# Coronary Artery Segmentation and Artery Tree Extraction on Angiographic CT Images

Jan Mika Dietz[1], Emile Gaudinot[1], Philipp von Mengersen[1], Nick Pralle[1]

[1]Group Cake, ML4MIP, Technische Universität Berlin,
28 February 2025

**SUMMARY**
This project explores machine learning-based segmentation of coronary arteries in CTA images to aid in coronary artery disease (CAD) assessment. Using a dataset of 800 3D CTA images with segmentation masks, we evaluate deep learning models, including UNETR, UNet, MedSAM, and nnU-Net, within a standardized pipeline. Additionally, we extract graph-based representations to analyze coronary structures. Our approach ensures systematic training, evaluation, and comparison, aiming to enhance automated CAD diagnosis and treatment planning. The code for this repository can be found in the TU-Cloud* and on GitHub†.

**Key words:** medical image processing – machine learning – coronary artery – segmentation – graph extraction.

## 1 INTRODUCTION

### 1.1 Motivation

Cardiovascular disease is a leading global health challenge, with coronary artery disease (CAD) being the most prevalent form. A key contributor to CAD is the pathological narrowing of coronary vessels, caused by the accumulation of atherosclerotic plaque. This narrowing leads to reduced blood flow and insufficient oxygen supply to the heart, resulting in significant morbidity and mortality caused by Myocardial infarction or cardiac arrest.

Diagnosing CAD involves a range of methods, including physical examinations, electrocardiograms (ECGs), laboratory tests, and imaging techniques like X-ray coronary angiography (XCA) or computed tomography angiography (CTA). Among these, CTA has emerged as a safer, more cost-effective, and less time-consuming alternative to traditional XCA for assessing stenoses and atherosclerotic plaques. This non-invasive method involves injecting the patient with a radiocontrast agent followed by high-speed CT scanning, enabling detailed visualization of coronary structures.

An essential step in the diagnosis and quantification of CAD is the segmentation of coronary arteries in CTA images. Accurate segmentation not only aids in the detection of stenoses but also supports the evaluation of treatment options such as percutaneous coronary intervention (PCI) and coronary artery bypass grafting (CABG). The PCI is a non-surgical procedure that uses a catheter (a thin flexible tube) to place a small structure called a stent to open up blood vessels in the heart. CABG aims at bypassing the blocked portion of the coronary artery with a piece of a healthy blood vessel from elsewhere in the body, attaching it below and above the blockage. To address this critical need, our project leverages a dataset of 800 3D CTA images and their corresponding segmentation masks to develop a machine learning model capable of precise and efficient coronary artery segmentation and subsequent processes [3].

### 1.2 Project goals

The project is divided into two key tasks. The first task involves the development of a machine learning model for the automated segmentation of coronary arteries. This includes designing and training the model to generate accurate segmentation masks from CTA images, as well as incorporating pre- and post-processing steps to improve the overall performance.

The second task focuses on extracting a graph-based representation of the coronary tree to better understand its structure. This graph representation will later enable exploratory analyses, such as segment labeling, addressing graph incompleteness due to stenosis or thrombosis, and classifying left/right dominance. Together, these tasks aim to enhance the automated analysis of CAD, providing a valuable contribution to clinical diagnostics and treatment planning.

### 1.3 First Steps

As a starting point for our project, our group initially presented the paper on the UNETR model [5], sparking an interest in testing this architecture for our segmentation task. Beyond UNETR, we were also drawn to exploring other approaches, including a custom UNet implementation, MedSAM, and nnU-Net, all of which were highlighted during the presentations in class. To ensure our experiments

* `https://tubcloud.tu-berlin.de/s/3JkEoaCrnyYyZog`
† `https://github.com/pvmeng/ML4MIP`

were conducted systematically, we decided to develop a fixed machine learning pipeline. This pipeline was designed to standardize the configuration, training, and evaluation processes across all experiments, enabling consistent and comparable results.

Before beginning the standardized experiments, each group member took the time to familiarize themselves with the programming environment and the provided resources. Following this initial phase, we commenced the setup of the machine learning pipeline, laying the foundation for our experiments.

## 2 METHODOLOGY

### 2.1 Data pre-processing

Data preprocessing transforms heterogeneous raw data into a unified dataspace optimized for the segmentation task while considering computational constraints. This results in two main strategies: (A) downsampling the raw images or (B) splitting them into several patches. Depending on the chosen method, an appropriate inference strategy must also be applied. Our work primarily focuses on 3D models, but we also explored 2D strategies, testing them with nnU-Net while initially considering MedSAM for evaluation. In the following, we briefly describe our preprocessing pipeline. Most components of our pipeline are implemented using MONAI [1]. Figure 1 illustrates the complete processing pipeline, utilizing the Patching Strategy followed by inference and post-processing.

**2.1.0.1 Voxel Spacing** Voxel spacing refers to the distance between the centers of adjacent voxels, defining the resolution of 3D data. The raw images have varying voxel spacing, which needs to be standardized to provide the model with a coherent representation of the data. To achieve this, we unify the spacing across each dimension using the geometric mean, resulting in a spacing of *(0.35, 0.35, 0.5)*. The full voxel spacing distribution is shown in Figure A1.
We choose the geometric mean because it minimizes interpolation artifacts and ensures a balanced representation across different resolutions.

**2.1.0.2 Value Scaling** CTA images typically have a single grayscale channel representing the Hounsfield Unit (HU) of the respective tissue. However, the value ranges vary across different images and can be within [-1000, 3000]. To provide the model with a unified representation, we consistently scale these values. Several strategies can be applied for this purpose. We experimented with both *min-max scaling* and *normalization*.
A key drawback of *min-max scaling* is that, due to varying value ranges across different images, the resulting distribution of background and foreground intensities can differ significantly between images, as illustrated in Figure A4. In contrast, the *normalization* yields a more consistent distribution of foreground and background voxel intensities (see Figure A5).
Despite the non-unified distribution introduced by min-max scaling, we did not observe a significant difference in segmentation performance between the two scaling strategies. Consequently, we primarily adopted min-max scaling and consider it the default unless stated otherwise.

**2.1.0.3 Patch or Crop** After achieving unified voxel spacing and value scaling, the model also requires equally sized inputs. Instead of resampling the volumes to a fixed shape, we apply a patch or crop operation: if the volume is smaller than the target shape, we add empty values at the edges, distributed equally on each side. Otherwise, we crop the volume to match the target shape.
This approach is advantageous over resampling since it preserves voxel spacing. To determine the target shape, we analyzed the shape distribution after applying unified voxel spacing, as shown in Figure A2. We selected a target shape of *(600, 600, 280)*. For the x and y dimensions, we chose *600*, as it closely approximates the geometric mean. Along the z-dimension, where variance is much smaller, we selected *280*, which accommodates all values while introducing only minimal padded regions.

