dice score tends to measure approximately the average performance, IoU will tend to measure the worst case performance.

2.7.5 Hausdorff distance

The pixel-wise Hausdorff distance (HD) measures how far two subsets of a metric space are from each other. It is the maximum distance of a set to the nearest point in the other set. For the two point sets, S_{gt} and S_{p}, the ground truth and predicted segmentation maps, the bidirectional HD between them is

$$\mathrm{HD}(S_{\mathrm{gt}}, S_{\mathrm{p}}) = \max(\mathrm{h}(S_{\mathrm{gt}}, S_{\mathrm{p}}), \mathrm{h}(S_{\mathrm{p}}, S_{\mathrm{gt}})) \tag{2.28}$$

where $\mathrm{h}(S_{\mathrm{gt}}, S_{\mathrm{p}})$ is the directed Hausdorff distance given by

$$\mathrm{h}(S_{\mathrm{gt}}, S_{\mathrm{p}}) = \max_{s_{\mathrm{gt}} \in S_{\mathrm{gt}}} \min_{s_{\mathrm{p}} \in S_{\mathrm{p}}} \|s_{\mathrm{gt}} - s_{\mathrm{p}}\|_2 \,. \tag{2.29}$$

HD is highly sensitive to the boundary area of a segmentation class because it measures the maximum distance of a set of points to the nearest point in another set, effectively capturing the worst-case scenario for the given class. This makes it especially well suited for the segmentation of medical images, where object boundaries are important. It is also intuitive to interpret, since a HD of 6 pixels means that the largest discrepancy between the segmented boundary and the ground truth is 6 pixels. The metric is however highly sensitive to outliers, which is why [53] proposed using a quantile method.

2.7.6 Wilcoxon signed-rank test

The Wilcoxon signed-rank test is a non-parametric statistical hypothesis test used to determine whether the median of paired differences between two sets of metric values is significantly different from zero [54]. It is typically used when the data is not normally distributed, which is required by parametric tests like the t-test. To use the test to check if one model is significantly better than another, let X and Y be the sets of scores for one class for models 1 and 2 respectively. With n observations, the test procedure is as follows:

1. Calculate $D = X - Y$ for each pair of observations. Discard the pairs if $D = 0$.
2. Assign ranks to the absolute differences, starting from the smallest. Ties are given the average rank.
3. Calculate T^+ and T^-, the sum of ranks for positive and negative differences.

4. Use $T = \min(T^+, T^-)$ as the test statistic.

5. Let the null hypothesis H_0 state that the median difference between the two sets of observations is zero. Let H_1 state that there is is a significant difference.

6. Obtain a critical value for a given significance level, typically 0.05 by using a reference table of critical values. Compare T to the critical value. If the absolute value of T is less than or equal to the critical value, the null hypothesis is rejected, indicating a significant difference between the two sets of observations.

2.8 U-Net

U-Net, a fully convolutional neural network introduced in 2015, represents a significant advancement in the field of biomedical image segmentation [5]. The architecture is specifically designed to efficiently process images with a limited amount of training data. The U-Net structure is characterized by its symmetrical layout, consisting of a contracting path to capture context and a expanding path for precise localization. This design enables it to capture both high-level and low-level features, making U-Net particularly adept at tasks like medical image segmentation, where fine details are crucial. Its effectiveness and efficiency have made U-Net a popular choice in medical imaging and other areas where precise image segmentation is essential [55].

Architecture

The network architecture consists of a contracting path followed by an expanding path depicted in Figure 2.6. This architecture allows the network to compress the the dimensions of the latent representations, while regaining the spatial dimensions of the input image in its output. The dimensionality reduction increases the receptive field in each step, while the dimensionality re-expansion is suitable to return to full resolution in the map. The contracting path is typically formed by repeating two convolutions, each followed by a nonlinear activation. Then the resulting feature maps are passed through a maxpool layer in order to half the spatial dimension. Each time downsampling is performed, the number of channels increases. This part of the network is known as the encoder part, meaning it distills the information from the original image into a compact representation which is spatially small, but has many feature channels.

The operations performed in the decoder essentially reverse those from the encoder. The expanding path follows a repeating pattern of transposed convolutions to increase the spatial dimension and half the feature channels, concatenation with the correspondingly cropped feature map from the contracting path and two convolutions, each followed by a nonlinear activation. Finally, a 1×1 convolution maps the multi-channel feature maps of the last step of the expanding path to the desired number of classes

for each pixel, producing the final segmentation map. This last layer essentially serves as a pixel-wise classifier, assigning each pixel in the input image to a specific class based on the learned features throughout the network. Since the output usually has the same shape as the original input-image, a loss between the two can easily be calculated and used to train the network.

Training

The original U-Net was trained with stochastic gradient descent (SGD), using the cross entropy loss function from 2.12 between the softmax of the output feature map and the ground truth. The paper emphasizes the importance of a good weight initialization scheme, pointing out that ideally, the initial weights should be set such that every feature map has unit variance. The authors therefore recommend using the method proposed by [56], namely drawing the weights from a Gaussian distribution with $\sigma = \sqrt{\frac{2}{N}}$. Here, N is the number of incoming connections to one neuron in the layer.

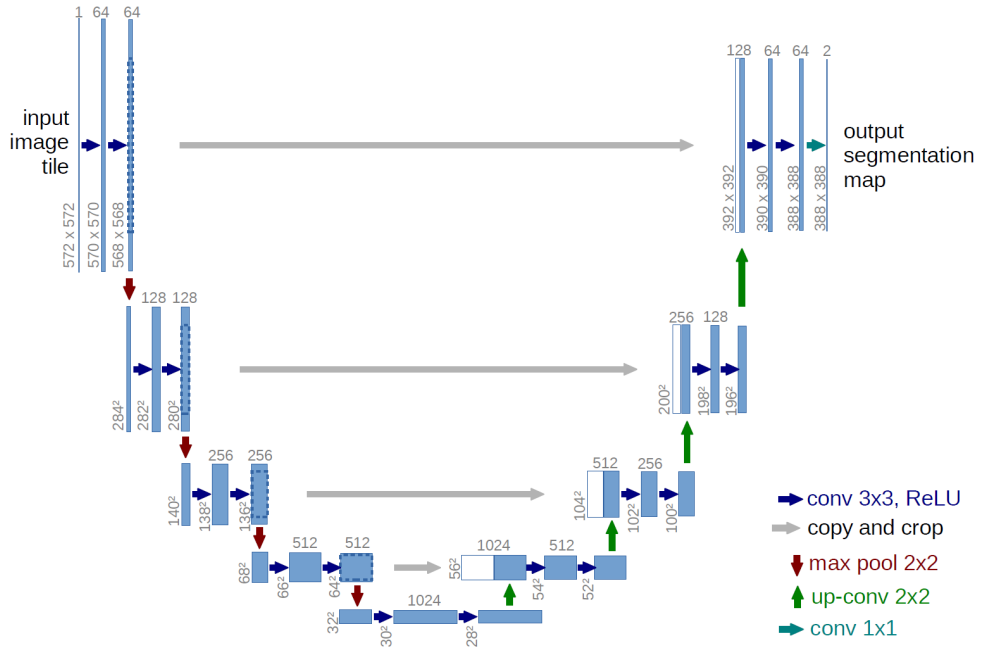


Figure 2.6: U-net architecture. Blue boxes are multi-channel feature maps. Source: Taken from [5].

2.9 nnU-Net

2.9.1 General concept and motivation behind nnU-Net

The no-new-U-Net (nnU-Net) framework [7] is a seminal advancement in the domain of medical image segmentation, revolutionizing the approach towards handling diverse medical imaging datasets. This framework is designed to autonomously adapt its configuration in order to suit the different characteristics inherent to each dataset it encounters. This self-regulatory mechanism not only applies to the neural network architecture but also extends to preprocessing steps and hyperparameter settings. The idea is grounded in what the authors call the underlying complexity of the high-dimensional optimization problem that is implicitly posed by configuring a deep learning method for biomedical image segmentation.

The methodology of nnU-Net is based on the standard U-Net [5] architecture described in Section 2.8 and the hypothesis that a well-configured vanilla U-Net is hard to beat. The network itself remains largely unchanged, however the training and preprocessing procedure is heavily modified. This includes an extensive data augmentation process which is fixed for all configurations and chosen based on empirical observations. In addition, nnU-Net automates much of the manual tuning process, which has historically been a bottleneck in both image segmentation and deep learning as a whole. The framework employs both a 2D, 3D and a cascade of two 3D U-Net configurations which are chosen based on the dataset's spatial properties.

The significant impact of nnU-Net lies in its universal applicability and robustness across a wide range of segmentation tasks. It has demonstrated exceptional performance in various benchmarking challenges, often surpassing specialized, tailor-made solutions. This universality is underpinned by nnU-Net's ability to generalize across different medical imaging modalities, including CT, MRI and ultrasound images, making it a versatile tool in the whole medical imaging domain.

nnU-Net's approach is not only methodologically innovative but also pragmatic, reducing the entry barrier for medical image analysis. It enables medical researchers and clinicians with limited machine learning experience to quickly train and deploy models out of the box. Additionally it can serve as a baseline or starting point for AI researchers. This has profound implications for accelerating research and improving diagnostic accuracy in the medical field.

2.9.2 Technical details

The first step in the nnU-Net pipeline is to decide which hyperparameters to use and what preprocessing steps to take. To this end, the process starts by generating a dataset-specific pipeline fingerprint consisting of a **dataset fingerprint**, **fixed parameters** and **inferred parameters**. The following chapter investigates these

key pieces and goes into detail on some of the more relevant ones.

The **dataset fingerprint** comprises a set of dataset-specific properties, such as image shapes (number of voxels), intensity distribution, modality, voxel spacing and the number of classes as well as the number of training cases. To combat large image sizes of certain datasets such as brain images, nnU-Net crops all images to their non-zero regions. Furthermore, the mean μ, standard deviation σ and the 0.5 and 99.5 percentiles of the image intensities are calculated across all voxels for each class. This fingerprint is used in conjunction with a set of heuristic rules to get the inferred parameters.

The **fixed parameters** are dataset-agnostic, meaning they are designed to remain constant irrespective of the dataset on which nnU-Net is being trained. The rationale behind these fixed parameters is rooted in empirical evidence: they consistently yield satisfactory performance across a diverse range of datasets. The fixed parameters include the learning rate scheduler, objective function, optimizer, data augmentation, architecture template as well as the training and inference procedures.

The rule-based **inferred parameters** largely depend on the dataset fingerprint and are calculated using a set of predefined heuristics. These parameters include the normalization scheme, resampling strategies for images and annotations, target spacing, network topology, patch size, batch size and whether to employ a cascade of 3D U-Nets. The normalization scheme is Z-score per image for all image types except CT, where a global normalization scheme is used. For isotropic data, the default median spacing of training cases is calculated independently for each axis, using third-order spline interpolation for data and linear interpolation for one-hot encoded segmentation maps. For anisotropic data, the target spacing on the out-of-plane axis is set as the 10th percentile of observed spacings to enhance resolution and reduce artifacts. Resampling on this axis utilizes nearest neighbor interpolation for both data and segmentation maps. Input patch size is dependent on the available GPU memory, but is chosen to be as large as possible with a minimum batch size of 2.

These parameters are then used to construct a customized pipeline for each specific dataset, essentially bypassing the need for a deep learning expert to manually fine-tune the model for each new dataset. This approach significantly streamlines the adaptation of the nnU-Net framework to novel data.

Network architecture

All U-Nets configured and trained by nnU-Net are derived from a template that closely resembles the original U-Net [5], described in Section 2.8, as well as the 3D analogue [57], except a few variations which will be highlighted below. By default it does not utilize recent suggestions on architectural modifications, such as adding residual connections [58], attention mechanisms [59] or dense connections [60], although this can easily be switched on within the nnU-Net framework. Instead, the U-Nets by

default only use plain convolutions, instance normalization and Leaky ReLU activation functions. Strided convolutions are used for downsamplig instead of max pooling. In order to maximize the patch size, most networks train on very small batch sizes, but with a minimum of 2. As a consequence, nnU-Net, by default, employs instance normalization instead of batch normalization typically used in the standard U-Net architecture.

The number of downsampling operations, and therefore the network's depth, is determined by the requirement to reduce feature maps to a minimum size of 4×4 pixels, ensuring adequate aggregation of contextual information. The number of convolutional layers is determined by $5 \cdot k + 2$, where k is number of downsampling steps (2 in the encoder, 2 in the decoder plus the transposed convolution). The last 2 come from the initial processing of the input.

nnU-Net uses Deep Supervision, as described in Section 2.4, by attaching loss functions to upscaled versions of all but the first two feature maps in the decoder. The loss is the same as the global one, namely the average of Dice and cross entropy, as explained below.

Training

nnU-Net defines an epoch as 250 weight updates and trains all its networks for 1000 of these epochs. Stochastic gradient descent (SGD) is used with momentum (0.99) and a polynomial learning rate scheduler, starting with a learning rate of 0.01. Foreground regions are oversampled to reduce the effect of class imbalance. The loss function is the average of the Dice and cross entropy loss, defined in 2.13 and 2.12.

nnU-Net employs an extensive set of data augmentations, which are computed on the fly using the CPU during training. Let $x \sim U(a, b)$ indicate that x is drawn from a uniform distribution between a and b. The following augmentations are used regardless of dataset:

- Rotation and scaling with probability of 0.2 each, which results in probabilities of 0.16 for only rotating, 0.16 for only scaling and 0.08 for both being applied. For 2D, the angle of rotation is drawn from $U(-180, 180)$, while for 3D it is drawn from $U(-30, 30)$. The scaling factor is drawn from $U(0.7, 1.4)$ for all patch types.
- Gaussian noise with a probability of 0.15 and variance drawn from $U(0, 0.1)$.
- Gaussian blur with a probability of 0.2. The width of the Gaussian kernel used to blur is sampled from $U(0.5, 1.5)$.
- Brightness, where voxel intensities are multiplied by a factor drawn from $U(0.7, 1.3)$ with probability 0.15.

- Contrast, where voxel intensities are multiplied by a factor drawn from $U(0.65, 1.5)$ with probability 0.15.
- Simulation of low resolution by first downsampling by $U(1, 2)$ using nearest neighbor interpolation and then upsampling to the original size with cubic interpolation with probability 0.25.
- Gamma adjustment with probability 0.15. Patch intensities are scaled to be in $[0, 1]$. Then a per-voxel intensity transform is applied: $i_{new} = i_{old}^{\gamma}$, where γ is drawn from $U(0.7, 1.5)$.
- Mirroring along all axes with probability 0.5.

Postprocessing

nnU-Net runs connected component analysis on the predicted segmentations and removes all but the largest component for each class. This is applied on a per-class basis and only if it increases validation performance in terms of the Dice score after cross-validation.

2.10 Datasets

2.10.1 CAMUS

The CAMUS (Cardiac Acquisitions for Multi-structure Ultrasound Segmentation) dataset introduces one of the largest publicly available dataset for training and evaluating machine learning models in 2D ultrasound [3]. It contains clinical exams of 2D echocardiographic sequences, optimized for left ventricular ejection fraction ($\mathrm{LV_{EF}}$) acquisition, with two and four-chamber views from 500 patients at the University Hospital of Saint-Etienne, France. Additionally, annotations for the left ventricle (LV), the myocardium (MYO) and the left atrium (LA) are provided as pixel-wise labels. In total there are 2000 image-annotation pairs from 500 patients. All images are 256×256. The entire dataset was obtained using GE Vivid E95 ultrasound scanners, capturing at least one full cardiac cycle for each patient. No images were removed during collection, and as a result, the quality of the images in the dataset varies. This was intentionally done with the purpose of simulating a real-life setting in the clinic where problems such as parts of the wall not being visible or difficulties with probe orientation could occur. A sample of images from the dataset is shown in Figure 2.7.

2.10.2 HUNT4

`Helseundersøkelsen i Nord-Trøndelag` is a large population-based health study that includes health information and biological material from the residents of Trøndelag.

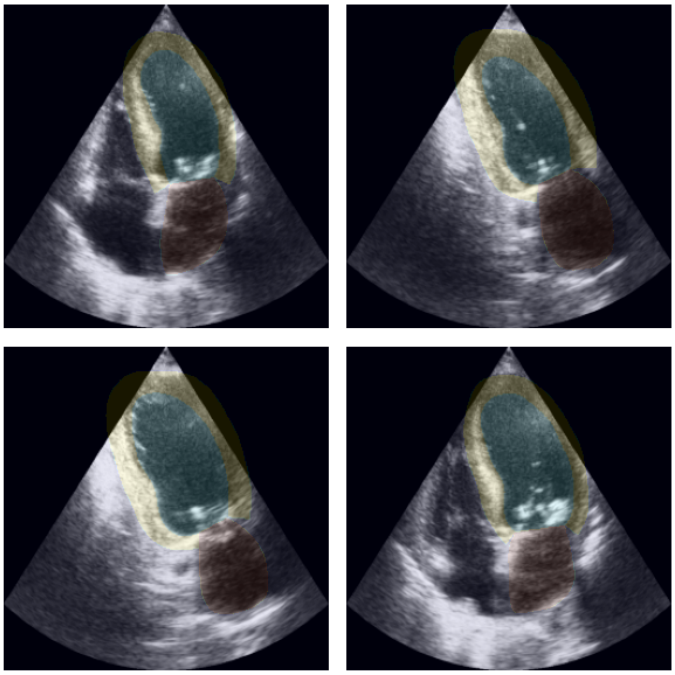


Figure 2.7: Images from CAMUS overlayed with the ground truth segmentations.

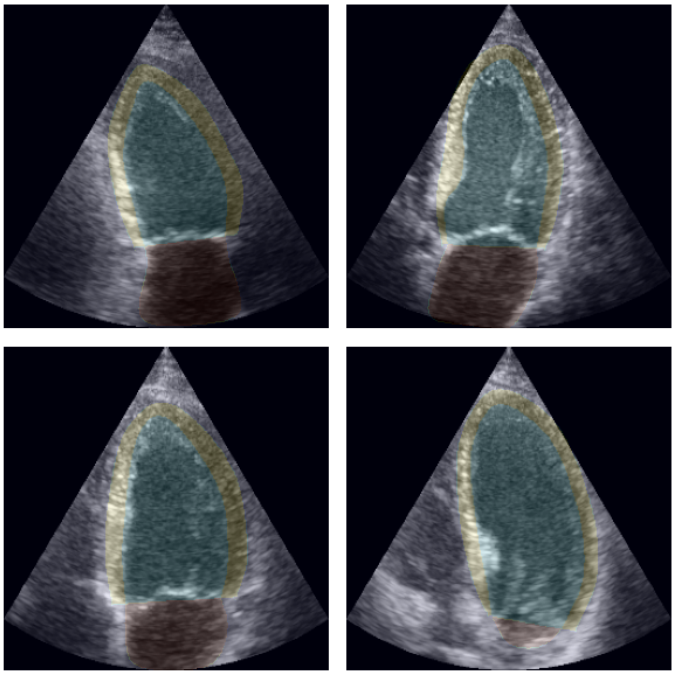


Figure 2.8: Images from HUNT4 overlayed with the ground truth segmentations.

Since the first round of data collection in 1984, 250000 people from Trøndelag have participated. It has an ultrasound dataset (HUNT4Echo) that comprises 2462 patient examinations of LV-focused two- and four chamber views [61]. 311 of the patient exams contain pixel-wise segmentation labels for LV, MYO and LA. This fraction is used as the `HUNT4 annotation set`. Sample images are shown in Figure 2.8.

2.10.3 Differences

The HUNT4 and CAMUS datasets cannot be directly combined and jointly trained on due to differences in annotation conventions and imaging techniques. The annotations for the myocardium are significantly thicker in CAMUS compared to HUNT4. Despite both datasets presenting the same clinical views, images in HUNT4 are LV-focused, while in CAMUS they are not. The annotations in HUNT4 are regarded as more clinically accurate [62]. There is also significantly less variance in the images of HUNT4. Despite this, CAMUS is used for most experiments in this project because it is publicly available, while HUNT4 is not.

Chapter 3

Method

This chapter describes the experiments carried out in order to bridge the gap between nnU-Net and a basic U-Net on the CAMUS dataset. The first sections outline the tools and frameworks used to implement and train the networks as well as the data preprocessing. Then, a baseline is established for both networks, and their architectures and differences are explained. Next, the plan for incrementally adding components of nnU-Net to the basic U-Net is documented. Most additions are from nnU-Net, while some are inspired by other papers. Some of the elements that increase performance are kept for downstream experiments, this is documented in the chapter covering results. The idea is thus to end up with a model containing all of the components that improve performance without adding unnecessary complexity or computational requirements. All models are evaluated based on the Dice coefficient and Hausdorff distance in pixels, as described in 2.22 and 2.28. Anatomical outliers are also counted[1]. To verify whether or not the distribution of the model metrics are statistically significantly different from one another, the Wilcoxon signed-rank test, explained in Section 2.7.6, is used. Chapter 4 shows the results and comparisons between the models and extensions covered in this chapter.

3.1 Frameworks and hardware

3.1.1 Training neural networks using PyTorch

Python is the programming language of choice for many data scientists and machine learning practitioners, thanks in part to powerful deep learning libraries like PyTorch [63]. Developed by Facebook's AI Research lab, PyTorch is renowned for its flexibility, ease of use, and native support for dynamic computational graphs that allow for intuitive deep learning model development. It provides automatic differentiation and abstractions for performing backpropagation. All models in this project were

[1]This is a subjective metric performed by the author, who is not a cardiologist.

developed using PyTorch.A simple recipe for defining and training a neural network in the framework is as follows:

- Define the model architecture by deciding what layers the data should flow through. Then, implement the forward pass through the model. These steps are illustrated in Pseudocode 1.
- Prepare the Dataset. Here, PyTorch needs to know where to read files from and how to process them before returning a single image-label pair, shown in Pseudocode 2. This is where augmentations are applied on the fly and normalization is done. A Dataloader is then used to return batches of samples from the defined dataset to allow parallel computing.
- Define the loss function and optimizer:
  - Choose a loss function appropriate to the task at hand, e.g. Cross-entropy or Dice.
  - Select an optimizer, e.g. SGD or ADAM.
- Train the Model:
  - Loop over the epochs where each epoch represents a full pass through the dataset where each datapoint in the dataset is used once.
  - For each epoch, iterate over the Dataloader to retrieve batches of data.
  - For each batch, perform the following steps:
    1. Zero the gradients to ensure they are not accumulated from previous iterations.
    2. Forward pass the batch through the model.
    3. Calculate the loss between the output and the actual labels.
    4. Perform the backward pass to calculate the gradients.
    5. Update the weights using the optimizer.
- Test the Model:
  - Disable gradient computation.
  - Run the model test set to assess its performance on unseen data.
  - Calculate an appropriate metric to determine how well the model is performing.

3.1.2 Remote development on a GPU-server

GPUs are essential in accelerating the computation-intensive process that is training neural networks. The models in this project are all trained on a server with an Nvidia Quadro P5000 GPU with 16GB VRAM. Although this is an old (2016) and somewhat

Pseudocode 1 General structure of a U-Net in PyTorch.

```
class Model(nn.Module):
  def __init__(self, no_classes)
    super().__init_()
    self.encoder ← nn.Sequential(nn.Conv2d(1, 32, (3, 3)), ...)
    self.decoder ← nn.Sequential(nn.Conv2d(128, 64, (3, 3)), ...)
  def forward(self, x)
    x ← softmax(decoder(encoder(x)))
    return x                                   ▷ x is the output segmentation
```

Pseudocode 2 Basic dataset in PyTorch.

```
class CamusDataset(Dataset):
  def __len__(self)
    return (len(data)
  def __get_item__(self, idx)
    image ← read_image(idx)
    label ← read_label(idx)
    image, label ← transform(image, label)     ▷ Optionally apply augmentations
    return image, label
```

outdated GPU, it is sufficient given the small size of both the models and the dataset involved in this study. The models are developed and trained through SSH-tunnelling in VSCode. Versioning is done using git.

3.2 Data

CAMUS is used for the development, testing and tuning of all components in this project. HUNT4 is used to evaluate the best models and compare their ability to generalize between datasets with varying variance.

For both datasets, an 80/10/10 split for train/val/test is used. For CAMUS this yields 1600 training images, 200 for validation and 200 for testing purposes. For HUNT4 the split gives 846 training images, 105 for validation and 105 for testing. The files were first converted from `.nii` to `.npy` (binary numpy arrays) for easier handling.

3.3 Baselines

Two baselines are established. One for a small vanilla U-Net similar to U-Net 1 [3] and another for nnU-Net [7]. The former is simple and significantly smaller than the latter. See Table 3.1 for a comparison. The goal of the following experiments is to analyze the nnU-Net and incrementally add components to the U-Net 1 baseline to figure out what the most important parts are in terms of performance. All of the experiments using the small U-Net follow the following training scheme:

Architecture	Number of channels	Lowest resolution	Upsampling scheme	Downsampling scheme	Normalization scheme	Batch size	Activation function	Loss function	# Trainable parameters	Augmentations
U-Net 1	$32 \downarrow 128 \uparrow 16$	8×8	Nearest neighbor interpolation	2×2 Maxpool	-	32	ReLU	Dice	2M	-
nnU-Net	$32 \downarrow 512 \uparrow 32$	4×4	Transposed convolutions	Strided convolutions	InstanceNorm	49	Leaky-ReLU	Dice+CE	33M	Rotating and scaling, Gaussian noise, Gaussian blur, Brightness, Contrast, Low-res sim, Gamma and Mirroring

Table 3.1: Architectural differences between the two baselines. Number of channels indicate the number of channels at the first, deepest and last feature map.

