



Universiteit
Leiden

Master Computer Science

Coronary Artery Extraction using Geometrical
Deep Learning on Medical Imaging Data

Name: T. Snelleman
Student ID: s1655949
Date: August 2, 2022
Specialisation: Artificial Intelligence
Supervisor: [N. Hampe (MSc.), AMC]
Supervisor: [B.M. Renting (MSc.), LIACS]
Supervisor: [Dr. J.N. van Rijn, LIACS]
Supervisor: [Prof. Dr. H.H. Hoos, LIACS]
Supervisor: [Prof. Dr. I. Išgum, AMC]

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

Coronary artery disease is the leading cause of death worldwide. To treat patients with such disease, clinicians need to identify the affected vessel(s). However, this analysis is cumbersome and costly. Automatic coronary artery extraction may assist experts in this task. Current state-of-the-art approaches achieve good results but do not address the difficulties found in the most complex cardiovascular systems. Current methods do not extract the complete artery in the complex population, due to locally similar non-arterial structures. To tackle this issue, we propose to employ geometrical deep learning methods on raw output graphs to differentiate between artery and other blood vessels or structures resembling blood vessels. We hypothesize that such techniques can outperform the state of the art by utilizing non-local information, such as geometry. To this end, we developed Graph U-NET II, which consists of graph convolutions, node pooling layers, and skip connections. We have found that, on our test set of 50 patients, our best ensemble method achieves an overlap with the reference data of 87.0 %. Moreover, it achieves a recall on the positive class, or sensitivity, of 86.0 %. In terms of overlap, our method exceeds the state-of-the-art graph agnostic approach (87.0 % vs. 80.4 %). However, our method does not outperform the state of the art in terms of sensitivity (87.1 % vs. 86.0 %). These results indicate a more clinically relevant model compared to the state of the art, due to the substantial increase in overlap.

Contents

1	Introduction	5
2	Background and Approach	7
2.1	Data	7
2.2	Coronary Centerline Extraction	8
2.3	Coronary Artery Centerline Tree Construction	10
2.4	Proposed Solution	10
3	Related Work	12
3.1	Structural Anatomy Classification	12
3.1.1	3D Convolutional Neural Networks for Automatic Coronary Centerline Extraction	12
3.1.2	Graph Attention Networks for Segment Labeling in Coronary Artery Trees	13
3.2	Geometrical Deep Learning	13
3.2.1	Directed Acyclic Graph Neural Networks	14
3.2.2	Edge Contraction Pooling for Graph Neural Networks	14
3.3	Hyperparameter Optimization	15
4	Methods	17
4.1	Method Training Techniques	17
4.1.1	Loss and Optimization	17
4.1.2	Dropout and Weight Decay	17
4.1.3	Precision-Recall Curve	18
4.2	Baselines	18
4.2.1	XGBoost Classifier	18
4.2.2	Multilayer Perceptron	18
4.3	Graph Neural Network operators	19
4.3.1	Convolutional Layers	19
4.3.2	Graph Pooling Operators	20
4.4	Graph Neural Network Architectures	21
4.4.1	Graph Convolutional Neural Networks	23
4.4.2	Graph Convolutional Pooling Networks	23

4.4.3	Graph U-NET II	23
5	Experimental Setup	25
5.1	Metrics	25
5.2	Training and Validation	26
6	Experiments	27
6.1	Graph Agnostic Baselines	27
6.1.1	XGBoost	27
6.1.2	Multi-layer Perceptron	29
6.2	Graph Neural Networks	31
6.2.1	Graph Convolutional Neural Network	31
6.2.2	Graph Convolutional Pooling Networks	32
6.2.3	Graph U-NET II	35
6.3	Comparison	38
7	Discussion	42
8	Conclusion and Future Work	43
Appendices		
A	Data	
B	Experiments	

1 Introduction

Cardiovascular diseases are one of the most deadly diseases worldwide [22], among which ischemic heart disease is the most frequent [21]. Along with other tools, Coronary Computed Tomography Angiography (CCTA) scans enable experts to determine the cause of and solutions to such diseases. To partially relieve the experts of this cumbersome analysis of evaluating the medical state of the arteries, we propose to automatically extract the arteries from the scans. We use the methods of Wolterink et al. [49] to propose a new approach for a fully automatic extraction method. Schaap et al. [35] define fully automatic extraction as requiring no human interaction.

For the task, we utilize 200 CCTA scans of patients, where the coronary arteries have been manually labeled by experts. There are multiple ways to extract the arteries from the scans, where we will use a tracking-based approach using deep learning. Tracking-based approaches in artery extraction methods use seed points to start a sequential extraction in the CCTA scan, where in each step a new centerline point of the artery is determined. For coronary artery tracking methods, deep learning techniques are currently the state of the art [34]. In our approach, we combine these tracking-based methods with geometrical deep learning, to create a fully automatic extraction method. We use a pipeline of convolutional neural networks (CNNs) by Wolterink et al. [49] to extract the centerlines of coronary arteries from the CCTA scans. The coordinates of these centerlines describe the locations of image patches. Patches are small segments from the scans containing information about the blood vessel, such as a center and a radius. To evaluate an approach, reference markers were placed on each CCTA scan by experts for comparability. These reference markers together create the reference tree. As measurement metrics, we apply overlap and sensitivity, e.g. recall on the positive class. To achieve a high sensitivity on the reference tree, the method has been configured to be oversensitive by Hampe et al. [16] and it extracts on average over 87.17 % noise. Noise in this case can be any extracted patch unrelated to the coronary arteries, which can be veins or other objects resembling vessels. To identify this noise, the current state of the art uses local patch information to determine if the data is related to the cardiovascular system [49] [34]. This task is complicated, as many other structures share representative qualities with the arteries, thus causing many false positives produced by the tracking method. Thus, the objective presents us with a binary classification task. Locally restricted data may not provide enough information to a model to differentiate between arteries and vessels, due to the local resemblance of arteries with veins. Furthermore, calcified or even stenosed arteries that may feign the end of an artery. Thus, assessing a longer sequence of patches may be beneficial to determine the label of the patches, as geometric information is only represented on a more global level. In previous works, the local information was insufficient, which led to the requirement of all segments being connected to the Ostia. The Ostia are the origins of the two main arteries that spring from the aorta.

To address this potential shortcoming of local information, where single centerline points are too ambiguous to determine a label, we leverage the graph structure in the arteries using geometrical deep learning. In state-of-the-art local information approaches, connectivity with the Ostia is required to avoid other blood vessels to be confused with the arteries. Our approach allows us to drop the requirement of connectivity with the Ostia, as the disjointed segments

can yield enough information to determine whether it represents an artery or noise. We create a graph structure of image patches by presenting the patches as nodes and their connectivity as edges. Other approaches have investigated the use of the field of geometrical deep learning in coronary artery segmentation, such as Wolterink et al. [48], who aim to detect stenoses in the arteries using Graph Neural Networks. Hampe et al. [17] determined the type of coronary artery using Graph Attention Networks. Hampe et al. [16] introduces graph neural networks as a classification method after centerline point extraction, to separate the arteries from noise.

In this thesis, we bring new approaches to this task, by employing different methods from the field of geometrical deep learning. In the field of geometrical deep learning, the non-euclidean data relations are used to train models that can learn from these relations between data points [5]. For this purpose, we developed a modified node pooling layer, based on Edge Pool [8], called Cluster Pool. We use both Edge Pool and Cluster Pool to create a modified Graph U-NET structure by Gao and Ji [15]. We have observed that our modification of Graph U-NET yields better F_1 -scores for node predictions on our data set than the original method by Gao and Ji [15]. The structure of the resulting trees is more suitable for manual error correction, as removing redundant parts takes less effort for the expert than adding in missed segments. Furthermore, the structure contains less irrelevant data, allowing for easier interpretation of the data.

With our geometrical deep learning approach to coronary artery extraction, we make the following contributions:

- We introduce Graph Neural Networks to the work by Wolterink et al. [49] to create a new approach to a fully automated process [35], aimed to improve performance in terms of sensitivity and overlap.
- We created a new node pooling layer, Cluster Pool, based on Edge Pool [8] that allows the pooling of nodes using graph component analysis.
- We contribute a modified Graph U-NET [15] structure, called Graph U-NET II, with our new layer making it more efficient at creating node representations for our node classification task.
- We created an approach that works on a local level (node) to determine segments in the artery tree. These segments are difficult to determine due to mistakes made by the tracker, such as missing bifurcations or tracking beyond the end of an artery.

In our experiments, we have found that Graph U-NET II achieves a sensitivity of 86.0 % and an overlap of 87.0 % on the test set compared to the reference trees. We show that this is competitive with state-of-the-art approaches and presents a substantial improvement in terms of overlap.

For this thesis, the new operators were developed in Python. The source code is available at the Github Repository (<https://github.com/thijssnelleman/CoronaryArteryDetection>).

2 Background and Approach

Cardiovascular diseases are the leading cause of death globally, causing nearly 18 million deaths each year [22]. Among these diseases is ischemic heart disease, which causes the heart muscles' effectivity to be reduced as the coronary arteries supply the muscles with a lessened amount of blood, due to coronary stenosis. Coronary stenosis, i.e. narrowing of the coronary arteries due to the formation of an atherosclerotic plaque [26], presents a serious risk to the patient's wellbeing. At the stenosed coronary artery, there is a risk of full occlusion of the artery. Heavily stenosed or occluded arteries may lead to myocardial infarction. Myocardial infarction is a condition that may lead to acute death or heart failure [46]. Ischemic heart disease was the number one cause of death in 2019, accounting for 13 % of deaths worldwide [21] [47].

In order to diagnose patients with these anomalies, experts typically use Coronary Computed Tomography Angiography (CCTA) [38]. These scans use a contrast agent, injected into the patient's blood to make the coronary arteries visible during the scanning procedure. The scan produces an analysis of the patient's heart, by creating a three-dimensional image. This procedure is timed with the patient's heartbeat and breath, as this will otherwise cause motion artifacts in the image. Both of these factors can introduce noise, causing the resulting data to be subject to loss of quality in the representation of the patient's cardiovascular structure. Thus a robust approach is required to deal with these challenges.

Using this information, experts attempt to determine the cause of a patient's symptoms and determine a strategy to reduce the chance of occlusion of the coronary artery, to avoid myocardial infarction and other possible complications. One solution, for example, is to place a coronary stent during a percutaneous coronary intervention procedure. The stent is a small metallic tube that aims to counteract stenosis by increasing blood flow in the artery. To determine whether a patient must undergo coronary intervention, the expert analyses the CCTA scan to estimate the extent and location of the coronary disease. This is a time-consuming and costly process that reduces the amount of patients a trained clinician can assist. The number of experts able to conduct such an analysis is limited and the quality of the analysis is paramount to the patient's wellbeing.

One notable possibility to increase both the throughput of experts as well as the quality of analysis is to enable machine learning to assist them in their objectives. The first step toward automating this process is to extract the coronary arteries from the 3D CCTA scan. This step was approached in the work of Wolterink et al. [49]. When the coronary artery centerlines are extracted, the resulting structure can then be easily represented in a three-dimensional structure to an expert for analysis, thus avoiding the cumbersome search in CCTA scans for the cardiovascular system. Furthermore, this structure can also be automatically analyzed for anomalies which can then be presented to an expert for verification [45] [25].

2.1 Data

In this work we use data extracted from CCTA scans acquired by several scanners from various companies. The general construct of acquiring these images is the same. Each patient is injected with an iodine-based contrast agent in their bloodstream. This material is then exposed

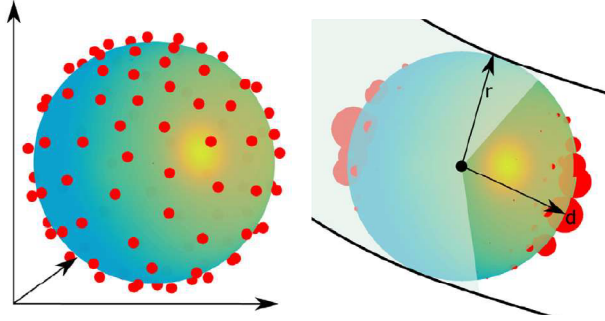


Figure 1: Representation of the directional sphere used by Wolterink et al. [49]. On the left, the distribution of directions is represented on a sphere. On the right, the distribution over the directions is visualized. The darker section of the sphere represents the minimum requirement of the 60° angle in order to be considered for the next step of the direction, to avoid the tracker from going backwards. Image from Wolterink et al. [49].

to x-rays in the scanner, which causes a visual reaction. Afterward, a three-dimensional representation is reconstructed. The presence of the iodine illuminates the blood, which allows for visual identification of the patient’s coronary arteries in the CCTA scans. In our work, the first step toward automating this identification process is to extract the scan sections representing any kind of blood vessel.

2.2 Coronary Centerline Extraction

Wolterink et al. [49] describe a deep convolutional neural network model, trained towards detecting and extracting patches of CT scans deemed relevant using the lumens’ intensity of the contrast fluid. The task presented to the CNN was as follows: Given a $19 \times 19 \times 19$ voxel input patch, determine the orientation and the radius of a coronary artery at the given patch. Wolterink et al. [49] argue that although the increase of input size would yield more information for the CNN to base its decisions on, this has a counterproductive effect due to the increase of irrelevant input which makes the learning of descriptive representations harder. By keeping the information more local, the model is thus able to generalize more to the task at hand, instead of being misled by irrelevant data.

The possible directions of the artery are presented as a classification problem, rather than a regression over angles, to allow for multiple directions to be yielded. A regression in multiple angles would be problematic, as centerline points vary in the amount of directions to be regressed. To transform this problem from regression into classification, a subset of all possible directions is represented as classes distributed over a sphere. The direction with the highest value in the probability distribution over the sphere is selected as a continuation for the tracker. To avoid the tracker moving backward through the blood vessel, only directions with an angle less than 60° are considered. The directional sphere is visualized in Figure 1. The last output node yields a regression value, which represents the vessel’s radius at the given point.

To ensure generalization to the task as well as robustness in the model, Wolterink et al. [49] explicitly produce translated training samples, where the authors adjusted the coordinates outside of the labeled reference trees, as to demonstrate negative samples. This is to give the model the opportunity to see a situation where it is “off track” from the arteries. An example

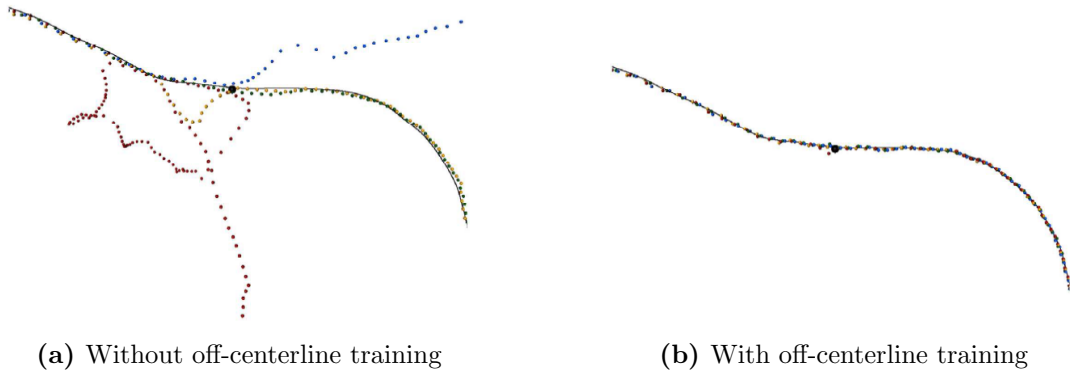


Figure 2: Two results of centerline extractions. On the left, we see the result of a model that has not been trained with off-sample examples. On the right, is the result of a model that has been supplied with randomly translated samples. Image from Wolterink et al. [49].

demonstrating the necessity of these translated samples can be seen in Figure 2.

The CNN tracker requires seed points, or starting points, in the CCTA scan to initialize the tracking. In order to fully automate this process, the paper suggests using another CNN model with a likewise architecture that extracts seed points from the CCTA scan. Although the paper ranks third in publicly evaluated methods, only being outperformed by Friman et al. [14] and Schaap et al. [36] in terms of overlap and accuracy in the MICCAI Coronary Artery Challenge (CAT08), this can be a completely independent and automatic process, whereas the two previously mentioned papers require both start and end points of coronary arteries to be indicated manually. This makes it an important stepping stone toward completely automatic extraction of the complete coronary artery tree from CCTA data using convolutional neural networks.