**2.1.0.4 (A) Downsampling** After the previous preprocessing steps, we have unified volumes that remain too large to process directly within our training setup. A straightforward approach to reducing the volume size is to downsample the entire volume to a shape that is small enough to meet resource constraints while still maintaining sufficient resolution for the segmentation task.
However, as discussed in Section 3, downsampling and subsequent upsampling introduce interpolation artifacts, which impact segmentation performance. We experimented with different shapes and report our findings in later sections.

**2.1.0.5 (B) Patches** An alternative to downsampling is splitting the input volume into multiple patches of a specific target shape. Unlike downsampling, these patches retain the same resolution as the unified volume, but they provide less spatial context.
During inference, we aim to segment the entire volume, so we construct a grid-like structure with overlapping patches. For training, however, we randomly extract cropped patches of the desired target shape, introducing high variability into the dataset. To improve efficiency, we precompute these randomly cropped patches and directly use them for training. We experimented with different target shapes and present the results in Section 3.
We also explored alternative cropping strategies, such as crops with a higher probability of occurring near the center, modeled using a Gaussian distribution, as well as randomly selecting crops that ensure a positive foreground voxel at the center. We analyzed the proportion of patches containing at least one foreground voxel, as well as the overall percentage of foreground voxels across all patches, as shown in Figure A3. Although there is a strong class imbalance between foreground and background voxels in randomly cropped patches, we chose to retain the random cropping strategy, as it best reflects the actual distribution of segmented areas. In contrast, the other strategies introduce a bias that often leads to over-segmented patches. To further address class imbalance, we incorporate adjustments in our loss function, as discussed in Section 2.2.2, to ensure effective training.

### 2.2 Segmentation

#### *2.2.1 Model architectures*

The widely increasing accessibility of medical imaging accelerated research on image segmentation methods for medical image processing. In the context of the project several methods were evaluated to elaborate advantages of different methods. Ultimately, a configured version of the UNet was used for main inference.

**2.2.1.1 U-Net** The UNet architecture was originally introduced in 2015 by researchers at the University of Freiburg im Breisgau to address segmentation challenges in biomedical imaging [13]. A

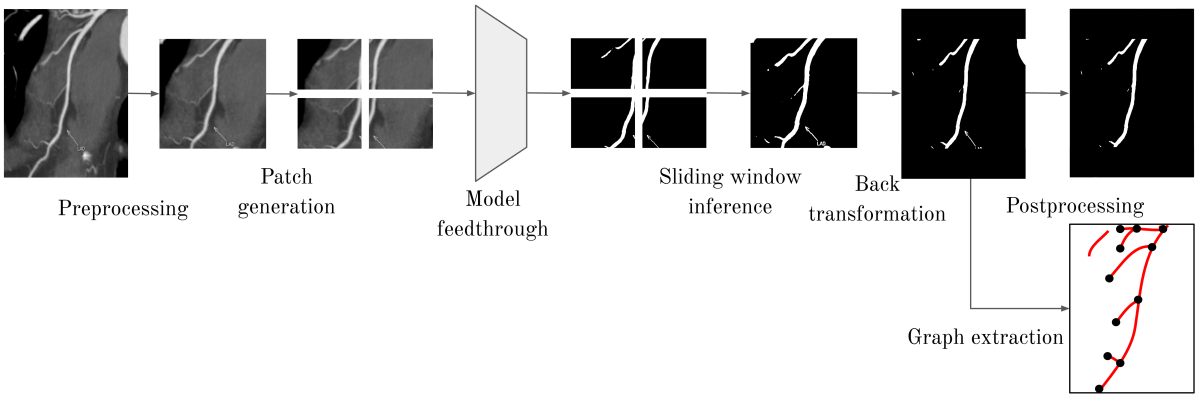



**Figure 1.** Processing Pipeline

year later, it was extended to the 3D domain to handle volumetric data [2]. The UNet architecture benefits from merging of high resolution information with lower-dimensional context related information and achieves high performance in the past. The UNet consists of a contracting and expansive path of convolutional layers with skip connections. The context is generated by successive down-sampling of the data using pooling. Primary configuration parameter of the model are the number of levels from the top to the bottleneck as well as the convolutional filter numbers. In the context of the project the 3D version was used to take advantage of the three-dimensional nature of the data and the concomitant higher spatial information. Several model configurations were tested throughout the project to identify an efficient and performative configuration.

**2.2.1.2 UNETR** UNet utilizes convolutions in the contracting and expansive path as main trainable model parameters. The small filter size of convolutional layers limits single layers to learn on local dependencies of small distance in the data. Attention-based mechanisms encounter this limitation [15]. In the UNETR model, the contracting path utilizes a chain of self-attention blocks to enable contextual understanding of the data [5]. In the beginning of the project, UNETR models were trained and evaluated.

**2.2.1.3 nnU-Net** The enormous usage of UNet models in not only medical image processing suggested research on standardized methods to identify parameters of well-performing UNet models. In this context, nnU-Net [6] was developed as a framework of automatic UNet configuration to reach high performance. Contrary to the other models nnU-Net was not trained as part of our data pipeline since it comes with its own pre- and post-processing system. These integrated modality suggested it as an independent benchmark model. In nnU-Net several UNet configurations were tested in the beginning of the project to get a reference for baseline model performance. Despite UNet models trained as part as the pipeline nnU-Net also trained two-dimensional versions of the model which generally leads to lower memory utilization as well as slightly lower model performance.

**2.2.1.4 MedSam (ViT-b)** MedSAM is a foundation model designed to enable universal medical image segmentation. [9] It has been developed on a large-scale medical image dataset with 1,570,263 image-mask pairs, covering 10 imaging modalities, including CT-scans. It relies on the Segment Anything Model (SAM, [7]). The image is propagated through an image encoder which processes the image and generates a set of embeddings. These represents the essential features of the image. Alongside with the image, a prompt input accepts various types of prompts, such as points, boxes, or masks, which guide the segmentation process by indicating the regions of interest within the image. MedSAM is a native 2D model. To gnerate a tree-diemsnional mask 3 inferences could be performed on each voxel: a first one considering z-slices, then x-slices and finally y-slices. A majority vote could then generate the final prediction. This could be a way to make use of the 3D structure of the data, since only considering the z-slices independently of each other leads to a significant information loss. Due to time constrains as well as implementation related difficulties MedSAM was not tested in the project period.

### *2.2.2 Loss functions*

The focus on the segementational accuracy of the model suggested a maximization of correct class predictions over the image. However, the sparse nature of the coronary arteries suggested a loss function aware of class imbalance.

**2.2.2.1 Binary Cross Entropy Loss** The binary cross-entropy (BCE) loss measures the cross entropy between the probability distributions of the predictions and the labels. Minimizing the cross-entropy between the distributions implies a minimization of the Kullback–Leibler divergence between the label distribution and distribution of predictions. [17] For a single scalar sample the loss can be formulated as

$$l = -w(y_{\text{true}} \log(y_{\text{pred}}) + (1 - y_{\text{true}}) \log(1 - y_{\text{pred}})).$$

For (mini-)batched training this atomic loss is aggregated over all voxels using averaging or summation: $\mathcal{L}_{\text{BCE}} = \mathbb{E}[l]$. Since the Kullback–Leibler divergence describes a divergence between the two probability distributions, it is not aware of class imbalances. Class imbalances might lead to the prediction of the most common class for all outputs at the cost of the other classes to minimize the overall error. This limitations is moderated by weighted binary cross-entropy loss, which enables weighting of different classes and sample regions.