- 100 epochs with early stopping (patience=50)
- Adam optimizer with initial learning rate of 0.001
- Batch size 32

For a fair comparison, the nnU-Net model is also trained for 100 epochs. This is unproblematic because it converges before the 100 epochs anyway.

3.4 Working with nnU-Net

Although nnU-Net "provides unprecedented out-of-the-box segmentation performance for essentially any dataset we have evaluated it on", knowing how to modify and tune the framework is a precondition for the experiments in this project. There are two ways of modifying nnU-Net. The first is to modify the customization options inside the nnU-Net framework. This already gives a good amount of modification options. However, for some changes we need to extend the framework by changing the actual code. Both approaches are used and documented below.

Using customization options in the nnU-Net framework

The `plans.json` configuration files are implemented as an abstraction to make changes easier. They let users customize the model within the bounds of the framework, without the the need to touch the code. The file itself is divided into `global` and `local` settings that allow users to make different configurations which are selected during training. The global settings are applied to all configurations, while the local settings define a unique training configuration. The global settings usually remain unchanged, but include the classes used for reading and writing images and handling the labels. The more specific local settings comprise most of the parameters that are usually tuned in standard deep learning projects. These include target spacing, patch size, batch size, normalization scheme, number of filters in the last convolutional layer etc. The complete list of tunable parameters are listed on the authors Github[2]. The configurations, composed of the local settings, can inherit from one another. When

[2]https://github.com/MIC-DKFZ/nnUNet/blob/master/documentation/explanation_plans_files.md

a new configuration is to be tested, it will usually inherit from a base configuration and only a selection of parameters will be changed. Listing 3.1 shows an example of a configuration which would produce a smaller network than the default. These files remove the need to understand the underlying code, making them practical for physicians and other user who are not experts in deep learning. This approach does however come with limitations, particularly as the training loop remains fixed.

```
"2d_bs32_small": {
            "inherits_from": "2d",
            "batch_size": 32,
            "UNet_base_num_features": 16,
            "unet_max_num_features": 64
        }
```

Listing 3.1: nnU-Net custom configuration in plans file. The custom configuration inherits from the `2d` base-config and changes the batch size, number of channels in the first layer and the maximum number of channels in the deepest layer.

Extending nnU-Net

If changing the plans files does not offer the needed customizability, the next step is to modify the code directly. The most common changes are to either replace elements of the training procedure or integrate new network architectures. To modify various components of the training loop, such as the loss function, sampling method, data augmentation scheme, learning rate scheduler and optimizer, the most straightforward approach is to create a custom `trainer` class inside the nnU-Net framework , which inherits from the default trainer class. Then, the desired selection of options from the defaults can be overwritten, e.g. replacing the activation functions or disabling deep supervision. Implementing a new architecture, such as adding residual connections or attention mechanisms, is more elaborate. The authors recommend building a new self-configuring network that uses a GPU-memory estimation method to decide how big the network can be and how large patches it can train on. This approach falls outside the scope of this project, but details are available on the nnU-Net Github[3].

Several experiments are run with nnU-Net using a combination of the two methods described above. First, an experiment is run where the entire data augmentation (DA) scheme is disabled. This is already implemented as an alternative trainer-class in the framework. The only change needed is to train with the validation augmentations, which turns the (image, label) pair into tensors, effectively skipping the augmentation steps. Further, training is restricted to 100 epochs, which is also implemented in the trainer class. Finally, plans files are used to set the batch size to 32.

Next, deep supervision (DS) is switched off, meaning the model trains without both

[3] https://github.com/MIC-DKFZ/nnUNet/blob/master/documentation/extending_nnunet.md

DA and DS. Disabling DS can be achieved with a small change to the code, i.e., by negating a boolean value in the constructor method of the trainer class.

3.5 Normalization scheme

nnU-Net uses Instance normalization (IN) because the networks configured by the framework usually train on very small batch sizes, which BN is poorly suited for [64].

In PyTorch, normalization layers are treated just like any other layer, they are added in the constructor method of the model class. The first addition to the baseline U-Net 1 is BN, but IN is also tested.

3.6 Activation functions

A selection of the activation functions presented in 2.1.2 are tested. The literature on nnU-Net reports best results with Leaky ReLUs. Specifically, models are trained and tested using ReLU, Leaky ReLU, Mish and GELU in experiments where all other parameters are kept the same.

3.7 Postprocessing

The images in CAMUS and HUNT4 only contain one instance of each target structure. This prior information can be applied to the output prediction during postprocessing. Several techniques are tested, including connected component analysis (CCA) and the morphological operations opening and closing. The latter two are both based on the fundamental operations `erosion` and `dilation` [65].

Erosion shrinks or thins objects in a binary image. The erosion of an image A by a structuring element B is defined as:

$$A \ominus B = \{z | B_z \subseteq A\} \tag{3.1}$$

where B_z is the translation of B such that its origin is at z. In this work, all operations use the default structuring element, namely a 3×3 grid with connectivity 1: $\begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$.

Dilation is the dual of erosion and expands or thickens objects in a binary image. It is defined as:

$$A \oplus B = \{z | (\hat{B}_z \cap A) \neq \emptyset\} \tag{3.2}$$

In words, the dilation of A by B is the set of all displacements, z, such that the foreground elements of $\hat{B}$ overlap at least one element of A.

Opening is used to remove small objects from an image while preserving the shape and size of larger objects. Opening of A by B is the result of applying erosion of A by B, followed by dilation of the result by B.

$$A \circ B = (A \ominus B) \oplus B. \tag{3.3}$$

Closing is the dual of opening and is used to fill small holes or gaps in images. This is achieved by doing dilation followed by erosion

$$A \bullet B = (A \oplus B) \ominus B. \tag{3.4}$$

Since these operations are binary, they are performed on a per-class basis.

Ghaffari et al. [66] use connected component analysis to do brain tumour segmentation with U-Nets to discard false positives. nnU-Net also uses a similar method. Therefore, a simple algorithm is devised: Remove all but the largest connected component of each class. These three techniques are tested on U-Net 1. In addition, experiments are run on nnU-Net to understand the impact of its postprocessing engine.

3.8 Data augmentation

Traditional deep learning models generally require large amounts of data to perform well. [67] showed that using data augmentation to artificially increase the size of the training data can achieve the benefit of big data in domains where data is limited, such as MIC. Data augmentation is the next addition to the basic U-Net. Several experiments are performed, where the idea is to incrementally add the augmentations used by nnU-Net, described in Section 2.9. To this end, five configurations with varying strength are tested and the parameters are shown in Table 3.2. Next, experiments are run with only the affine transformations shift, scale and rotate. Finally, the whole augmentation pipeline from nnU-Net is applied and tested. The augmentations are implemented using Albumentations[4], which is a Python package offering fast and flexible image augmentations. It provides access to a large selection of augmentations covering most of the ones commonly used in deep learning based computer vision. The library also has native support for transforming both image and label, which is useful when doing segmentation.

In order to understand how augmentations enrich and extend the dataset, Figure 3.1 shows the five configurations from Table 3.2 along with the original images. The illustration shows that as more aggressive augmentations are applied, the images stop

[4] https://albumentations.ai/docs/api_reference/full_reference/

Config \ Augmentation	Shift	Scale	Rotation angle	Gamma adjustment	Gaussian noise	Blackout probability	Brightness, Contrast, Saturation	Hue
I	$(-0.05, 05)$	$(-0.1, 0.05)$	$(-5, 5)$	$(90, 110)$	$(5, 10)$	0.25	$(0.95, 1.05)$	$(-0.05, 0.05)$
II	$(-0.10, 10)$	$(-0.2, 0.1)$	$(-10, 10)$	$(85, 115)$	$(10, 25)$	0.25	$(0.90, 1.10)$	$(-0.10, 0.10)$
III	$(-0.15, 15)$	$(-0.3, 0.15)$	$(-15, 15)$	$(80, 120)$	$(20, 35)$	0.25	$(0.85, 1.15)$	$(-0.15, 0.15)$
IV	$(-0.20, 20)$	$(-0.4, 0.20)$	$(-20, 20)$	$(70, 130)$	$(30, 45)$	0.25	$(0.80, 1.20)$	$(-0.20, 0.20)$
V	$(-0.25, 25)$	$(-0.5, 0.25)$	$(-30, 30)$	$(60, 140)$	$(40, 50)$	0.25	$(0.75, 1.25)$	$(-0.25, 0.25)$

Table 3.2: Data augmentation experiment plan. The Blackout box is a rectangle with lengths drawn randomly between 25 and 65 pixels and placed randomly in the image. All other parameters are inputs to augmentations available in the Albumentations-package.

looking realistic.

3.9 Loss function

nnU-Net uses the average of Dice and Cross entropy loss. The authors rationale behind this is that the Dice loss addresses the class imbalance well, but say that directly optimizing the evaluation metric comes with drawbacks [7]. More precisely, this is in part due to the fact that their oversampling scheme skews the class distribution seen during training. Empirical results in the literature also show an increase in both training stability and final evaluation accuracy when combining the two losses. Therefore, an experiment is run where the Dice loss is replaced by the average of Dice and Cross entropy loss.

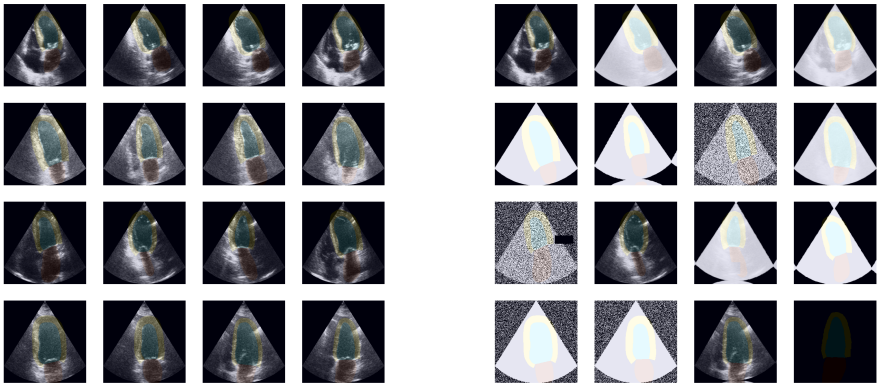
3.10 Deep supervision

The next addition to the basic U-Net from nnU-Net is deep supervision. This requires modification of the network architecture similar to what is shown in Figure 2.4. First, 1×1 convolutions are used to reduce the number of channels down to the number of classes. Then, the feature maps are upscaled using nearest neighbor interpolation, before being run through a softmax activation. Finally, the forward-method of the model needs to not only return the final segmentation, but also the intermediate upsampled feature maps. The modifications needed are shown in Pseudocode 3. The listing omits the code for running the image through the model and only contains the new additions required for DS.

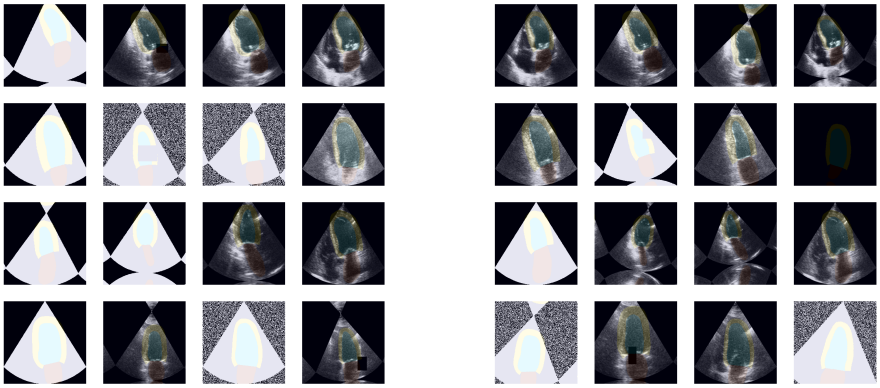
The new loss function is then chosen to be

$$L = L_{\text{final}} + \sum_{i=1}^{n} \lambda_i L_{\text{auxiliary}_i} \tag{3.5}$$

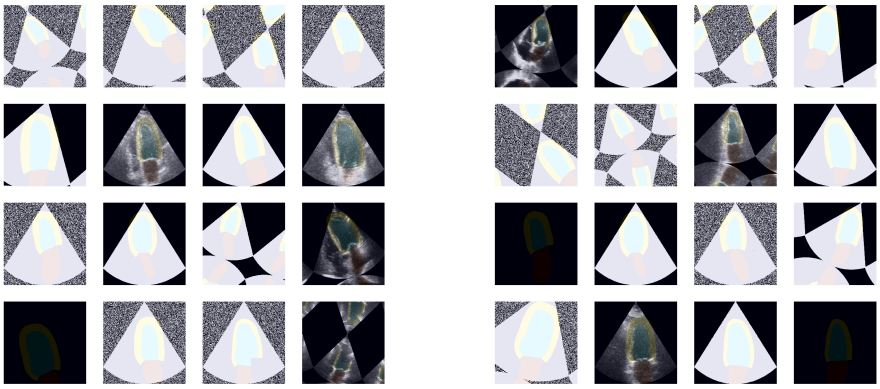
where λ is set to 0.3, L_{final} is the loss between the final prediction and the ground truth, and $L_{\text{auxiliary}_i}$ is the loss between the upscaled intermediate feature map i and the ground truth.



Original images (left) and Config I (right)



Config II (left) and Config III (right)



Config IV (left) and Config V (right)

Figure 3.1: Strength of augmentations are in ascending order top to bottom. Configurations correspond to Table 3.2. Original images are in the top left.

Pseudocode 3 Modified `init` and `forward` methods to include deep supervision

```
def __init__(self, no_classes)
    self.ds1 ← nn.Conv2d(128, no_classes, (1, 1))
    self.ds2 ← nn.Conv2d(64, no_classes, (1, 1))
    self.ds3 ← nn.Conv2d(32, no_classes, (1, 1))
def forward(self, x)
    ds1_out ← softmax(interpolate(self.ds1(up1_x)))      ▷ up_i is the feature map
    ds2_out ← softmax(interpolate(self.ds2(up2_x)))      ▷ from the decoder
    ds3_out ← softmax(interpolate(self.ds3(up3_x)))
    return x, ds1_out, ds2_out, ds3_out                  ▷ x is the output segmentation
```

Architecture	Number of channels	Lowest resolution	Normalization scheme	Batch size	Scheduler	Loss function	# Trainable parameters	Augmentations
U-Net 1	$32 \downarrow 128 \uparrow 16$	8×8	BatchNorm	32	-	Dice	2M	Shift, Scale, Rotate
U-Net 2	$64 \downarrow 512 \uparrow 32$	8×8	BatchNorm	16	ReduceLROnPlateau	Dice	11M	Shift, Scale, Rotate
U-Net 3	$64 \downarrow 1024 \uparrow 64$	16×16	BatchNorm	8	ReduceLROnPlateau	Dice	31M	Shift, Scale, Rotate
U-Net 0	$16 \downarrow 64 \uparrow 16$	64×64	BatchNorm	64	ReduceLROnPlateau	Dice	119K	Shift, Scale, Rotate

Table 3.3: Network architectures with varying size. Number of channels indicate the number of channels at the first, deepest and last feature map.

In order to understand the impact deep supervision has on guiding the intermediate layers in the decoder to make reasonable predictions, upscaled feature maps are extracted for models both with and without DS enabled. The idea is to compare how realistic the predictions made by the early layers in the decoder look for models with and without DS. Although realistic intermediate feature maps do not directly correlate to good final predictions, it can help to visualize what happens inside the black-box model.

3.11 Bigger U-Nets

With nnU-Net being ~ 16 times larger than the basic U-Net, as shown in Table 3.1, a natural extension to our model is to increase the size. Therefore, three networks with varying sizes, shown in Table 3.3, are trained. The goal is to find the needed capacity of a model to learn the distribution of the training set.

U-Net 2 is the first extension with 11M parameters - a significant increase from the 2M parameters of U-Net 1. This growth is attributed to an increase in the number of channels at each stage of both the encoder and decoder, which cumulatively results in a rapid escalation in the total number of parameters. Compared to U-Net 1s 128 channels at the bottleneck, U-Net 2 uses 512.

U-Net 3 features 31M parameters which is close to the one configured by nnU-Net. The main difference between this and U-Net 2 is that it has 1024 channels in the bottleneck and only downsamples the image to a 16×16 spatial size.

Finally, U-Net 0 is a tiny U-Net with just two downsampling operations, meaning the smallest feature maps are 64×64. It has 119K parameters.

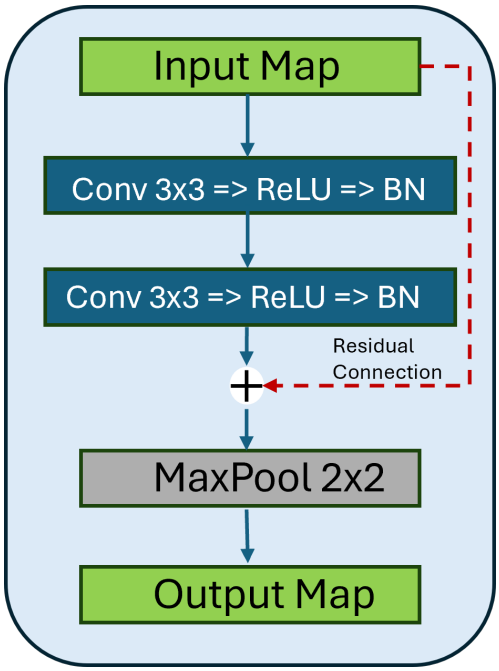


Figure 3.2: Convolution block with residual connection. The output is the addition between the processed input, after convolutions are applied, and the actual input itself. Finally, the result is run through a maxpool.

3.12 Architectural changes

A wide range of architectural modifications are suggested by methods subsequent to U-Net. These include, but are not limited to, residual connections [58], transposed convolutions [44] and attention mechanisms [59]. The first two are compared against U-Net 1 to investigate their impact on a small network trained on limited data.

Residual connections improve gradient flow by letting gradients bypass layers during backpropagation. Although they were initially implemented to mitigate the vanishing gradient problem in very deep networks, [68] show that residual U-Nets exhibit superior performance on segmentation tasks compared to vanilla U-Nets. The idea is to add residual connections to convolution blocks where the input is added to the output of the block. Figure 3.2 shows how an image is processed through such a block. Since all convolution layers maintain the spatial size of the input, it can be directly added to the output. In conjunction with the skip connections in the U-Net between the contracting and expanding path, the residual connections help achieve a smooth loss curve and avoid vanishing gradients. To check whether these connections increase performance, the convolution blocks in U-Net 1 are replaced with residual blocks and compared to the baseline. All other parameters are kept the same.

Transposed convolutions are used as an alternative to traditional upsampling schemes. The main difference is that the filters are learned, while for traditional upsampling methods, such as nearest neighbor interpolation, the images are scaled up based on

Figure 3.3: Checkerboard patterns produced when using transposed convolutions for image generation. Source: [69].

fixed rules. [69] show that transposed convolutions are prone to creating checkerboard artifacts, see Figure 3.3. This is especially the case when the kernel size is not divisible by the stride. To avoid this, the authors recommend disentangling the spatial scaling from the learning of image features, allowing each process to be optimized separately. In practice, this means using traditional upsampling schemes first, before running the image through normal convolution layers, such as in the original U-Net 1 baseline.

An experiment is run where the nearest neighbour interpolation scheme is replaced by transposed convolutions in order to see if it impacts performance. A side-effect of using learned upsampling is the increase in network parameters. To avoid the increased capacity of the model impacting performance, the network with transposed convolutions is scaled down to have approximately the same number of parameters as U-Net 1.

3.13 Results on HUNT4

In order to verify that the addition of the components described above have an impact on performance across multiple datasets, and not just tailored to CAMUS, the models are also trained and tested on HUNT4 to check if the improvements are reflected on another dataset. Similarly, models trained on CAMUS are tested on HUNT4, and vice versa, to see how well the models are able to generalize. There are inherent challenges here, because, as noted in Section 2.10.3, the myocardium-annotations are thicker on CAMUS, which is hard for a model to account for. To compare, the baseline model trained on CAMUS is first tested on HUNT4 without having seen any images from that dataset during training. Then the model is retrained on HUNT4. Finally, the best model, containing all improvements, is run through the same experiment. In order to investigate the capability of the models to generalize from a dataset with low variance, such as HUNT4, to a dataset with relatively higher variance, such as CAMUS, the models are also trained on HUNT4 and tested on CAMUS. Metrics are

reported and predictions are visualized in order to understand how well the models generalize to other similar datasets.

Chapter 4

Experiments & results

This chapter serves as a summary of the experiments carried out in order to find the most impactful components of nnU-Net in terms of performance. It is structured incrementally, meaning it starts with the U-Net 1 baseline model from Table 3.1, before adding the components described in Chapter 3 and evaluating each addition. All experiments are evaluated based on the Dice score and pixel-wise Hausdorff distance. Boxplots are included for each addition.

4.1 Baselines

The metrics for the baselines are shown in Table 4.1. Figure 4.1 shows the corresponding boxplots. The nnU-Net baseline uses all the parameters which are default to the framework, meaning no changes are done. Ensembling is disabled by default for 2D images. This is the best performing nnU-Net tested. The U-Net 1 baseline predicts $\frac{42}{200}$ anatomical outliers. Figure 4.2 shows four randomly selected predictions. For all cases, the model outputs are nonsensical.

The nnU-Net baseline achieves an average dice score of 0.89 and average Hausdorff distance of 8.54 after 100 epochs. Figure 4.1 shows that the spread in score distributions of these two metrics are substantially different.

Model	Dice LV	Dice MYO	Dice LA	Hausdorff LV	Hausdorff LV	Hausdorff MYO	# Anatomical Outliers
U-Net 1	0.90	0.82	0.86	10.13	12.87	12.04	42
nnU-Net	0.93	0.86	0.89	7.10	9.21	9.31	6

Table 4.1: Baseline metrics.

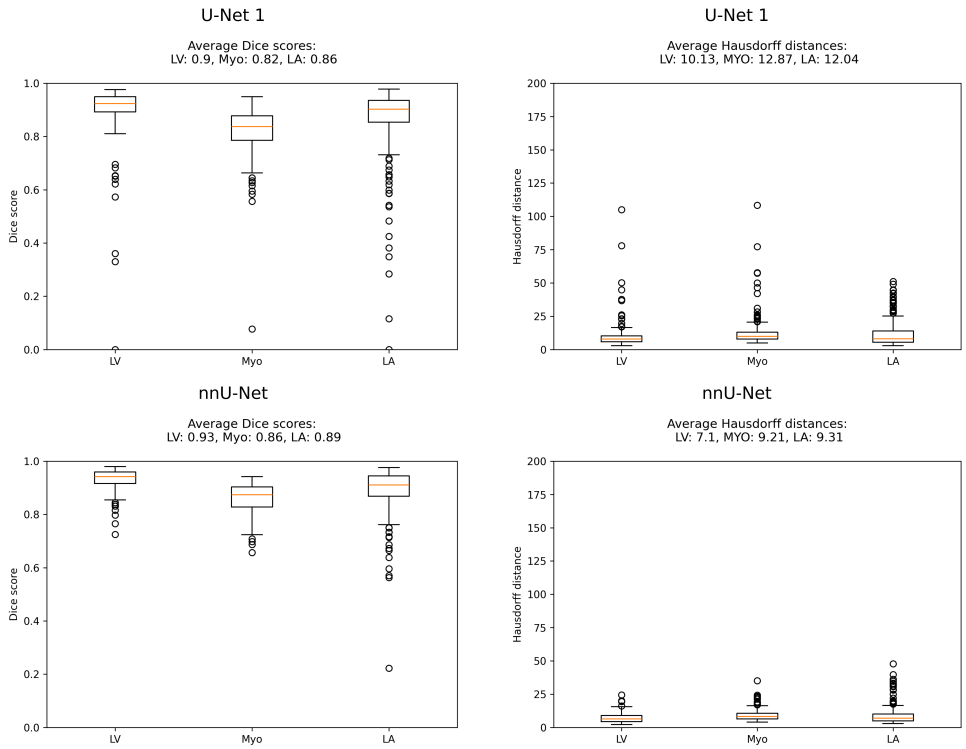


Figure 4.1: Boxplots of the Dice score and Hausdorff distances for the two baselines for the left ventricle lumen (LV), myocardium (MYO), and left atrium (LA)

4.2 Experiments with nnU-Net

This section covers the experiments where some parts are removed from nnU-Net to investigate their impact on performance. The first experiment aims to examine the influence of data augmentation. Thus, the entire data augmentation scheme is disabled. This results in the Dice scores and Hausdorff distances shown in Figures 4.3 and 4.4. The difference in both metrics between the results with the baseline and the results without augmentations are statistically significant with $p < 0.05$, using the Wilcoxon signed-rank test, except the Dice score for the LA class.

To investigate the importance of deep supervision, the next experiment trains without both data augmentation and deep supervision. Here, nnU-Net achieves the results shown in Figures 4.5 and 4.6. With $p < 0.05$ both the Dice score and Hausdorff distance is significantly worse than the baseline for all three classes. However the difference is not statistically significant compared to only disabling data augmentations, indicating that for nnU-Net, augmentations are more important that deep supervision.

