Comparing U-Mamba and SegMamba Architectures for Coronary Artery Segmentation

Oliver Sanchez*, Lars Talian Stangebye-Hansen†
*Department of Mechanical and Industrial Engineering, Norwegian University of Science and Technology
†Department of Computer Science, Norwegian University of Science and Technology
*olivesa@ntnu.nu †larststa@ntnu.no

Abstract—Mamba based architectures, employing mamba state space models, have shown great promise in tasks requiring long-term context and efficiency in computation. One such task is medical image segmentation. A task which many mamba architectures have excelled at over the past months. However, leveraging mamba's efficiency and dependency abilities for coronary artery segmentation has, to the best of our knowledge, not been done and is therefore explored in this investigation. Two separate mamba architectures are explored, U-Mamba encoder and SegMamba. With a third architecture, nn-UNet, used as a baseline. The results show that the nn-UNet architecture yields slightly higher performance than both Mamba architectures. It was concluded, however, that these results were not an accurate reflection of the mamba architectures performance ceiling due to limitations in compute and the authors' inadequate ability to maximize the performance of the mamba models.

1 Introduction

For a great while transformers and CNN architectures have dominated the field of medical image segmentation due to their ability to see patterns in data and segment detailed structures. However, recently architectures leveraging Mamba state space models (SSMs) have shown immense results and potential to improve upon the transformer and CNN architecture in medical image segmentation. In light of these results this paper aims to investigate the performance of Mamba architectures in the segmentation of coronary arteries. To the best of our knowledge, there has been no published work of using Mamba architectures for coronary artery segmentation and hence the motivation to do as such. To further explain the motivation of the authors, it is worth highlighting the potential societal impact of the task being investigated. Computer tomography scans are of crucial importance in detecting coronary artery disease, and segmenting these scans can significantly speed up the process. Freeing up time and saving labor for the medical professionals that are always in a shortage. In addition to speeding up screenings, which can potentially save the lives of patients in need of early intervention. This investigation will, concretely, explore the use of U-Mamba encoder and SegMamba architectures in coronary artery segmentation, using nn-UNet as the baseline model.

2 Methods

2.1 Mamba SSM

The Mamba state space model is like a state space model in the sense that it gets an input that it applies a matrix operation on before adding noise and then producing an output. This output then represents not only the input and some noise but because it was multiplied by the internal matrix of the Mamba SSM it also carries information of previous states, this is what gives the Mamba SSM its long-distance dependency. The matrix operations (in essence just weights), are then backpropagated and updated to maximize the Mambas SSM's ability to represent an internal state of the previous data.

2.2 Dataset

Our investigation used the official dataset from the ASOCA Challenge, as it was introduced as a benchmark dataset within coronary artery segmentation research. It consists of 40 Computed Tomography Coronary Angiography (CTCA) scans, 20 labeled as diseased and 20 labeled as normal. For the purpose of our investigation, which focuses on segmentation of these scans, rather than full classification, we simply use the entire dataset of 40 CTCA scans, and corresponding labels, without the diseased and normal classes.

2.3 Model Requirements

For image segmentation of coronary arteries there are a few important features that an architecture needs to be capable of. Two of which are defining of a good model. Firstly, there is the ability of long-range spatial dependencies, i.e., knowing that there is an artery that is in the neighborhood and that it is likely that it will continue through the volume in question. Secondly, an architectures ability to understand the finer features is also of great significance in the model, as the arteries are detailed structures whose minute details can reveal a great deal about their health. Other aspects of the model that play a role is its ability to model the complex topology of coronary arteries, but also be very robust to differences in their topology as there is great variability, person to person, in their coronary arteries (see Figure 1). The purpose of segmenting the arteries is to allow for quicker analysis of the coronary arteries. This analysis is only valuable if it is reliable and accurate. Both reliability and accuracy in the human analysis is contingent on a detailed segmentation. We will discuss and explore

three architectures that meet these requirements to varying degrees.

(a) Labels for CTCA 1 (b) Labels for CTCA 17

Fig. 1: Displays the great differences in the structures of the coronary arteries in difference samples.