**2.2.2.2 Dice-Sørensen coefficient based loss** The Dice-Sørensen coefficient [4] measures the sample similarity for classification tasks. Generally, it measures the ratio of the intersection of the two classification masks over the sum of overall active elements in two masks each. For per-class probability predictions a scalar product from can be used to calculate a uncertainty-aware intersection in the Dice-Sørensen coefficient

$$s = \frac{2|y_{\text{true}} \cdot y_{\text{pred}}|}{|y_{\text{true}}|^2 + |y_{\text{pred}}|^2}.$$

A DICE-Loss can be formulated by $\mathcal{L}_{\text{DICE}} = -\mathbb{E}[s]$ to maximize the similarity of labels and predictions. Smoothing constants were added to the denominator and numerator to ensure numerical stability. When calculating for underrepresented target classes it will not bias overrepresented classes and therefore can be used to face classification tasks with class-imbalances under these constraints.

**2.2.2.3 Combined loss** In the context of the project a combination of the BCE- and Dice-Loss was used to profit from the global distribution similarity-related approach of the BCE loss as well as the local intersection-focused approach of the Dice-Loss. The combined loss is represented by a weighting of the indiviual losses:

$$\mathcal{L} = \lambda_{\text{DICE}}\, \mathcal{L}_{\text{DICE}} + \lambda_{\text{CE}}\, \mathcal{L}_{\text{BCE}}.$$

The hyperparameter selection and reasoning are described in 3.1.

### 2.3 Data post-processing

#### *2.3.1 Inference*

The inference strategy is crucial for generating the segmentation mask at its original resolution and size. Depending on the preprocessing strategy, different methods are required to achieve this.

**2.3.1.1 Resize and Interpolation** If downsampling (see Section 2.1.0.4) was applied after running model inference on a low-resolution input volume, the output must be upsampled to a unified size of *(600,600,280)*. This is achieved using PyTorch trilinear interpolation [10]. We experimented with different interpolation strategies, including scaling logits, probabilities, and binary values. Our observations indicate that binary interpolation performed well in terms of centerline Dice, whereas logits and probabilities resulted in better overall segmentation Dice scores. Binary interpolation is used as default.

**2.3.1.2 Sliding Window Inference** If the model was trained on patches (see Section 2.1.0.5), a grid-like structure with overlapping patches is constructed to segment the entire volume. By default, overlapping regions are equally weighted to ensure a smooth and consistent reconstruction of the full segmentation. The sliding window inference was implemented using MONAI [1].

**2.3.1.3 Reshape to Original Size** After applying one of the previously mentioned methods, we obtain segmentation volumes of the unified shape *(600, 600, 280)*. In the final step, the output volumes are reshaped back to their original format by first restoring the original pixel dimensions and then adjusting the shape of the source volume using a patching or cropping operation.

#### *2.3.2 Connected Component Analysis*

After performing inference on the input volume, we obtain the raw segmentation mask. This can be further refined using connected component analysis to enhance segmentation performance. To achieve this, we filter connected components based on three criteria: (i) the size of the component, (ii) its minimum distance to one of the $n$-largest components, and (iii) the number of largest components to consider.
This approach allows us to remove small segmentation islands that are either insufficiently connected or falsely segmented. Depending on the use case, we apply different configurations for filtering: For segmentation, our goal is to maximize the number of correctly segmented voxels to achieve optimal performance metrics. However, in graph extraction, an excessive number of unconnected components can negatively impact the results. To address both scenarios, we experimented with multiple configurations and present our results in Section 3.3.2.

### 2.4 Graph extraction

The Graph Extraction process further transforms the segmented artery tree into a structured graph representation consisting of nodes and edges. The nodes correspond to voxels in the segmentation mask that represent either branching points or leaf points. Edges connect these nodes wherever an artery segment exists between them. Additionally, each edge includes a sequence of points that more precisely delineate the course of the respective artery segment. In the following we describe the processing steps that are applied on the post-processed binary segmentation volume:

#### *2.4.1 Skeletonization and Graph Conversion*

Skeletonization reduces a shape to its essential structure while preserving connectivity and topology, making it useful for extracting centerlines of binary structures like blood vessels. We employ the `skeletonize` function from `skimage.morphology` [16], which iteratively thins object boundaries to obtain a one-pixel-wide representation. However, it is important to note that the resulting skeleton does not perfectly reflect the true centerline and often introduces additional small paths and loops, increasing the number of branching points in some areas. This issue is further addressed in Section 2.4.3.
Given the skeleton, we can define the neighborhood of each voxel using a simple $3^3$ kernel and construct an undirected graph. Each skeleton voxel serves as a node, and edges are formed by connecting neighboring voxels within the skeleton.

#### *2.4.2 Merge to two connected components*

The initial graph contains as many connected components as the previous segmentation volume. Since there are only two coronary arteries, the components may need to be connected. This is achieved by merging the smallest component with its closest neighbor by adding an edge between the nearest nodes. This process is repeated until only two components remain.

#### *2.4.3 Graph reduction*

The final graph should include only nodes that are either leaf or branching points. Thus, only nodes with a degree of 1 or greater than or equal to 3 are relevant. Nodes with a degree of 2 merely serve as connectors between these relevant nodes and need to be processed to form the edges. This is achieved by traversing the connecting nodes from each branch or leaf node until another branch or

leaf node is reached. An edge is then added between the source and destination nodes, while all nodes along the path are collected to provide a more detailed representation of the artery course. This process is repeated for each unvisited connecting node of every branch or leaf node until all nodes have been processed. During this process, clusters of nodes occasionally remained without proper reduction. To address this, the nodes within each cluster were merged into a single node. This was achieved by ensuring that no other node existed within a 10-voxel radius of each node. This threshold was determined through qualitative testing to balance accuracy and consistency in node reduction.