Wolterink et al. [49] present their results with leave-one-patient-out cross-validation. Per patient, they have used experts' opinions on the image quality as well as the calcium score to outline two impactful biases to the quality of the model. The calcium score indicates the medical condition of the patients and the increase will result in less visible or harder to track arteries. The calcium causes plaque on the artery wall, thus narrowing the vessel and decreasing blood flow and a lower lumens intensity. The results are evaluated in four metrics, of which total overlap with the reference data is most relevant. The results on the CAT08 test-set show quite varying results, although there is a correlation between the image quality and/or calcium score and the resulting metrics. The CNN model achieves on average a total overlap (OV) of 93.7 %. This indicates that the model makes precise predictions, e.g. generates few false positives. The clinically relevant overlap (OT) is higher, yielding 97.0 %. The metrics are fully defined by Schaap et al. [35].

They trained and verified the quality of their work on various CCTA data sets, both public and private, such as the CAT08 from Schaap et al. [35] and an internal data set, consisting of 50 CCTA scans acquired at the University Medical Center of Utrecht. In the latter data set, an expert manually indicated for each scan markers along the visible main arteries and side branches of coronary arteries, separated by approximately 10mm. The markers include a measurement of the radius of the vessel which varied from 0.45mm to 3.51mm.

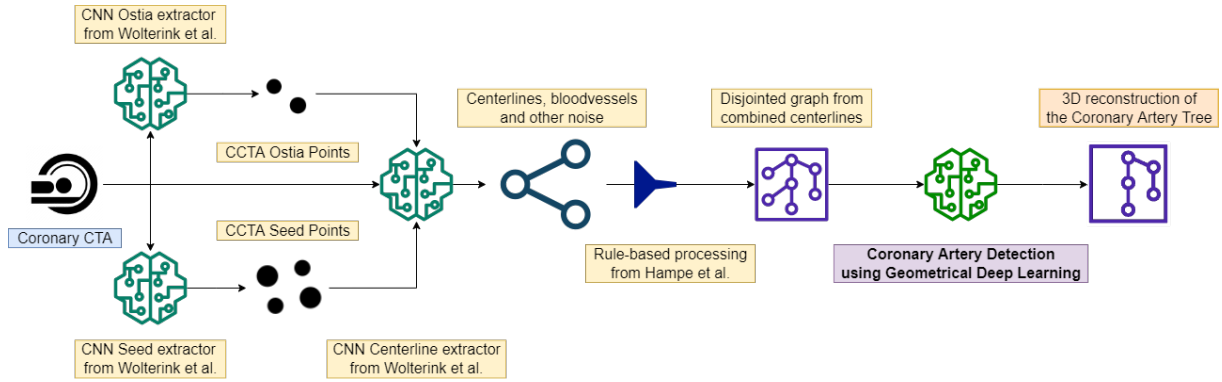


Figure 3: A flow chart of the proposed solution to the problem. The CCTA is used as input for three separate CNNs, the outputs of two are used as input for the third [49]. Then, the produced centerlines are processed using a set of rules [17]. This serves as the input for our model, which will detect which piece of data is part of the cardiovascular system, thus producing the reconstruction of the coronary artery tree.

2.3 Coronary Artery Centerline Tree Construction

The data yielded by the CNN model from Wolterink et al. [49] contains a vast proportion of noise, as per design. In general, about 87 % of extracted centerline points are not related to the coronary arteries. This is due to two main factors. First of all, the model sensitivity for our data set has been increased to minimize the loss of coronary artery data. In contrast to the work of Wolterink et al. [49], we use 400 seed points to start tracking instead of 200. Furthermore, seed points have an increased minimum distance of 3mm instead of 1mm, to reduce the number of redundant seed points. This effectively also leads to more seed points. This is to increase the sensitivity of the output when compared to the expert reference labeling; By increasing the number of relevant seed points, we look to extract as much of the reference tree as possible, at the cost of extracting a substantial amount of false negative centerline points in the process. Thus a certain amount of present noise can be interpreted as inevitable.

Secondly, the model has no understanding of the geometrical graph structure that the arteries represent, which leads to the absence of global features of the arteries, such as orientation, geometry, and their positioning in the tree. Arteries always form a graph tree structure, where blood will flow from the aorta around the heart. We can use the centerline points to reconstruct these, where each point will be presented as a node, and the edges indicate whether two patches are in a sequence in the same vessel. Since the method by Wolterink et al. [49] already extracts entire centerlines, it is rather straightforward to create a graph component from a centerline. In order to combine centerlines that are connected through a bifurcation, the overlap of the two centerlines was measured and combined to form one graph, where each node is unique [17].

2.4 Proposed Solution

In Figure 3 the full pipeline of our method is shown. First, three CNNs from Wolterink et al. [49] are used to generate the centerlines. Afterward, they are combined into a, possibly disjointed, graph [17]. Then the work from this thesis is used to filter out any noise, to create the graph representing the coronary arteries. This pipeline turns the work from Wolterink et al. [49] into

a new fully automated method, requiring no human intervention, that aims to reduce the number of false positives.

The data set that was extracted for our method, was acquired by executing the pipeline from Figure 3 up to the "Disjointed graph from combined centerlines" step. These graphs are compared against the reference trees annotated by experts, using the definitions from Schaap et al. [35], to determine the ground truth labels for the training of models in this thesis. Nodes, or centerline points, that are within the artery radius of a node in the reference tree are marked as true, otherwise false. We recorded a sensitivity of 91.2 % on the reference tree and an overlap score of 95.4 % with the reference tree on the acquired data set. Since our method does not take in any information except for this data set, as we can see in Figure 3 for example the CCTA scan does not serve as an input to our model, the sensitivity score will serve as an upper limit for our model. This is because we do not acquire any new data points in this step, but only filter false positives from the tree.

3 Related Work

First, we will look into other works that aim to solve similar problems with machine learning on coronary artery data. Secondly, we will review works within the field of geometrical deep learning, to explore applications of graph-structured data. Finally, we will look into the field of hyperparameter optimization that automatically optimizes hyperparameters and architectures of models and algorithms.

3.1 Structural Anatomy Classification

In this section, we will review two papers related to coronary artery extraction and classification. First, we will look at the work of Salahuddin et al. [34], where the authors present the state-of-the-art methods for automatic coronary centerline extraction. Secondly, we will review the paper by Hampe et al. [17], where they present graph attention networks (GAT) for coronary artery classification.

3.1.1 3D Convolutional Neural Networks for Automatic Coronary Centerline Extraction

Salahuddin et al. [34] proposed **AuCoTrack**, a three-phased pipeline to automatically extract the coronary arteries from CCTA scans. The three phases, or modules, are comprised of the following steps: First, the authors employ two 3D convolutional neural networks (CNN), named Direction and Bifurcation Classification Network (DBC-Net) and Stop Patch Classification Network (STC-Net), to extract the patches in the CCTA that contain parts of the coronary arteries. This is used for the third phase, called tracker, to extract the centerlines of the arteries. The tracker is initialized at the two Ostia of the aorta, which are also obtained automatically, thus creating a fully automated extraction method. This thesis most closely relates to the STC-Net, since in our work we aim to determine whether a patch is deemed to be containing the coronary artery. The approach in this thesis is distinctive from theirs due to the usage of geometrical deep learning, since we aim to leverage the geometrical structure of the arteries, whereas they apply another CNN to a local patch using only local features. Later on in this thesis, we will compare our results to theirs.

The first question that comes to mind when reviewing their phases is why the authors decided to create two separate tracking CNNs: One for bifurcation and direction prediction (DBC-NET), and one to determine whether the patch is part of the arteries at all (STC-NET). The input to both networks is the same (image patch), yet one outputs two values, the direction classification and bifurcation probability, and the other only the artery probability. This could be due to the labeling of centerline points that do not belong to the coronary arteries, as it could be problematic that they do not have any direction nor bifurcation labels.

Another question of interest is that of automatic Ostia detection, as it is the main contributing factor by Salahuddin et al. [34] to differentiate between a semi-automatic and automatic method [35]. Salahuddin et al. [34] refer to this as “Model-Based Segmentation” method described by Ecabert et al. [11]. Ecabert et al. [11] propose a method to automatically fully

segment the heart, consisting of the four chambers, the myocardium, and the great vessels [31]. In our work, based on Wolterink et al. [49], we apply separate CNNs to indicate possible seed points for the network, as shown in Figure 3. They aim to only extract two seed points (Ostia), to track and detect bifurcations from there, whereas our method does not detect bifurcations but instead looks for multiple seeds and merges the structure into a tree afterward. Our method has a clear advantage with the increased amount of seed points in terms of sensitivity but does yield a significant amount of false positives thus decreasing the overlap with the reference tree.

The STC-Net from Salahuddin et al. [34] is aimed to detect the end of an artery, which can be fairly ambiguous [34]. The ambiguity does not only spring from inter-rater variability, as experts may judge differently from one another, but also local centerline points may suffer different degrees of calcification in patient’s arteries and thus feign the end of an artery. Where our work goes beyond theirs is the impact of severely calcified or stenosed arteries on the tracking method. A heavily calcified or stenosed artery may feign the end of an artery. To overcome this, we propose in Section 2.3 to substantially increase the number of seed points and that can yield a disjointed graph. A disjointed segment that is still considered part of the coronary arteries can not be detected in the work of Salahuddin et al. [34], while possible with our method.

3.1.2 Graph Attention Networks for Segment Labeling in Coronary Artery Trees

Hampe et al. [17] presented a Graph Attention Network to determine the anatomical labels of the coronary arteries [31]. They use the semi-automatically extracted trees from CCTA scans based on the work of Wolterink et al. [49]. The derived graph is reduced in terms of nodes by aggregating linear node segments in the tree to become one node. This leads to a simplified representation of the tree structure. This makes sense since the labeling task between bifurcations can not differ due to the definition in the anatomical reference standard [31]. Finally, the graph is translated into a line graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ s.t. the edges \mathcal{E}' in the original graph became the new nodes \mathcal{V} to create a node labeling task.

A graph neural network is applied for the classification task, using a special type of graph convolution called Graph Attention Networks [44]. This network employs attention layers for nodes, to determine given a node \mathcal{V}_i , the relevance of a node \mathcal{V}_j ’s features toward node \mathcal{V}_i if \mathcal{V}_i and \mathcal{V}_j are in each others neighborhood. They achieve an average F_1 -score of 92.4 % over all segments and patients. Although the task is significantly different from this work, binary classification versus multi-class for example, the data, and its relations are the same but the labels are different. The authors show that the anatomical structure can be leveraged by a graph neural network to represent the internal relations of the coronary arteries, which is the shared objective between our work and theirs.

3.2 Geometrical Deep Learning

In order to leverage the graph structure to create a representation of a node and its neighborhood, we will use Graph Convolutions [6] and have seen the application of Graph Attention

Layers [44] in this domain [17]. In this section, we explore related methods that use the tree-like graph structure our data set represents.

3.2.1 Directed Acyclic Graph Neural Networks

Thost and Chen [43] presents a Message Passing layer for directed acyclic graphs. Message passing in Graph Neural Networks entails the concept of updating a node’s features n_i based on its neighborhood’s features. Thost and Chen [43] suggests an alternative method to pass this information by leveraging the partial order that directed acyclic graphs represent compared to regular graphs, thus making not all neighbors equal.

In regular message passing, all nodes are processed in parallel, e.g. the graph representation at \mathcal{G}_i is used to create the next graph iteration \mathcal{G}_{i+1} for every node in the graph. This means that each node uses the nodes from the previous iteration to create the next. In directed acyclic graph neural networks (DAGNN), the authors use the partial order of the graph to propagate information through the graph. This partial order yields a sequence of updates so that the impact of each node’s neighborhood on the node itself can be determined using the updated neighborhood in the graph \mathcal{G}_{i+1} instead of \mathcal{G}_i . This brings the advantage that a node’s representation is updated with the most recent information. The sequential manner of their work is both an advantage and a disadvantage, whereas the computational speed is at a disadvantage. The approach does not allow for complete parallel computation, as we cannot calculate the features of a current node and a succeeding node at the same time.

Coronary arteries create an acyclic graph where a direction can be synthesized: The flow of blood with Ostia as roots is the most straightforward interpretation. However, it is not a completely logical interpretation when reviewing it as a node’s neighborhood, as its predecessors describe a node’s locality just as much as its successors. This creates a bi-directional interpretation of the graph. To make use of this, the authors suggest alternating the directed graph representation \mathcal{G} to an inverted graph $\tilde{\mathcal{G}}$ where each edge has been reversed. This work could be relevant to ours due to this shared underlying graph and the possibility of alternating the direction of the graph. The depth of our graphs could reduce the propagation of information between roots and leaves, which could have a possible negative impact as this could produce less informative node representations. Furthermore, the gaps in our graph produced by the tracker can severely prohibit the propagation of information, as this yields many small segments with their own roots.

3.2.2 Edge Contraction Pooling for Graph Neural Networks

Diehl [8] presents a pooling layer to pool nodes by singular edge relations. This was originally proposed in Diehl et al. [9]. It aims to enable GNNs to generalize over simplified graph representations by reducing certain node clusters to singular nodes, to improve performance on classification tasks using graph or node representations. The general idea behind this is that graphs can contain closely related node clusters that yield a better graph representation when combined.

To determine which edges in a graph should be contracted, they calculate an edge score based

on the features of the two participating nodes. The highest scoring edges are contracted until half of the nodes have been clustered. Each cluster can only contain two nodes at maximum. When nodes are clustered, the cluster features are calculated as an addition of the nodes' features with the edge score as a factor. Later on, when nodes are unpooled, this factor is reverted. The operator can be injected into existing frameworks. The authors show their results in comparison of with and without this operator and show a significant increase on the Cora [29], CiteSeer [40] and other benchmarks for five different architectures.

Since our graphs have many node sequences that can be reduced for better interpretation, the method by Diehl [8] is very applicable to this work. Not only does it yield better graph and node representations, but it also allows to revert a pooling step, later on, thus making it available for node classification tasks. The learned grouping, or clustering, of nodes, is particularly sensible for our task, as the node sequences have hidden segments produced by the tracker, such as a wrongly tracked artery or tracking beyond the end of an artery. Later on, we will explore the limitation of the two-node maximum for clusters.

3.3 Hyperparameter Optimization

Hyperparameters control the learning procedure and the architecture of the model. A human/expert-based approach, or manual search, to discovering the right hyperparameters for a method tends to have some form of grid search which has been shown to perform worse than random search [3]. The field of automated hyperparameter optimization aims to reduce the workload of experts and yield better algorithm configurations.

Differential Evolution Hyperband Awad et al. [1] present Differential Evolution Hyperband, an HPO method. DEHB is a successor to Bayesian Optimization Hyperband [12]. Awad et al. [1] show that DEHB achieves a stronger and more robust final performance, especially for more high-dimensional problems. The latter is important, as the hyperparameters of state-of-the-art algorithms are many, creating high-dimensional search spaces.

The method is built of two parts: Differential Evolution, a population-based evolutionary algorithm, and Hyperband, a successive halving strategy that distributes its bets over the population with increasingly higher budgets. The Differential Evolution generates a population of hyperparameter configurations and uses mutation, crossover, and selection operators to create new generations. To produce a new generation, for each specimen a mutation with a difference vector is generated with three random parents. Selection is based on an objective function and performed by successive halving. An important modification the authors made is that the mutation step does not select from the population but a separate parent pool.

In the work of Awad et al. [1], the authors show the performance of the algorithm on Reinforcement Learning methods such as Proximal Policy Optimization [39] on the Cart Pole environment as well as on Neural Architecture Search benchmarks [50] and Hyperparameter Optimization benchmarks.

This method not only offers more robust results than its predecessor but also a great speedup in wall-clock time. It is also a model-free optimization method, as no probabilistic modeling

is done to determine the next generation of the population. This reduces latency between generations. As the models we aim to use can be rather expensive due to their number of parameters and complex layers, this method offers an alternative to manual search. Not only can it reduce the workload of the expert, but also better hyperparameters. The exploration of the hyperparameter space conducted by the algorithm can then be used as an argument of its relative quality: The more attempts the method has had to review the search space the better the argument is that the found hyperparameters are near optimal.

4 Methods

In this section, we will describe the methods and algorithms we will use to train our models. In Section 4.1, we will look into the training techniques in general for the models, such as the loss function and regularization methods. In Section 4.2 we look into two methods that will serve as graph agnostic baseline methods. In Section 4.3 we describe various operators and introduce new operators from the geometrical deep learning domain. In Section 4.4 we describe the architectures of our models based on the operators described in Section 4.3.

4.1 Method Training Techniques

In this section, we will describe training techniques for our models. Section 4.1.1 describes the loss function and optimizer for our neural networks. Section 4.1.2 describes the regularization techniques we will use to prevent overfitting on the training set for our neural networks. In Section 4.1.3 we describe our approach to determine the optimal threshold for a model.