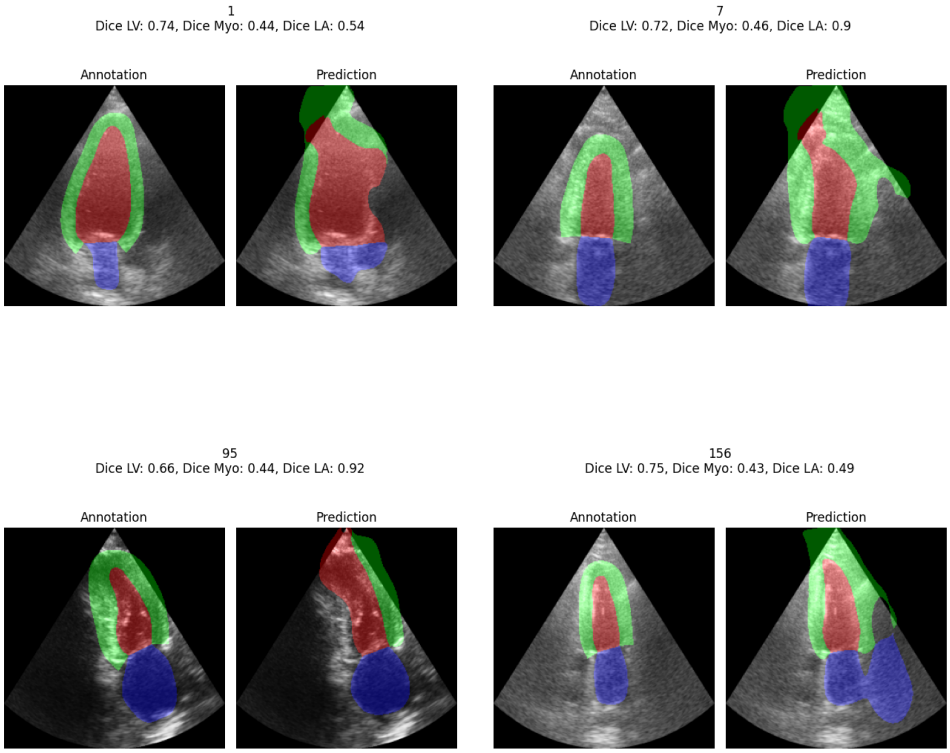


Figure 4.2: Randomly selected predictions from U-Net 1 baseline.

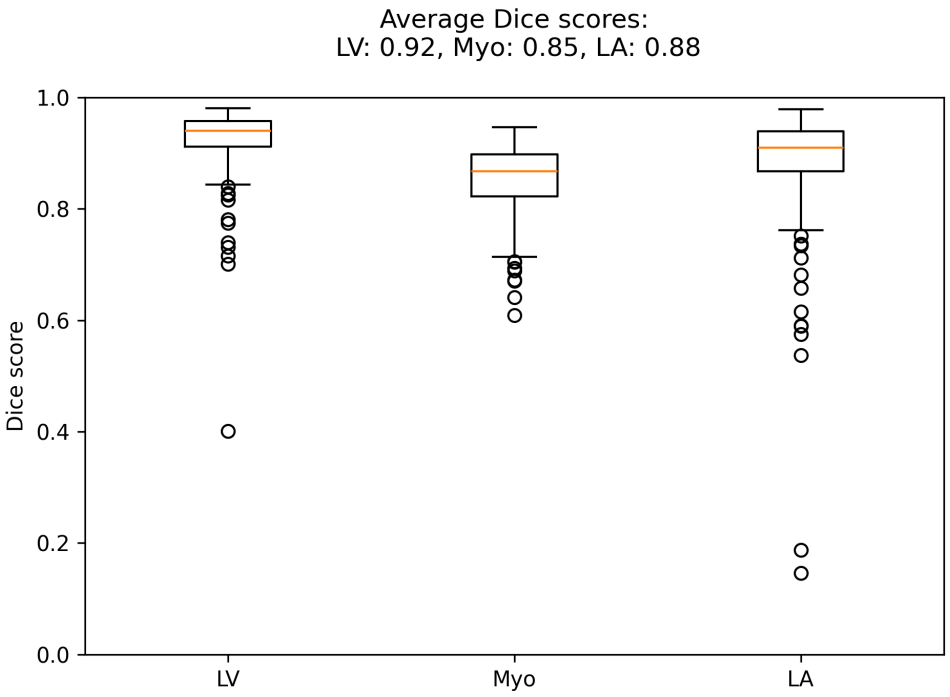


Figure 4.3: Dice scores for nnU-Net trained without data augmentation.

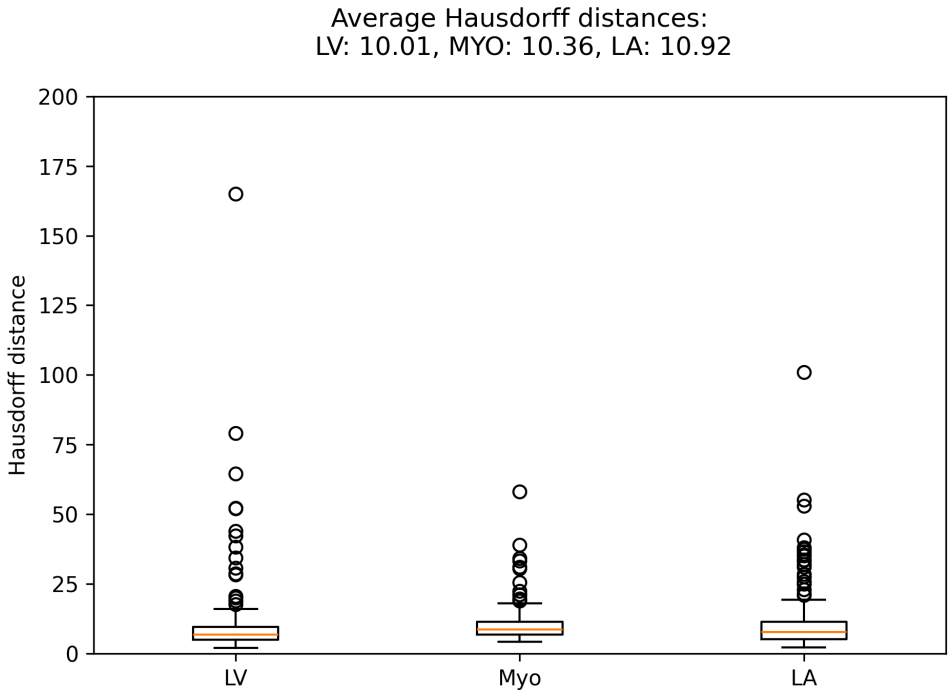


Figure 4.4: Hausdorff distances for nnU-Net trained without data augmentation.

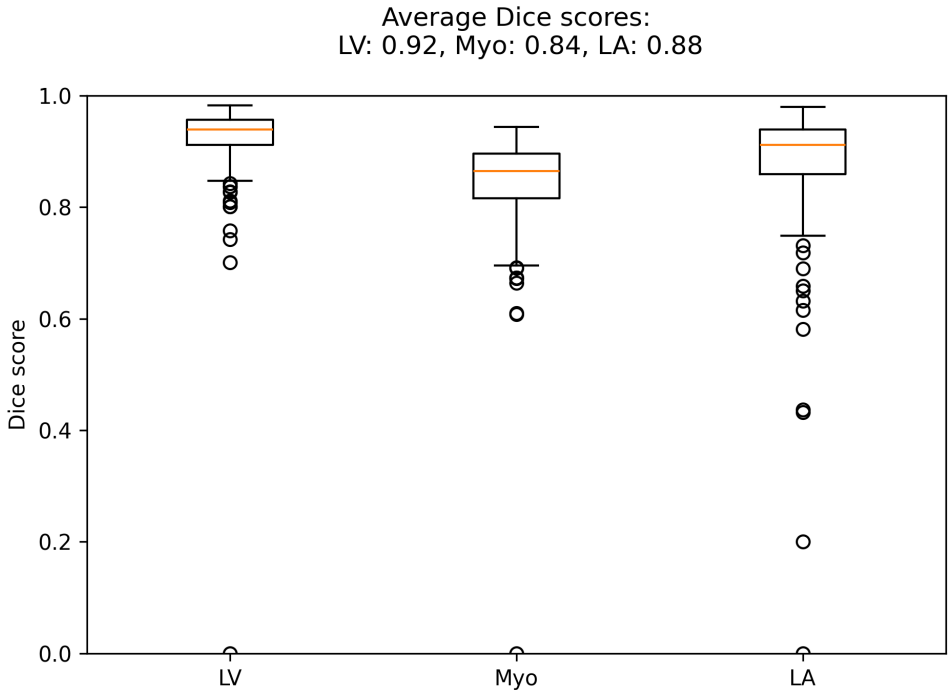


Figure 4.5: Dice scores for nnU-Net trained without data augmentation and deep supervision.

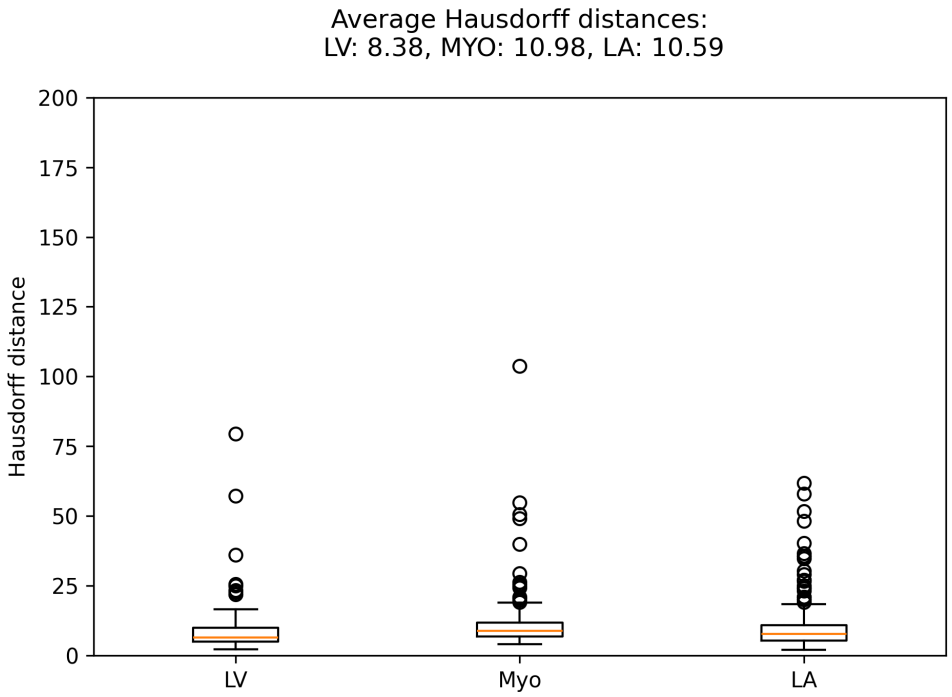


Figure 4.6: Hausdorff distancess for nnU-Net trained without data augmentation and deep supervision.

Effect of postprocessing

To examine the effect of the connected component analysis done by nnU-Nets post-processing engine, the predictions of the models above are compared before and after postprocessing is applied. Results are summarized in Table 4.2. Using the Wilcoxon signed-rank test with $p < 0.05$, the difference in both metrics is significant between before and after postprocessing only for some of the architectures. In particular, post-processing does not improve performance significantly on the baseline. For the model trained without augmentations, both metrics improve for LV and MYO, but not for LA. The model trained without augmentations or deep supervision is statistically significantly worse for all classes on both metrics with $p < 0.05$ when postprocessing is enabled. The noticeable difference in Hausdorff distance specifically is likely due to the fact that it measures the maximum distance. Thus, the worst-case distance is more prone to outliers, which the postprocessing helps reduce. Postprocessing is more important for models trained without regularization techniques that help generalization ability, as illustrated in Figure 4.7. It shows a significant increase in Hausdorff distance when postprocessing is disabled.

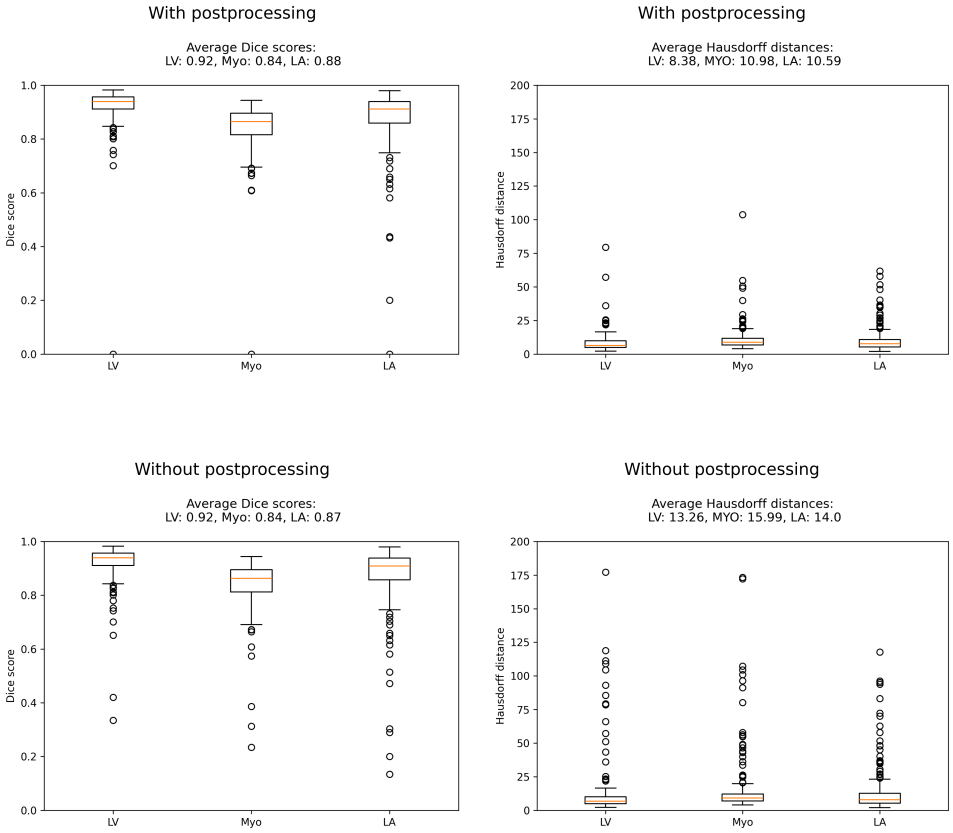


Figure 4.7: Difference with and without postprocessing for nnU-Net trained without data augmentation and deep supervision in terms of Dice score and Hausdorff distance.

nnU-Net Architecture	**Post-processing?**	**Dice LV**	**Dice MYO**	**Dice LA**	**Hausdorff LV**	**Hausdorff LV**	**Hausdorff MYO**
Baseline	No	0.93	0.86	0.89	7.43	9.21	9.47
Baseline	Yes	0.93	0.86	0.89	7.10	9.21	9.31
No DA	No	0.92	0.84	0.87	11.85	18.22	14.57
No DA	Yes	0.92	0.85	0.88	10.01	10.36	10.92
No DA, no DS	No	0.92	0.84	0.87	13.26	15.99	14.00
No DA, no DS	Yes	0.92	0.84	0.88	8.38	10.98	10.59

Table 4.2: Effect of postprocessing (pp) for different nnU-Nets.

Config	Dice LV	Dice MYO	Dice LA	Hausdorff LV	Hausdorff LV	Hausdorff MYO	# Anatomical Outliers
Baseline	0.90	0.82	0.86	10.13	12.87	12.04	42
Baseline w/ BN	0.91	0.84	0.86	10.08	11.64	12.38	26
Baseline w/ IN	0.92	0.84	0.87	10.25	13.43	13.63	26

Table 4.3: Dice score and Hausdorff distance for the U-Net 1 baseline with Batch normalization (BN) and Instance normalization (IN).

Activation	Dice LV	Dice MYO	Dice LA	Hausdorff LV	Hausdorff LV	Hausdorff MYO
ReLU	0.91	0.84	0.86	10.08	11.64	12.38
Leaky ReLU	0.91	0.84	0.86	9.44	12.19	10.65
GELU	0.91	0.84	0.88	9.36	11.80	9.98
MISH	0.92	0.84	0.87	8.97	12.27	11.15

Table 4.4: Model metrics from testing activation functions.

4.3 Normalization scheme

Table 4.3 summarizes the results from adding normalization to the baseline U-Net 1. The boxplots are visualized in Figure 4.8. There is an increase in performance for both schemes compared to the baseline, with Batch normalization giving the highest overall increase in performance. Both normalization schemes reduce the number of outliers drastically, from $\frac{42}{200}$ to $\frac{26}{200}$. Therefore, BN is used in all subsequent models.

4.4 Activation functions

Table 4.4 summarizes the results from testing a selection of activation functions. Since they are all variations of ReLU, similar performance is expected. Although all activation functions give comparable results, the Gaussian error linear unit (GELU) and MISH performs best. MISH is used in subsequent experiments.

4.5 Postprocessing

In light of the results in Section 4.2 showing that postprocessing significantly improves performance on models trained without augmentations and deep supervision, similar tests are run on the smaller U-Net. The three postprocessing techniques described in 3.7 are tested on the baseline model with and without BN. The results are summarized in Table 4.5. While the Dice scores are more or less consistent for all techniques, as illustrated in Figure 4.10, the Hausdorff distance varies to a larger extent depending on what postprocessing is applied. This is illustrated with boxplots in Figure 4.9. From these results, it is concluded that a) postprocessing the output helps improve the segmentations, and b) that connected component analysis (CCA) is the best performing technique. Therefore it is included as a part of the inference pipeline in

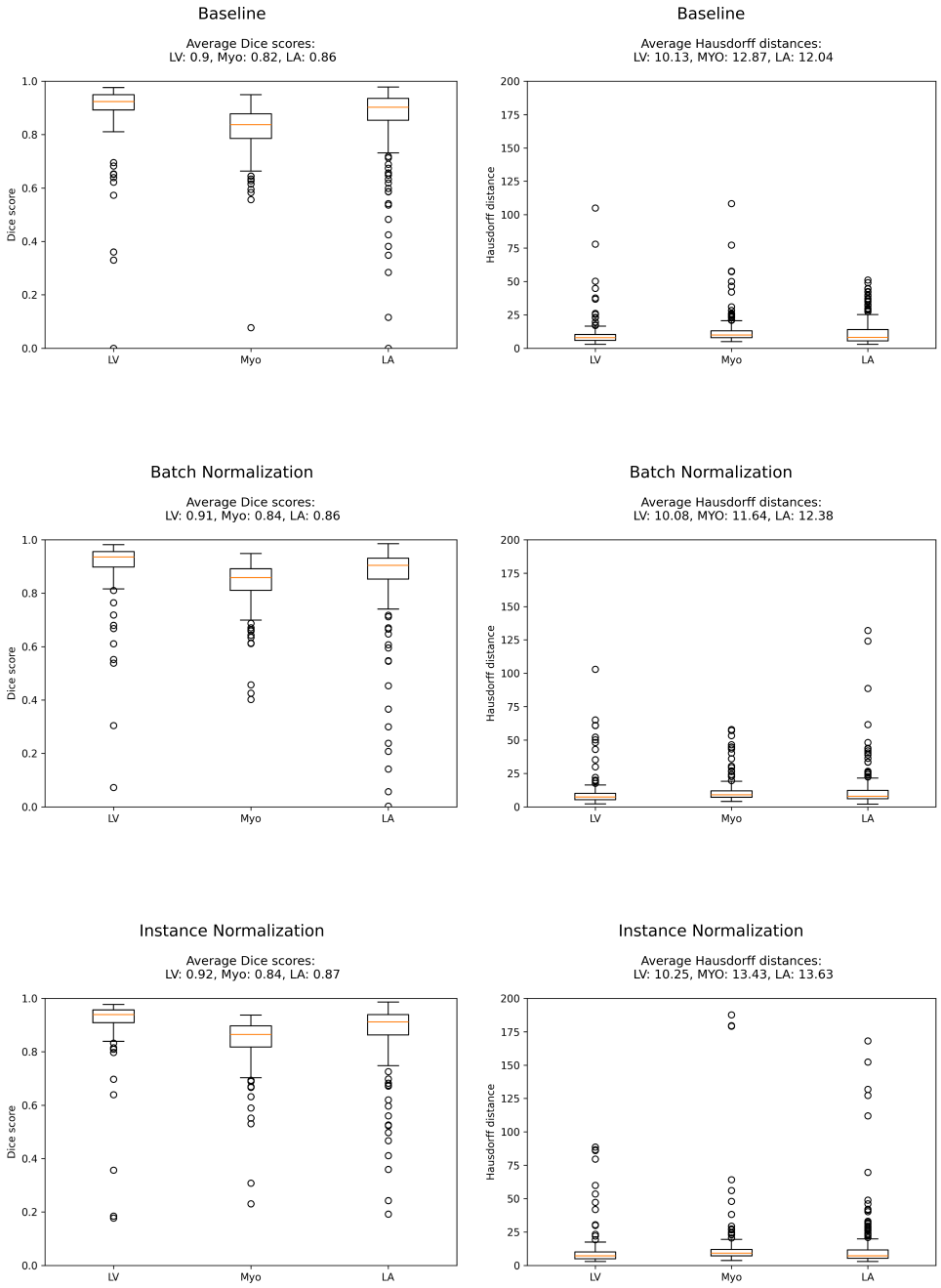


Figure 4.8: Dice score and Hausdorff distance for the U-Net 1 baseline with Batch and Instance normalization.

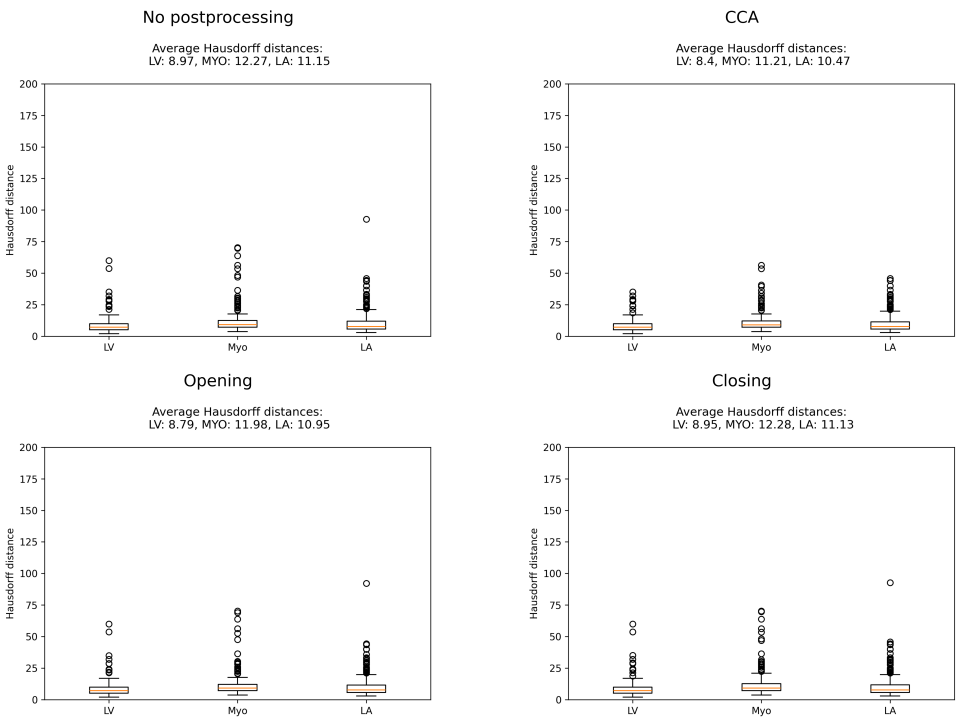


Figure 4.9: Hausdorff distances for different postprocessing techniques on the U-Net 1 baseline with BN.

all subsequent experiments.

4.6 Optimizing data augmentation

The next addition to U-Net 1 from the nnU-Net pipeline is data augmentation. Initial experiments are run with the configurations in Table 3.2, with accompanying results in Table 4.6 and Figures 4.15 and 4.16. The results show that a moderate degree of augmentations is optimal. The number of anatomical outliers are reduced from 26 down to 10 for the best scheme. It is concluded that configuration II is best suited out of the five configurations using the same transforms.

U-Net Architecture	**Post-processing?**	**Dice LV**	**Dice MYO**	**Dice LA**	**Hausdorff LV**	**Hausdorff LV**	**Hausdorff MYO**
Baseline	No	0.90	0.82	0.86	10.13	12.87	12.04
Baseline	Yes, CCA	0.90	0.82	0.85	9.76	12.02	12.07
Baseline	Yes, Opening	0.90	0.82	0.86	10.24	12.84	12.06
Baseline	Yes, Closing	0.90	0.82	0.86	10.14	12.88	12.03
Baseline w/ BN	No	0.92	0.84	0.87	8.97	12.27	11.15
Baseline w/ BN	Yes, CCA	0.92	0.84	0.87	8.40	11.21	10.47
Baseline w/ BN	Yes, Opening	0.92	0.84	0.87	8.79	11.98	10.95
Baseline w/ BN	Yes, Closing	0.92	0.84	0.87	8.95	12.28	11.13

Table 4.5: Effect of postprocessing for different U-Net configurations. Both models are run with connected component analysis (CCA), opening and closing.