#### *2.4.4 Directed graph & root node calculation*

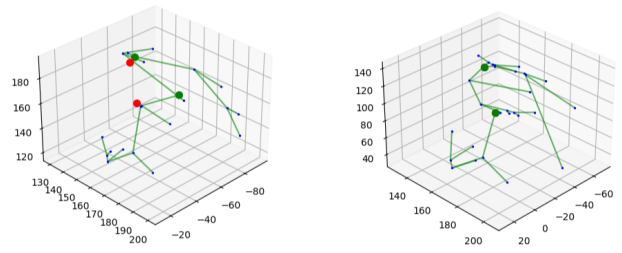
In the final step, it is necessary to determine the root nodes for each connected component in the graph and direct all edges accordingly, ensuring that paths extend from each root node to the leaf nodes.
After analyzing ground truth graphs and their root nodes, we observed that these root nodes are often positioned at a high location and aligned toward the center. Based on this observation, we devised a heuristic that selects the highest-positioned nodes as root nodes for the left and right subtrees.
Since the ground truth graphs are represented in world coordinates, the graph must first be transformed accordingly. This can be achieved by multiplying the affine matrix with homogeneous voxel coordinates. Once translated into world coordinates, the heuristic can be applied ⋆ .
We also experimented with a more refined heuristic that considers not only the vertical position of nodes within the graph but also their alignment toward the center. One approach involves computing the centroid of all graph nodes and setting its $z$-coordinate to the maximum value within the volume. Nodes could then be ranked based on their Euclidean distance to this high-positioned centroid, and the closest node for each subtree could be selected as the root node. However, at the time of submission, this heuristic was not yet mature enough for integration into our pipeline.
While the initial heuristic was not always perfectly accurate, it frequently produced correct results or was at least very close (see Figure 2).
Based on these root nodes, we directed the graph by traversing it using depth-first search (DFS). An edge was added between a source and a destination node if an undirected edge existed between them and the destination node was visited after the source node.

## 3 EXPERIMENTS & RESULTS

### 3.1 Model training

During the project several model architectures and configurations were trained and tested. Starting with models with low parameter counts for evaluation of simple training and model parameters as activation functions and data pre-processing. Beside of these tests different architectures as UNet and UNETR were tested and compared. Finally, several configurations of the preferred UNet model were trained and the effect of hyperparameters was analyzed.

⋆ At the time of submission, we mistakenly applied the heuristic before translating to world coordinates, causing it to not function as expected.



(a) Approximate determination (b) Accurate determination

**Figure 2.** Root node computation. We have the ground truth roots in red, and our computations in green. In 2a, we are approximate but close. In 2b, we have an accurate determination (green is exactly over red).

#### *3.1.1 nnU-Net training*

Since nnU-Net was intended to be used as a benchmark only and was not integrated in our pipeline it was trained independent from the other models. The model was trained using the 2D-, 3D-lowres and partially by the 3D-fullres strategy. Parameters were mainly determined by nnU-Net. An overview over the regarding configurations is provided in the configuration file `plans.json` in the supplementary material.

#### *3.1.2 Optimization*

All other models were fitting using the AdamW optimizer [8]. To promote convergence of loss a linear learning rate scheduler was used. Different learning rate decay rates were tested. Finally, the initial learning rate was set to $\gamma_{\text{init}} = 10^{-2}$ and decayed to $\gamma_{\text{final}} = 10^{-4}$ at the end of the last epoch. During first testing phases the models where trained 20 epochs. The final model was finetuned for 100 epochs. All models were trained in a (mini-) batched fashion. The batch size was generally set to maximal possible size of samples fitting to the GRAM substracted by model weights and intermediate tensor representations. In most cases the batch size was defined to be $n = 4$. Different loss configurations of the combined DICE-CE loss (2.2.2.3) were tested.
In the Patch Scenario, a high weighting of $\lambda_{\text{BCE}}$ led to a model that predominantly predicted the background class, only occasionally learning weights that correctly classified arteries. This issue was particularly evident when using datasets with non-uniform patch sampling (Section 2.1.0.5). In contrast, training with only DICE loss ($\lambda_{\text{BCE}} = 0$) resulted in high false positive rates due to a slight bias toward patches containing arteries. To address this, we fixed the loss parameters to $\lambda_{\text{DICE}} = 1$ and $\lambda_{\text{BCE}} = 0.3$ for all training runs. Unless otherwise stated, PyTorch default values were used.
For the Downsampling Scenario, training with only DICE loss was sufficient, as both foreground and background pixels were consistently present in the training samples.

#### *3.1.3 Training environment*

All models were trained on a central training server provided by the Institute of Computer-assisted Cardiovascular Medicine of the Charité Universitätsmedizin Berlin in the context of the academic project. The GPU-accelerated training was performed on Nvidia RTX2080 Ti GPUs on the system. `PyTorch` was used as main

model optimization framework. The `MONAI` framework was used to support model development.

#### *3.1.4 Dataset configuration*

The CTA images were split into three subsets of 720, 20, and 60 images for training, evaluation, and testing, respectively. After assignment to a subset, different datasets were derived from the original CTA images using the methods described in Section 2.1. Initially, data samples were processed at training time, but later, datasets were cached to accelerate training.
For each sample, 20 patches were cropped, resulting in a total of $800 \times 20 = 16,000$ training samples. However, in each epoch, only 800 crops were used in an interleaved fashion, ensuring that after 20 epochs, all patches had influenced the gradients. Due to the spacing settings in the post-processing pipeline and the relatively low patch coverage of the CTA images, even more patches could potentially be generated from each image.
For final model training, the crop positions of the patches were uniformly sampled.

### 3.2 Evaluation metrics

During model development, several evaluation metrics were used. In the context of graph extraction and post-processing, structure-based metrics were preferred over intersection-based metrics. Conversely, the latter were prioritized during hyperparameter tuning while training the segmentation model.

- Dice-Sørensen coefficient - The Dice-Sørensen coefficient $S$ was already introduced in 2.2.2.2 as part of the relating loss. It essentially quantifies how similar the predicted $P$ and ground truth segmentation $L$ are in proportion to the positives in the prediction and label:

$$S = \frac{2|P \cap L|}{|P| + |L|}.$$

  This score was used throughout the project as the main performance metric.
- Jaccard index - The Jaccard index is another set based similarity measure. It quantifies the ratio of the intersection of the two sets to their union. The Jaccard index $J$ relates to the Dice-Sørensen coefficient $S$ by the $J = \frac{S}{2-S}$.
- Hausdorff distance - The Hausdorff distance quantifies the maximum spatial dissimilarity between the predicted and ground truth arterial structures. Unlike overlap-based metrics, it measures the maximum of all points distance between the point itself and the closest point on the other set [12]. Here it iterates over all points on the predicted artery volume and measures the distance to the corresponding ground truth artery mask, and vice versa. In our context, the Hausdorff distance is sensitive to boundary discrepancies and can capture errors in the overall shape and extent of the segmented vessels, including missed branches or inaccurate vessel boundaries.
- Centerline Dice coefficient - The centerline Dice coefficient (clDice) is a metric used to evaluate the accuracy of coronary artery segmentation and graph extraction. Unlike the standard Dice-Sørensen coefficient that assesses volumetric overlap, the clDice focuses on the structural representation of the coronary artery centerline [14]. It measures the spatial agreement between the predicted and ground truth centerlines, represented by its skeleton. Specifically, it quantifies the proportion of the predicted centerline that falls within the artery volume of the ground truth artery, and vice versa. This metric is crucial for assessing the fidelity of the extracted arterial pathways.