4.1.1 Loss and Optimization

Loss functions evaluate how well an algorithm performs on a given data set. In this thesis, we measure binary cross-entropy loss [51] to calculate the loss between the prediction and the ground truth distribution. The formula is shown in Equation 1.

$$L_{BCE} = -\frac{1}{N} \cdot \sum_{n=1}^N \left[y_n \cdot \log \hat{y}_n + (1 - \hat{y}_n) \cdot \log(1 - \hat{y}_n) \right] \quad (1)$$

In Equation 1 y_n denotes the class label for item n , \hat{y}_n the predicted label for item n . To reduce the impact of the extreme class imbalance, we have weighted the loss in favor of the positive class with a factor 4 which was selected through manual optimization from the range of $[1, 8]$ determined from Table 3 based on the class imbalance.

For all neural networks, we employ the Adam optimizer from Kingma and Ba [23], selected for its known speed, general quality, and robustness. The Adam optimizer is an extension of stochastic gradient descent that combines the advantages of AdaGrad [10] and RMSProp [33].

4.1.2 Dropout and Weight Decay

To prevent overfitting and thus improve the generalization of our models, we use dropout and a form of L2 regularization, also known as weight decay [24] [42]. Dropout refers to ignoring, or “dropping out”, certain neurons in a network with a probability p . This is to reduce the inter-dependence of neurons in a network, to counter overfitting. In weight decay, a penalty is calculated for the weights and added to the loss function. This is to penalize the size of the weights in a network and is calculated as $W_d = \lambda \cdot \sum_i^n w_i^2$, where λ is the factor determining the impact of the weights on the loss.

4.1.3 Precision-Recall Curve

It is important to determine the right threshold to use for our model's output probabilities, as this can have a major impact on the performance of our model. The precision-recall curve describes, given a ground truth and a set of corresponding probabilities, the impact of shifting a threshold on the precision and recall metric. When visualized, it illustrates the trade-off between these two metrics. We calculate the precision-recall curve over the positive class since we optimize on the positive class F_1 -score. We then calculate the F_1 -score of every precision and recall combination as Equation 5 and determine the highest F_1 -score. The threshold corresponding to this F_1 -score is then selected as the optimal threshold. We determine this value on the validation set, and not on the training set, to avoid overfitting.

4.2 Baselines

As baseline methods for our task, we will use graph-agnostic methods, e.g. methods that only take in single nodes and their labels to train. This is to show the effectiveness of the nodes' features and demonstrate the impact of the use of the edges of the graph later on. We have selected two baseline methods, XGBoost for its widely known capabilities and Multi-layer Perceptron as a neural network baseline. The motivation for these baselines are described in detail in their corresponding sections.

4.2.1 XGBoost Classifier

In 2016 XGBoost was presented by Chen and Guestrin [7]. The XGBoost classifier is a gradient boosted ensemble tree method, that creates multiple decision trees during training time, and the final prediction is the sum of probabilities over all the trees. The decision trees are evaluated such that a new tree in the forest reduces the loss when combined with the previous tree. Thus, the method is able to learn from its previous mistakes. The loss between these steps is reduced by using gradient boosting [13]. To improve upon this existing method, Chen and Guestrin [7] present Extreme Gradient Boosting, which employs more advanced techniques to allow the model to fit better to the data, such as second partial derivatives and regularization.

We have opted to use this method because of its widely known capabilities in many practical problems [2], including the competition website Kaggle, where in 2015 out of 29 winning solutions 17 of these used XGBoost [7]. Furthermore, its short training time and few hyper-parameters to optimize make it an easily applicable method.

4.2.2 Multilayer Perceptron

The multilayer perceptron (MLP), also known as a fully connected neural network, is the standard and most widely known type of artificial neural networks [27]. MLPs are fully connected neural networks with one or more hidden layers and use a non-linear activation function in their neurons. MLPs are able to model non-linearly separable data. We use MLPs as a baseline to show the impact of the more complex neural networks later on.

Activation Functions As our activation functions in the hidden layers of any neural network we selected the Rectified Linear Unit, or ReLU [28] which is defined as $f(x) = \max(0, x)$. This activation function has been shown to be a very effective activation function in deep learning [37]. ReLU does not cause any gradient problems as its classical counterpart the sigmoid activation function when $x > 0$. The sigmoid activation function runs into disappearing gradients as the function saturates at zero and one for high negative and positive values, respectively. The ReLU activation function is also very computationally efficient, as it takes only one comparison to calculate the result.

The sigmoid activation function is used in our model as the output layer's activation function, as we use a single output node for binary classification [30]. The sigmoid activation function is defined as $f(x) = \frac{1}{1 + e^{-x}}$.

4.3 Graph Neural Network operators

In graph neural networks, the aim is to leverage the graph structure to better represent the nodes or the graph itself. We make use of a message-passing framework to do so. In short, this means that a node's representation is an aggregation of the combination of its neighborhood's features. When applied sequentially it allows the features of one node to flow into and beyond its own neighborhood, thus passing on its information around the graph. Note that a node's neighborhood is unordered: It is thus important to define a permutation invariant aggregation function when combining a node's features with its neighborhood.

In this section, we will first describe graph convolutions. Then we will describe two pooling operators that allow the reduction of the size of the graph. Graph convolutions enable the network to have a receptive field. Pooling operators aim to reduce the size of the graph, thus making the receptive field much larger. Finally, we will describe three architectures that combine these methods together to create five types of models.

4.3.1 Convolutional Layers

We use the graph convolutional layers from Bruna et al. [6] to share the node information with its neighborhood. The graph convolutions by Bruna et al. [6] are a special case in the message-passing framework, namely, graph attention networks [4]. In message-passing, nodes send "messages" containing their features to their neighbors. Then, each node combines its features with the information received from its neighbors through a permutation invariant aggregation function. The messages of the connected nodes are then fed to a linear layer to create a learned representation of a node's neighborhood.

Each convolutional layer in a network thus allows node information to travel "one hop": Each node shares its features with its neighborhood, and if applied sequentially, each step in the sequence allows the information to travel one neighborhood further. However, in our case, we have a very sparse adjacency matrix, as shown in Table 3, thus allowing information in the graph to travel to many neighborhoods beyond its own would require many convolutional layers. For example, the maximum segment length, a linear sequence of nodes, in Table 3 is

192 nodes. This means that, if the first and last node in this segment were to directly exchange information through convolutions, we would require 192 convolutional layers.

4.3.2 Graph Pooling Operators

Node pooling operators allow graph neural networks to reduce the number of nodes in the graph to create a more global representation of the graph. In our work, we selected the Edge Pooling operator because it learns which nodes to pool together based on the node features. Furthermore, it has a reverting pooling operator, which is a soft requirement when dealing with our node labeling task.

In theory, we could proceed with the task on a reduced graph, however, the more the variance of labels within a cluster, the less well we are able to perform the task in these clusters. By unpooling the graph, and returning to the original structure of the input, the nodes are separated and can be individually classified by the model, solving this limitation. An illustration of this can be seen in the appendix, in Figure 22.

Edge Pooling To determine which nodes are to be pooled together in a graph, the Edge Pooling operator assigns a score to each edge using the formula in Equation 2 [8]. Based on this edge score, a selection procedure is executed to determine which edges are to be contracted.

$$s(e_{ij}) = \text{softmax}(\vec{r}_j)_i + 0.5, \quad \vec{r}_j = \parallel_{k \in \mathcal{N}_j} \phi(v_k || v_j) \quad (2)$$

In Equation 2, a score $s(e_{ij})$ is defined over an edge e_{ij} using function $s(e_{ij})$. We create a vector \vec{r}_j for node j by concatenating the output of network ϕ for all other incoming edges $k \in \mathcal{N}_j$ in the neighbourhood \mathcal{N}_j of node j [4]. In Equation 2 ϕ is a learned function that defines a score given the concatenated input features of two nodes $v_k || v_j$. The softmax function is applied to vector \vec{r}_j and the edge corresponding to node i is selected. Thus we have applied a node-local softmax for edge e_{ij} based on node j . Finally, 0.5 is added to all scores to avoid a division by zero issue later on. Furthermore, the authors state that this addition helps with the backpropagation in their work [8].

Using the input graph as $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, we calculate the edge score $s(e_{ij})$ for every edge e_{ij} in \mathcal{E} . The aim here is to, based on these scores, reduce the number of nodes by 50 % by placing them in clusters together. This results in a fixed proportion, in contrast to other pooling operators such as Top-k pooling [15]. As we give preference to higher scoring edges, the scores are first sorted from high to low. Then, we loop through the scores. An edge is contracted if neither of the participating nodes is already in a cluster and the number of nodes pooled is less than half. This is an important rule in Edge Pool: No cluster can contain more than two nodes in a layer. We then pool the clusters into nodes by using Equation 3.

$$CF(e_{ij}) = s(e_{ij}) \cdot (v_i + v_j), \quad v_i = v_j = CF(e_{ij})/s(e_{ij}) \quad (3)$$

In Equation 3, we define the calculation of a cluster's features $CF(e_{ij})$ over an edge e_{ij} , as each cluster can only contain two nodes. The new node's features, which are formed from the

cluster, are defined by adding the node feature vectors v_i and v_j together element-wise and multiplying them by their edge score $s(e_{ij})$. To revert this clustering step, or unpooling, we define in Equation 3 a second equation. There, we state that to recreate node features of v_i and v_j , we take the new node's features $CF(e_{ij})$ and divide them by the original edge score $s(e_{ij})$. Note that this operator makes the two nodes v_i, v_j indistinguishable from one another. Furthermore, it is this division operator that must refrain from using any edge score equal to zero.

In the construct Edge Pool, each cluster of nodes can only contain 2 nodes. This limitation may not be a logical choice for our problem, since the segments we wish to detect in our graphs may be longer than two nodes. Thus we developed a modification to this pooling operator, called Cluster Pool which is one of the contributions of this thesis.

Cluster Pooling The cluster pooling method eliminates the premise of edge pooling where an edge can only be contracted if both of the participating nodes are not already in a cluster. This is desirable for our task, as we aim to pool longer segments of the graph together that are strongly related. This leads to a few alterations in the process.

We calculate the edge scores as before using Equation 2. We select the top quantile of q edges $\mathcal{E}' \subset \mathcal{E}$ that corresponds to half the nodes $|\mathcal{E}'| = |\mathcal{V}|/2$. These selected edges create a new subgraph $\mathcal{G}' = \{\mathcal{V}, \mathcal{E}'\}$. Since this new subgraph only has the edges we want to contract, we can detect the graph components to determine the clusters [19], denoted here as Graph Component Detection or GCD for short. This algorithm's complexity is linear in the number of nodes, $\mathcal{O}(|\mathcal{V}|)$. We calculate the components c of the new subgraph \mathcal{G}' using this algorithm, where clusters are defined as a set of edges $\mathcal{E}'_c \subset \mathcal{E}'$.

To calculate the cluster's features, we make an adaption for multiple edges in Equation 4.

$$CF(\mathcal{E}'_c) = \sum_{e_{ij} \in \mathcal{E}'_c} s(e_{ij}) \cdot (v_i + v_j) \quad (4)$$

In Equation 4, we define the features of a cluster \mathcal{E}'_c as a summation over the edges in the subset $e_{ij} \in \mathcal{E}'_c$. For each edge, we multiply its edge score with the element-wise addition node features v_i, v_j . This construct is based upon the fact that, if each cluster \mathcal{E}'_c consists of only one edge, the edge pool operator is executed. The entire process is summarized in Algorithm 1. The unpooling operator in the cluster pooling operator is simplified in comparison to edge pool, as we assign each participating node the cluster's features as $v_i = v_j = CF(\mathcal{E}'_c)$.

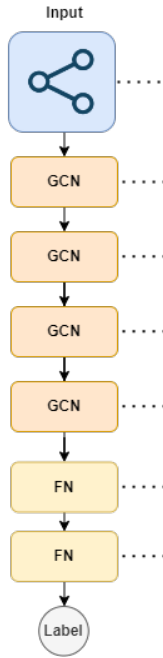
4.4 Graph Neural Network Architectures

In this section, we will discuss the various proposed architectures that we have constructed using the techniques described in Section 4.3.

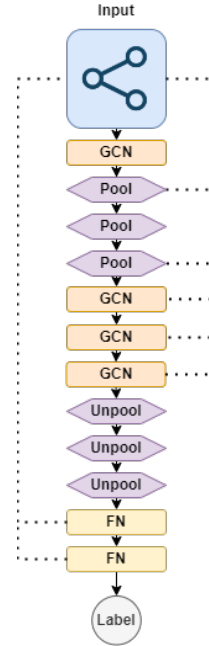
Algorithm 1 Cluster Pooling Algorithm

Require: \mathcal{V} : set of Nodes, \mathcal{E} : set of Edges, s : function of Equation 2

- 1: $S = \{s(e) : \forall e \in \mathcal{E}\}$
 - 2: $q = \text{topQuantile}(S, |\mathcal{V}|/2)$
 - 3: $\mathcal{E}' = \{e : \forall e \in \mathcal{E}, s(e) > q\}$
 - 4: $Clusters = GCD(\mathcal{V}, \mathcal{E}')$ ▷ Graph Component Detection [19]
 - 5: $\mathcal{V}_{clusters} = \{CF(\mathcal{E}'_c) : \forall \mathcal{E}'_c \in Clusters\}$ ▷ Eq. 4
 - 6: $\mathcal{V}' = \{v : \forall v \in \mathcal{V}, v \notin Clusters\}$ ▷ Update the graph nodes
 - 7: $\mathcal{V}_{out} = \mathcal{V}' \cup \mathcal{V}_{clusters}$
 - 8: $\mathcal{E}_{out} = \mathcal{E} \setminus \mathcal{E}'$
 - 9: return $\mathcal{V}_{out}, \mathcal{E}_{out}$
-



(a) An illustration of the Graph Convolutional Neural Network architecture. Here, GCN denotes a graph convolutional layer, and FN denotes a fully connected layer. Skip connections are shown with dotted lines.



(b) An illustration of the Graph Convolutional Pooling Neural Network. GCN denotes graph convolutional layer, Pool denotes either an Edge Pool or Cluster Pool layer, Unpool denotes the reversal of a pooling operator and FN denotes a fully connected layer. Skip connections are shown with dotted lines.

Figure 4: Two of the architectures constructed from our operators. In Figure 4a we see an architecture with convolutional layers and fully connected layers. In Figure 4b we see an architecture employing pooling operators as well. Both architectures include skip connections with dotted lines.

4.4.1 Graph Convolutional Neural Networks

We use the graph convolutional layers to create a Graph Convolutional Neural Network for our experiments. The architecture consists of four convolutional layers followed by two fully connected layers. This creates a neighborhood of four hops, entailing that each node can gather information from up to four hops away. Furthermore, we also created skip connections to concatenate each layer's output to the input. The architecture is illustrated in Figure 4a.

4.4.2 Graph Convolutional Pooling Networks

We use the graph convolutional layers and the pooling operators to create Graph Convolutional Pooling neural networks for our experiments. Using these layers, we created an architecture consisting of a convolutional layer, followed by three pooling layers, followed by three convolutional layers. Afterward, the three pooling steps are reversed and two fully connected layers are applied. There is a skip connection from the input two the fully connected layer, as well as two skip connections from the third pooling layer to the third and fourth convolutional layer. The architecture is illustrated in Figure 4b.

4.4.3 Graph U-NET II

Gao and Ji [15] present a geometric deep learning variant of U-NET [32]. U-NETs have shown to be a very effective architecture for medical imaging data [41]. U-NETs employ an encoder-decoder architecture, where the data is first downsampled to a more global structure, and afterward upsampled to allow the global structure to flow into the local structure. A distinctive element of U-NET's are the skip connections: After each up-sampling step, the original information present before the downsampling is concatenated to the output. This yields for each local element both a local and global representation of the element. The advantage of these dual representations caused by the skip connections is that each element now both describes itself and its position in the global structure thus making it easier to distinguish compared to taking only either representation. In the paper by Gao and Ji [15], the authors describe a graph-based variant, that allows up- and downsampling of graphs in terms of nodes and edges. They make use of a modified graph convolution layer inspired by Bruna et al. [6] and a new graph pooling layer, gPool, to represent the operations in the original U-NET. In each layer of the architecture, a convolutional layer is applied followed by a pooling layer. This is the downsampling phase, to create a more global structure of the graph. Afterward, the graph is upsampled by using unpooling operations and convolutional layers in between. The upsampling reconstructs the graph to its original size, and by using the skip connections, each node has both a global and a local representation.

5 Experimental Setup

In this section, we will describe the methods we use to evaluate our models. In Section 5.1 we will discuss the metrics used in our experiments to measure the quality of the models. In Section 5.2 we describe the training and validation setup we use to determine the performance of each method.

5.1 Metrics

We evaluate each model using the F_1 -score on the positive class, due to the high-class imbalance as demonstrated in Table 3. The F_1 -score and its components are calculated using the formulas of Equation 5.

$$F_1 = 2 \cdot \frac{p \cdot r}{p + r}, \quad p = \frac{TP}{TP + FP}, \quad r = \frac{TP}{TP + FN} \quad (5)$$

In Equation 5, p and r are the precision and recall, respectively, and TP , FP , and FN stand for True Positive, False Positive, and False Negative, respectively, entailing the outcome of prediction compared to the ground truth. We also present the precision-recall curve, to demonstrate the impact of the threshold that is used to transform the probabilities into binary classes. An important advantage of the precision-recall curve (PR curve) is that it is not affected by the class imbalance, as neither precision nor recall takes into account the True Negatives which would be over-represented in our case.