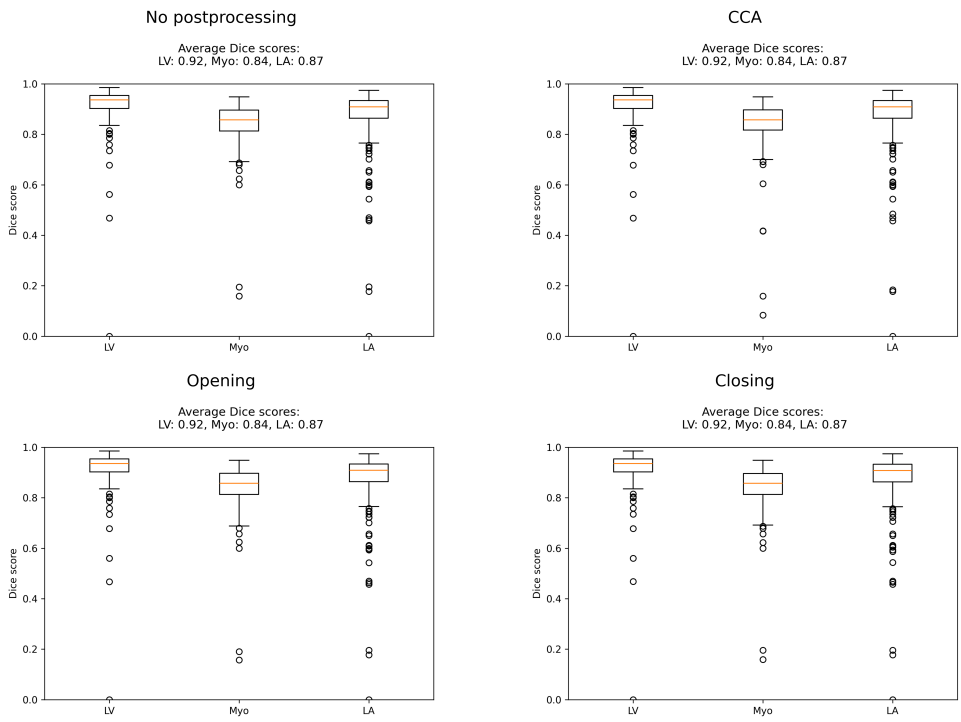


Figure 4.10: Dice scores for different postprocessing techniques on the U-Net 1 baseline with BN.

Second, an experiment is run where only the affine geometric transforms shift, scale and rotate are applied. The parameters are kept the same as configuration II, shown in Table 3.2. Surprisingly, this yields, with $p < 0.05$ a significantly higher Dice score and lower Hausdorff distance across all classes using the Wilcoxon signed-rank test. Figures 4.11 and 4.12 show that especially the Hausdorff distance is reduced considerably. Although only training with affine augmentations gives the best overall metrics, this model does have more anatomical outliers compared to models trained with more aggressive augmentations.

The final experiment uses exactly the same augmentations and parameters as the nnU-Net defaults, which are outlined in Section 2.9.2. The results are shown in Figures 4.13 and 4.14. The metrics are with $p < 0.05$ significantly worse than the configuration II-model above for LV and MYO on the Dice metric, and significantly worse for LV in terms of Hausdorff distance. Compared to the experiment with only geometric augmentations, the model performs significantly worse across both metrics for all classes with $p < 0.05$.

4.7 Selecting the best loss function

Replacing the original Dice loss function with the average of Dice loss and Cross entropy gives the Dice score and Hausdorff distance shown in Figures 4.17 and 4.18.

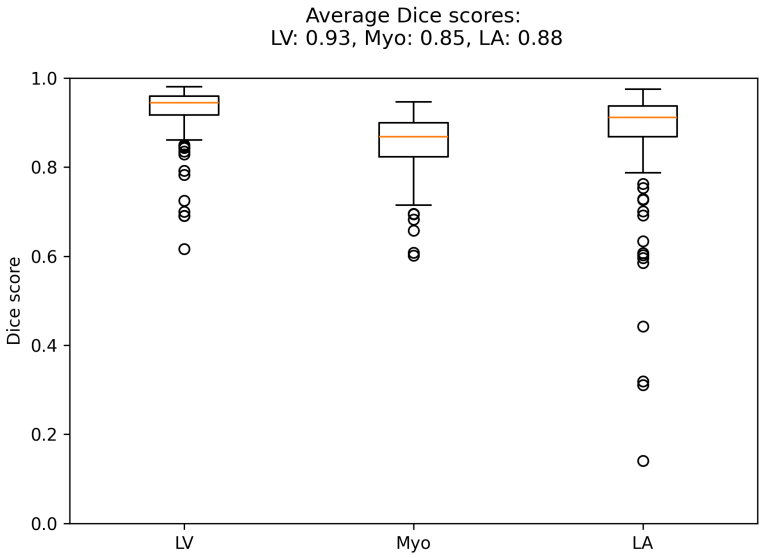


Figure 4.11: Dice score for U-Net trained with only geometric augmentations.

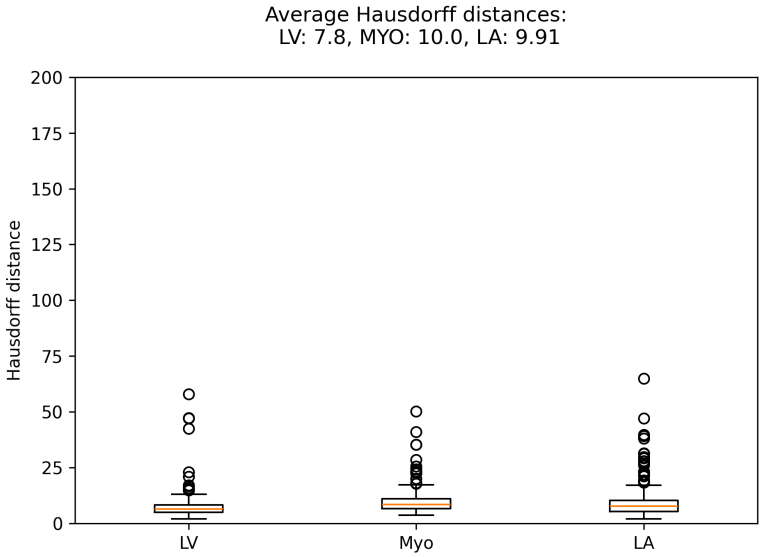


Figure 4.12: Hausdorff distance for U-Net trained with only geometric augmentations.

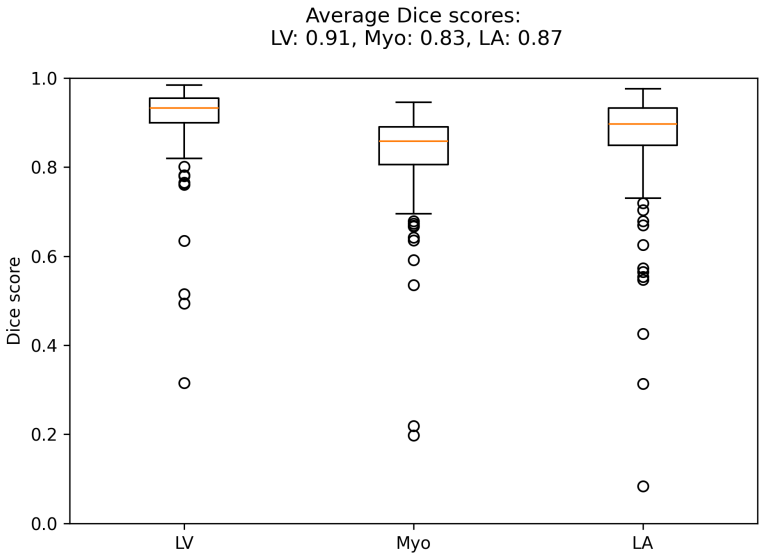


Figure 4.13: Dice score for U-Net trained with nnU-Net augmentations.

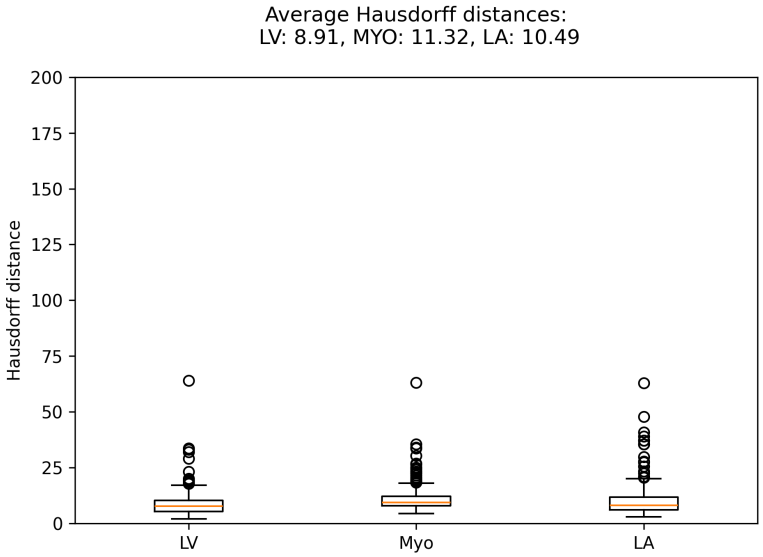


Figure 4.14: Dice score for U-Net trained with nnU-Net augmentations.

Config	Dice LV	Dice MYO	Dice LA	Hausdorff LV	Hausdorff LV	Hausdorff MYO	# Anatomical Outliers
Baseline	0.90	0.82	0.86	10.13	12.87	12.04	42
Baseline w/ BN	0.92	0.84	0.87	8.40	11.21	10.47	26
I	0.92	**0.85**	**0.88**	8.38	11.13	9.93	12
II	0.92	**0.85**	**0.88**	8.22	10.96	10.56	**10**
III	0.92	0.84	0.87	8.57	11.13	10.72	**10**
IV	0.91	0.83	0.87	9.46	12.67	10.94	14
V	0.82	0.58	0.69	27.27	55.12	26.17	66
Affine	**0.93**	**0.85**	**0.88**	**7.80**	**10.00**	**9.91**	12
nnU-Net transforms	0.91	0.83	0.87	8.91	11.32	10.49	16

Table 4.6: Dice score and Hausdorff distance for U-Net 1 with different augmentation configurations

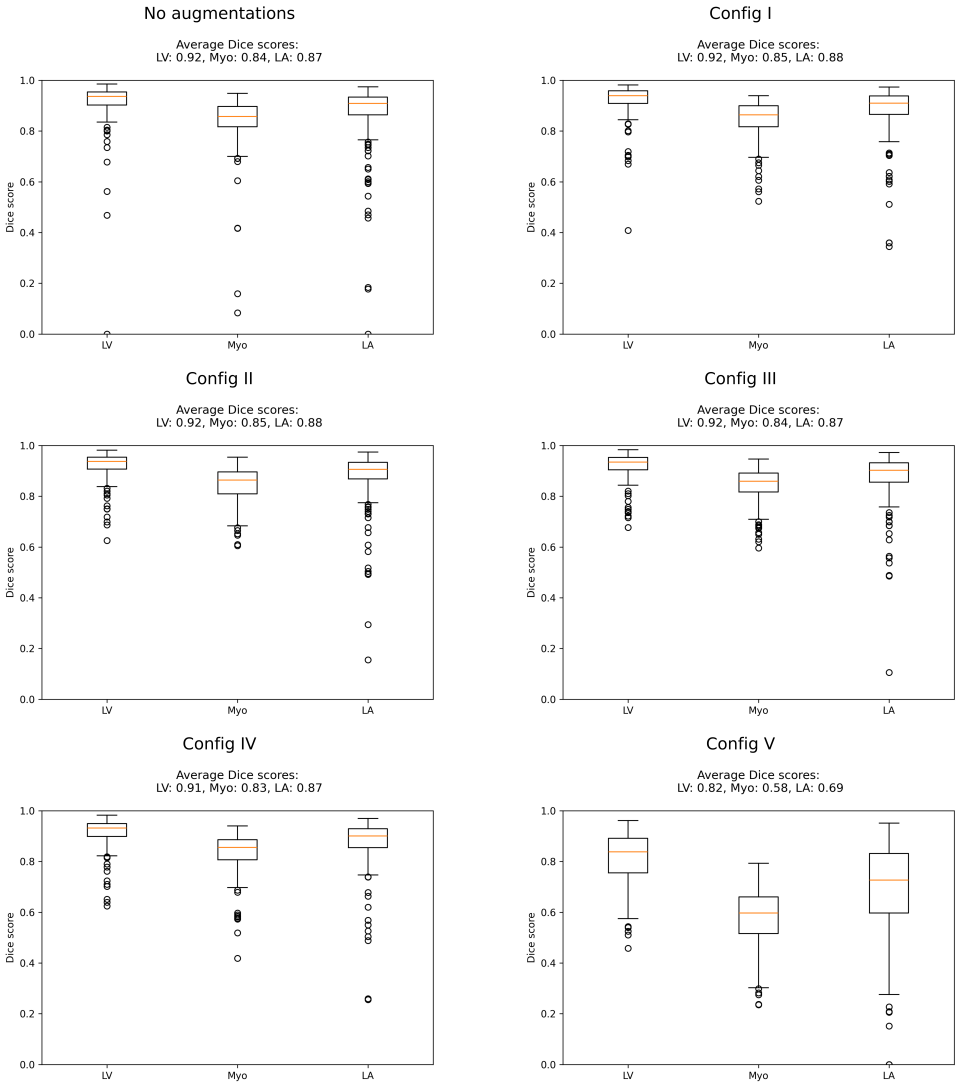


Figure 4.15: Dice scores for the five configurations in Table 3.2.

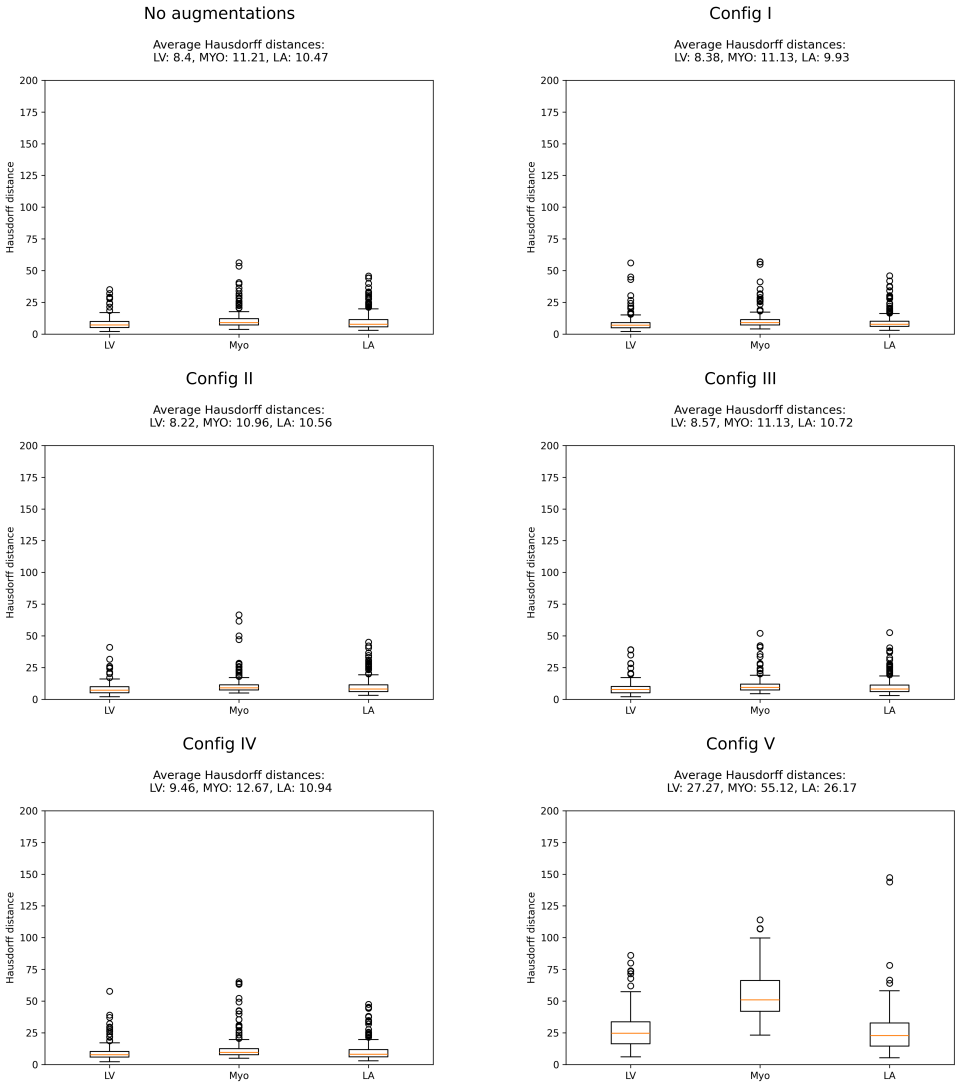


Figure 4.16: Hausdorff distances for the five configurations in Table 3.2.

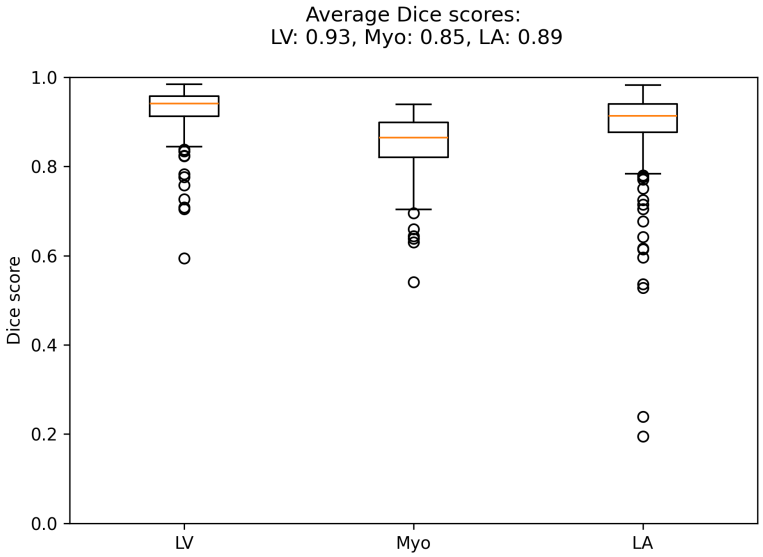


Figure 4.17: Dice score with average of Dice loss and Cross entropy loss as the objective function.

With $p > 0.05$, none of these results are significantly different from only using the Dice loss. However, the number of anatomical outliers are halved from 12 to 6, indicating increased robustness of the model. This is further underpinned by looking at Figures 4.19 and 4.20 which shows a predicted sample from the same model with and without the addition of Cross entropy loss.

4.8 Deep supervision

Adding deep supervision to the baseline U-Net with Batch normalization gives the results in Figures 4.21 and 4.22. The metrics of the new model are, with $p < 0.05$, statistically significantly better than without deep supervision for the Hausdorff distance of the LV, but not significantly better for the remaining metrics and labels. Compared to the model with affine augmentations, it performs significantly worse on all metrics except the Hausdorff distance of the LV.

Figures 4.23 and 4.24 show the results when combining the affine augmentations with deep supervision. Both metrics are, with $p < 0.05$, statistically significantly better compared to training without deep supervision.

To investigate how deep supervision forces the decoder blocks to make valid predictions with respect to the label, the upscaled feature maps of a model with and without DS are visualized. Figure 4.25 shows upscaled feature maps generated by a model trained with DS. The feature maps from the first layers of the decoder are somewhat pixelated because they are upscaled by a factor of 16. Figure 4.26 shows the feature maps of a model trained without DS from the same image in the test set. It is clear

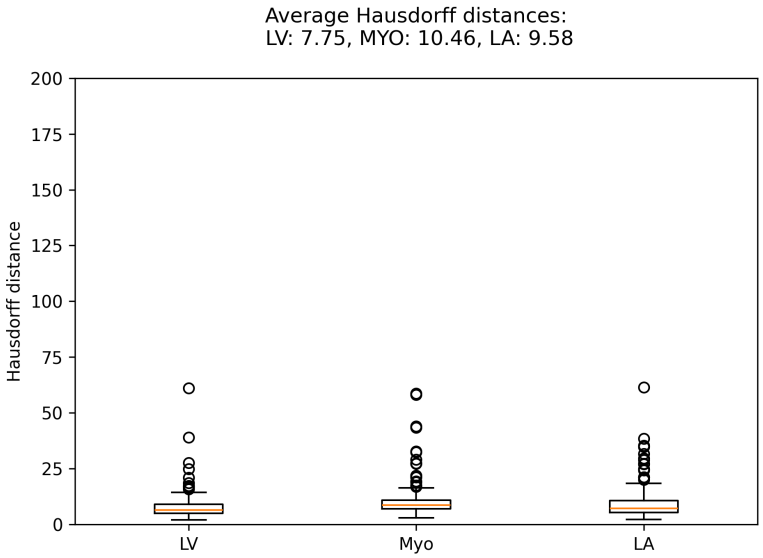


Figure 4.18: Hausdorff distance with average of Dice loss and Cross entropy loss as the objective function.

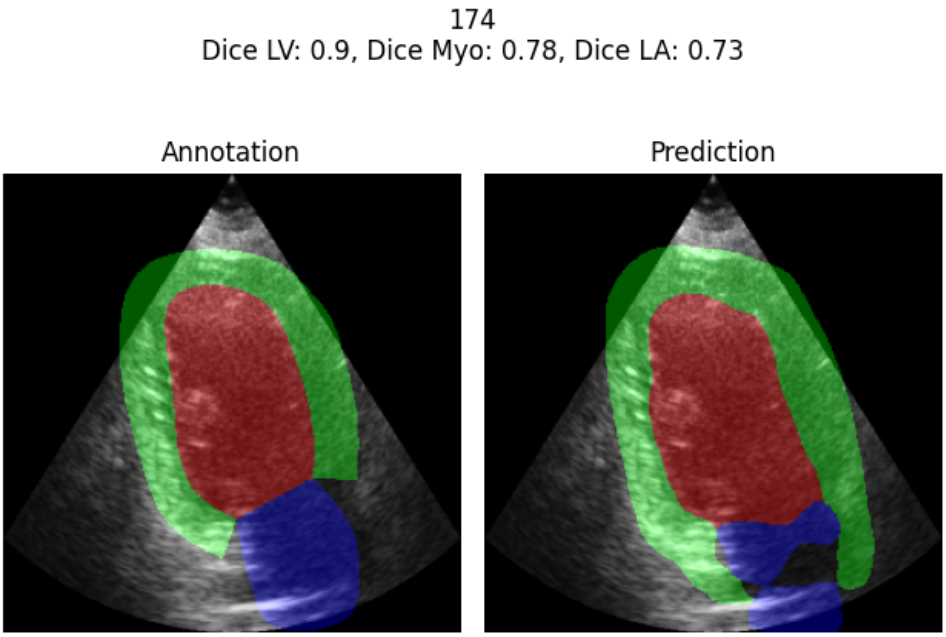


Figure 4.19: Prediction with Dice loss as objective function.

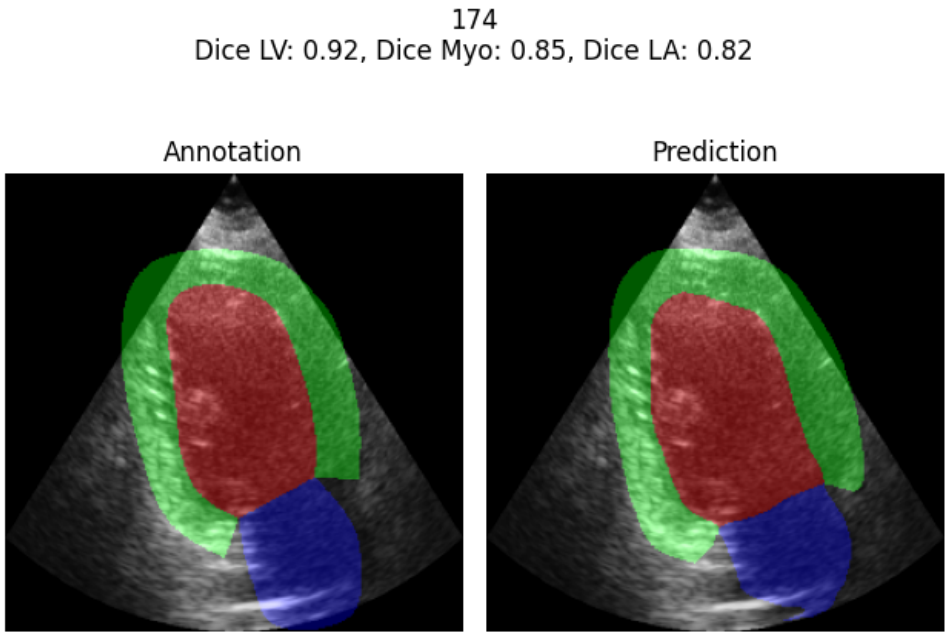


Figure 4.20: Prediction with average of Dice loss and Cross entropy as objective function.

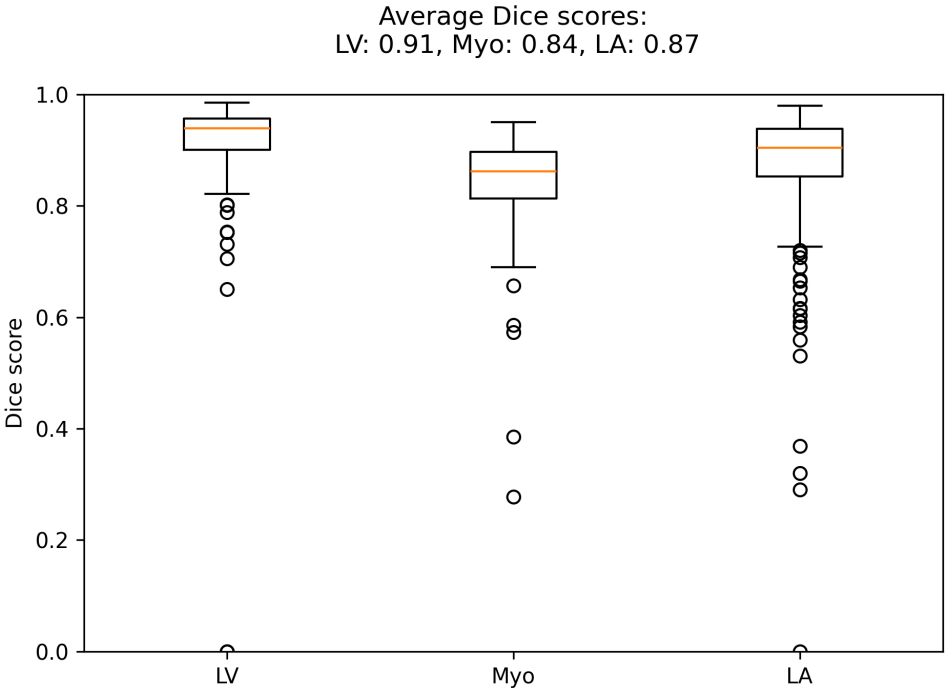


Figure 4.21: Dice score of the model trained with deep supervision. No augmentations are applied.