3 METHODOLOGIES

3.1 nn-UNet Architecture

The UNet is one of the most successful deep neural networks in history. Introduced in 2015 it is made of two parts, an encoder and a decoder [1]. The encoder takes in an input of relatively large dimensions, and applies convolutions to downsample it and extract feature maps, this is done in the same way that convolutional neural networks (CNNs) process images. These feature maps are a dimensionally reduced representation of the original input that summarizes some 'features' that it has. The relevance of these features to the task is initially close, if not totally, none. Feature selection is, initially, purely random. Training the model changes the weights of these feature extractors, essentially making them look for certain features that makes the model overall better at its task, as defined by the loss function. This aggregation of features, producing lower dimensions, occurs repeatedly, multiple times. Each convolution has different weights, finds different features, and operates at a differing depth (lower convolutions naturally can find features of larger spatial context). After the encoder (i.e. all these convolutions applied in sequence, with activation and pooling in between), the representation of the input is at its smallest dimension, and all the features of the entire input are represented by a relatively tiny tensor. This layer is called the bottleneck layer, for obvious reasons. From this bottle neck layer the decoder is supposed to reconstruct an image that is of the same dimensions as the original input. What the decoder does is essentially use all the encoded features of the input as an argument to a complex function that performs some complex function on the input. This is to say that the network as a whole can be looked at as a function, with an input and an output, and the operations that this function applies to the input is not defined definitely and is not a function in the common mathematical definition. The function is more of an abstract way of defining what the network does to an input. With data and backpropagation the network incrementally learns a better function to minimize the loss, hence it is the loss that defines what function to be learned and the architecture that decides how complex such a function can be. In the case of coronary artery segmentation this function is locating all data points that represent the coronary arteries and output them, while labeling the data points not representing coronary arteries as background. A key feature of the U-Net that enables it to do this reconstruction of sorts from such little information in the bottleneck layer are skip connections. Shallow layers in the encoder have their feature maps saved and then these are concatenated with the feature maps in the decoder, giving the decoder critical information about the fine details that might have been lost through the downsampling. Without these skip connections the decoder would not be able to recreate the intricate parts of the input. As specified earlier, these intricate parts are essential to the task of segmenting coronary arteries.

Seeing as the UNet was to serve as a baseline in our investigation we chose to use a popular version of the UNet that optimizes many of the classical tunable parts of the network [2]. We chose to use the nn-UNet as our baseline model. The nn-UNet is an architecture similar to the one outlined here above, but it optimizes many of the tunable parameters and functions making it very quick to implement, which is why we chose it. The architecture was proposed in 2018 and is built upon the classical UNet but with certain features that make it more streamlined. The nn-UNet automatically optimizes augmentation and preprocessing, in addition to employing loss functions and an adaptive learning rate scheduler that fit our task well. The exact nn-UNet pipeline used was from an open source github repository that was published based off of the LightM-UNet research Appendix B. However, the code is the same as the original nn-UNet release [3]. Seeing as the nn-UNet was purely intended as a baseline for this investigation the use of a rather automatic training pipeline was deemed reasonable as no great amount of work was planned with the nn-UNet. This does not mean that the architecture of the nn-UNet was not fine tuned to any extent. Fine tuning the nn-UNet was seen as essential to establish a fair comparison between the baseline and new models, giving the new models an otherwise unfair advantage. The nn-UNet pipeline had multiple advanced techniques to maximize performance of the network in addition to the traditional data augmentation and mixed loss function. The data augmentation used was spatial transformations, noise and blur addition, and intensity adjustments. A mix of cross-entropy and dice loss was used for loss. To fully maximize the performance the pipeline used deep supervision, incorporating a weighed perceived loss at intermediate levels into the overall training loss, in addition to compilation and cascade training. Compiling the network makes it run more efficiently, while cascade training uses lower resolution training to guide the training of the higher resolution. The trained nn-UNet in this investigation used only full resolution training, due to complex architecture-data errors with cascade training.