Neighbourhood Probability Difference We also measure the difference in probabilities between neighboring nodes. This is to indicate how smooth the predictions change over the nodes in the graph and to demonstrate the impact of the neighborhood on the representation of the nodes. This is calculated over each edge, by $\mathcal{E}_{ij} = |p_i - p_j|$ where p_i and p_j are the probabilities of nodes i, j involved in edge \mathcal{E}_{ij} . We only consider edges that are part of a graph component where at least one node either has a ground truth label as true or one node is predicted as true. These components are considered difficult and are thus important to show the impact of the neighborhood on its representation, as local features will likely not be sufficient on their own. Furthermore, as the negative labels are overrepresented in our data set, these would reduce the variance artificially as many negative labels are relatively easy to detect.

Reference Tree Metrics In order to compare our results against the state of the art, we require methods to compare the output trees from our models to the reference trees. In contrast, the previously discussed metrics are measured in quality on the data set that was generated as output from Wolterink et al. [49], as described in Section 2.4, to measure the performance of the geometrical deep learning methods. These metrics will serve to directly compare against the reference trees, to demonstrate the performance of the entire pipeline. From the work of Schaap et al. [35], we take two well-known metrics in the field of coronary

artery extraction. We use the sensitivity metric to determine how much of the reference artery tree has been extracted. As a second metric, we take overlap, to determine how well the predicted tree and the reference tree overlap. In these metrics we use the following terms: TPM , the true positives in the predicted trees, TPR the true positives in the reference trees, FP the false positives in the predicted trees, and FN , the false negatives in the reference trees.

Sensitivity The sensitivity is calculated over the reference tree as: $\frac{TPR}{TPR + FN}$ [35]

Overlap The measurement overlap between the reference tree and the predicted tree. This differs from the F_1 -score as it takes in both the positives and negatives on the reference trees as well as the predicted trees. This is calculated as $\frac{TPR + TPM}{TPM + TPR + FN + FP}$ [35]

5.2 Training and Validation

k -fold cross validation To validate each method, we use k -fold cross-validation. We partition the training set in k sections, where in k folds each section serves once as the validation set. This validation set is used to validate the model's performance as well as to determine the threshold that leads to an optimal F_1 -score by calculating the precision-recall curve. At the end of each fold, the model parameters are restored to the epoch, e.g. iteration step, with the highest F_1 -score on the validation set. Followingly, this model is tested on the test set thus generating k evaluations on the test set. We present the mean and standard deviation of the scores as the model's performance on the task, to demonstrate the model's average performance and the variation between the folds.

Model Ensembling k -fold cross validation yields ten models and thus ten labels on the test set. However, to enable the application of our method and to evaluate the metrics of Section 5.1 we require one set of labels to compare e.g. one model. Therefore we will use the ensemble of models to produce one classification of the test set. We will consider the sum of votes for both classes, and select the majority vote as the final label. This is because we want to preserve the threshold that has been determined for each model, whereas averaging the probabilities would also require the averaging of thresholds. In case of a tie, we select the positive class, as we are interested in keeping as much of the positive class as possible.

6 Experiments

In this section, we will evaluate the results produced by the methods described in Section 4. We evaluate each method with 10-fold cross-validation. A proprietary data set of 200 patients were made available for this thesis. The data was annotated by experts from the Amsterdam Medical Center but remains private as the data is not owned by the Amsterdam Medical Center. Of these 200 patients, 50 were randomly selected for the test set. The remaining 150 patients are used for cross-validation. Information on the distributions of patients can be found in the appendix, in Table 3. In Section 6.1 and Section 6.2 we compare the results to the ground truth on the output trees from Wolterink et al. [49]. In Section 6.3, we compare these results and evaluate the best output to the original reference trees.

It would be more cumbersome for experts to manually add arteries that are missing in the output trees than to remove them, and as our work looks to relieve experts we will evaluate the performance of our methods using two F_1 -scores. First, we will look at the achieved F_1 -score. Secondly, we will also measure the F_1 -score at 95.0 % recall. This latter score is an indication of clinical relevance for experts: We aim to extract as many of the coronary arteries from the graph, and experts can not make a proper analysis if too few of the arteries remain in our representation resulting in a counterproductive approach. The threshold of 95.0 % recall has been agreed upon by domain experts from the Amsterdam Medical Center.

For each method, the given hyperparameters such as learning rate and dropout values were manually optimized.

6.1 Graph Agnostic Baselines

First, we will evaluate the graph agnostic methods, to analyze the separability of the labels when the graph structure is not taken into consideration by the classifying method hence each node is treated as a separate sample.

6.1.1 XGBoost

We trained the XGBoost classifier with one hundred estimators with a thousand leaves and evaluated the model on the validation set with the area under the PR curve. The results on the test set are depicted in Figure 6.

In Figure 6, the precision-recall curve is shown. The curve is smooth but with a steep decline when approaching 100 % recall. As a result of this, the difference between the achieved F_1 -score of **0.7903** \pm **0.0013** and the F_1 -score at 95.0 % of **0.5630** is substantial, deviating 22.73 %. The distribution of F_1 -score as obtained by 10-fold cross-validation shows a small standard deviation between the folds. This indicates that the F_1 -score presented here shows that there is little variance in the algorithm’s performance, demonstrating the robustness of XGBoost. However, even though the area under the curve is high and the variance between the folds is low, the F_1 -score at 95.0 % recall is relatively low, thus indicating the limited clinical applicability in practice of this approach.

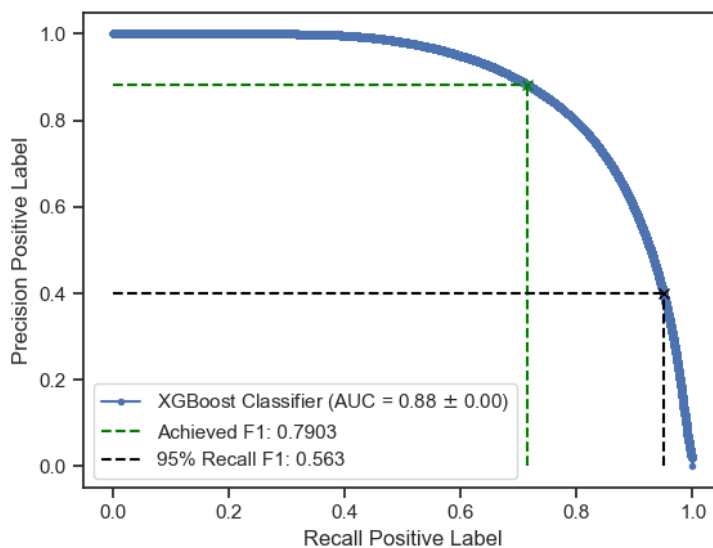
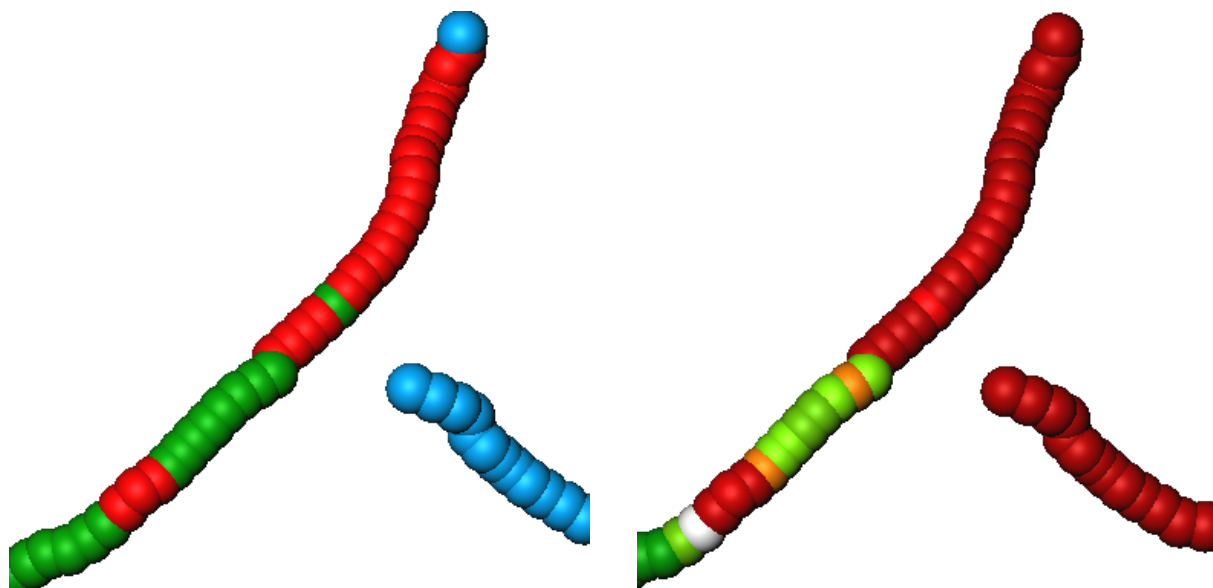


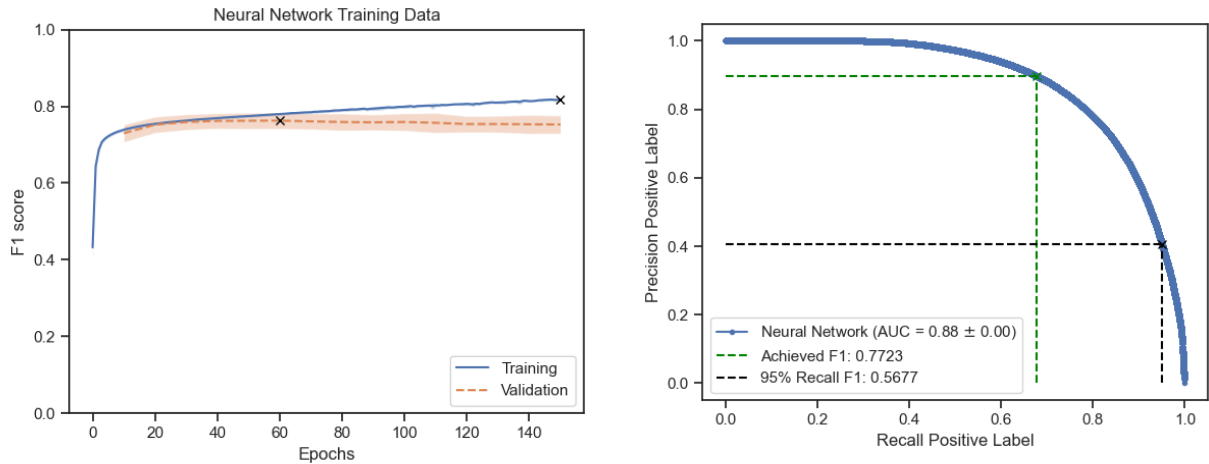
Figure 6: The XGBoost precision-recall curve averaged over ten folds on the test set. The achieved F_1 -score is 0.7903 ± 0.0013 , and the F_1 -score for 95.0 % recall 0.5630 .



(a) A segment of the XGBoost model's labels on subject 2. Green indicates true positive, red false negative, yellow false positive, and blue true negative.

(b) A segment of the XGBoost model's probabilities on subject 2. Red indicates approaching negative label, green approaching positive label. White indicates a fifty-fifty split.

Figure 7: XGBoost examples on test set subject 2.



(a) Multi-layer perceptron learning curve over 10 folds. The folds are averaged and depicted with one standard deviation. The dashed line shows the validation set results. The y-axis represents the F_1 -score for the positive class. For both lines, the best-achieved score has been marked with a black cross.

(b) Multi-layer perceptron precision-recall curve over 10 folds. The folds are averaged and depicted with one standard deviation. Plotted on the curve are the achieved F_1 -score of 0.7723 ± 0.0087 on the test set, as well as the F_1 -score of 0.5677 at 95 % recall.

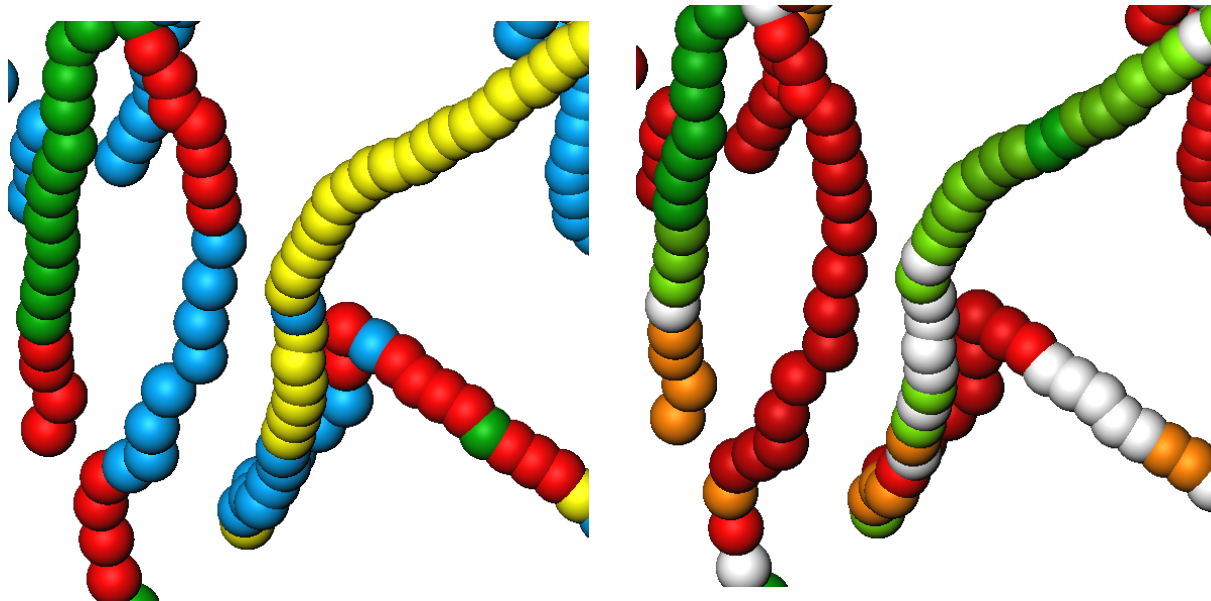
Figure 8: Multi-layer Perceptron training and test set results.

In Figure 7 a segment of one of the folds on subject 2 is depicted. One observation to make in Figure 7a compared to Figure 7b, is the single green node at the center. This node reflects the relatively low threshold that has been selected for this method, averaging at 19.0 %, as the probabilities of its neighbors are shown to be close. This indicates that the model tends towards predicting lower probabilities. A second observation is the difference in probabilities in Figure 7b, as the nodes differentiate substantially from colors between neighboring nodes. The Neighbourhood Probability Difference is on average 0.0868 ± 0.1800 . These jumps in probabilities can cause gaps in the output tree, which is demonstrated in Figure 7. Finally, we see that, although the achieved F_1 -score of the XGBoost method is high, it does have a low recall due to relatively few true positives. The F_1 -score at 95.0 % recall is relatively low, suggesting that shifting the threshold will not result in a clinically applicable model.

6.1.2 Multi-layer Perceptron

We use seven fully connected layers each with 128 parameters for our experiments with a multi-layer perceptron (MLP). We spend 200 epochs of training time on each fold, where after each 10th epoch the validation set was used to measure the performance of the model, as well as to determine the model's corresponding classification threshold. A learning rate of $\alpha = 5 \cdot 10^{-4}$ was used with L2 regularization factor of $1 \cdot 10^{-4}$. The results are illustrated in Figure 8.

The training curve of Figure 8a depicts that around epoch sixty the average best validation score has been achieved. Both methods of regularization described in Section 4.1.2 have been tried, and L2 regularization was selected empirically. Figure 8b shows the precision-recall curve on the test set. We achieved on average an F_1 -score of 0.7723 ± 0.0087 , which declines quickly at 95.0 % recall to 0.5677 . This is a decrease of 20.46 %. Although the achieved



(a) A segment of the MLP model's labels on subject 3. Green indicates true positive, red false negative, yellow false positive, and blue true negative.