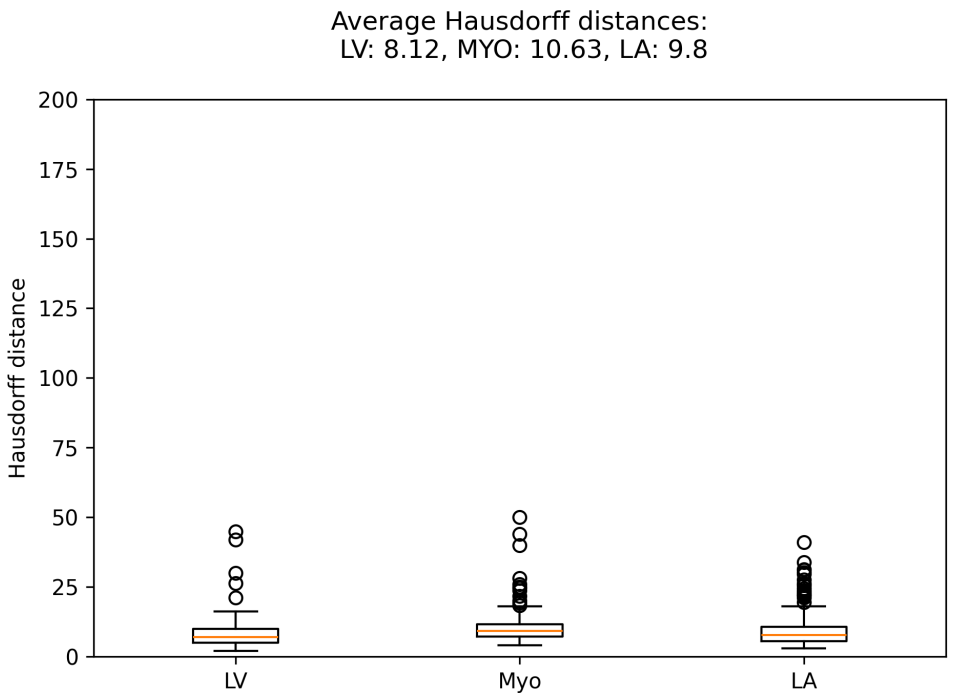


Figure 4.22: Hausdorff distance of the model trained with deep supervision. No augmentations are applied.

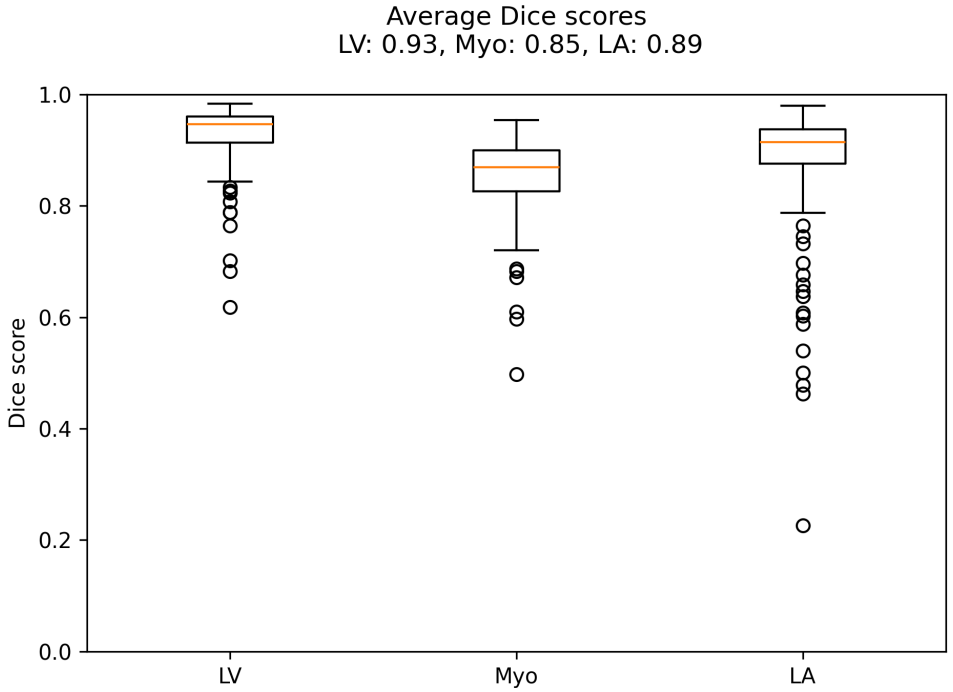


Figure 4.23: Dice score of the model trained with deep supervision and affine data augmentations.

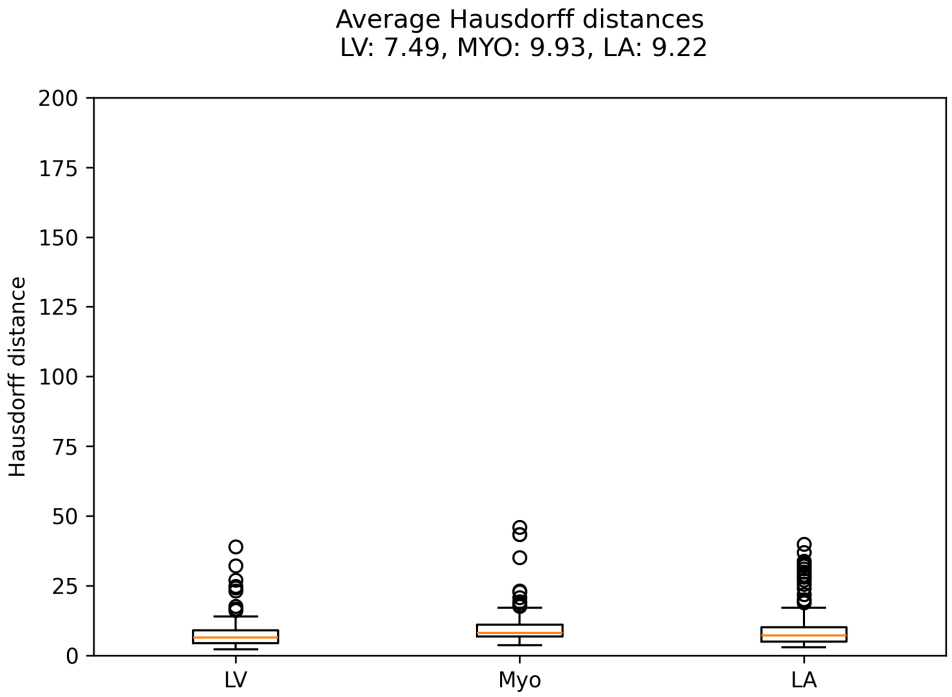


Figure 4.24: Hausdorff distance of the model trained with deep supervision and affine data augmentations.

that the intermediate layers of models trained with DS predict more anatomically valid segmentation maps than models without.

To investigate how DS impacts model complexity, the number of learnable parameters are found using 2.16. For the example in Pseudocode 3, with 4 output classes, DS will add a total of $(128+1)\cdot 4+(64+1)\cdot 4+(32+1)\cdot 4 = 908$ additional parameters. For a network with 2M parameters, this is negligible for all intents and purposes.

4.9 Bigger U-Nets

The U-Net 1 network has 2M trainable parameters. To investigate if this size is sufficient for the task, the models outlined in Table 3.3 are tested. All models are trained with the affine augmentations discussed in Section 3.8, deep supervision enabled, nearest neighbour interpolation and maxpooling as upsampling and downsampling schemes, and finally ReduceLROnPlateau[1] as the learning rate scheduler. Table 4.7 compares the new networks with the current best U-Net 1. Boxplots of Dice scores and Hausdorff distances are visualized in Figures 4.27 and 4.28. The results show that none of the larger networks perform better than the standard 2M parameter one, while the smaller U-Net 0 performs much worse. It is therefore concluded that the size of U-Net 1 is sufficient to learn the distribution in the CAMUS dataset.

[1] https://pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.ReduceLROnPlateau.html

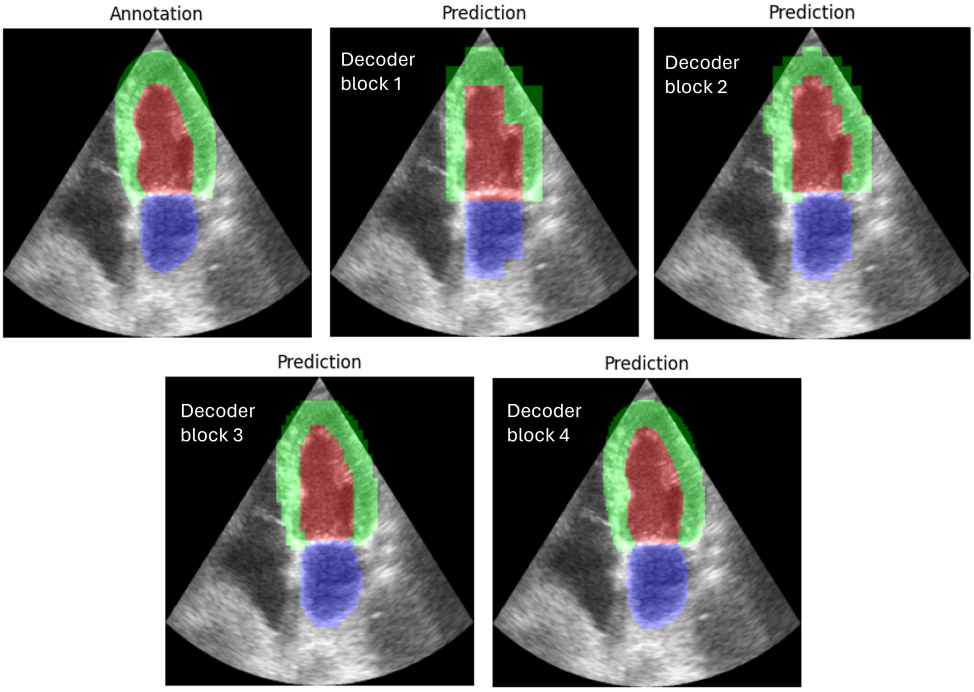


Figure 4.25: Upscaled feature maps of a model trained with deep supervision. All feature maps are from the same input, but extracted at different levels of the decoder. The images are of predictions generated during inference.

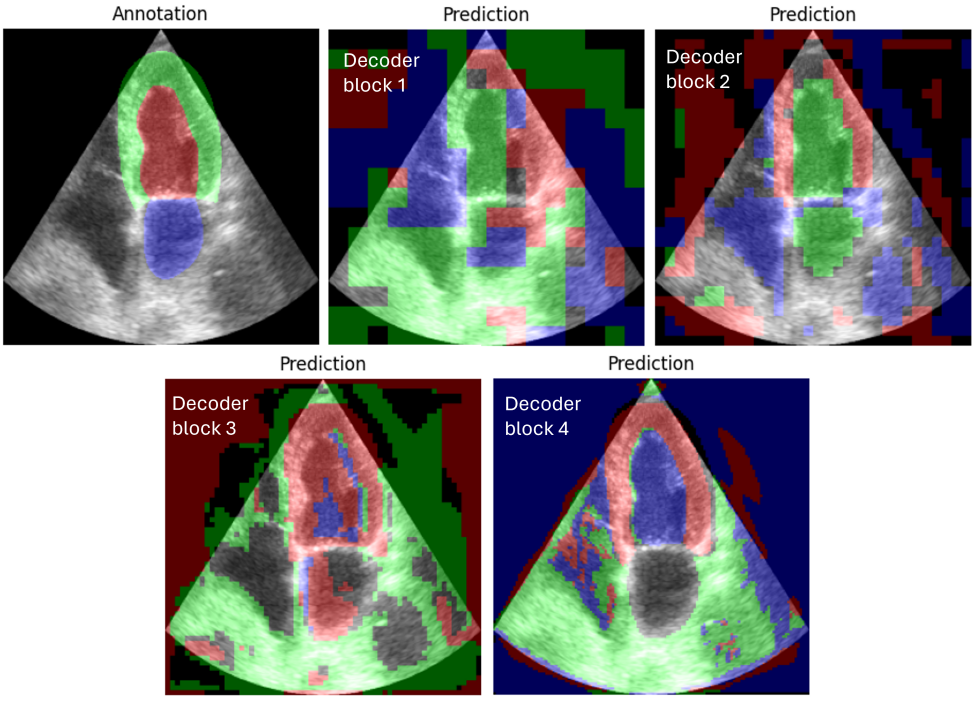


Figure 4.26: Upscaled feature maps of a model trained without deep supervision.

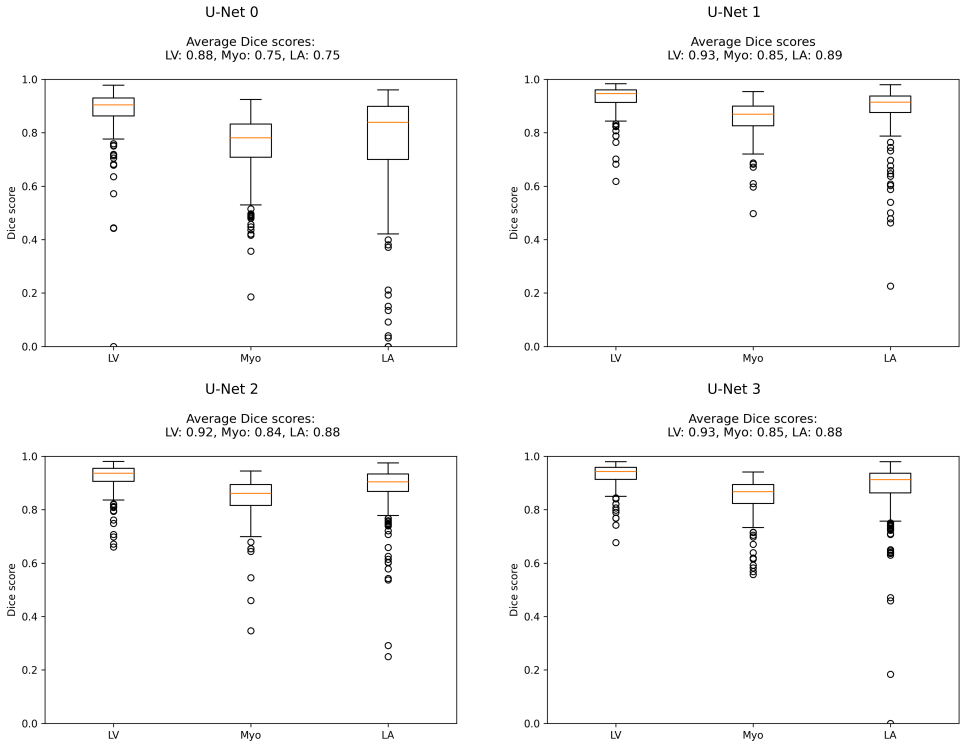


Figure 4.27: Boxplots of Dice scores for networks with varying size.

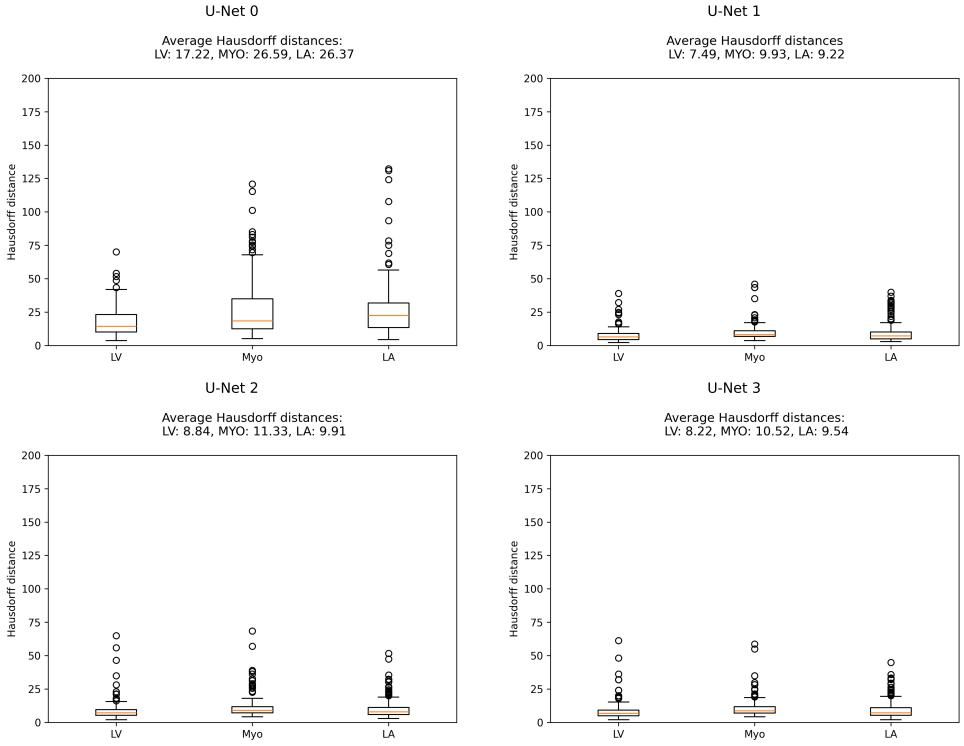


Figure 4.28: Boxplots of Hausdorff distances for networks with varying size.

Architecture	Dice LV	Dice MYO	Dice LA	Hausdorff LV	Hausdorff LV	Hausdorff MYO
U-Net 1	0.93	0.85	0.89	7.49	9.93	9.22
U-Net 2	0.92	0.84	0.88	8.84	11.33	9.91
U-Net 3	0.93	0.85	0.88	8.22	10.52	9.54
U-Net 0	0.88	0.75	0.75	17.22	26.59	26.37

Table 4.7: Model metrics for the the networks in Table 3.3.

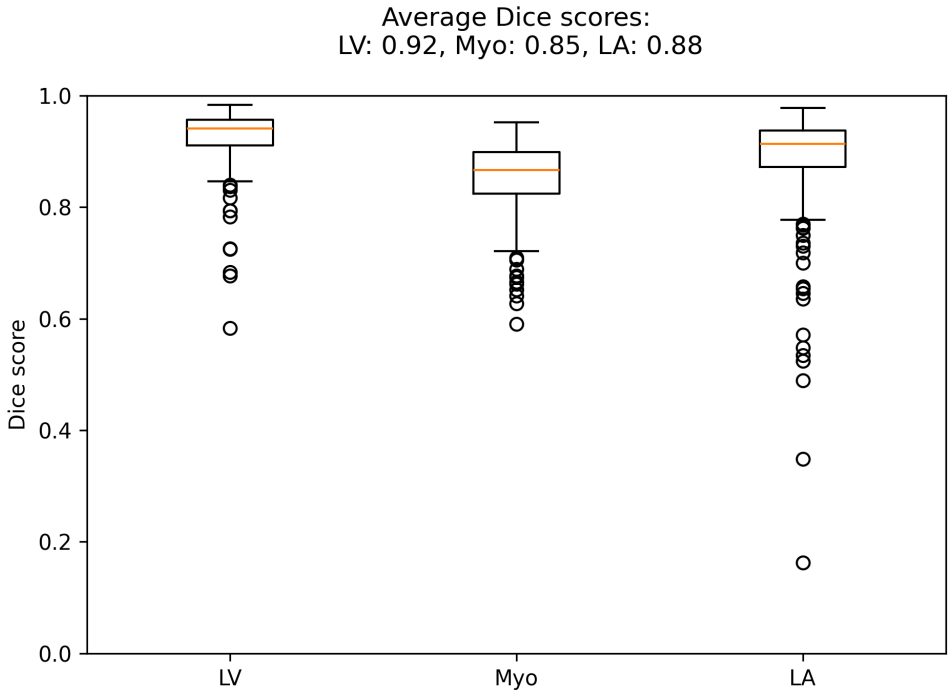


Figure 4.29: Boxplot of Dice score for network with residual connections.

4.10 Architectural changes

The results from training models where residual connections are added and transposed convolutions replace the nearest neighbour upsampling are summarized in Table 4.8. Figures 4.29 and 4.30 show the boxplots of the metrics for the network with residual connections. Compared to the standard U-Net 1, adding the residual connections does not improve performance.

Figures 4.31 and 4.32 show boxplots of the metrics for networks trained with transposed convolution for upscaling the feature maps. With $p > 0.05$, this model performs significantly worse for both metrics.

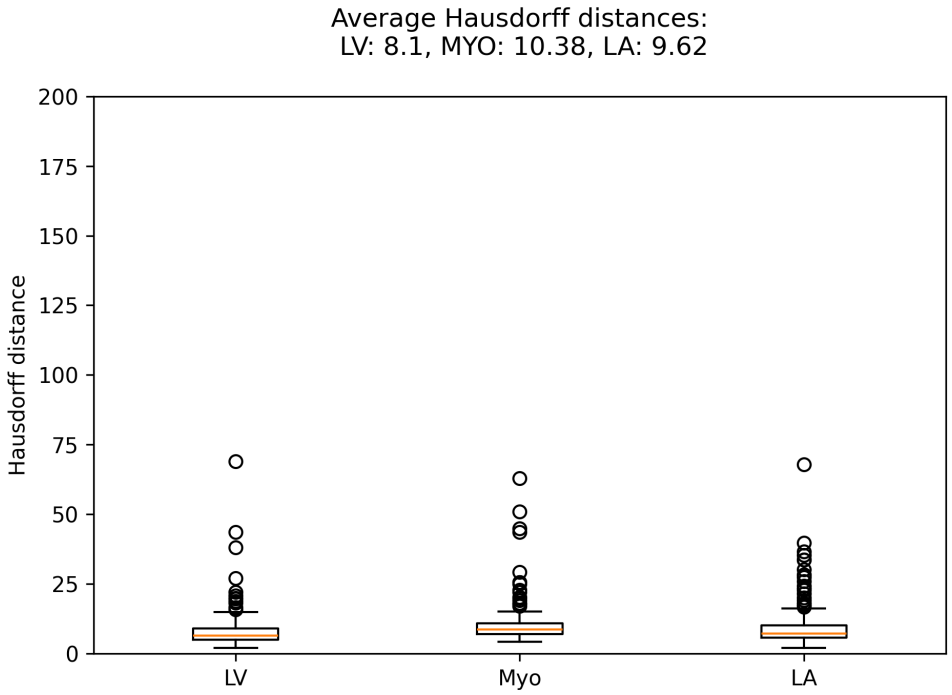


Figure 4.30: Boxplot of Hausdorff distance for network with residual connections.

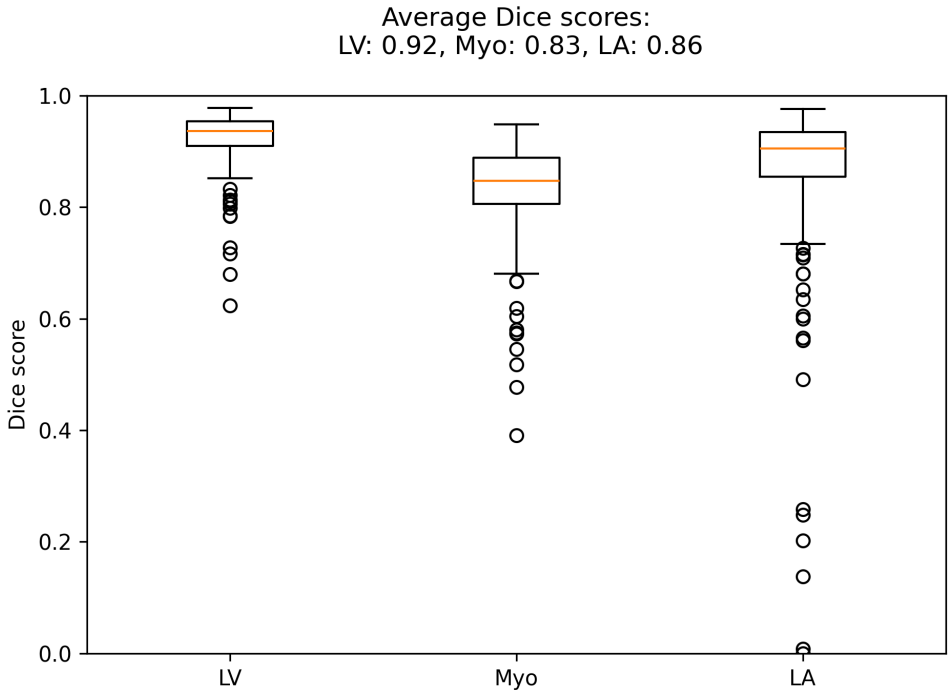


Figure 4.31: Boxplot of Dice score for network with transposed convolution as upsampling scheme.