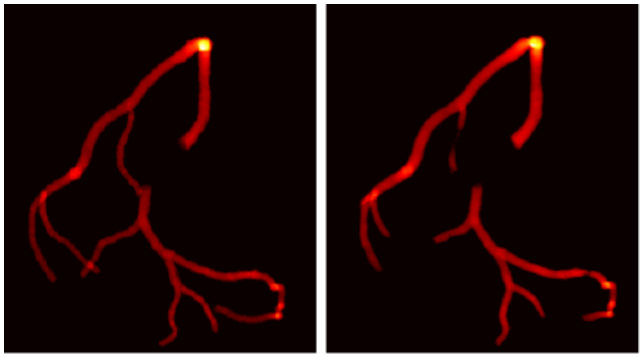


**Figure 3.** 2D Projection of Segmentation Mask: Brighter regions indicate a higher number of positive voxels along the reduced dimension. Reference (left) and Prediction (right).

To eventuate the performance of the graph extraction pipeline three main metrics were used. For all graph measurements the ICC(3,1) was calculated to compensate for different raters. Alternatively also the Mean Absolute Error (MAE) is computed. The evaluated graph measurements were the graph depth, the overall graph length and the total node number (graph size) [3].

### 3.3 Results

#### *3.3.1 Pre-processing and Model configuration*

We tested different learning scenarios with respect to downsampling and patching, various model architectures, and input shapes, as shown in Table 1. The best-performing model in terms of Dice score is the 3D nnU-Net.
For models trained using our pipeline, we observe that the Patch 3D approach outperforms the Downsampling approach. Another clear pattern emerges: increasing the input shape leads to better performance. Initially, we experimented with isomorphic shapes, but later switched to non-isomorphic shapes, such as *(192, 192, 96)*. This choice better aligns with the postprocessed volume shape of *(600, 600, 280)*, as it maintains a more proportional fit.
Regarding model architectures, we primarily focused on UNet and were unable to train many configurations of UNETR due to the slow training and loss convergence.
While patch-based models perform well in both Dice score and clDice score, the downsampling approach shows significantly weaker Dice performance, though its clDice score remains competitive. This can be explained by the necessary upsampling process, which introduces interpolation artifacts that negatively impact the Dice score but preserve the topological structure. Given these result we decided to use the 3D Patch *(192,192,96)* model for submission.

#### *3.3.2 Post-processing configuration*

As we can see in Figure 3, the predicted segmentation mask is very similar to the reference. However, some parts are missing, while other areas exhibit over-segmentation. As mentioned in Section 2.3.2, we can remove unwanted segmentation artifacts using filtering strategies based on Connected Component Analysis. To determine the optimal configuration, we evaluated the performance using the provided evaluation script on both the segmentation task

| Learning Scenario | Patch (3D) | | | Downsampling (3D) | | | | 2D |
|---|---|---|---|---|---|---|---|---|
| **Model Architecture** | **UNet** | | | **nnU-Net** | **UNETR** | **UNet** | | **nnU-Net** |
| **Input-Shape** | 96:96:96 | 128:128:128 | 192:192:96 | 96:160:160 | 96:96:96 | 128:128:128 | 192:192:96 | 512:512 |
| **Loss-Function (Weights)** | DICE_CE (1,.3) | DICE_CE (1,.3) | DICE_CE (1,.3) | DICE_CE | DICE | DICE | DICE | DICE_CE |
| **Epochs** | 200 | 200 | 200 | 1000 | 50 | 50 | 50 | 1000 |
| **Test Dice** | 0.68 | 0.72 | **0.77** | *(0.82)** | 0.43 | 0.53 | 0.53 | 0.78 |
| **Test Centerline** | 0.74 | 0.75 | **0.83** | - | 0.53 | 0.69 | 0.80 | - |

**Table 1.** Comparison of Model Architectures in Different Learning Scenarios. *Pseudo-DICE Score obtained by nnU-Net.

and the graph extraction task. The results for the Segmentation (Figure A6) and Graph Extraction (Figure A7) can be found in the appendix.

For the segmentation problem, we noticed that the performance metrics were barely impacted by our post-processing, except for the Hausdorff distance. This observation is intuitive, as the Hausdorff distance is strongly influenced by outliers, which can be effectively removed through post-processing. Based on this, we employed a filter configuration with `size`=25, `distance`=50, and $n$-`largest`=2.

For the graph extraction task, more variance was observed in the results. To determine the optimal configuration, we considered the overall mean across the evaluated performance metrics. The resulting optimal configuration was `size`=1, `distance`=0, and $n$-`largest`=3. This configuration corresponds to selecting the three largest components.

## 4 DISCUSSION

### 4.1 Challenges

During this project, we faced technical challenges alongside team-related difficulties. Varying levels of coding experience made task distribution complex, requiring efficient project management. Limited GPU resources further constrained our workflow, making it essential to coordinate around differing class schedules and workloads. Due to technical issues, the adequate allocation of the GPU resources between the groups was not possible. This led to resource conflicts while training.

Since our main priority was establishing the structured pipeline, limited time was available for the graph extraction task, resulting in some inaccuracies. For example, the transformation to the required coordinate system was mistakenly performed at the wrong step, leading to poor root node detection. The overall time frame was very tight, making it challenging to balance exploring new strategies while refining specific approaches to meet the submission goals.

### 4.2 Considerations regarding Future Work

Due to time and resource constraints, many aspects could not be tested extensively. Nevertheless, our pipeline provides a strong foundation for future experiments, which has been our main focus and accomplishment.

Regarding the segmentation task, further fine-tuning of the hyperparameters of the UNet model is an important direction, along with making an informed decision between a custom model and nnU-Net. Exploring alternative model architectures, such as UNETR, could also lead to improvements.

For graph extraction, refining skeletonization techniques could enhance the accuracy and robustness of the extracted graphs. Another promising path might be training a network directly to predict the graph structure rather than relying on the segmentation step.

As mentioned in the course slides, future work on segment labeling, vessel occlusion, and graph incompleteness (caused by stenosis or thrombus), as well as the classification of left/right dominance, presents interesting downstream applications. A Graph Neural Network (GNN) could serve to tackle these challenges by leveraging the inherent relational structure of vascular graphs. For segment labeling, a GNN could learn vessel hierarchies and anatomical relationships. Incomplete graphs could be compared to intact graphs to identify missing regions. Left vs. Right Coronary Dominance can also be inferred using a machine learning classification system, leveraging features like vessel length, branching points, and perfusion patterns for accurate prediction.

## 5 FINAL REMARKS

Overall, we thoroughly enjoyed working on this hands-on project and collaborating as a team. Throughout the process, each of us gained valuable insights and skills, sparking our interest in conducting further experiments with the pipeline we developed. We sincerely thank for this opportunity.

## REFERENCES


[1] M. Jorge Cardoso et al. “MONAI: An open-source framework for deep learning in healthcare”. In: *arXiv preprint arXiv:2211.02701* (2022). URL: `https://arxiv.org/abs/2211.02701`.