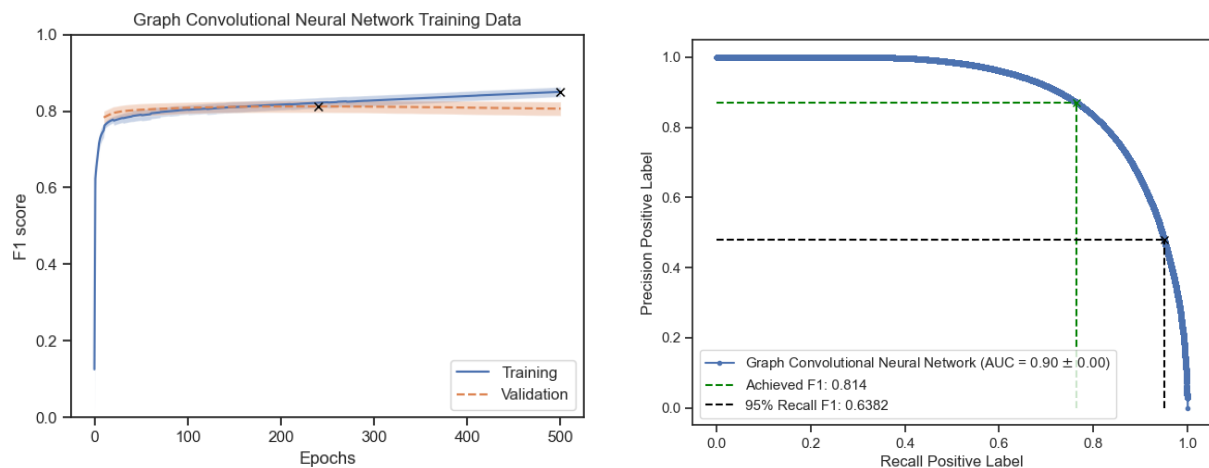
(b) A segment of the MLP model's probabilities on subject 3. Red indicates approaching negative label, green approaching positive label. White indicates a fifty-fifty split.

Figure 9: Multi-layer perceptron (MLP) examples on test set subject 3.

F_1 -score for the neural network model is slightly lower (1.8 %) compared to XGBoost, the decline of F_1 -score is slightly less steep in higher recall scores, increasing by 0.47 %.

Figure 9 shows an example of the MLP on test set subject 3. Here we show a dense section of the patient, where several segments are in close proximity of one another. Figure 9a depicts the predicted labels compared to the ground truth. The figure shows that the entire center-left segment is not recognized as an artery until the end. In this example, the tracker made a mistake in the center part of this segment and this needs to be removed. The probabilities of the nodes depicted in Figure 9b demonstrate that there is little to no variation in the center-left segment until the lower section which could be explained through graph agnosticism. In the right section in Figure 9a, we see that in these segments the model yields a substantial amount of errors as well, as it predicts many false positives and false negatives but the latter are close in their probabilities in Figure 9b. Finally, the left segment in Figure 9b ends in false-negative labels but we can see that the probabilities are decreasing smoothly. The Neighbourhood Probability difference was measured to be 0.0777 ± 0.1300 , a slight decline compared to the XGBoost approach.

The performance of these two graph agnostic baselines shows that the features of individual nodes provide enough information to enable the achieved performance demonstrated in this section. However, we have also seen that their results are limited and the methods struggle when it comes to the 95.0 % recall F_1 -score, indicating limited applicability in clinical settings.



(a) GCN training curve over 10 folds. The folds are averaged and the shaded area depicts one standard deviation. The dashed line shows the validation set evaluations. The y-axis represents F_1 -score for the positive class. For both lines, the best-achieved score has been marked with a black cross.

(b) GCN precision-recall curve over ten folds. The folds are averaged and depicted with one standard deviation. The dashed lines on the plot are illustrating the achieved F_1 -score of 0.8140 ± 0.0035 and the F_1 -score at 95.0 % recall of 0.6382

Figure 10: Graph Convolutional Neural Network (GCN) training and test set results.

6.2 Graph Neural Networks

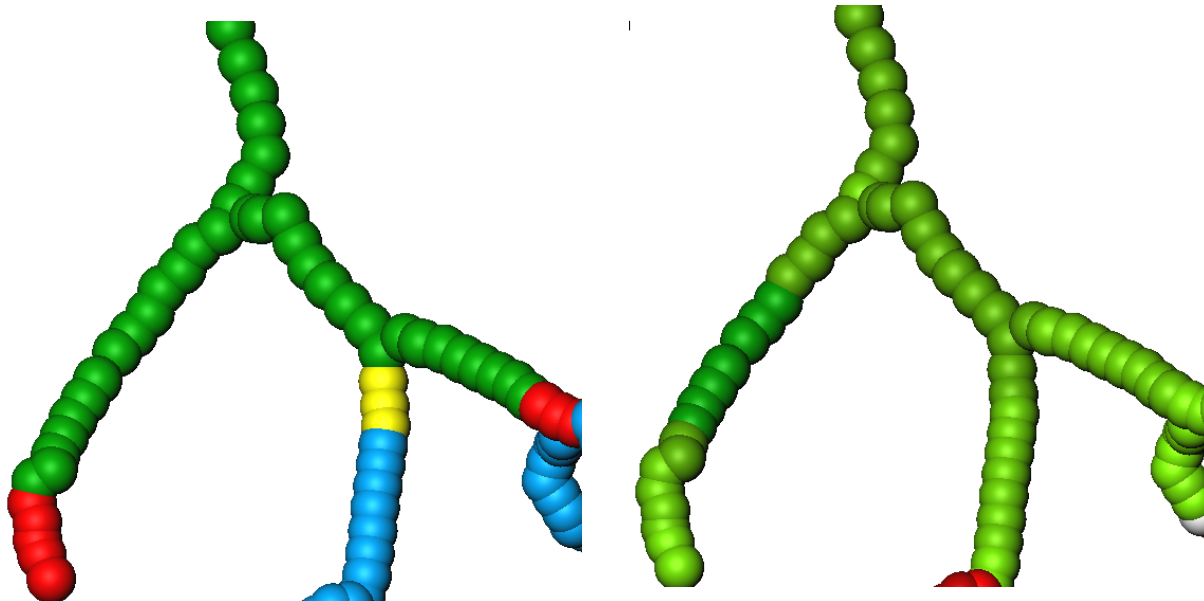
In this section, we present our experimental results for the various graph neural network architectures as described in Section 4.4.

6.2.1 Graph Convolutional Neural Network

For our Graph Convolutional Neural Network (GCN) experiments, we trained a six-layer model, four convolutional layers followed by two linear layers where for each layer the input features were concatenated to the output of the previous layer as skip connections. The width of each layer was set to 128 parameters. We trained the model with a learning rate of $\alpha = 2 \cdot 10^{-4}$ and an L2 regularization of $1 \cdot 10^{-4}$. Each fold was trained for 500 epochs. The results are depicted in Figure 10.

The training curve in Figure 10a shows in the first one hundred epochs, the validation score is actually higher than the training score. In the middle section, they cross one another and in the last one hundred epochs, although afterward, the increase in the training curve is not too steep. We also see that on average, the best validation score is achieved halfway through, suggesting that an increase in training epochs will not contribute to a better model. The PR-curve in Figure 10b shows the results on the test set. The model achieved on average an F_1 -score of 0.8140 ± 0.0035 and an F_1 -score at 95.0 % recall of 0.6382 . The drop between these two is substantial, at 17.58 %.

Figure 11 depicts a segment of the model’s result on subject 4. Figure 11a illustrates the predicted labels compared to the ground truth. We see here that both errors, false positive and negative, occur at the end of the arteries. Furthermore, the transition of probabilities is smooth in these areas as well, indicating a slow shift between remove and keep. The Neighbourhood



(a) A segment of the GCN model's labels on subject 4. Green indicates true positive, red false negative, yellow false positive, and blue true negative.

(b) A segment of the GCN model's probabilities on subject 4. Red indicates approaching negative label, green approaching positive label. White indicates a fifty-fifty split.

Figure 11: Graph Convolutional Neural Network (GCN) examples on test set subject 4.

Probability Difference is 0.0333 ± 0.0560 , which is much less compared to the XGB and MLP results in Section 6.1, suggesting that the introduction of the graph to the problem allows for a better representation of the node among its neighborhood.

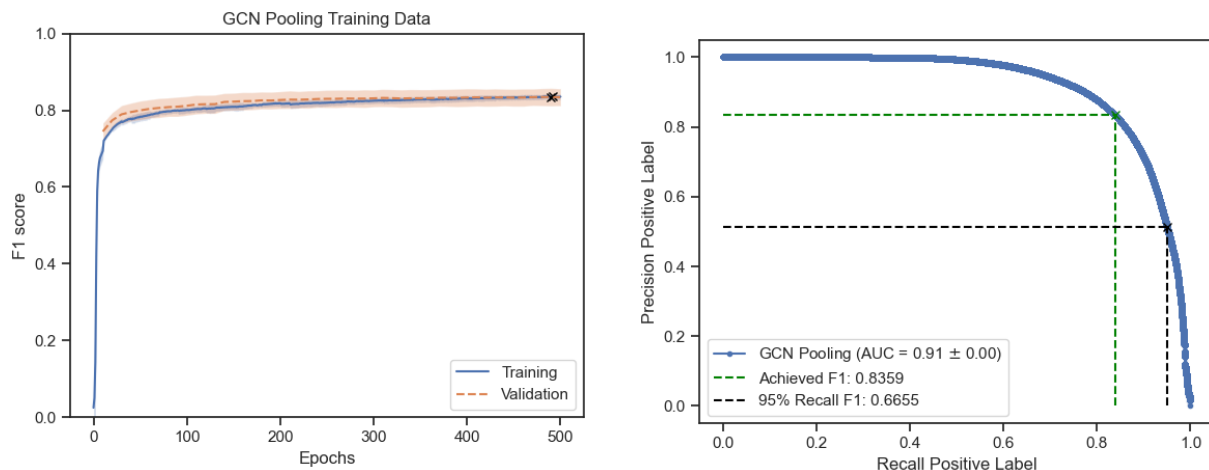
6.2.2 Graph Convolutional Pooling Networks

In this section, we will evaluate two types of models where the architectures are almost the same: One convolutional layer, followed by three pooling layers, three convolutional layers, and finally two linear layers. Before the two linear layers, all pooling is reverted. There are skip connections of the input to the first pooling layer, and the two linear layers. The output of the third pooling layer also has a skip connection to the second and the third convolutional layer.

The difference between the two models is the type of pooling used, Edge Pooling versus Cluster Pooling as described in Section 4.3.2. Both models were trained for 500 epochs with a 10 % dropout on each layer. First, we will look into a GCN with Edge Pooling.

For the Edge Pooling model, we ran each fold with a learning rate of $\alpha = 5 \cdot 10^{-5}$. The results are illustrated in Figure 12. The training graph depicted in Figure 12a shows that in the first two hundred epochs, the validation performance is on average higher than the training performance. Later on, the two seem to merge more and in the end achieve their best evaluations at about the same score and epoch. The increase in performance is decreased towards the end, however, since the validation score is about the same as the training score, it is unclear what the impact of an increase in training epochs would have. Finally, we see that the training curve has a small area of standard deviation in comparison to the validation curve.

Figure 10b illustrates the precision-recall curve over the test results of the ten folds. The model



(a) GCN Edge Pooling training curve over 10 folds. The folds are averaged and depicted with one standard deviation. The dashed line shows the validation set evaluations. The y-axis represents the F_1 -score for the positive class. For both lines, the best-achieved score has been marked with a black cross.

(b) GCN Edge Pooling precision-recall curve over ten folds. The folds are averaged and depicted with one standard deviation. The dashed lines on the plot are illustrating the achieved F_1 -score of 0.8359 ± 0.0016 and the F_1 -score at 95.0 % recall of 0.6655

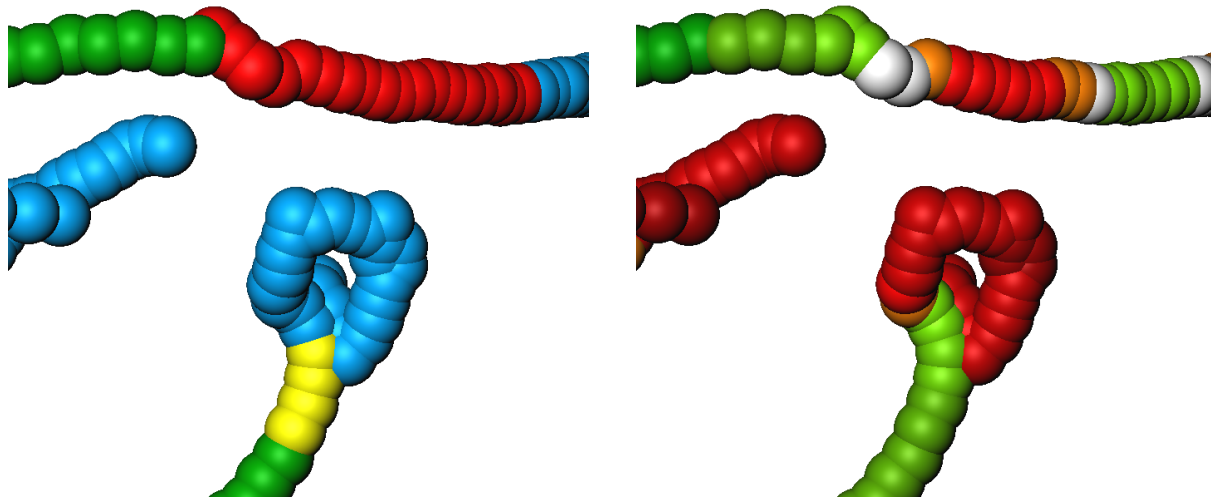
Figure 12: GCN Edge Pooling training and test set results.

achieved on average an F_1 -score of 0.8359 ± 0.0016 and a 95.0 % recall F_1 -score of 0.6655 . This is an improvement of 0.0219 and 0.0273 respectively to the results in Section 6.2.1. The difference between the two F_1 -scores is still substantial, decreasing 17.04 % between the two. This is slightly less than the GCN.

In Figure 13 a segment of the model on a test set case is shown. Figure 13a illustrates a segment of the model’s labels compared to the ground truth. The segment shows two difficult situations to predict: The linear segment at the top, which has some noise in its shape in the middle, and the clustered section at the bottom, which is at the end of a correct segment. In the top segment, we see that the model predicted the segment to end much higher in the tree, due to this noise. The results in Figure 13b show that the predictions are a bit varying on this part, indicating that it does partially learn this part of the problem. The model has a restrictive cut-off, classifying many nodes with an unnecessary negative label. In the bottom section, we see that the model performs a bit better. It correctly assesses the clustered part to be incorrect, and its probabilities in Figure 13b are very distinctive in this case. However, the cut-off does have some false positives but these are very few. The model has a neighborhood probability difference of 0.0272 ± 0.0609 .

For the Cluster Pooling model, we ran each fold for 500 epochs with a 10 % dropout and a learning rate of $\alpha = 5 \cdot 10^{-5}$. The architecture is the same as before, however, we now applied our cluster pooling layer instead of edge pooling. The results are depicted in Figure 14.

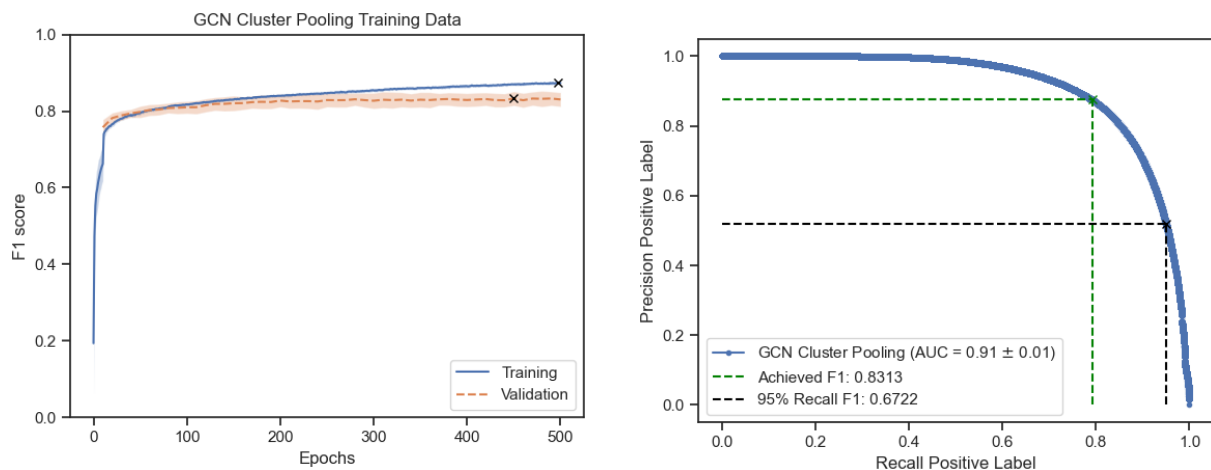
Figure 14a depicts the training results of the GCN with cluster pooling. From about three hundred epochs and onward the validation curve is flattening, but there is still some slight improvement. Figure 14b depicts the precision-recall curve on the test set. It shows that the model achieved an F_1 -score of 0.8313 ± 0.0060 and an F_1 -score at 95.0 % recall of 0.6722 . The achieved F_1 -score is a decrease from our previous result with Edge Pooling, declining



(a) A segment of the GCN Pooling model's labels on subject 117. Green indicates true positive, red false negative, yellow false positive, and blue true negative.