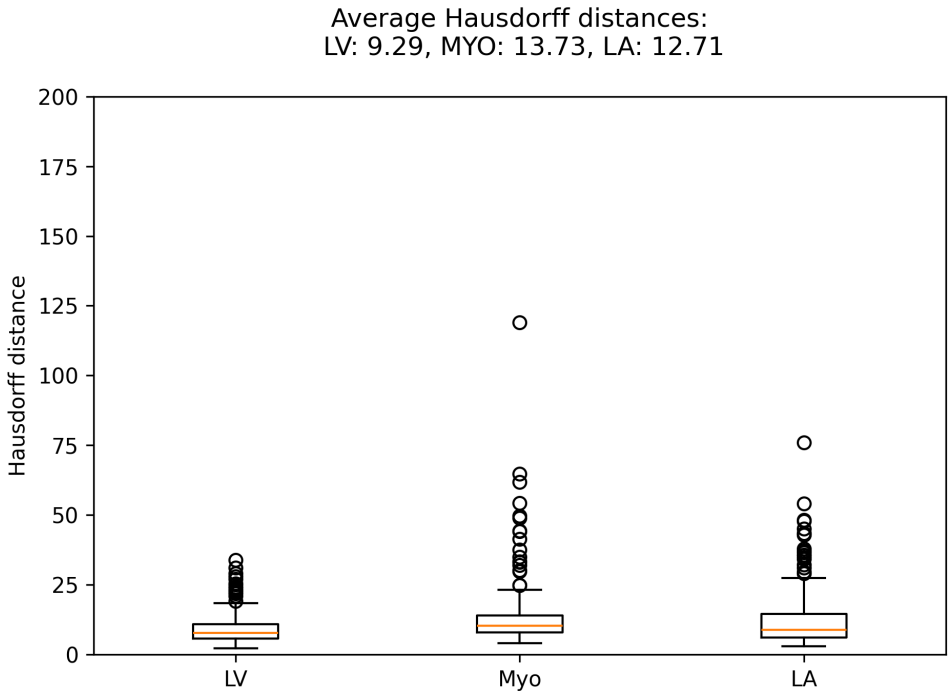


Figure 4.32: Boxplot Hausdorff distance for network with transposed convolution as upsampling scheme.

Architecture	**Dice LV**	**Dice MYO**	**Dice LA**	**Hausdorff LV**	**Hausdorff LV**	**Hausdorff MYO**
U-Net 1 w/ residual connections	0.92	0.85	0.88	8.10	10.38	9.62
U-Net 1 w/ transposed conv	0.92	0.83	0.86	9.29	13.73	12.71

Table 4.8: Model metrics for networks with residual connections and transposed convolutions.

Component	Description
Model	U-Net 1
Downsampling scheme	2×2 Maxpool
Upsampling scheme	Nearest neighbor interpolation
Normalization scheme	Batch normalization
Postprocessing	Yes, connected component analysis
Data augmentation	Shift(0.1), Scale(-0.2, 0.1), Rotate(10)
Loss function	Average of Dice + Cross Entropy
Activation function	Mish
Deep supervision	Yes, $\lambda = 0.3$ for all auxiliary losses

Table 4.9: Details on the final model.

4.11 The final model

The final U-Net model includes all components and techniques covered above that showed an increase in performance. The results are summarized in Table 4.9. The network itself uses the same number of channels, has the same lowest resolution, batch size, number of learnable parameters and up-and downsampling scheme as the U-Net 1 in Table 3.1. Along with adding the new components, the normalization scheme, activation function and loss function are replaced. The model achieves the Dice scores $0.93, 0.85, 0.89$ and the Hausdorff distances $7.49, 9.93, 9.22$ for the LV, MYO and LA respectively. Boxplots are shown in Figures 4.23 and 4.24. The difference in overall Dice score and Hausdorff distance between nnU-Net and the final model is not statistically different, with $p = 0.87$ and $p = 0.56$ respectively. This indicates that there is no difference in median performance between the two models.

4.12 Results on HUNT4

This section covers the results from testing both the baseline vanilla U-Net 1 model from Table 3.1 and the final best model on HUNT4. Two tests were run for each model: initially, both models were trained on the CAMUS dataset and subsequently tested on HUNT4. Following this, they were retrained on HUNT4 and then tested again on the same dataset. Additionally, the same models are trained on HUNT4 and tested on CAMUS. The results of the tests on HUNT4 are summarized in Table 4.10. The Dice scores for the models trained on CAMUS are shown in Figure 4.33, while the Dice scores for the models trained on HUNT4 are shown in Figure 4.35. Similarly, Hausdorff distances are shown in Figures 4.34 and 4.36 for the models trained on CAMUS and HUNT4 respectively. The metrics show that both models perform better when trained and tested on the same dataset. While the performance difference between the baseline and the best model in this setting is statistically significant, it is much smaller than when only using the CAMUS dataset.

Training the same models on HUNT4 and testing on CAMUS gives the Dice scores and Hausdorff distances summarized in Table 4.11. Figures 4.37 and 4.38 show the

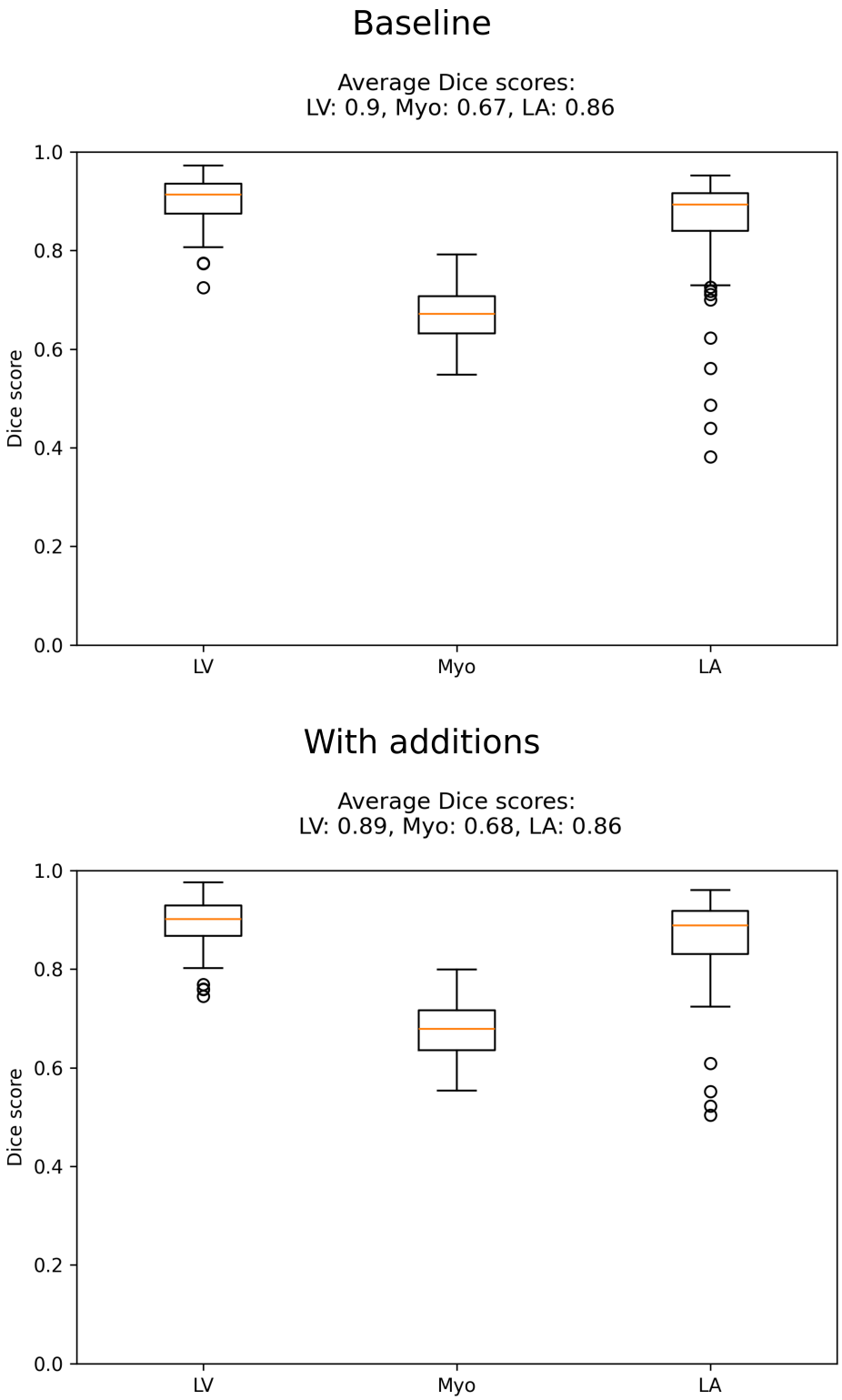


Figure 4.33: Boxplots of Dice scores for models trained on CAMUS and tested on HUNT4.

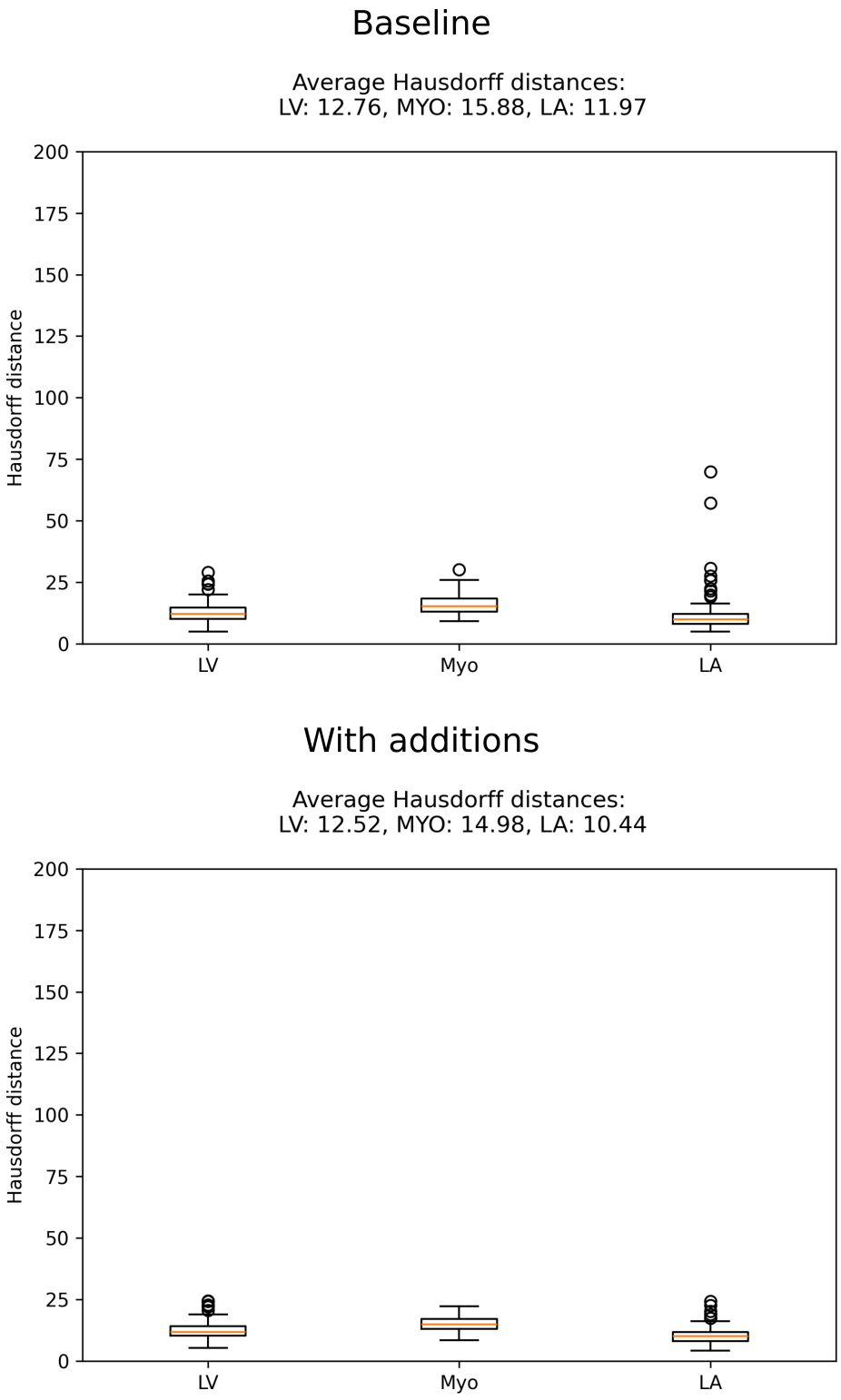


Figure 4.34: Boxplots of Hausdorff distances for models trained on CAMUS and tested on HUNT4.

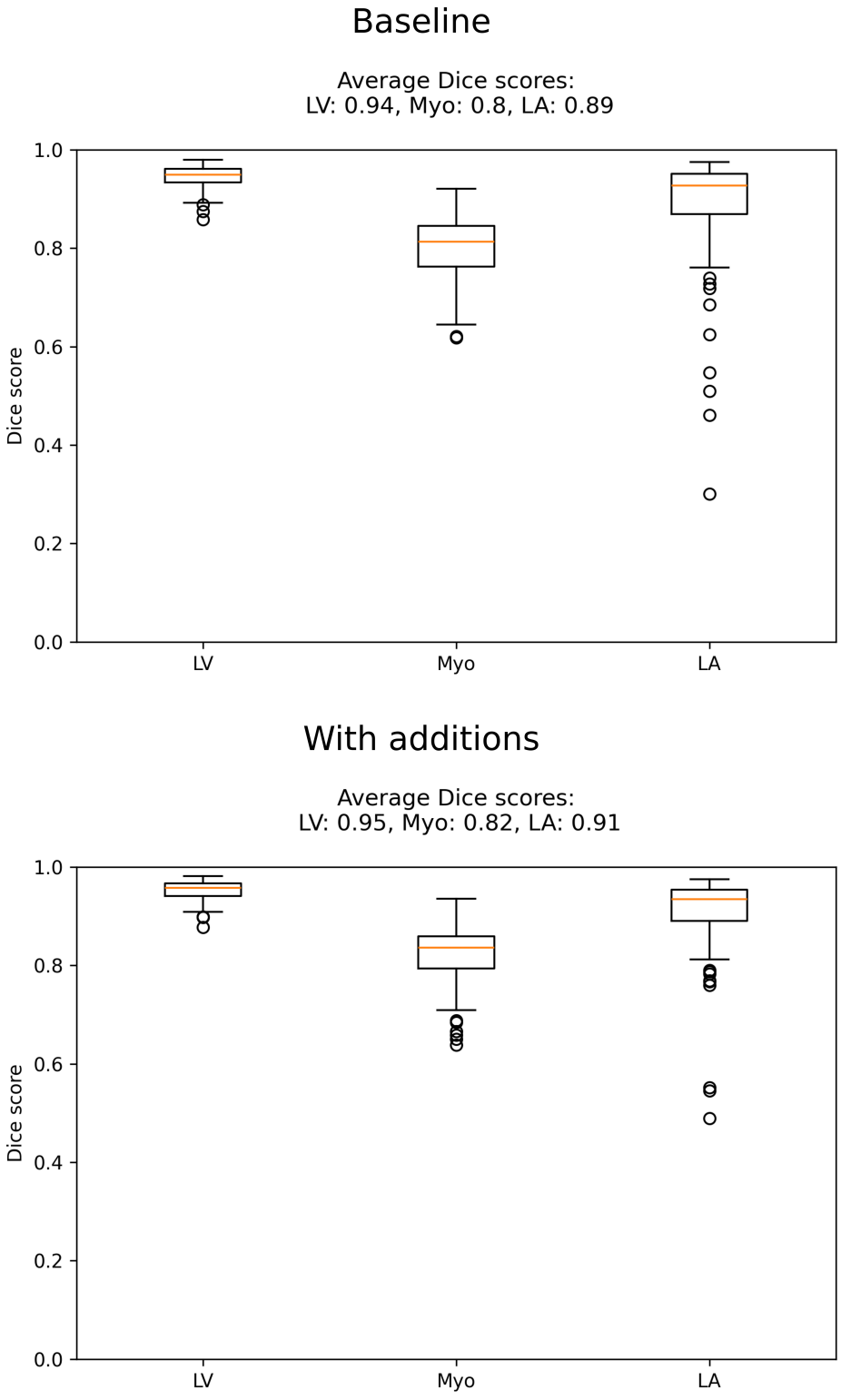


Figure 4.35: Boxplots of Dice scores for models trained and tested on HUNT4.

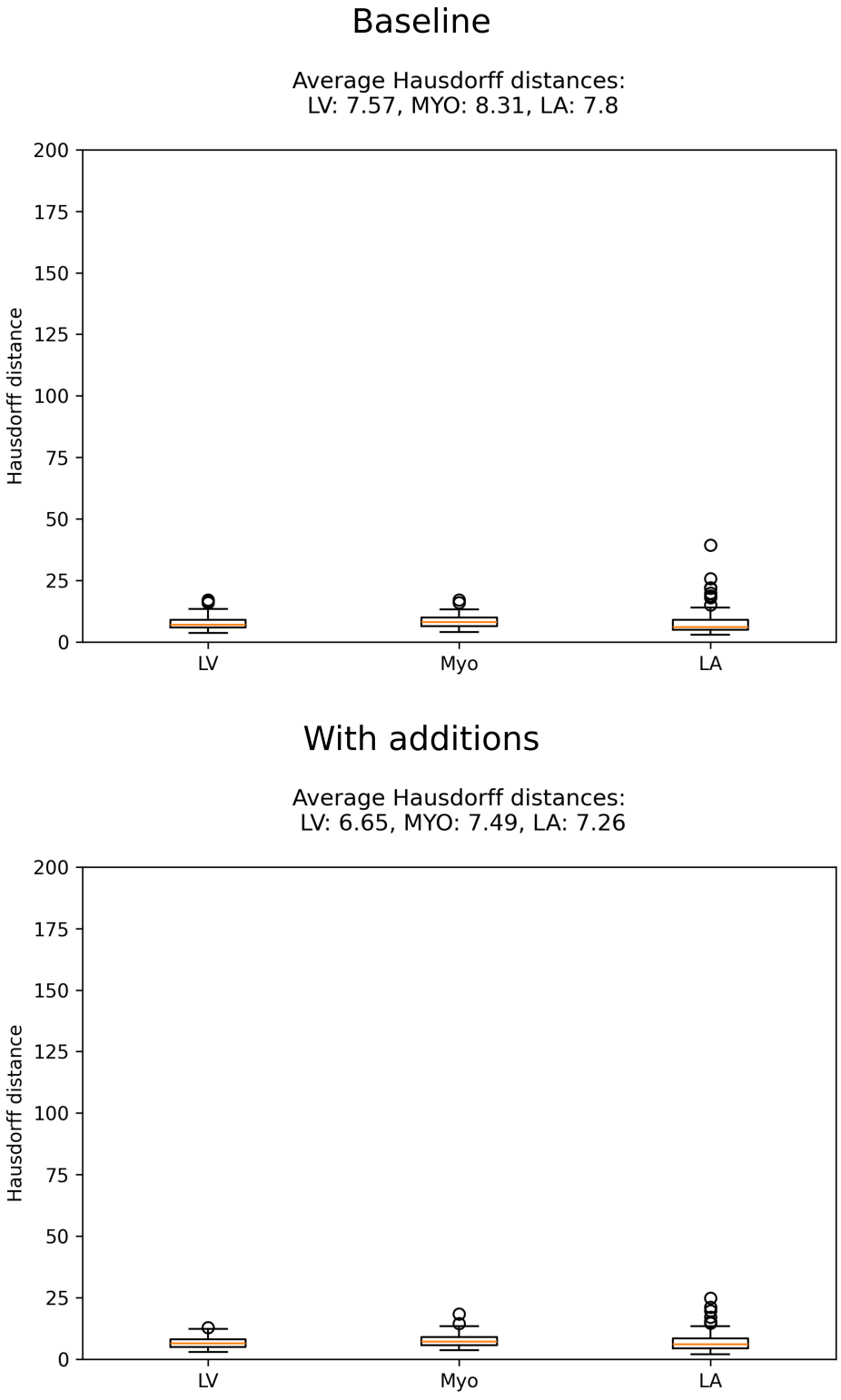


Figure 4.36: Boxplots of Hausdorff distances for models trained and tested on HUNT4.

Architecture	Trained on	Dice LV	Dice MYO	Dice LA	Hausdorff LV	Hausdorff LV	Hausdorff MYO	# Anatomical Outliers
U-Net 1 baseline	CAMUS	0.90	0.67	0.86	12.76	15.88	11.97	4/105
U-Net 1 baseline	HUNT4	0.94	0.80	0.89	7.57	8.31	7.80	2/105
U-Net 1 w/ additions	CAMUS	0.89	0.68	0.86	12.52	14.98	10.44	0/105
U-Net 1 w/ additions	HUNT4	0.95	0.82	0.91	6.65	7.49	7.26	0/105

Table 4.10: Metrics for the two models tested on HUNT4. Additions refer to the components in Table 4.9.

Architecture	Dice LV	Dice MYO	Dice LA	Hausdorff LV	Hausdorff LV	Hausdorff MYO	# Anatomical Outliers
U-Net 1 baseline	0.81	0.68	0.62	25.68	33.96	32.90	67/200
U-Net 1 w/ additions	0.86	0.66	0.75	17.86	21.17	22.50	36/200

Table 4.11: Metrics for the two models trained on HUNT4 and tested on CAMUS. Additions refer to the components in Table 4.9.

corresponding boxplots.

Because the myocardium is annotated thicker on CAMUS than HUNT4, the models trained on CAMUS consistently predict thicker myocardiums, even on HUNT4. Figure 4.39 shows two randomly selected predictions from the best model trained on CAMUS and tested on HUNT4.

4.13 Qualitative results

Figures 4.40 and 4.41 show the median and worst case performance of the final U-Net 1 model with all relevant techniques from nnU-Net when trained and tested on CAMUS. Figures 4.42 and 4.43 show the median and worst case performance of the same model, but trained on HUNT4 and tested on CAMUS.

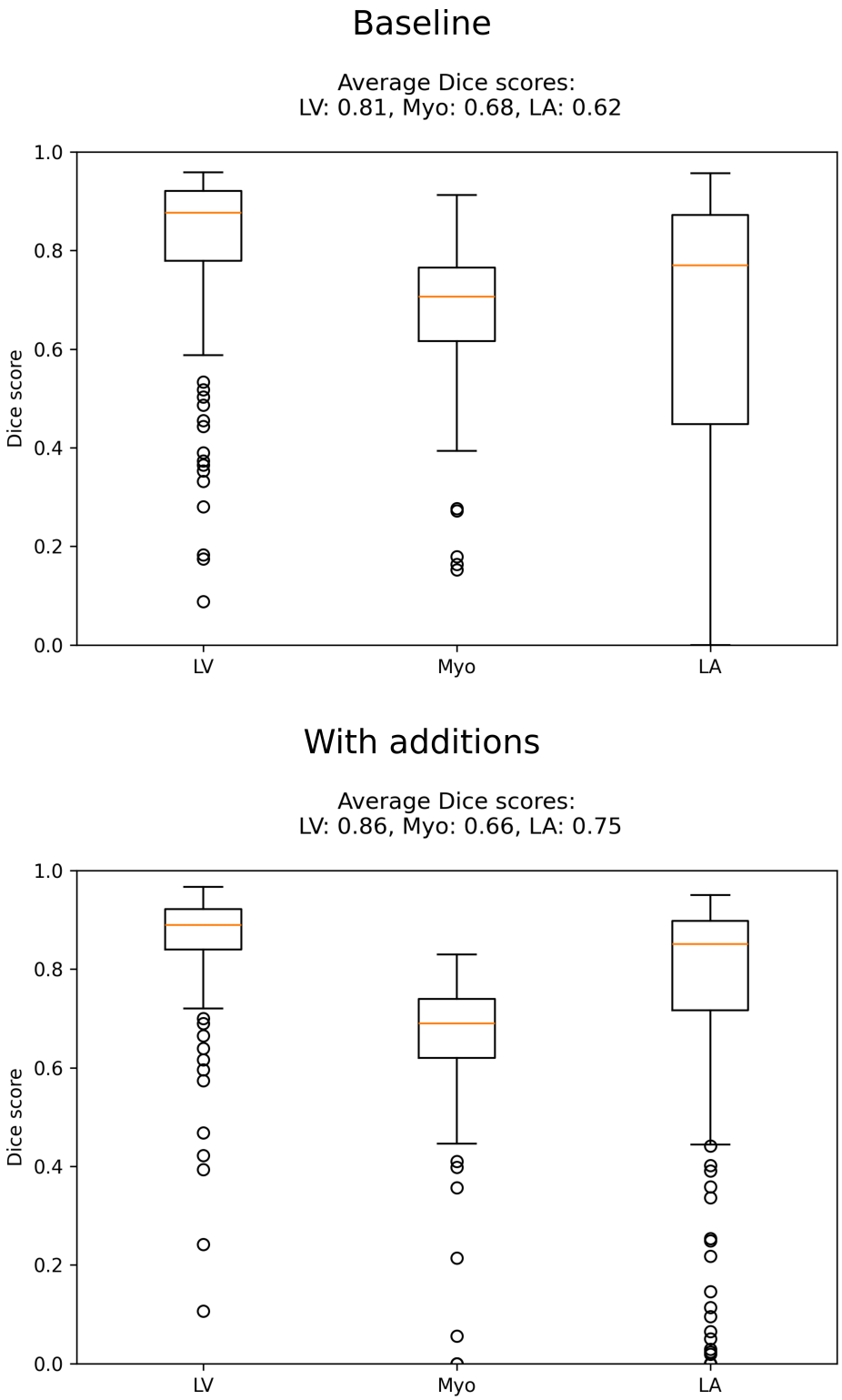


Figure 4.37: Boxplots of Dice scores for models trained on HUNT4 and tested on CAMUS.