[2] Özgün Çiçek et al. “3D U-Net: Learning Dense Volumetric Segmentation from Sparse Annotation”. In: *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2016*. Springer International Publishing, 2016, pp. 424–432. ISBN: 9783319467238. DOI: `10.1007/978-3-319-46723-8_49`. URL: `http://dx.doi.org/10.1007/978-3-319-46723-8_49`.

[3] Institute of Computer-Assisted Cardiovascular Medicine. “Machine Learning for Medical Image Processing”. Module Lecture Slides. 2024.

[4] Lee R. Dice. “Measures of the Amount of Ecologic Association Between Species”. In: *Ecology* 26.3 (July 1945), pp. 297–302. ISSN: 1939-9170. DOI: `10.2307/1932409`. URL: `http://dx.doi.org/10.2307/1932409`.

[5] Ali Hatamizadeh et al. “UNETR: Transformers for 3D Medical Image Segmentation”. In: *2022 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. IEEE, Jan. 2022, pp. 1748–1758. DOI: 10.1109/wacv51458.2022.00181. URL: http://dx.doi.org/10.1109/WACV51458.2022.00181.

[6] Fabian Isensee et al. “nnU-Net: a self-configuring method for deep learning-based biomedical image segmentation”. In: *Nature Methods* 18.2 (Dec. 2020), pp. 203–211. ISSN: 1548-7105. DOI: 10.1038/s41592-020-01008-z. URL: http://dx.doi.org/10.1038/s41592-020-01008-z.

[7] Alexander Kirillov et al. *Segment Anything*. 2023. DOI: 10.48550/ARXIV.2304.02643. URL: https://arxiv.org/abs/2304.02643.

[8] Ilya Loshchilov and Frank Hutter. *Decoupled Weight Decay Regularization*. 2017. DOI: 10.48550/ARXIV.1711.05101. URL: https://arxiv.org/abs/1711.05101.

[9] Jun Ma et al. “Segment anything in medical images”. In: *Nature Communications* 15.1 (Jan. 2024). ISSN: 2041-1723. DOI: 10.1038/s41467-024-44824-z. URL: http://dx.doi.org/10.1038/s41467-024-44824-z.

[10] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32 (NeurIPS 2019)*. 2019, pp. 8024–8035. URL: v://arxiv.org/abs/1912.01703.

[11] Sebastian Porsdam Mann et al. “Guidelines for ethical use and acknowledgement of large language models in academic writing”. In: *Nature Machine Intelligence* 6.11 (Nov. 2024), pp. 1272–1274. ISSN: 2522-5839. DOI: 10.1038/s42256-024-00922-7. URL: http://dx.doi.org/10.1038/s42256-024-00922-7.

[12] R. Tyrrell Rockafellar and Roger J. B. Wets. *Variational Analysis*. Springer Berlin Heidelberg, 1998. ISBN: 9783642024313. DOI: 10.1007/978-3-642-02431-3. URL: http://dx.doi.org/10.1007/978-3-642-02431-3.

[13] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “U-Net: Convolutional Networks for Biomedical Image Segmentation”. In: *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*. Springer International Publishing, 2015, pp. 234–241. ISBN: 9783319245744. DOI: 10.1007/978-3-319-24574-4_28. URL: http://dx.doi.org/10.1007/978-3-319-24574-4_28.

[14] Suprosanna Shit et al. “clDice - a Novel Topology-Preserving Loss Function for Tubular Structure Segmentation”. In: *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2021, pp. 16555–16564. DOI: 10.1109/cvpr46437.2021.01629. URL: http://dx.doi.org/10.1109/CVPR46437.2021.01629.

[15] Ashish Vaswani et al. *Attention Is All You Need*. 2017. DOI: 10.48550/ARXIV.1706.03762. URL: https://arxiv.org/abs/1706.03762.

[16] S. van der Walt et al. “scikit-image: image processing in Python”. In: *PeerJ* 2 (2014), e453.

[17] Wikipedia contributors. *Cross-entropy — Wikipedia, The Free Encyclopedia*. [Online; accessed 27-February-2025]. 2025. URL: https://en.wikipedia.org/wiki/Cross-entropy.

## ACKNOWLEDGMENTS

In the context of the report and software development large language models were utilized to support the implementation and research process. Any use of generative AI in this report adheres to ethical guidelines for use and acknowledgment of generative AI in academic research. [11] Each author has made a substantial contribution to the work, which has been thoroughly vetted for accuracy, and assumes responsibility for the integrity of their contributions.