(b) A segment of the GCN Pooling model's probabilities on subject 117. Red indicates approaching negative label, green approaching positive label. White indicates a fifty-fifty split.

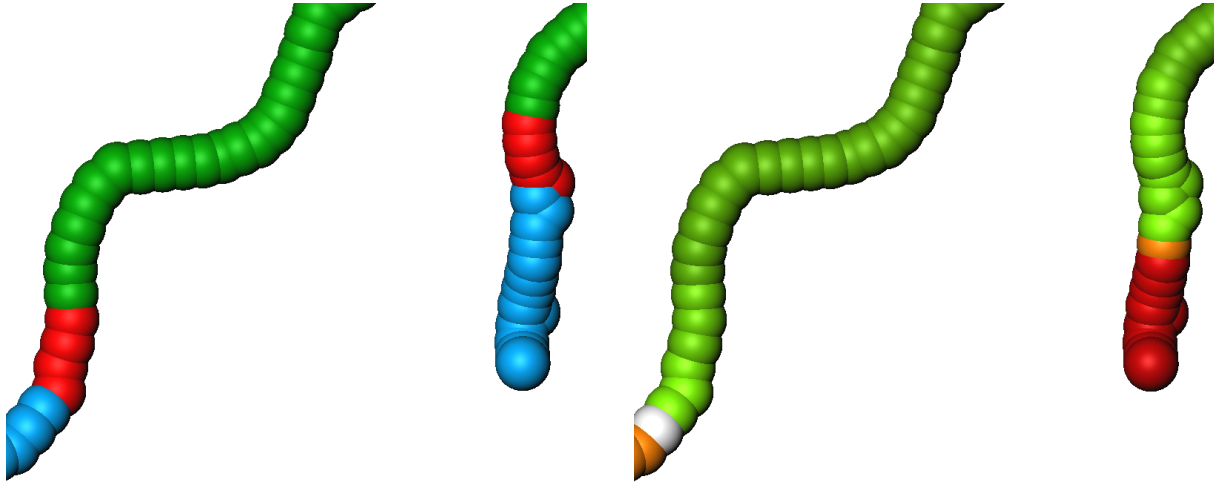
Figure 13: GCN with Edge Pooling examples on test set subject 117.



(a) GCN Cluster Pooling training curve over 10 folds. The folds are averaged and depicted with one standard deviation. The dashed line shows the validation set evaluations. The y-axis represents the F_1 -score for the positive class. For both lines, the best-achieved score has been marked with a black cross.

(b) GCN Cluster Pooling precision-recall curve over ten folds. The folds are averaged and depicted with one standard deviation. The dashed lines on the plot are illustrating the achieved F_1 -score of 0.8313 ± 0.0060 and the F_1 -score at 95.0 % recall of 0.6722

Figure 14: GCN Cluster Pooling training and test set results.



(a) A segment of the GCN Cluster Pooling model’s labels on subject 5. Green indicates true positive, red false negative, yellow false positive, and blue true negative.

(b) A segment of the GCN Cluster Pooling model’s probabilities on subject 5. Red indicates approaching negative label, green approaching positive label. White indicates a fifty-fifty split.

Figure 15: GCN with Cluster Pooling examples on test set subject 5.

0.49 %. However, the 95 % recall F_1 -score increased 0.67 %.

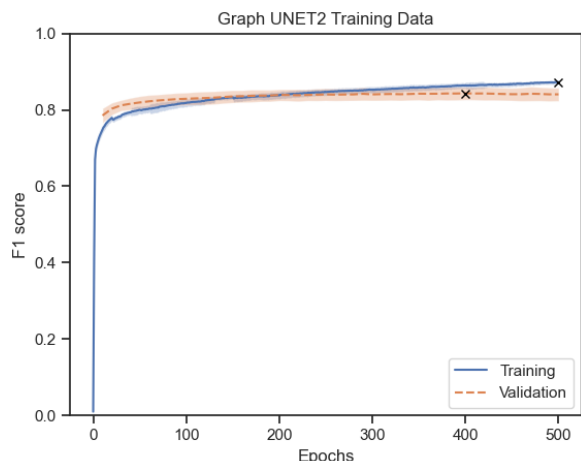
Figure 15 illustrates a segment of the model’s result on test set subject 5. Figure 15a shows that in both segments, the model cuts off the ends just a bit too early. When we compare this to the probabilities of Figure 15b, it does depict a smooth transition in these segments from positive to negative labels. This is indicating that the model does learn the task at hand, however, the threshold is in this particular segment too high as still a few nodes are incorrectly classified as negative. The GCN Cluster Pool has a neighborhood probability difference of 0.0277 ± 0.0537 .

6.2.3 Graph U-NET II

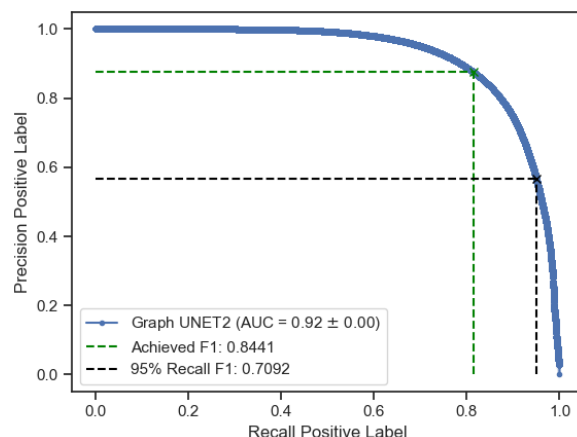
We ran our Graph U-NET II experiments with a depth of three, a dropout probability of 10 %, and a learning rate of $\alpha = 5 \cdot 10^{-4}$. The size of the hidden layers is 128. First, we will look into Graph U-NET II with Edge Pooling layers. In Figure 16 the results of the Graph U-NET II with Edge Pooling are illustrated.

Figure 16a illustrates the training curve of the model. From epoch three hundred and onward the incline of the validation curve is flattening, achieving on average its best validation score at about four hundred epochs. Figure 16 depicts the precision-recall curve on the test set. The model achieved an F_1 -score of 0.8441 ± 0.0027 and at a recall of 95.0 % of 0.7092 . There is much less difference between these two F_1 -scores than for example in the baselines. This is also evident in the PR curve, as it shows a much steeper curve and an increase in PR-AUC.

Figure 17 illustrates a segment of the model’s performance on test set subject 6. Figure 17a the labels on the segments are shown. We see here that in the bottom section it has completely failed to properly recognize an entire artery segment. At the top, we see a segment where labeling did go correctly, and only the end was cut off a bit too soon. In Figure 17b shows the probabilities of these segments. Here we can see that in the top segment there is a smooth

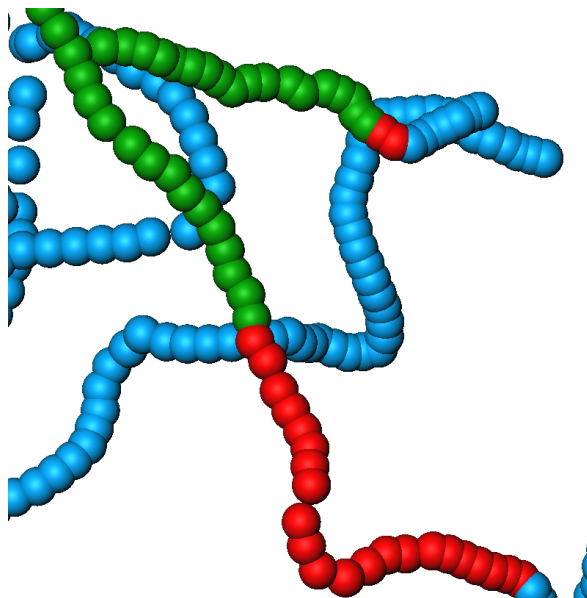


(a) Graph U-NET II Edge Pooling training curve over 10 folds. The folds are averaged and depicted with one standard deviation. The dashed line shows the validation set evaluations. The y-axis represents the F_1 -score for the positive class. For both lines, the best-achieved score has been marked with a black cross.

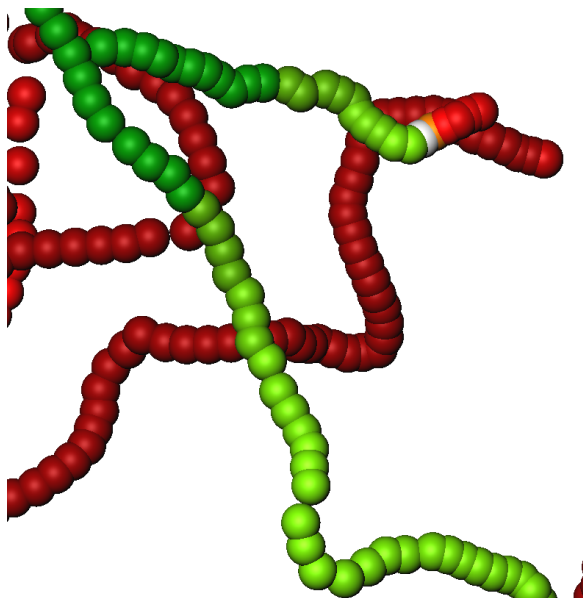


(b) Graph U-NET II Edge Pooling precision-recall curve over ten folds. The folds are averaged and depicted with one standard deviation. The dashed lines on the plot are illustrating the achieved F_1 -score of 0.8441 ± 0.0027 and the F_1 -score at 95.0 % recall of 0.7092

Figure 16: Graph U-NET II Edge Pooling training and test set results.

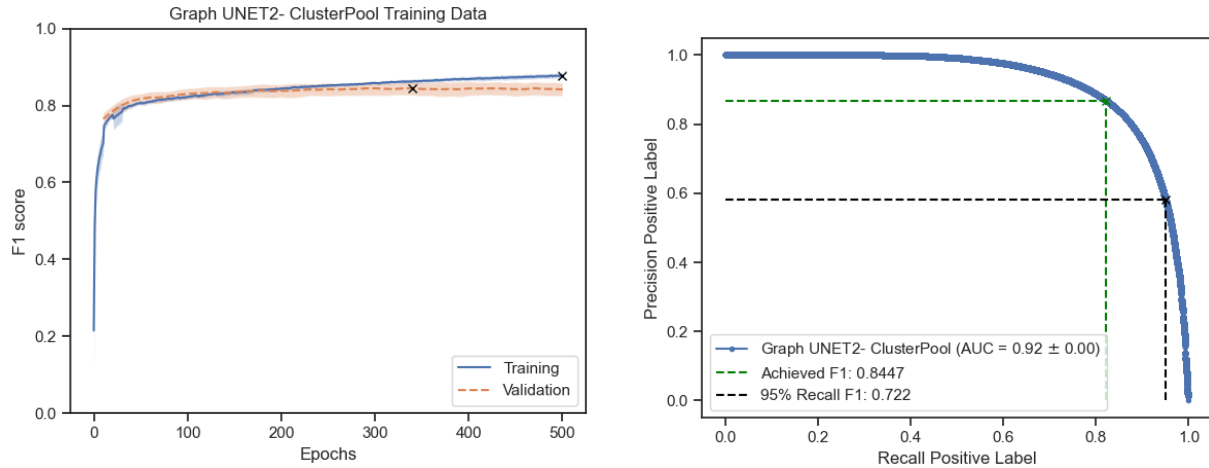


(a) A segment of the Graph U-NET II Edge Pooling model's labels on subject 6. Green indicates true positive, red false negative, yellow false positive, and blue true negative.



(b) A segment of the Graph U-NET II Edge Pooling model's probabilities on subject 6. Red indicates approaching negative label, green approaching positive label. White indicates a fifty-fifty split.

Figure 17: Graph U-NET II with Edge Pooling examples on test set subject 6.



(a) Graph U-NET II Cluster Pooling training curve over 10 folds. The folds are averaged and depicted with one standard deviation. The dashed line shows the validation set evaluations. The y-axis represents the F_1 -score for the positive class. For both lines, the best-achieved score has been marked with a black cross.

(b) Graph U-NET II Cluster Pooling precision-recall curve over ten folds. The folds are averaged and depicted with one standard deviation. The dashed lines on the plot are illustrating the achieved F_1 -score of 0.8447 ± 0.0025 and the F_1 -score at 95.0 % recall of 0.7220

Figure 18: Graph U-NET II Cluster Pooling training and test set results.

transition going on and the two false negatives actually have a pretty good probability. In the bottom segment, we see that every node that has been selected to be removed actually had a high probability. This segment shows us how important and difficult it is to select the right threshold. The model has a neighborhood probability difference on the test set of 0.0196 ± 0.0331 .

In the next section, we will evaluate the impact of exchanging the Edge Pool layers in Graph U-NET II with our Cluster Pool Operator. Figure 18 the results of the Graph U-NET II with Cluster Pooling are illustrated.

Figure 18a shows the training curve of the model. In the last hundred epochs, the training curve and validation curve start to deviate. The best-achieved validation score is around 350 epochs. In Figure 18b the PR-curve on the test set is shown. It shows an F_1 -score of 0.8447 ± 0.0025 . This is a very slight increase compared to the results of Graph U-NET II with edge pooling, only increasing the average by 0.06 %. However, the performance of this model of an F_1 -score at 95.0 % recall is 0.7220 , is a substantial increase from our previous model, improving 1.28 %. This shows that there is some merit to using this pooling operator, as it does signal a more clinically relevant model.

Figure 19 an example segment of the model on test set subject 7 is illustrated. Figure 19a shows a segment of subject 7 label's compared to the ground truth. In the left section, we see a very large proportion of false negative nodes. As we aim to extract the coronary arteries from the data, these mistakes are very detrimental to our output. In the center section, we see three arteries that are properly detected, with some small but acceptable mistakes towards the end. Towards the right, we see a relatively small mistake, where a sizable end piece of the artery was removed. If we compare these mistakes with the probabilities, as shown in Figure 19b, it does indicate that the model does indicate for all of these false negatives that there are some

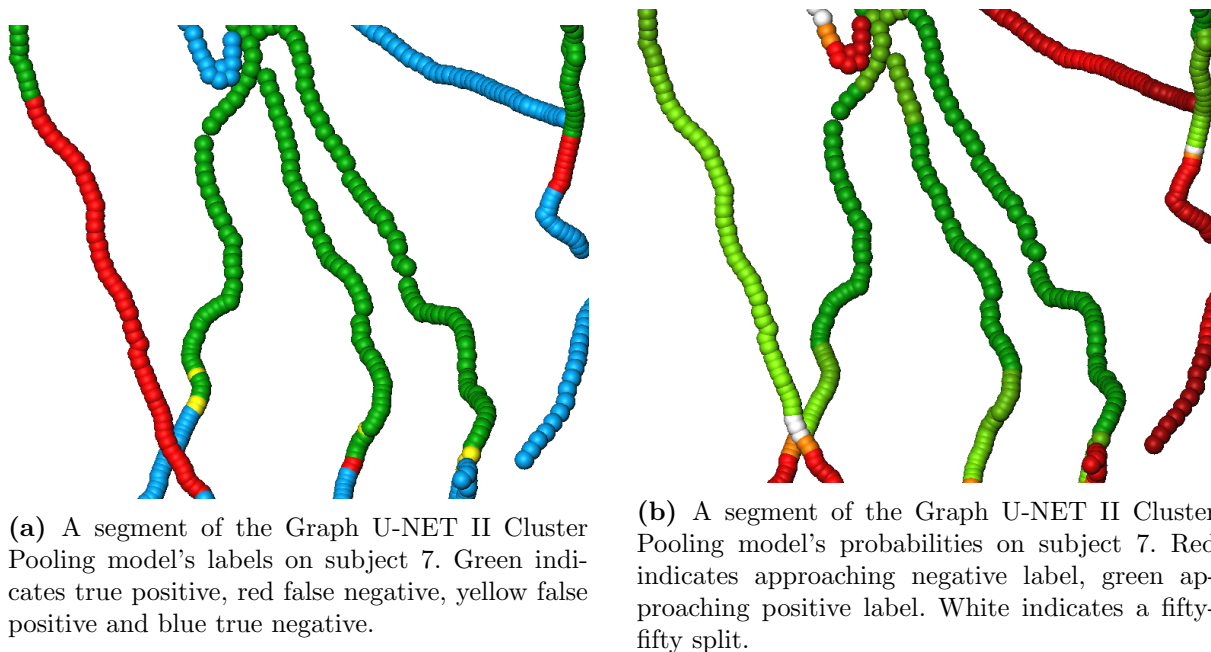


Figure 19: Graph U-NET II with Edge Pooling examples on test set subject 7.

indicators for the model, as many of the nodes do have a probability of over 50.0 %. Graph U-NET II with Cluster Pooling has a neighborhood probability difference of 0.0186 ± 0.0374 . This is visualized in Figure 19b, as we see smooths transitions in almost every segment of the tree, including the false negatives.