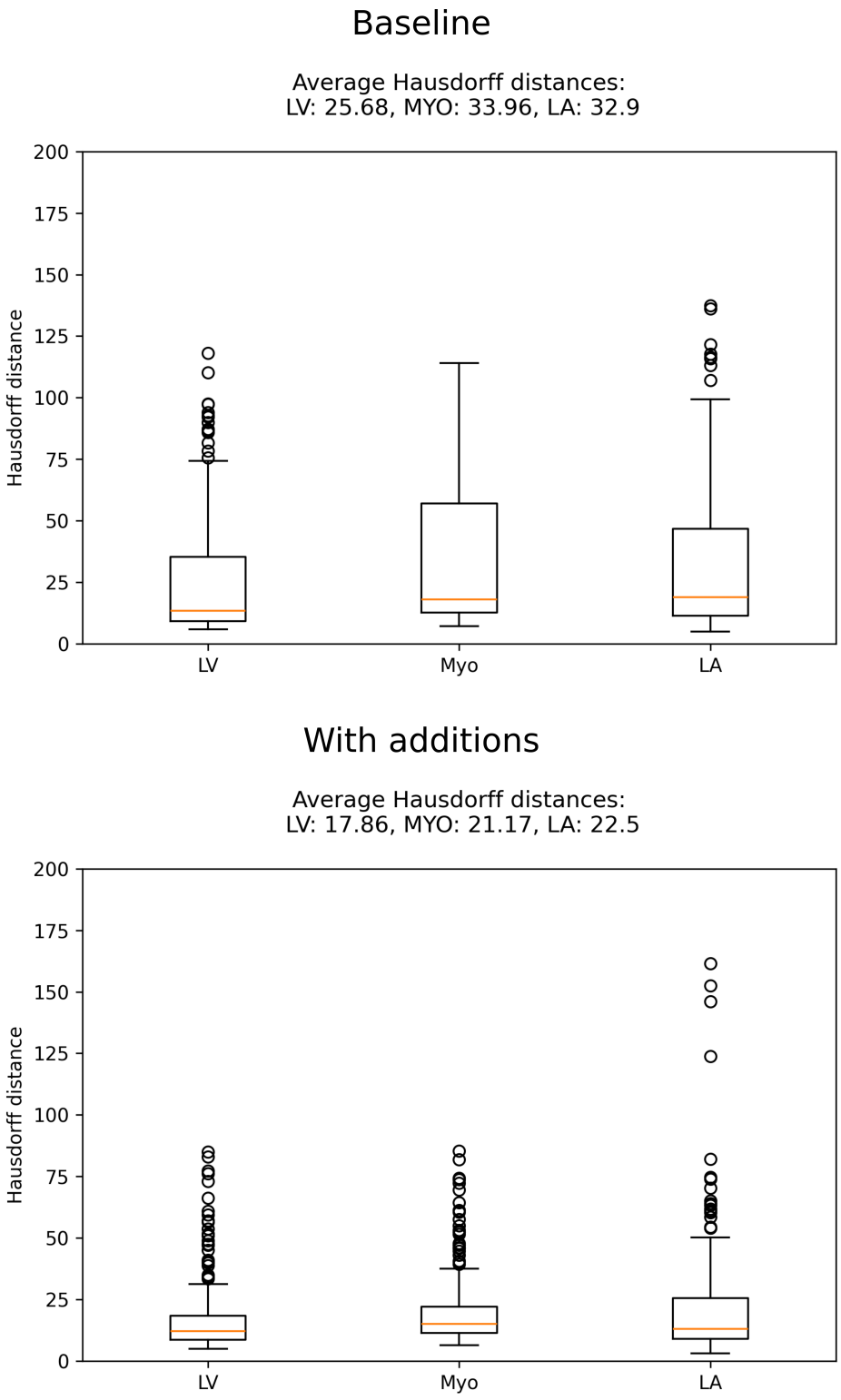


Figure 4.38: Boxplots of Hausdorff distances for models trained on HUNT4 and tested on CAMUS.

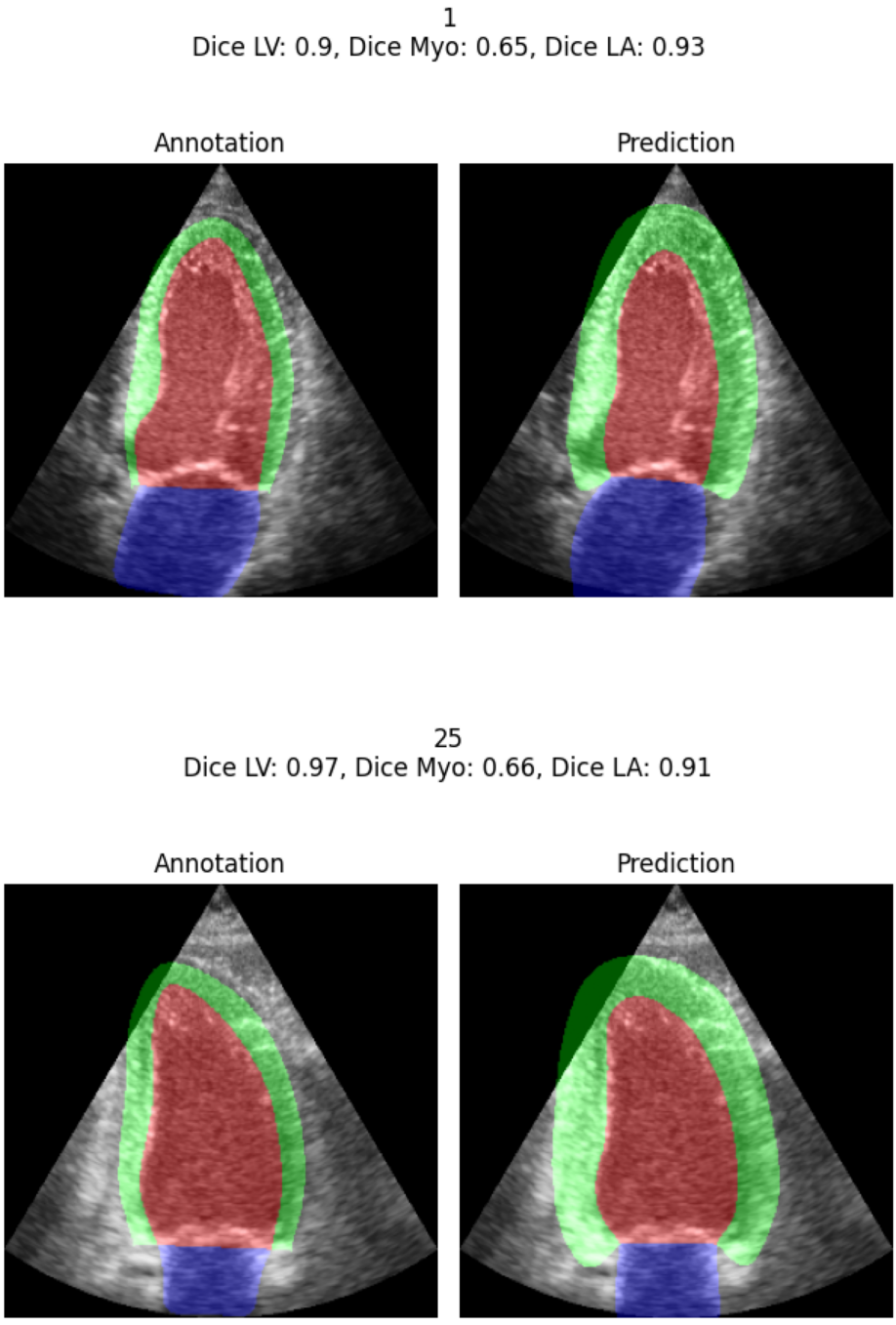


Figure 4.39: Predictions on HUNT4 by a model trained on CAMUS.

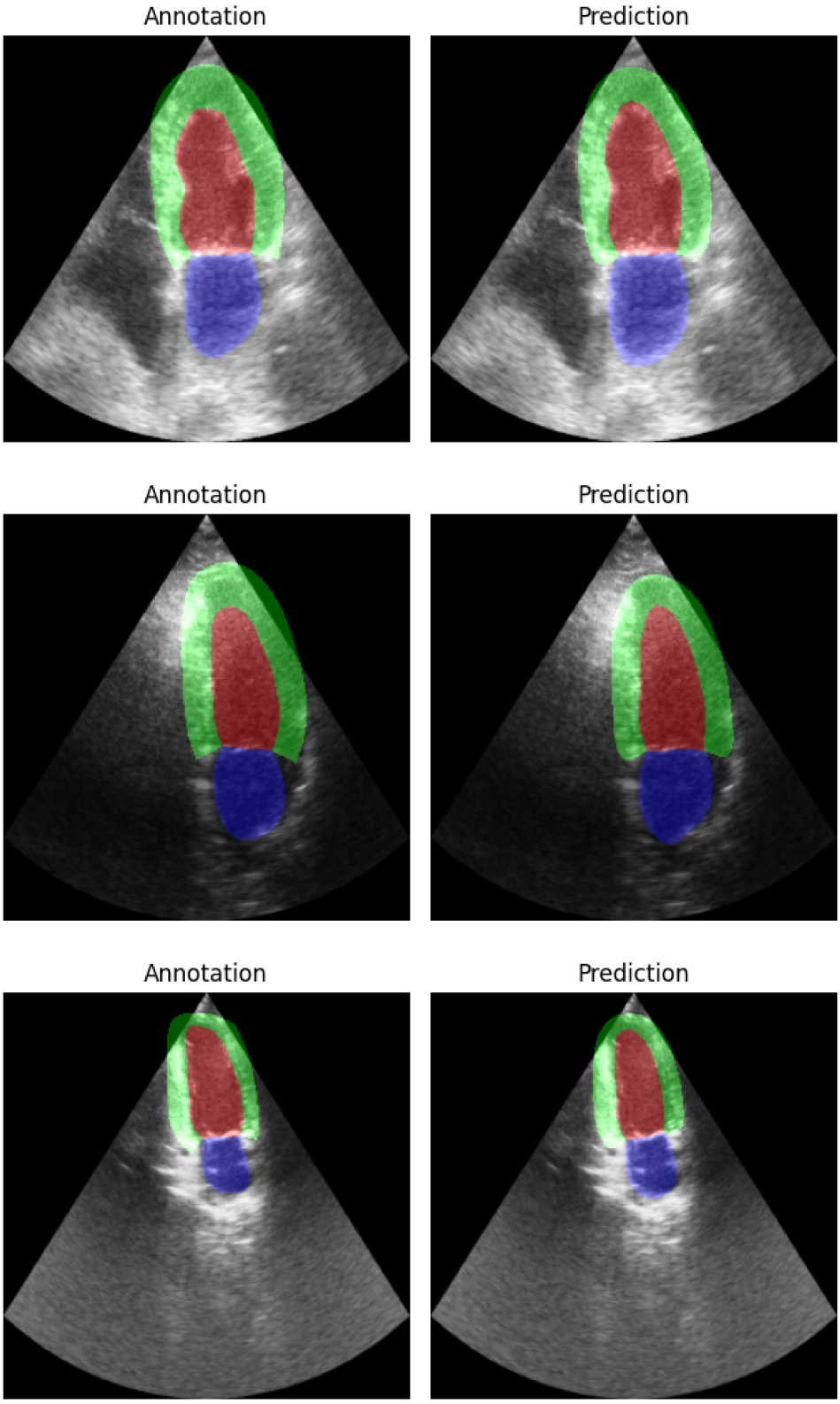


Figure 4.40: Median cases for the final model trained and tested on CAMUS. The cases are randomly selected segmentations with Dice scores close to the global average.

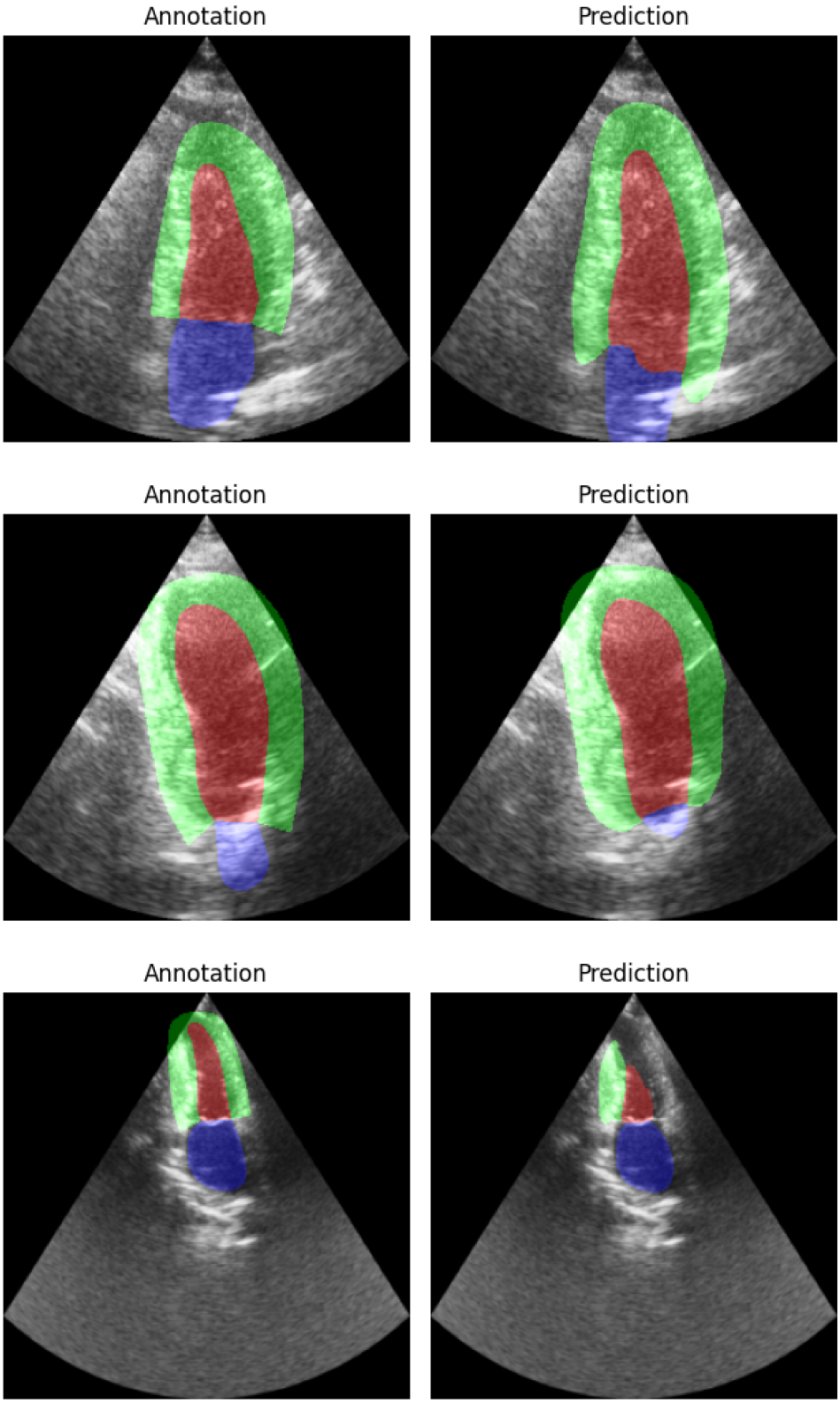


Figure 4.41: Worst cases for the final model trained and tested on CAMUS. The cases are selected based on the lowest average Dice scores.

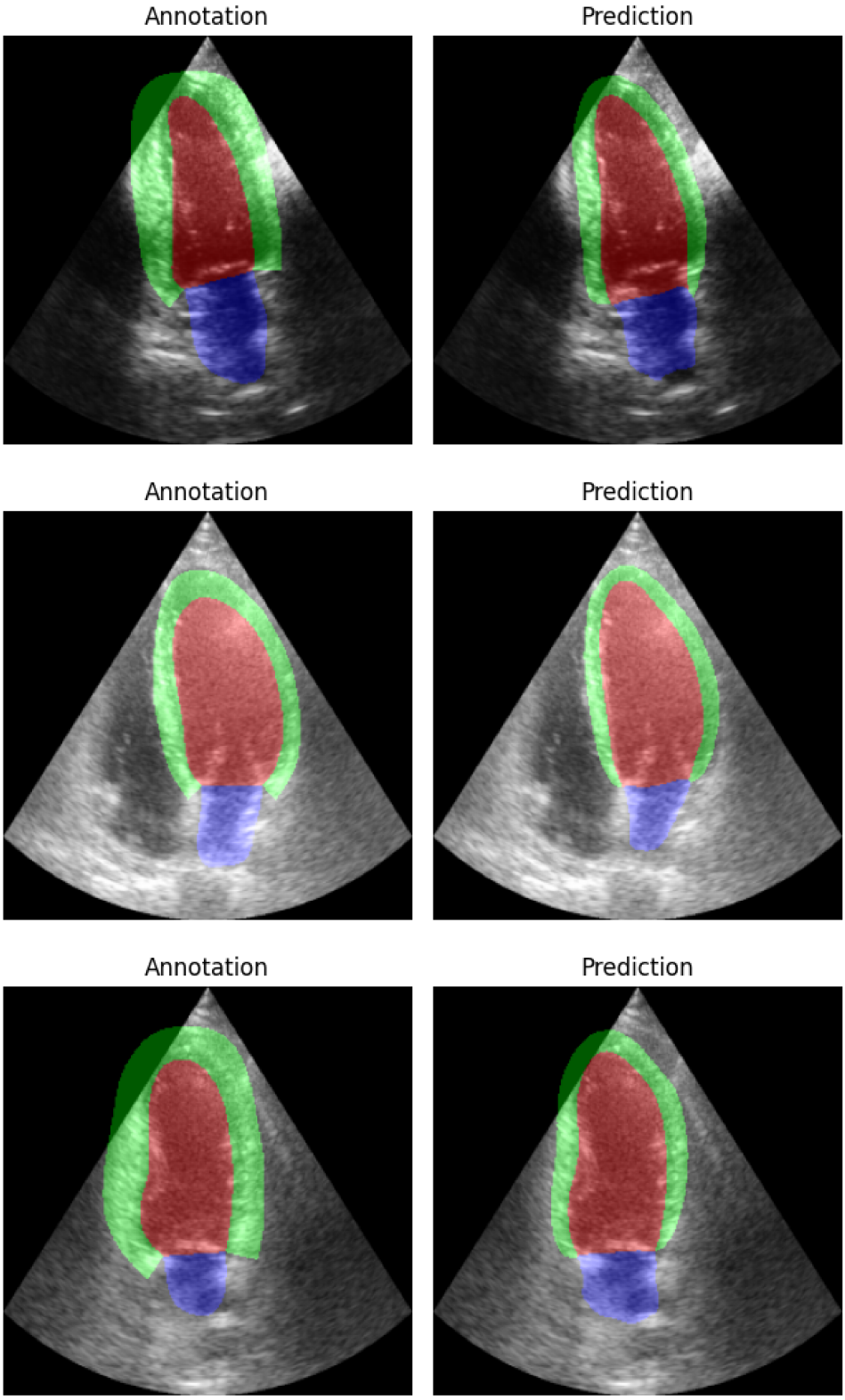


Figure 4.42: Median cases for the final model trained on HUNT4 and tested on CAMUS. The cases are randomly selected segmentations with Dice scores close to the global average.

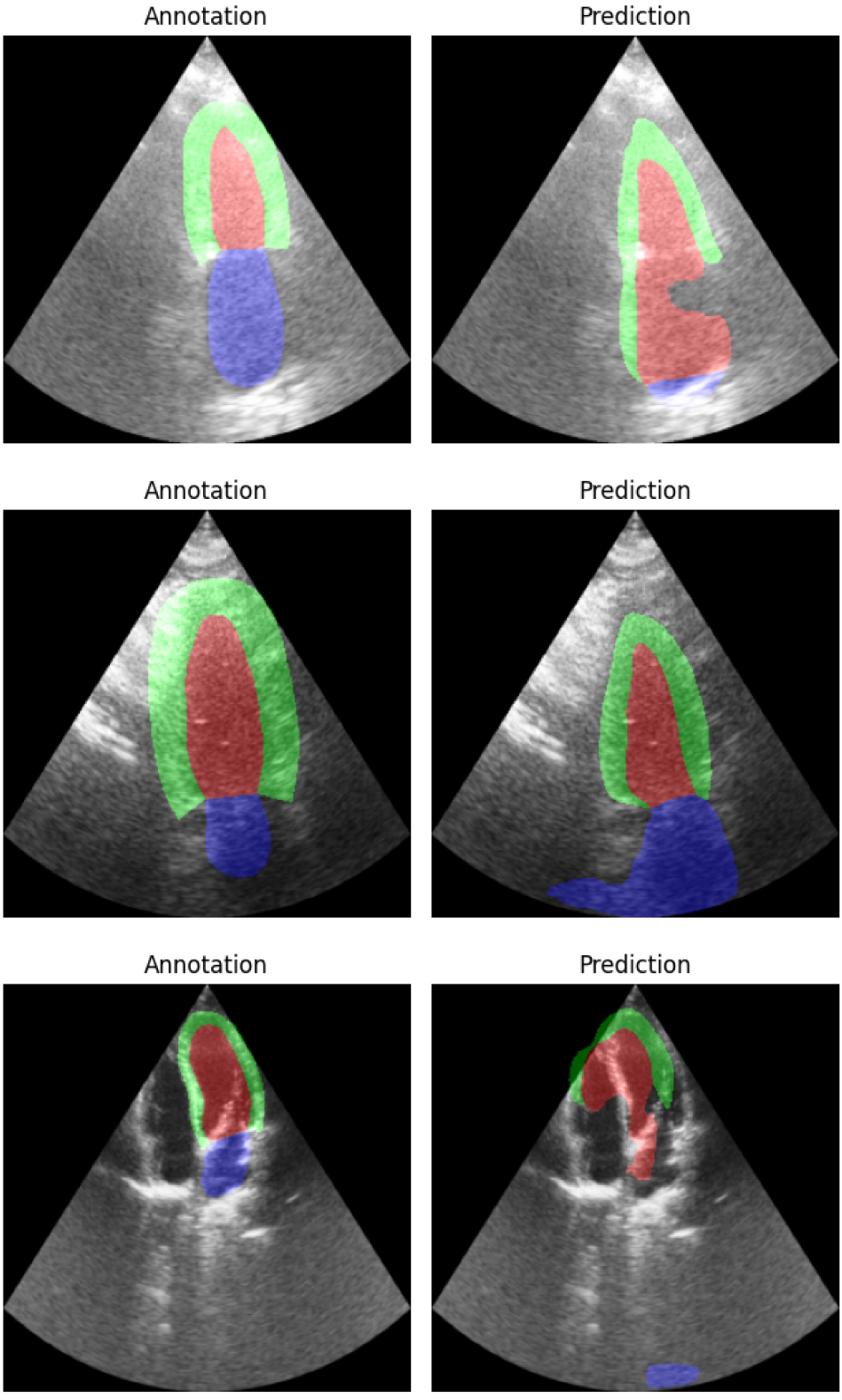


Figure 4.43: Worst cases for the final model trained on HUNT4 and tested on CAMUS. The cases are selected based on the lowest average Dice scores.

Chapter 5

Discussion

This chapter gives a discussion and reflection on the thesis. It discusses the results of the experiments run and puts them into context within the field of deep learning based segmentation in 2D echocardiography. The first section summarizes the results from the ablation study on nnU-Net, while the second discusses the applicability of nnU-Nets components to a smaller and faster U-Net.

5.1 nnU-Net

Regarding **RQ1** (*What are the most important components of nnU-Net for 2D cardiac ultrasound segmentation?*), the results, presented in Section 4.2, indicate that a robust data augmentation scheme in combination with deep supervision are the most important components for nnU-Nets success. The connected component analysis performed as a postprocessing step in the nnU-Net pipeline has a large impact on the Hausdorff distance. The experiments testing the effectiveness of postprocessing show that for a model trained without data augmentation and deep supervision, the Hausdorff distance is improved by $\mathbf{27-45}\%$, depending on the class. For the baseline model including both DA and DS, the Hausdorff distance is improved by $\mathbf{2-5}\%$. Since the Hausdorff distance essentially measures the worst-case, this shows that a) DA and DS are important components to both improve generalization and help reduce outlier predictions, and b) postprocessing can help minimize the distance between predicted and annotation boundaries by removing erroneous disconnected regions.

5.2 U-Net 1

With regards to **RQ2** (*How do the components from nnU-Net affect performance on a smaller U-Net?*), Sections 4.3 to 4.13 show that many of the key components from nnU-Net also improve performance for the smaller U-Net. The ones found to

be most important are discussed below. The baseline U-Net 1 fails to consistently predict accurate shapes. Although the Dice scores suggest adequate performance, the number of anatomical outliers presented in Table 4.1 reveals a different aspect of the model's capabilities. This limitation is further highlighted by the sample predictions in Figure 4.2, which demonstrate that the model has not learned the essential anatomical structures. This discrepancy between the Dice scores and the visual evidence from the predictions points to significant deficiencies in the model's understanding of the anatomical details.

Normalization scheme and postprocessing

From Table 4.3, it is evident that the model performs significantly better when trained on images and intermediate feature maps that have their intensities normalized. Normalizing the batches ensures that the model trains on inputs that are on the same scale, which [36] show is critical for model convergence. The postprocessing experiments show that while both opening and closing improve the Hausdorff distance, connected component analysis similar to nnU-Nets postprocessing engine is the best choice. For the baseline U-Net 1 with batch normalization, the best postprocessing scheme improves Hausdorff distance by $\mathbf{7-9}\%$.

Data augmentation and deep supervision

After normalization layers, data augmentation is shown to be the single most important component. Interestingly, despite its significant impact on the number of anatomical outliers, the augmented images are not entirely plausible in a clinical setting, see Figure 3.1. This indicates that the transformed images do not necessarily need to look realistic to the human eye. By extension this suggests that augmentation development should not be driven by visual aesthetics alone, but rather focus on the unbiased performance metrics.