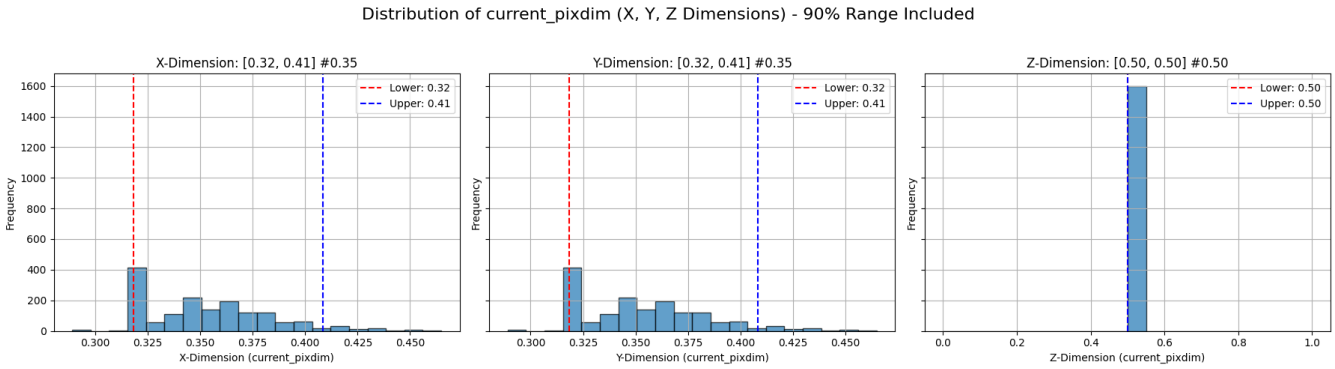



**Figure A1.** Voxel Spacing Distribution

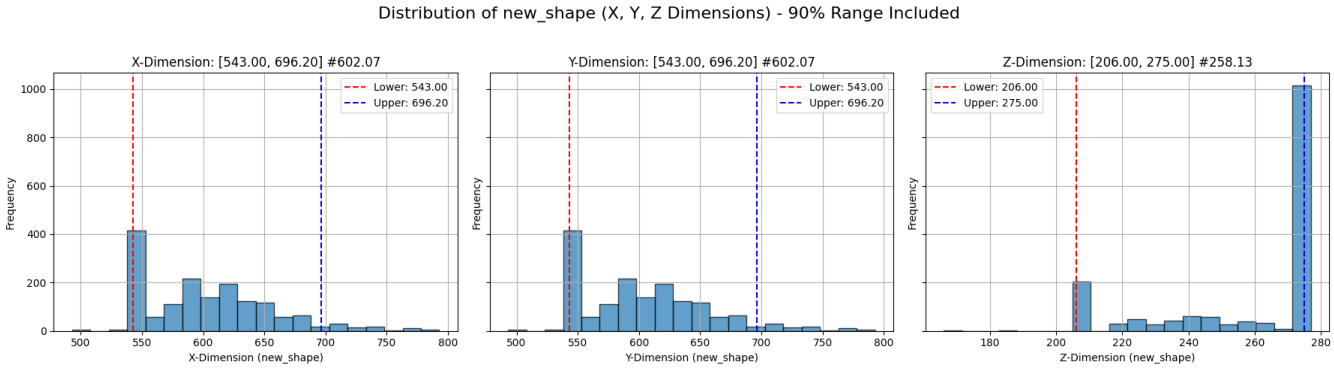



**Figure A2.** Shape Distribution

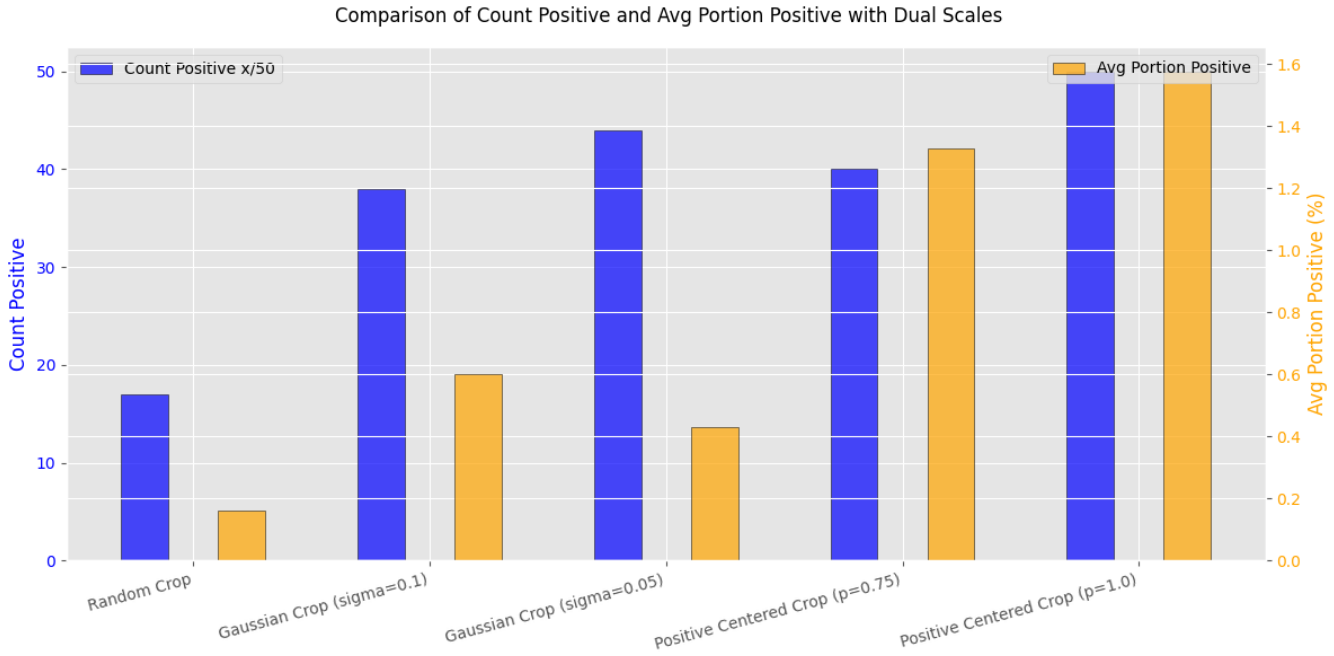



**Figure A3.** Class Imbalance of Different Cropping Strategies. The Evaluation was performed on a small holdout set of 50 examples.