6.3 Comparison

In Table 1 we show the results of every model as described in Section 6.1 and Section 6.2, separating the baselines with the GNNs with a dashed line. The best results are boldfaced.

The results of Table 1 have in general a small standard deviation. The graph neural networks show a clear improvement in F_1 -scores compared to the graph agnostic baselines. The table shows that Graph U-Net II with Cluster Pooling achieved the best test set performance for the F_1 -scores. We also see a steady decrease in the neighborhood probability difference throughout the methods, with exception of the GCN Cluster Pool. The Graph U-NET II Edge Pool and Cluster Pool methods both display the lowest neighborhood differences, indicating that the probabilities of the nodes in the output graphs of these methods have in general smooth transitions.

To test the significance of the difference in the performance of our methods, we have conducted a Nemenyi test [18]. We ranked the methods by their performance, where the best-ranked method per patient obtained rank 1, and the worst performing method in terms of F_1 -score per patient obtained rank 7. As the number of data sets, we selected the number of patients in the test set, e.g. each patient represents a data set thus yielding fifty data sets. The average score F_1 -score of the methods was used to determine their rank. The results are depicted in Figure 20.

Figure 20 shows that there is no evidence for statistically significant improvement between

Method	F ₁ -score	F ₁ (r = 0.95)	NPD
XGB	0.7903 ± 0.0013	0.5630	0.0868 ± 0.1800
MLP	0.7723 ± 0.0087	0.5677	0.0777 ± 0.1300
GCN	0.8140 ± 0.0035	0.6382	0.0333 ± 0.0560
GCN Edge Pool	0.8359 ± 0.0016	0.6655	0.0272 ± 0.0609
GCN Cluster Pool	0.8313 ± 0.0060	0.6722	0.0277 ± 0.0537
Graph U-NET II Edge Pool	0.8441 ± 0.0027	0.7092	0.0196 ± 0.0331
Graph U-NET II Cluster Pool	0.8447 ± 0.0025	0.7220	0.0186 ± 0.0374

Table 1: Test Set F₁-scores of all presented models, the first column denoting the best-achieved F₁-score and the second its standard deviation. The third column contains the F₁-score at a 95.0 % recall. The fourth column contains the probability difference between neighboring nodes in the test set, indicating the abruptness of the change in probabilities. The dashed line separates the graph agnostic baselines (top). The best scores are marked in **bold**.

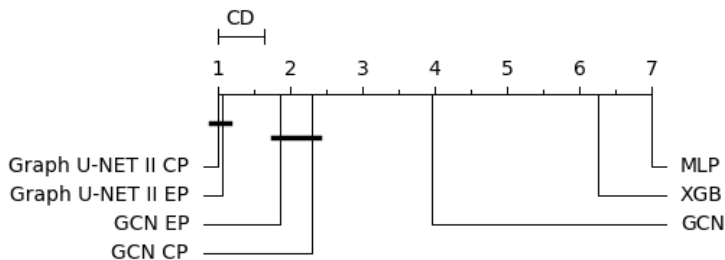
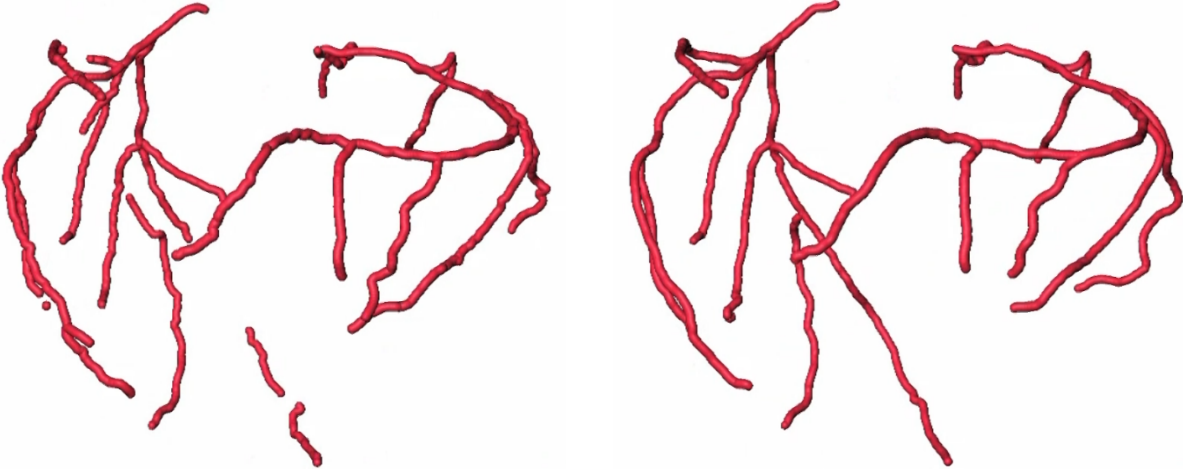


Figure 20: Nemenyi test on the results of Table 1. Here, the data sets are defined by the number of patients in the test set e.g. 50. CD denotes the critical difference, the minimal difference required to determine significantly different ranks, which is determined here to be 0.63. The thick line indicates that this threshold has not been met.



(a) The coronary artery tree of subject 1 as extracted by the pipeline.

(b) The coronary artery reference tree of subject 1 as annotated by the expert.

Figure 21: An example output graph of the pipeline from Figure 3 on subject 1, where we used an ensemble of ten Graph U-NET II Cluster Pool models as the geometric deep learning model. Figure 21a shows the output of the pipeline, Figure 21b illustrates the corresponding reference tree marked by domain experts.

the Graph U-NETs as well as between the two GCNs. The Graph U-NETs’ result is to be expected, as we have seen in the results of Table 1 that the margin was slim. We also see that the GCN with Edge Pooling is quite close to being critically different from GCN with Cluster Pooling. In contrast, there is a substantially significant improvement between the GCN and the graph agnostic baselines, as well as between the GCN and GCNs with pooling operators. This suggests that there is an added value to these operators and the architecture from Graph U-NET II.

To have some context for the quality of the results, we compare our best model to the state-of-the-art. We use the metrics from Section 5.1 to determine the quality of the output trees based on the reference trees. We compare our method against Salahuddin et al. [34] as both this thesis and Salahuddin et al. [34] present fully automatic methods.

Note that the comparison to Salahuddin et al. [34] is limited: It is not possible to compare the methods on the same data set as medical data, in general, is not public data. Furthermore, their code is not publicly available, so reproducing their model to run on our data set is also not easily possible. Thus we compare the results on their data set to ours directly. We have not opted to use the MICCAI challenge data set (CAT08), as this challenge only focuses on the four main centerlines of the coronary arteries. This is an issue, as we have focussed on extracting the entire coronary artery tree including the most difficult cases.

We have taken our best achieving method’s results as an ensemble model, Graph U-NET II with Cluster Pooling, to compare to the reference trees. This ensemble achieves an F_1 -score of 86.17 % on the test set, which increased by 1.70 % compared to the average performance of the models in the ensemble. The ensemble achieves a precision of 88.0 % on the reference trees. The results are summarized in Table 2. An example of our full pipeline on a subject

Metric	Salahuddin et al. [34]	Ours
Sensitivity	87.1 %	86.0 %
Overlap	80.4 %	87.0 %

Table 2: Reference Tree Metrics. Sensitivity shows what proportion of the reference tree was extracted. Overlap shows how much the predicted tree overlaps with the reference tree.

is depicted in Figure 21, where Figure 21a shows the output of our pipeline on the subject, and Figure 21b shows the reference tree. The automatically extracted coronary artery tree in Figure 21a indicates high performance to the reference tree of Figure 21b. A few mistakes can be seen, such as long gaps in the center. The bifurcation of this detached centerline is also not present, as the center-left artery is not connected.

Improvements Our results in Table 2 on our data set show a substantial improvement compared to the results by Salahuddin et al. [34] on their proprietary data set. We have shown an increase of 6.6 % in terms of overlap. As overlap is a combination of sensitivity and precision, this indicates that we have a much more precise prediction, that includes less noise. Although there is a decrease in sensitivity, indicating that their approach extracts more centerline points of the reference tree, our pipeline yields more usable results in practice as there is less noise surrounding the extracted coronary artery tree. As an example, we have shown the output on subject 1 in Figure 21, where we see that, although certain segments are missing, there is a limited amount of noise present making it straightforward to recognize the artery tree in the output.

7 Discussion

In this thesis, we aimed to leverage methods from the geometrical deep learning domain to aid in the automated extraction of the coronary arteries from CCTA scans. We used methods from Wolterink et al. [49] and Hampe et al. [16] to extract the centerlines and reviewed various approaches to filter out the noise these pipelines generated. To examine the effectiveness of geometrical deep learning techniques, we compared graph agnostic baselines to graph neural networks. We have attained increased F_1 -scores compared to the graph agnostic baselines and other graph neural networks using our newly developed techniques. We achieved a best F_1 -score of 86.17 % with Graph U-NET II ensemble on our test set. The experiments showed a contribution in performance when making use of the graph structure of the data. We have seen a significant increase in the F_1 metric in Table 1 between the graph agnostic baselines and the GNNs. We have seen that this enables the models to yield higher performance. Finally, we also saw a steep increase in the F_1 -score for 95.0 % recall, signifying substantial improvement towards clinically relevant models.

However, there is room for performance improvement on this task. There can be many causes of the difference between our achieved F_1 -scores and near-optimal F_1 -scores. One that we have identified as a possible issue is the many disjoint components and edge sparsity of the graphs, as can be seen in Table 3. As all of our geometrical deep learning methods revolve around exchanging information using edge relations between nodes, this sparsity brings limitations, as the information propagation GNNs is based on edges. One solution to this could have been the detection of missed edges: In each node, there are probabilities of a bifurcation being present through the directional vectors. This information could indicate that there is a missing section in some direction of a node. If this information is used effectively, a less disjointed graph could be created, thus increasing the propagation of information.

Furthermore, it is hard to make any significant statements about the added value of our Cluster Pooling layer compared to the Edge Pooling layer. Not only did Graph U-NET II barely improve with the Cluster Pooling operator in its best F_1 -score, the regular GCN with pooling operators actually showed *decreased* performance when combined with Cluster Pool. An argument for its advantage is that the F_1 -score of Graph U-NET II improved significantly at 95.0 % recall and yielded the optimal value. However, the training curves demonstrated a faster learning curve, that deviated substantially from the validation curve. This could indicate that there might be some possible merit to this layer, as there might be room for more generalization. However, this is only speculation. A better review of this operator could be conducted on benchmark data sets, but there are indications that there is room for improvement in this operator.

Another issue could be the differentiability of stenosed and narrowing arteries. In the current data set, we use nodes that represent patches of CCTA scans. These scans not only show the presence of contrast fluid but also calcified arteries. This is currently filtered out by the CNNs of Wolterink et al. [49] and not shown in the information of the node. However, to increase distinguishability between vessels and narrowed or stenosed arteries, this information could be key: The radius of a node could be expressed as including the plaque. Experts use this information during their analysis of a patient, thus it does not only make sense to use this information to extract the arteries but this plaque would also be represented in the output.

We have seen an improvement in our pipeline on our proprietary test set, including Graph U-NET II, in comparison to Salahuddin et al. [34] on their private data set. We achieved an overlap of 87.0 %, an improvement of 8.6 % to Salahuddin et al. [34]. We achieved a sensitivity of 86.0 %, which is a decrease of 1.1 % compared to the results of Salahuddin et al. [34]. This shows that our method sacrifices sensitivity for a substantial increase in overlap, thus creating a more clinically relevant model in comparison with the state-of-the-art method of Salahuddin et al. [34]. There are key differences between their approach and ours, as we have adopted geometrical deep learning methods that allowed a receptive field beyond a single node and have shown it to be very effective whereas their work focussed on local methods.

Finally, the quality of the hyperparameters is very likely not even close to optimal. Although many configurations have been tested in the presented experiments by employing manual search, the search space of the parameters is vast and only a fraction can be explored through manual optimization. We have set all hyperparameters manually, such as learning rates and scale of loss to adjust the class imbalance. We have shown no evidence that these configurations are optimal, and in order to do so an extensive search of the hyperparameter space should be conducted using HPO methods such as SMAC [20] or DEHB [1].

8 Conclusion and Future Work

We investigated the problem of coronary artery extraction from CCTA scans. Specifically, we looked for a new fully automated pipeline based on the work of Wolterink et al. [49]. Current works make use of local information methods to determine if a centerline point is within the coronary arteries. Instead, we aimed to make use of the graph structure in the data and reviewed methods from the field of geometrical deep learning to detect coronary arteries. For this purpose, we have developed new methods, a new node pooling layer, and Graph U-NET II, and demonstrated their performance on the problem. We evaluated their performance on the test set in terms of F1-score on the classification task in comparison to graph agnostic baselines and other classical GNNs.

Our results have demonstrated that there is a statistically significant improvement in Graph U-NET II in comparison to the other methods using a Nemenyi test [18]. Furthermore, we demonstrated that our entire pipeline, including the Graph U-NET II, has a substantial increase in performance in comparison to the work of Salahuddin et al. [34] in terms of overlap. However, for the cluster pooling operator, its significance is unclear. The graph neural networks in general significantly outperform the graph agnostic baselines and yield interpretable output probabilities. Our best performing method, Graph U-NET II, produced results that surpassed Salahuddin et al. [34] in overlap, as our model achieved 84.8 % compared to 80.4 % of Salahuddin et al. [34]. In sensitivity it scored 86.0 %, compared to 87.1 %, which is a slight decrease. However, this comparison can be seen as an indication only, as both the method proposed by Salahuddin et al. [34] and our methods were reviewed on proprietary data sets. For comparability, we open-source our code in the aforementioned Github Repository.

Through the observation of samples, we have the intuition of several critical issues that may contribute to improvement in the future, such as the sparsity of edges and the differentiability of the data. Graph neural networks offer a valid alternative to local classification methods,

such as convolutional neural networks, that are restricted to a single centerline point, and have shown their values in terms of the output trees' overlap with the reference trees.

In future work, to further enable these graph-based methods to propagate information through the graph, missed edges or bifurcations can be detected with the directional vectors of the nodes, as described in Section 2. This could lead to more seed points for the tracker described by Wolterink et al. [49], which can yield either completely missed segments or reconnect the artery to previously detected segments. The addition of extra nodes, and a more connected graph, would allow for better propagation of information and thus lead to better models.

The models presented here have little support for the quality of their hyperparameters. The hyperparameter settings have been selected manually and a search of the hyperparameter configuration space using HPO methods has shown to lead to better results in various algorithms and objectives [12] [1]. Furthermore, the experimentation with HPO methods on graph neural networks would be an interesting point of research on its own.

The result for fully automatic extraction of the full coronary artery tree here has limited comparability to state-of-the-art methods due to the lack of a publicly available data set. In future work, creating a new challenge data set on which methods are reviewed could lead to more comparable results for state-of-the-art approaches in coronary artery extraction.

To conclude, we have measured the effectiveness of various methods from the geometrical deep learning domain on the task of coronary artery extraction. We have seen that these methods offer a valuable alternative to graph agnostic approaches. We contributed new techniques to the domain of geometrical deep learning that increased the performance of our models for the task at hand. We have shown that the results of this thesis exceed the current state of the art.