The experiments show that although the nnU-Net augmentations work remarkably well for the networks configured by the pipeline, they might be excessive for smaller networks such as U-Net 1. Because the nnU-Net augmentation scheme is designed for networks an order of magnitude larger than the ones trained on in these experiments, applying them to smaller networks could lead to unnecessary noise in the input. An optimal augmentation scheme should therefore only use augmentations that improve performance. The affine augmentations are the most important, specifically shifting, scaling and rotating. They reduce the number of anatomical outliers from 26 to 12, while simultaneously giving statistically significantly higher Dice score and lower Hausdorff distances across all classes. The other augmentations, including Gamma adjustment, Gaussian noise and ColorJitter appear to add unnecessary noise to the data. Using them results in decreased Dice scores and increased Hausdorff distances, although they do reduce the number of anatomical outliers.

When combining the affine augmentations with deep supervision, the results are further improved. Combining these two methods both help the model learn the distribution better and increases convergence speed. Together with the fact that there is a negligible increase in model complexity, as detailed in Section 3.10, this shows that alongside data augmentation, deep supervision should be part of a successful U-Net architecture.

Looking at Figure 4.25 it is clear that deep supervision guides the intermediate layers of the decoder to produce anatomically valid shapes. By imposing losses on every block of the decoder, the model is effectively punished for not producing feature maps which are similar to the annotation, leading to reasonable segmentations already in the first block. When trained without DS, the feature maps learned by the first decoder block is not interpretable, as illustrated in Figure 4.26. For the later blocks they do however converge on shapes that resemble the annotations, but the labels do not match because of later layer operations. Although producing visually reasonable shapes in intermediate layers does not correlate directly to improved segmentation performance, it offers increased interpretability for the intermediate representations. Furthermore, the results in this work show that adding auxiliary losses at these layers do in fact offer increased performance. This is in line with Mishra et al. [70], who also show that using deep supervision for ultrasound image segmentation results in improved overall segmentation accuracy.

Model size and architectural changes

Section 4.9 compares the performance between U-Nets with varying number of parameters. The metrics do surprisingly not improve as model size increases. Instead, the medium-sized model U-Net 2 performs worse than both the standard U-Net 1 and the large U-Net 3. While the tiny U-Net 0 with 119K parameters achieves reasonable Dice scores, the Hausdorff distances are considerably worse. In fact, when looking at the predictions, it is clear that the model is not sufficiently complex. The predictions are fragmented and only a handful of the shapes are anatomically valid. This shows that even with the addition of the components from nnU-Net, there exists a lower bound for required model complexity in order to perform semantic segmentation with reasonable performance. The results also show that the 2M parameter U-Net 1 is sufficiently large and that the 33M network configured by nnU-Net might be excessive for the dataset used in this project. Although nnU-Net does not exhibit signs of overfitting, the increased complexity of the model will likely lead to worse generalization. It also demands more hardware resources and results in slower training and inference. Inference time is crucial because the goal is to use these models on portable ultrasound scanners in the clinic.

The results of the experiments with architectural modifications, such as learned upsampling using transposed convolutions in place of nearest neighbor interpolation and residual connections in the convolution blocks in Section 4.10 show that none of these

more sophisticated components increases performance. It is unexpected that newer and more complex deep learning techniques, which have enhanced performance in other models and tasks, do not yield improved results in this case. On the contrary, adding these components gave a slightly worse performance in this work. For transposed convolutions, when the filters are learned, it is natural to assume that the upsampling would be more tailored to the task and perform similar to traditional upsampling in the worst case, but the added complexity they bring is likely not needed on CAMUS. This evidence is supported by Leclerc et al. [3] who conclude more sophisticated architectures are not needed on CAMUS because the simple ellipse-like shapes encountered in 2D echocardiography are easily learned by simple networks.

Applying Vision Transformers (ViTs) [34] to the datasets would be interesting. Since their release, they are continuously used in state-of-the-art models in computer vision. This is however not possible in many MIC-applications because they require significantly more data than CNN-based models. Transformers do not possess the inductive biases inherent to CNNs, namely translation invariance and local connectivity, and thus they would likely be unable to generalize well on the datasets used in this project.

The final model and results on HUNT4

The final model comprises all components which are found to increase performance on all experiments done in this project. Despite it having 16 times less parameters than the U-Net configured by nnU-Net, there is no statistical difference in performance for any class or metric. This is remarkable, as it demonstrates that a compact network possesses sufficient complexity to accurately represent the distribution of the CAMUS dataset. Furthermore, it highlights that, given a sufficiently complex model for the data and task, the key determinant of performance lies in how the parameters are tuned, rather than their quantity. When it comes to segmenting out-of-distribution cases, the final model trained and tested on CAMUS gives varied results, as shown in Figures 4.40 and 4.41. The changes that improve performance on CAMUS also improve performance on HUNT4, which shows that the performance gains are not dataset specific.

Results from another experiment testing performance on out-of-distribution cases are shown in Figure 4.43, where the final model is trained on HUNT4 and tested on CAMUS. Here, the model struggles to give accurate segmentations when the input is different from the images it has seen during training, as can be seen by the large number of anatomical outliers. This is expected because HUNT4 has less variance and no extreme outliers in the training set. Thus, a more aggressive augmentation scheme could potentially have made the model more robust to extreme out-of-distribution cases, like the ones encountered in CAMUS. When training on CAMUS and testing on HUNT4, the model performs well, but consistently predicts a thicker myocardium because of the difference between the two datasets. This is also reflected in the Dice score for the MYO class in this experiment. Overall, the results in Section 4.12

indicate that, compared to CAMUS, HUNT4 is easier to learn, thus requiring fewer enhancements such as DA and DS.

A limitation of the model developed in this thesis is that, while nnU-Net provides state-of-the-art performance on a multitude of datasets and modalities out-of-the-box, the model in this work is only tuned on one dataset and tested on two datasets. It is also only trained one one modality. This implies that even if it achieves similar performance on CAMUS, there is no guarantee that the design choices made are universally applicable. Another difference is that the model size does not automatically scale like nnU-Net does, such that if a bigger, more complex dataset is encountered, there is no guarantee that U-Net 1 will have the required complexity.

Chapter 6

Conclusion

6.1 Conclusion

This work conducts an in-depth analysis of nnU-Net, identifying its critical components and undertaking a comprehensive ablation study to assess their impact and adaptability to a model approximately 16 times smaller. The resulting model, a custom U-Net, enhances the baseline U-Net 1 by integrating the most effective features of nnU-Net. The final model is effectively a significantly scaled-down version of nnU-Net where the key elements that contribute to its success are preserved. The custom model performs similarly to nnU-Net in terms of the Dice score and Hausdorff distance on the CAMUS dataset. With $p < 0.05$ using the Wilcoxon signed-rank test, there is no statistically significant difference between the models in terms of the two metrics. The baseline U-Net 1 achieves a Dice score of $0.9, 0.82, 0.86$ and a Hausdorff distance of $10.13, 12.87, 12.04$ for the LV, MYO and LA respectively. The final model improves the Dice scores to $0.93, 0.85, 0.89$ and the Hausdorff distances to $7.49, 9.93, 9.22$. This research demonstrates that the popularity of nnU-Net is not merely a function of the model's size, but rather its architectural features and component efficacy. The findings reveal that the considerably smaller U-Net, when optimized with nnU-Net's most critical components, is fully capable of segmenting 2D echocardiographic images with a precision which is on par with nnU-Net. The most important components are discovered to be the data augmentation scheme, deep supervision, loss function, normalization scheme and postprocessing. Other components such as residual connections, transposed convolutions used to upscale feature maps and choice activation function are deemed to be less important in this context.

6.2 Future work

The new nnU-Net paper shows promising results with residual connections and elements from ConvNeXt [35], which is worth investigating further. Even though the results in this work showed no increase in performance when using residual connections, this could be due to the significantly smaller network.

Given the impact of data augmentation, a larger dataset with consistent labelling could lead to better generalization. Since the two datasets used in this work have significant differences, both in terms of labeling and variance, training on a combination of the two was not possible. Potential future work could be to establish guidelines for how new datasets in cardiac ultrasound are labelled to ensure they are compatible with one another. nnU-Net should also be trained and tested on HUNT4 and compared to U-Net 1.

Finally, further investigations into inference speed should be done. This includes both inference speed on a GPU and for real-time segmentation on ultrasound scanners in the clinic in order to investigate the differences from nnU-Net. The models should also be included in automatic calculation of clinical measures such as left ventricular volume and ejection fraction. The accuracy of these measurements can then be used as another way of evaluating the performance of the final U-Net 1.

Bibliography

[1] Megan Lindstrom et al. 'Global burden of cardiovascular diseases and risks collaboration, 1990-2021'. In: *Journal of the American College of Cardiology* 80.25 (2022), pp. 2372–2425.

[2] Mariachiara Di Cesare et al. 'The Heart of the World'. en. In: *Glob Heart* (Jan. 2024), p. 11.

[3] Sarah Leclerc et al. 'Deep Learning for Segmentation Using an Open Large-Scale Dataset in 2D Echocardiography'. In: *IEEE Transactions on Medical Imaging* 38.9 (2019), pp. 2198–2210. DOI: 10.1109/TMI.2019.2900516.

[4] Iqbal H. Sarker. 'Deep Learning: A Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions'. In: *SN Computer Science* 2.6 (Aug. 2021). ISSN: 2661-8907. DOI: 10.1007/s42979-021-00815-1.

[5] Olaf Ronneberger, Philipp Fischer and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. arXiv: 1505.04597 [cs.CV].

[6] Geert Litjens et al. 'A survey on deep learning in medical image analysis'. In: *Medical Image Analysis* 42 (Dec. 2017), pp. 60–88. ISSN: 1361-8415. DOI: 10.1016/j.media.2017.07.005. URL: http://dx.doi.org/10.1016/j.media.2017.07.005.

[7] Fabian Isensee et al. *nnU-Net: Self-adapting Framework for U-Net-Based Medical Image Segmentation*. 2018. arXiv: 1809.10486 [cs.CV].

[8] Fabian Isensee et al. 'nnU-Net for Brain Tumor Segmentation'. In: *Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries*. Ed. by Alessandro Crimi and Spyridon Bakas. Cham: Springer International Publishing, 2021, pp. 118–132. ISBN: 978-3-030-72087-2.

[9] Guobin Zhang et al. 'Multiorgan segmentation from partially labeled datasets with conditional nnU-Net'. In: *Computers in Biology and Medicine* 136 (2021), p. 104658. ISSN: 0010-4825. DOI: https://doi.org/10.1016/j.compbiomed.2021.104658. URL: https://www.sciencedirect.com/science/article/pii/S0010482521004522.

[10] Rowland W. Pettit et al. 'nnU-Net Deep Learning Method for Segmenting Parenchyma and Determining Liver Volume From Computed Tomography Images'. In: *Annals of Surgery Open* 3.2 (2022). ISSN: 2691-3593. URL: https://journals.lww.com/aosopen/fulltext/2022/06000/nnu_net_deep_learning_method_for_segmenting.2.aspx.

[11] Lu Huo et al. 'Segmentation of whole breast and fibroglandular tissue using nnU-Net in dynamic contrast enhanced MR images'. In: *Magnetic Resonance Imaging* 82 (2021), pp. 31–41. ISSN: 0730-725X. DOI: https://doi.org/10.1016/j.mri.2021.06.017. URL: https://www.sciencedirect.com/science/article/pii/S0730725X21001053.

[12] Ashish Vaswani et al. 'Attention Is All You Need'. In: *arXiv preprint arXiv:1706.03762v7* (2017).

[13] Albert Gu and Tri Dao. *Mamba: Linear-Time Sequence Modeling with Selective State Spaces*. 2023. arXiv: 2312.00752 [cs.LG].

[14] Jieneng Chen et al. *TransUNet: Transformers Make Strong Encoders for Medical Image Segmentation*. 2021. arXiv: 2102.04306 [cs.CV].

[15] Ali Hatamizadeh et al. *Swin UNETR: Swin Transformers for Semantic Segmentation of Brain Tumors in MRI Images*. 2022. arXiv: 2201.01266 [eess.IV].

[16] Jun Ma, Feifei Li and Bo Wang. *U-Mamba: Enhancing Long-range Dependency for Biomedical Image Segmentation*. 2024. arXiv: 2401.04722 [eess.IV].

[17] Fabian Isensee et al. *nnU-Net Revisited: A Call for Rigorous Validation in 3D Medical Image Segmentation*. 2024. arXiv: 2404.09556 [cs.CV].

[18] Tom M Mitchell. *Machine learning*. Vol. 1. 9. McGraw-hill New York, 1997.

[19] G. Cybenko. 'Approximation by superpositions of a sigmoidal function'. In: *Mathematics of Control, Signals and Systems* (Dec. 1989), pp. 303–314. DOI: 10.1007/BF02551274.

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

[21] Ronald DeVore, Boris Hanin and Guergana Petrova. *Neural Network Approximation*. 2020. arXiv: 2012.14501 [math.NA].

[22] Michael A. Nielsen. *Neural Networks and Deep Learning*. 2018. URL: http://neuralnetworksanddeeplearning.com/.

[23] David E. Rumelhart, Geoffrey E. Hinton and Ronald J. Williams. 'Learning representations by back-propagating errors'. In: *Nature* 323.6088 (Oct. 1986), pp. 533–536. ISSN: 1476-4687. DOI: 10.1038/323533a0. URL: https://doi.org/10.1038/323533a0.

[24] Razvan Pascanu, Tomas Mikolov and Yoshua Bengio. *On the difficulty of training Recurrent Neural Networks*. 2013. arXiv: 1211.5063 [cs.LG].

[25] Lu Lu. 'Dying ReLU and Initialization: Theory and Numerical Examples'. In: *Communications in Computational Physics* 28.5 (June 2020), pp. 1671–1706. ISSN: 1991-7120. DOI: 10.4208/cicp.oa-2020-0165. URL: http://dx.doi.org/10.4208/cicp.OA-2020-0165.

[26] Dan Hendrycks and Kevin Gimpel. 'Gaussian Error Linear Units (GELUs)'. In: *arXiv preprint arXiv:1606.08415* (2016).

[27] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL].

[28] Tom B. Brown et al. *Language Models are Few-Shot Learners*. 2020. arXiv: 2005.14165 [cs.CL].

[29] Diganta Misra. *Mish: A Self Regularized Non-Monotonic Activation Function*. 2020. arXiv: 1908.08681 [cs.LG].

[30] Alexey Bochkovskiy, Chien-Yao Wang and Hong-Yuan Mark Liao. *YOLOv4: Optimal Speed and Accuracy of Object Detection*. 2020. arXiv: 2004.10934 [cs.CV].

[31] Juan Terven et al. *Loss Functions and Metrics in Deep Learning*. 2023. arXiv: 2307.02694 [cs.LG].

[32] Carole H. Sudre et al. 'Generalised Dice Overlap as a Deep Learning Loss Function for Highly Unbalanced Segmentations'. In: *Lecture Notes in Computer Science*. Springer International Publishing, 2017, pp. 240–248. ISBN: 9783319675589. DOI: 10.1007/978-3-319-67558-9_28. URL: http://dx.doi.org/10.1007/978-3-319-67558-9_28.

[33] Thomas N. Kipf and Max Welling. *Semi-Supervised Classification with Graph Convolutional Networks*. 2017. arXiv: 1609.02907 [cs.LG].

[34] Alexey Dosovitskiy et al. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. 2021. arXiv: 2010.11929 [cs.CV].

[35] Zhuang Liu et al. *A ConvNet for the 2020s*. 2022. arXiv: 2201.03545 [cs.CV].

[36] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. arXiv: 1502.03167 [cs.LG].

[37] Dmitry Ulyanov, Andrea Vedaldi and Victor Lempitsky. *Instance Normalization: The Missing Ingredient for Fast Stylization*. 2017. arXiv: 1607.08022 [cs.CV].

[38] Nanyi Fei et al. 'Z-Score Normalization, Hubness, and Few-Shot Learning'. In: *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2021, pp. 142–151. DOI: 10.1109/ICCV48922.2021.00021.

[39] Chen-Yu Lee et al. *Deeply-Supervised Nets*. 2014. arXiv: 1409.5185 [stat.ML].

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

[41] Yingbin Bai et al. *Understanding and Improving Early Stopping for Learning with Noisy Labels*. 2021. arXiv: 2106.15853 [cs.LG].

[42] Mikhail Belkin et al. 'Reconciling modern machine-learning practice and the classical bias–variance trade-off'. In: *Proceedings of the National Academy of Sciences* 116.32 (July 2019), pp. 15849–15854. ISSN: 1091-6490. DOI: 10.1073/pnas.1903070116. URL: http://dx.doi.org/10.1073/pnas.1903070116.

[43] Gabriela Csurka, Riccardo Volpi and Boris Chidlovskii. *Semantic Image Segmentation: Two Decades of Research*. 2023. arXiv: 2302.06378 [cs.CV].

[44] Jonathan Long, Evan Shelhamer and Trevor Darrell. *Fully Convolutional Networks for Semantic Segmentation*. 2015. arXiv: 1411.4038 [cs.CV].

[45] Enze Xie et al. *SegFormer: Simple and Efficient Design for Semantic Segmentation with Transformers*. 2021. arXiv: 2105.15203 [cs.CV].

[46] Kaiming He et al. *Mask R-CNN*. 2018. arXiv: 1703.06870 [cs.CV].

[47] Daniel Bolya et al. *YOLACT: Real-time Instance Segmentation*. 2019. arXiv: 1904.02689 [cs.CV].

[48] Ze Liu et al. *Swin Transformer: Hierarchical Vision Transformer using Shifted Windows*. 2021. arXiv: 2103.14030 [cs.CV].

[49] Alexander Kirillov et al. *Panoptic Segmentation*. 2019. arXiv: 1801.00868 [cs.CV].

[50] Feng Li et al. *Mask DINO: Towards A Unified Transformer-based Framework for Object Detection and Segmentation*. 2022. arXiv: 2206.02777 [cs.CV].

[51] Abdel Aziz Taha and Allan Hanbury. 'Metrics for evaluating 3D medical image segmentation: analysis, selection, and tool'. In: *BMC Medical Imaging* 15 (12th Aug. 2015), p. 29. DOI: 10.1186/s12880-015-0068-x.

[52] Paul Jaccard. 'Distribution de la Flore Alpine dans le Bassin des Dranses et dans quelques régions voisines.' In: *Bulletin de la Societe Vaudoise des Sciences Naturelles* 37 (Jan. 1901), pp. 241–72. DOI: 10.5169/seals-266440.

[53] D.P. Huttenlocher, G.A. Klanderman and W.J. Rucklidge. 'Comparing images using the Hausdorff distance'. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 15.9 (1993), pp. 850–863. DOI: 10.1109/34.232073.

[54] R. F. Woolson. 'Wilcoxon Signed-Rank Test'. In: *Wiley Encyclopedia of Clinical Trials*. John Wiley & Sons, Ltd, 2008, pp. 1–3.

[55] Xiangbin Liu et al. 'A Review of Deep-Learning-Based Medical Image Segmentation Methods'. In: *Sustainability* 13.3 (2021). DOI: 10.3390/su13031224.

[56] Kaiming He et al. *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*. 2015. arXiv: 1502.01852 [cs.CV].

[57] Özgün Çiçek et al. *3D U-Net: Learning Dense Volumetric Segmentation from Sparse Annotation*. 2016. arXiv: 1606.06650 [cs.CV].

[58] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].

[59] Ozan Oktay et al. *Attention U-Net: Learning Where to Look for the Pancreas*. 2018. arXiv: 1804.03999 [cs.CV].

[60] Simon Jégou et al. *The One Hundred Layers Tiramisu: Fully Convolutional DenseNets for Semantic Segmentation*. 2017. arXiv: 1611.09326 [cs.CV].

[61] Sindre Olaisen et al. 'Automatic measurements of left ventricular volumes and ejection fraction by artificial intelligence: clinical validation in real time and large databases'. In: *European Heart Journal - Cardiovascular Imaging* (Oct. 2023). DOI: 10.1093/ehjci/jead280.

[62] Gilles Van De Vyver et al. *Towards Robust Cardiac Segmentation using Graph Convolutional Networks*. 2023. arXiv: 2310.01210 [eess.IV].

[63] Adam Paszke et al. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. 2019. arXiv: 1912.01703 [cs.LG].

[64] Saurabh Singh and Shankar Krishnan. *Filter Response Normalization Layer: Eliminating Batch Dependence in the Training of Deep Neural Networks*. 2020. arXiv: 1911.09737 [cs.LG].

[65] Rafael C. Gonzalez and Richard E. Woods. *Digital image processing*. Prentice Hall, 2008. ISBN: 9780131687288.

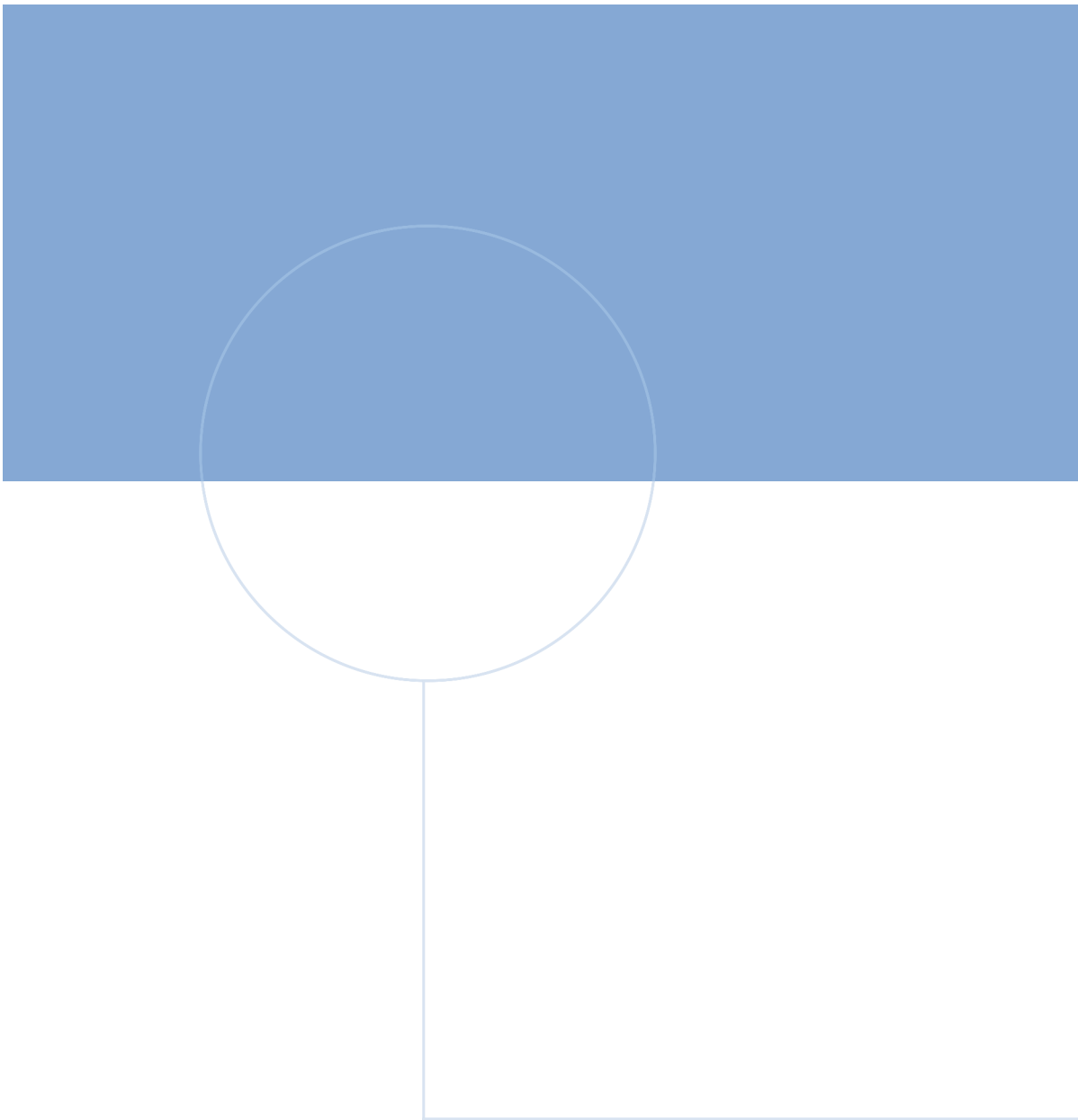
[66] Mina Ghaffari, Arcot Sowmya and Ruth Oliver. *Brain tumour segmentation using cascaded 3D densely-connected U-net*. 2020. arXiv: 2009.07563 [eess.IV].

[67] Connor Shorten and Taghi M. Khoshgoftaar. 'A survey on Image Data Augmentation for Deep Learning'. In: *Journal of Big Data* (July 2019). ISSN: 2196-1115. DOI: 10.1186/s40537-019-0197-0. URL: https://doi.org/10.1186/s40537-019-0197-0.

[68] Md Zahangir Alom et al. 'Recurrent residual U-Net for medical image segmentation'. en. In: *J Med Imaging (Bellingham)* (Mar. 2019).

[69] Augustus Odena, Vincent Dumoulin and Chris Olah. 'Deconvolution and Checkerboard Artifacts'. In: *Distill* (2016). DOI: 10.23915/distill.00003. URL: http://distill.pub/2016/deconv-checkerboard.

[70] Deepak Mishra et al. 'Ultrasound Image Segmentation: A Deeply Supervised Network With Attention to Boundaries'. en. In: *IEEE Trans Biomed Eng* 66.6 (Oct. 2018), pp. 1637–1648.



NTNU

Norwegian University of
Science and Technology