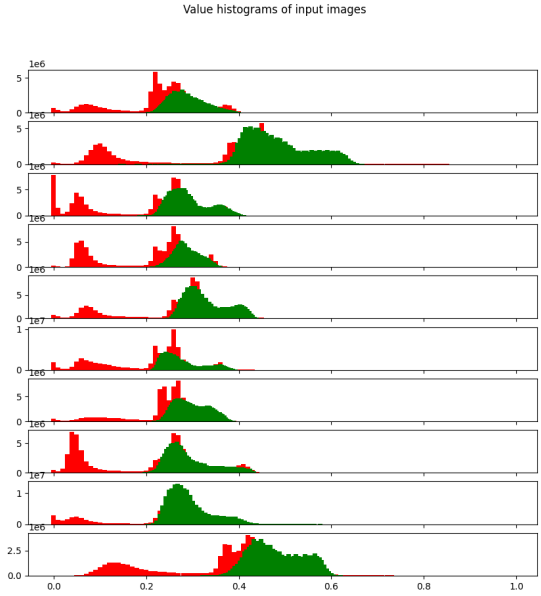



**Figure A4.** Min-Max Scaling

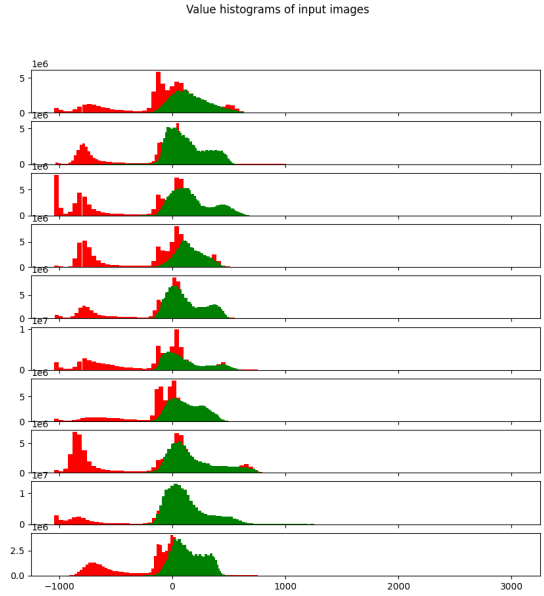



**Figure A5.** Normalization

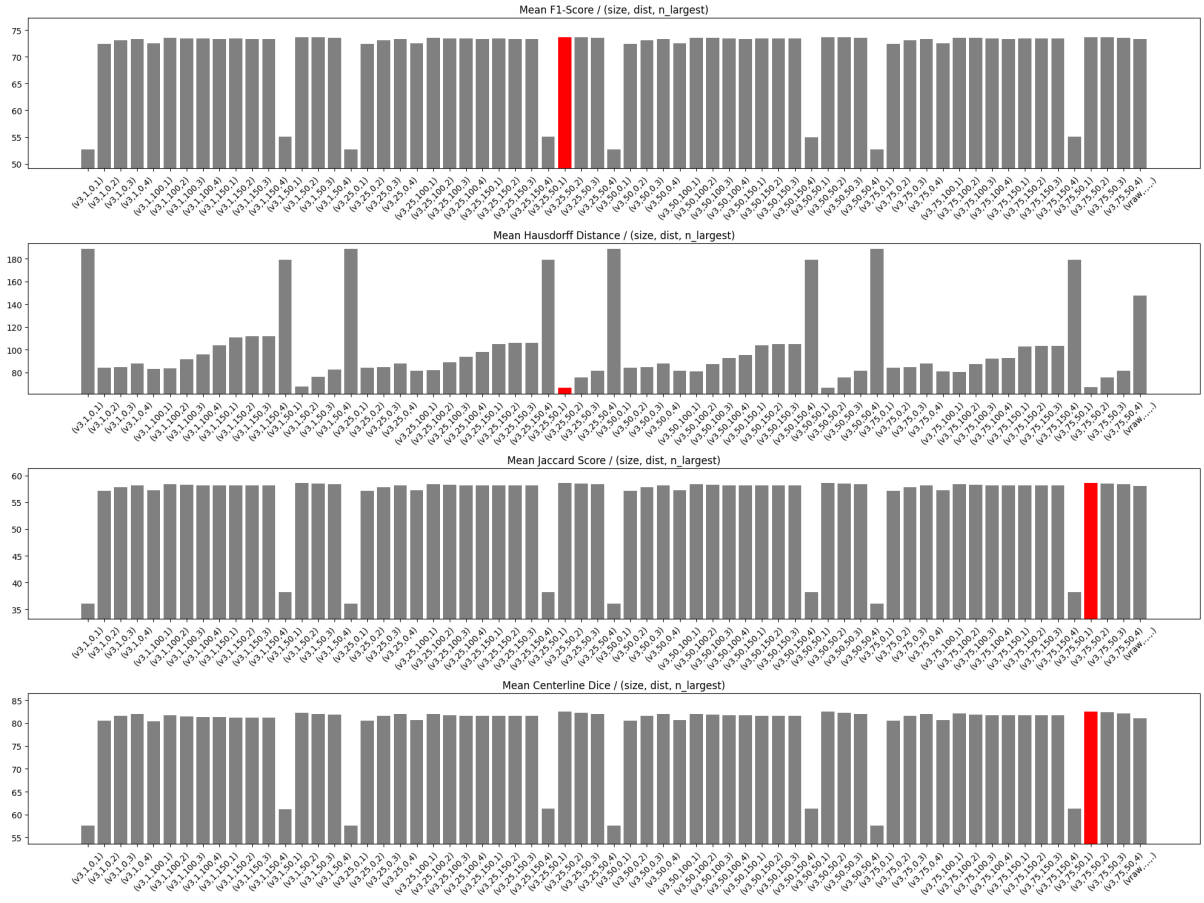



**Figure A6.** Post-Processing Evaluation for Segmentation. The table presents the mean values of different evaluation metrics across rows: 1st row – F1-Score, 2nd row – Hausdorff Distance, 3rd row – Jaccard Score, 4th row – Centerline Dice.

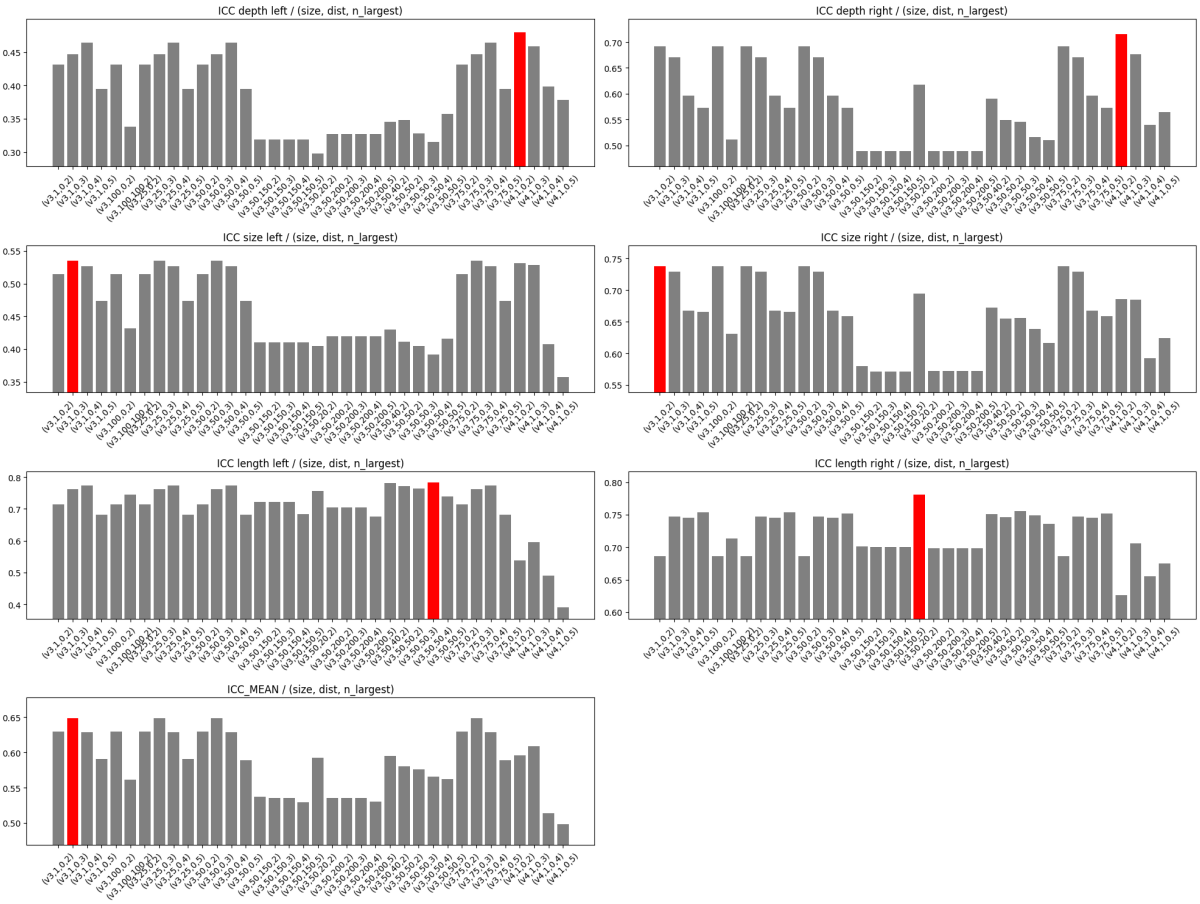



**Figure A7.** Post-Processing Evaluation for Graph Extraction Segmentation. The table presents the mean values of different evaluation metrics across rows: 1st row – ICC Depth, 2nd row – ICC Size, 3rd row – ICC Length, 4th row – Geometric Mean of the Metrics.