References

- [1] Noor Awad, Neeratyoy Mallik, and Frank Hutter. 2021. DEHB: Evolutionary hyperband for scalable, robust and efficient hyperparameter optimization. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence (IJCAI)*, Vol. 30. IJCAI, Montreal, Canada, 2147–2153. (Cited on p. 15, 43, 44)
- [2] James Bennett and Stan Lanning. 2007. The Netflix Prize. In *Proceedings of the KDD Cup Workshop*. Association for Computing Machinery, 1601 Broadway, Times Square, New York City, 3–6. (Cited on p. 18)
- [3] James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *Journal of Machine Learning Research* 13, 2 (2012), 281–305. (Cited on p. 15)
- [4] Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. 2021. *Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges*. arXiv, Imperial College London, New York University. <https://doi.org/10.48550/ARXIV.2104.13478> (Cited on p. 19, 20)
- [5] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. 2017. Geometric Deep Learning: Going beyond Euclidean Data. *IEEE Signal Processing Magazine* 34, 4 (2017), 18–42. (Cited on p. 6)
- [6] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2013. Spectral Networks and Locally Connected Networks on Graphs. In *Proceedings of the First International Conference on Learning Representations (ICLR)*. OpenReview, <https://openreview.net/group?id=ICLR.cc/2013/conference>. (Cited on p. 13, 19, 23, 24)
- [7] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Vol. 22. Association for Computing Machinery, 1601 Broadway, Times Square, New York City, 785–794. (Cited on p. 18)
- [8] Frederik Diehl. 2019. Edge Contraction Pooling for Graph Neural Networks. <https://doi.org/10.48550/ARXIV.1905.10990> (Cited on p. 6, 14, 15, 20)
- [9] Frederik Diehl, Thomas Brunner, Michael Truong Le, and Alois Knoll. 2019. Towards graph pooling by edge contraction. In *ICML 2019 Workshop on Learning and Reasoning with Graph-structured Data*. Proceedings of Machine Learning Research, <https://proceedings.mlr.press>. (Cited on p. 14)
- [10] John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research* 12, 61 (2011), 2121–2159. (Cited on p. 17)

- [11] Olivier Ecabert, Jochen Peters, Hauke Schramm, Cristian Lorenz, Jens von Berg, Matthew J. Walker, Mani Vembar, Mark E. Olszewski, Krishna Subramanyan, Guy Lavi, and Jürgen Weese. 2008. Automatic Model-Based Segmentation of the Heart in CT Images. *IEEE Transactions on Medical Imaging* 27, 9 (2008), 1189–1201. <https://doi.org/10.1109/TMI.2008.918330> (Cited on p. 12)
- [12] Stefan Falkner, Aaron Klein, and Frank Hutter. 2018. BOHB: Robust and efficient hyperparameter optimization at scale. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*. PMLR, <https://proceedings.mlr.press>, 1437–1446. (Cited on p. 15, 44)
- [13] Jerome H Friedman. 2001. Greedy function approximation: a gradient boosting machine. *Annals of Statistics* 29 (2001), 1189–1232. (Cited on p. 18)
- [14] Ola Friman, Milo Hindennach, Caroline Kühnel, and Heinz-Otto Peitgen. 2010. Multiple hypothesis template tracking of small 3D vessel structures. *Medical Image Analysis* 14, 2 (2010), 160–171. (Cited on p. 9)
- [15] Hongyang Gao and Shuiwang Ji. 2019. Graph U-Nets. In *Proceedings of the 36th International Conference on Machine Learning (ICLR) (Proceedings of Machine Learning Research, Vol. 97)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). PMLR, <https://proceedings.mlr.press>, 2083–2092. (Cited on p. 6, 20, 23, 24)
- [16] Nils Hampe, Sanne G. M. van Velzena, Jelmer M. Wolterinka, Carlos Collet, Jose P. Simao Henriques, Nils Planken, and Ivana Išgum. 2022. Graph Neural Networks for Automatic Extraction and Labeling of the Coronary Artery Tree in CT Angiography. *UNDER SUBMISSION* (2022). (Cited on p. 5, 6, 42)
- [17] Nils Hampe, Jelmer Wolterink, Carlos Collet, R. Planken, and Ivana Išgum. 2021. Graph attention networks for segment labeling in coronary artery trees. In *Proceedings of SPIE Medical Imaging 2021*. SPIE Press, 50–58. <https://doi.org/10.1117/12.2581219> (Cited on p. 6, 10, 12, 13, 14)
- [18] Myles Hollander and Douglas A. Wolfe. 1999. *Nonparametric Statistical Methods* (2nd ed.). Wiley-Interscience. (Cited on p. 38, 43)
- [19] John Hopcroft and Robert Tarjan. 1973. Algorithm 447: Efficient Algorithms for Graph Manipulation. *Commun. ACM* 16, 6 (jun 1973), 372–378. (Cited on p. 21, 22)
- [20] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2011. Sequential model-based optimization for general algorithm configuration. In *International Conference on Learning and Intelligent Optimization (LION)*. Springer, 507–523. (Cited on p. 43)
- [21] Moien A.B. Khan, Muhammad Jawad Hashim, Halla Mustafa, May Yousif Baniyas, Shaikha Khalid Buti Mohamad Al Suwaidi, Rana AIKatheeri, Fatmah Mohamed Khalfan Alblooshi, Meera Eisa Ali Hassan Almatrooshi, Mariam Eisa Hazeem Alzaabi, Reem Saif Al Darmaki, and Shamsa Nasser Ali Hussain Lootah. 2020. Global Epidemiology of Ischemic Heart Disease: Results from the Global Burden of Disease Study. *Cureus* 12 (7 2020), e9349. (Cited on p. 5, 7)

- [22] Taskeen Khan. 2022. *Cardiovascular diseases*. World Health Organization. <https://www.who.int/health-topics/cardiovascular-diseases> Accessed: 18-05-2022. (Cited on p. 5, 7)
- [23] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*. OpenReview. (Cited on p. 17)
- [24] Anders Krogh and John Hertz. 1991. A simple weight decay can improve generalization. *Advances in Neural Information Processing Systems* 4 (1991), 950–957. (Cited on p. 17)
- [25] Hong Zu Li and Pierre Boulanger. 2020. A Survey of Heart Anomaly Detection Using Ambulatory Electrocardiogram (ECG). *Sensors* 20, 5 (3 2020). (Cited on p. 7)
- [26] Peter Libby and Pierre Theroux. 2005. Pathophysiology of Coronary Artery Disease. *Circulation* 111, 25 (2005), 3481–3488. (Cited on p. 7)
- [27] Warren S. McCulloch and Walter Pitts. 1943. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics* 5 (1943), 115–133. (Cited on p. 18)
- [28] Vinod Nair and Geoffrey E Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML)*. Omnipress, Haifa, Israel, 807–814. (Cited on p. 19)
- [29] Galileo Namata, Ben London, Lise Getoor, Bert Huang, and U Edu. 2012. Query-driven active surveying for collective classification. In *Proceedings of the Tenth Workshop on Mining and Learning with Graphs (MLG)*, Vol. 8. Association for Computing Machinery, 1601 Broadway, Times Square, New York City, 1. (Cited on p. 15, 24)
- [30] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. 2021. Activation functions: Comparison of trends in practice and research for deep learning. *International Conference on Computational Sciences and Technology (ICCST)* (2021). (Cited on p. 19)
- [31] Friedrich Paulsen and Jens Waschke. 2013. *Sobotta Atlas of Human Anatomy* (15 ed.). Vol. 1. Urban & Fischer. 1–1000 pages. (Cited on p. 13)
- [32] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer Assisted Intervention (MICCAI), Proceedings, Part III*. Springer, 234–241. (Cited on p. 23)
- [33] Sebastian Ruder. 2016. An overview of gradient descent optimization algorithms. <https://doi.org/10.48550/ARXIV.1609.04747> (Cited on p. 17)
- [34] Zohaib Salahuddin, Matthias Lenga, and Hannes Nickisch. 2021. Multi-Resolution 3D Convolutional Neural Networks for Automatic Coronary Centerline Extraction in Cardiac

CT Angiography Scans. In *2021 IEEE 18th International Symposium on Biomedical Imaging (ISBI)*. Curran Associates, Inc., 91–95. <https://doi.org/10.1109/ISBI48211.2021.9434002> (Cited on p. 5, 12, 13, 40, 41, 43)

- [35] Michiel Schaap, Coert T. Metz, Theo van Walsum, Alina G. van der Giessen, Annick C. Weustink, Nico R. Mollet, Christian Bauer, Hrvoje Bogunović, Carlos Castro, Xiang Deng, Engin Dikici, Thomas O'Donnell, Michel Frenay, Ola Friman, Marcela Hernández Hoyos, Pieter H. Kitslaar, Karl Krissian, Caroline Kühnel, Miguel A. Luengo-Oroz, Maciej Orkisz, Örjan Smedby, Martin Styner, Andrzej Szymczak, Hüseyin Tek, Chunliang Wang, Simon K. Warfield, Sebastian Zambal, Yong Zhang, Gabriel P. Krestin, and Wiro J. Niessen. 2009. Standardized evaluation methodology and reference database for evaluating coronary artery centerline extraction algorithms. *Medical Image Analysis* 13, 5 (2009), 701–714. Includes Special Section on the 12th International Conference on Medical Imaging and Computer Assisted Intervention. (Cited on p. 5, 6, 9, 11, 12, 25, 26)
- [36] Michiel Schaap, Theo van Walsum, Lisan Neefjes, Coert Metz, Ermanno Capuano, Marleen de Bruijne, and Wiro Niessen. 2011. Robust Shape Regression for Supervised Vessel Segmentation and its Application to Coronary Segmentation in CTA. *IEEE Transactions on Medical Imaging* 30, 11 (2011), 1974–1986. <https://doi.org/10.1109/TMI.2011.2160556> (Cited on p. 9)
- [37] Jürgen Schmidhuber. 2015. Deep learning in neural networks: An overview. *Neural Networks* 61 (2015), 85–117. (Cited on p. 19)
- [38] U. Joseph Schoepf, Peter L. Zwerner, Giancarlo Savino, Christopher Herzog, J. Matthias Kerl, and Phillip Costello. 2007. Coronary CT angiography. *Radiology* 244 (7 2007), 48–63. <https://doi.org/10.1148/radiol.2441052145> (Cited on p. 7)
- [39] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. <https://doi.org/10.48550/ARXIV.1707.06347> (Cited on p. 15)
- [40] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. 2008. Collective classification in network data. *AI magazine* 29, 3 (2008), 93–93. (Cited on p. 15, 24)
- [41] Nahian Siddique, Sidike Paheding, Colin P Elkin, and Vijay Devabhaktuni. 2021. U-net and its variants for medical image segmentation: A review of theory and applications. *IEEE Access* 9 (2021), 82031–82057. (Cited on p. 23)
- [42] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research* 15, 56 (2014), 1929–1958. (Cited on p. 17)
- [43] Veronika Thost and Jie Chen. 2021. Directed Acyclic Graph Neural Networks. In *Proceedings of the 9th International Conference on Learning Representations (ICLR)*. Open-Review. <https://doi.org/10.48550/ARXIV.2101.07965> (Cited on p. 14)

- [44] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *Proceedings of the 6th International Conference on Learning Representations (ICLR)*. OpenReview. (Cited on p. 13, 14)
- [45] H W West, M Siddique, L Volpe, R Desai, M Lyasheva, K Dargas, C Shirodaria, S Neubauer, K Channon, M Y Desai, D E Newby, J C L Rodrigues, D Adlam, E D Nicol, and C Antoniadou. 2021. Automated quantification of epicardial adipose tissue on CCTA via deep-learning detection of the pericardium: clinical implications. *European Heart Journal* 42, 1 (10 2021). <https://doi.org/10.1093/eurheartj/ehab724.0199> arXiv:https://academic.oup.com/eurheartj/article-pdf/42/Supplement_1/ehab724.0199/41054144/ehab724.0199.pdf ehab724.0199. (Cited on p. 7)
- [46] Harvey D. White and Derek P. Chew. 2008. Acute myocardial infarction. *The Lancet* 372, 9638 (2008), 570–584. (Cited on p. 7)
- [47] WHO. 2020. *The top 10 causes of death*. World Health Organization. <https://www.who.int/news-room/fact-sheets/detail/the-top-10-causes-of-death> Accessed: 18-05-2022. (Cited on p. 7)
- [48] Jelmer M. Wolterink, Tim Leiner, and Ivana Išgum. 2019. Graph convolutional networks for coronary artery segmentation in cardiac CT angiography. In *International Workshop on Graph Learning in Medical Imaging*. Springer, 62–69. (Cited on p. 6)
- [49] Jelmer M. Wolterink, Robbert W. van Hamersvelt, Max A. Viergever, Tim Leiner, and Ivana Išgum. 2019. Coronary artery centerline extraction in cardiac CT angiography using a CNN-based orientation classifier. *Medical Image Analysis* 51 (Jan 2019), 46–60. (Cited on p. 5, 6, 7, 8, 9, 10, 13, 25, 27, 42, 43, 44)
- [50] Arber Zela, Julien Siems, and Frank Hutter. 2020. NAS-Bench-1Shot1: Benchmarking and Dissecting One-shot Neural Architecture Search. <https://doi.org/10.48550/ARXIV.2001.10422> (Cited on p. 15)
- [51] Zhilu Zhang and Mert Sabuncu. 2018. Generalized cross entropy loss for training deep neural networks with noisy labels. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems (NeurIPS)*, Vol. 32. Curran Associates, Inc., 8792–8802. (Cited on p. 17)

Appendices

In these appendices supplementary material is included on the thesis regarding the data used for training the models, and experiments regarding the Cluster Pooling operator.

A Data

Property	Average	Std. Dev	Min	Max
# Nodes	5344	± 796	3200	8942
# Edges	10798	± 1610	6478	17978
Positive Labels	669	± 159	335	1079
Negative Labels	4675	± 821	2413	8391
Segment Length	34	± 24	1	192
Edge %	0.0774 %	± 0.0120 %	0.0450 %	0.1266 %
Positive %	12.8323 %	± 3.7867 %	5.8682 %	25.7767 %

Table 3: Description of the distributions of the data set, containing 200 subjects. The table shows the overall distribution of the number of nodes, edges, labels, length of linear node segments and the percentage of edges of a fully connected graph.

Feature Name	Range	Description
X,Y,Z Coordinates	$[0, 1]^3$	Node coordinates centralized around the left heart chamber
Radius	$[0, 1]$	Radius of the blood vessel predicted by the tracker
Seed Net Value	$[0, 1]$	Previous network probability of being an artery
Directional Vector	$[0, 1]^3$	Flow of blood, determined by current node and preceding
Order	$[0, 1]$	Depth in the graph
Direction entropy	$[0, 1]$	Direction probability from tracker

Table 4: Node Features in the data set. The coordinates and radius were normalized. The directional vectors were normed into the range of 1. The order was normalized within each linear segment of nodes, such that each leaf has value 1.

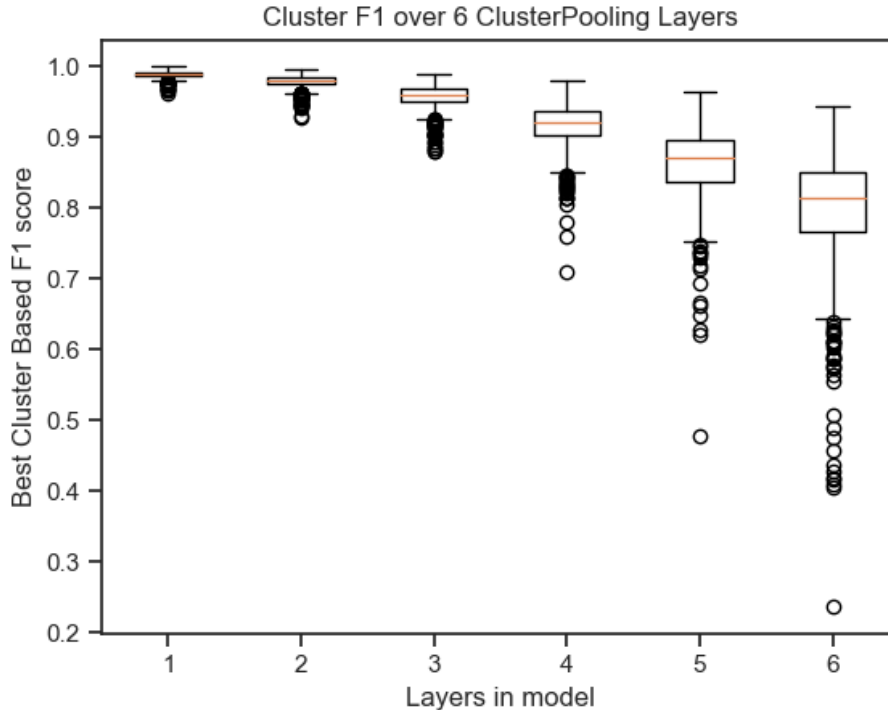


Figure 22: An analysis of Cluster Pooling’s impact in Graph U-NET II on the node classification task. What is visualized here over various training epochs, is the maximum achievable F1-score per layer if the classification was done per cluster instead. This can be interpreted as an diluting of the node labels, where the mixture of labels within a cluster decreases the separability of the data. This information was used to determine an acceptable depth for Graph U-NET II on the given problem.

B Experiments

Method	α	Dropout	Weight Decay	Width	# Layers
XGBoost	0.3	N/A	N/A	N/A	N/A
MLP	$5 \cdot 10^{-4}$	0.0	$1 \cdot 10^{-4}$	128	7
GCN	$2 \cdot 10^{-4}$	0.0	$1 \cdot 10^{-4}$	128	6
GCN Edge Pool	$5 \cdot 10^{-5}$	0.1	0.0	64	9
GCN Cluster Pool	$5 \cdot 10^{-5}$	0.1	0.0	128	9
Graph U-NET II Edge Pool	$5 \cdot 10^{-5}$	0.1	0.0	128	13
Graph U-NET II Cluster Pool	$5 \cdot 10^{-5}$	0.1	0.0	128	13

Table 5: Model hyperparameters for the experiments from Section 6. α is the learning rate.