3.2 U-Mamba Encoder Architecture

As previously mentioned, one of the primary requirements in an architecture for coronary artery segmentation is the ability of maintaining a large context capture; being able to understand long-term (spatial) dependencies. How an artery goes from one location to another is important for the model to understand, as well as the local context (continuity

of the artery in particular). A standard UNet does have deep layers and hence the deep kernels have a large receptive field. Their feature maps are, however, simply a collection of many large scale features and do not, specifically, model how the data changes with depth. For this, introducing a state space model is ideal. A state space model (SSM) integrated into the UNet is essentially a layer that keeps an internal state that has memory, giving it context with respect to the time dimension. In the case of medical imaging the time dimension is height. The state space model implemented is a Mamba SSM. The state transition matrix, control-input matrix, and the observation matrix of the Mamba are modeled as learnable weights. The matrices are learned through backpropagation and optimized to adapt their function effectively based on the input, as well as developing an internal state that can improve the segmentation by providing long term dependency. The module does this by having three matrices as outlined earlier, these are represented as A, Bt, and Ct, in Figure 2. The function of Bt and A is to change the internal state from the previous state to the current state. Bt models how the input should affect the change of state, while A models how the previous state should affect the current state. The current state that is found from using A and Bt is then pushed through Ct that produces the output. One of the key things of the Mamba SSM compared to other SSM implementations is that each of the three matrix functions, A, Bt, and Ct are able to learn how to most effectively change depending on the input. That is to say the function that each of the matrix apples is dynamically altered depending on the input, and the way it dynamically changes is learned through backpropagation. This ability of changing the operation based on the input gives the Mamba SSM linear scaling with sequence length. The other key aspect of the Mamba SSM is long-term dependency, this would more correctly be long-range in our spatial application. The output of the Mamba SSM is dependent on both the previous internal state and the current input. Dynamically adapting its process based on the input essentially allows it to decide what input is relevant and focus its processing on that input, embedding only relevant information into the new current state and hence optimizing the representation of the state of the module. Having such an ability of only focusing on relevant information makes for a very efficient representation of the internal state and enables the long-term dependency of the Mamba SSM to be quite powerful. The U-Mamba encoder employed in our investigation is available in the same training pipeline as the nn-UNet Appendix B, it was proposed by Canadian researchers in a paper released in January of 2024 [5]. Tuning the U-Mamba encoder network, in contrast to the nn-UNet, required more careful consideration and insight. Adjustments made were mostly with the foreground oversampling, random patching dimensions, as well as the architecture and kernel sizes at different layers; since the UMambaEnc is created in the same self-configuring manner as the nn-UNet tampering with these features was initially not done but we found performance in changing them slightly such as increasing kernel sized to fully leverage the last 15% of the GPU memory. These were tuned in addition to tuning classical hyperparameters such as learning rate, decay, and batch sizes. The already implemented training did, similarly to the nn-UNet, use advanced techniques such as deep supervision, skip connections and residual blocks. Isolating the Mamba SSM layers as the single big difference between the architectures, laying the base for a fair comparison.

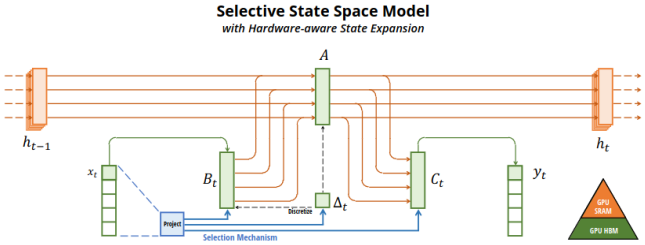


Fig. 2: an overview over how the data flows through a Mamba SSM, taken from the pioneering Mamba paper [4]

3.3 SegMamba Architecture

The third architecture that will be investigated was proposed by Xing et al in late February of 2024, and leverages the Mamba SSM for 3D segmentation specifically [6]. While the U-Mamba encoder architecture uses a Mamba SSM layer to capture long term dependency in no particular direction, SegMamba significantly improved upon this architecture by integrating what they call Tri-oriented Mamba (ToM). The Tri-oriented Mamba is in essence three parallel Mamba SSMs that each deal with dependency in one of the three orthogonal directions; axial, coronal, and sagittal. The output of each model is then processed together into a single representation for the dependencies of the data. This representation encapsulates the dependencies more comprehensively than the one-directional Mamba SSM. The 3-dimensional input into the mamba ssm is reduced to the three dimensions, and sequenced such that the SSM receives one dimensional vectors representing how the data changes in one part of the input in one of the three axis. To explain this mathematically, imagine the input is a 3x3 cube, then there are nine separate 1-d vectors that represent the propagation in each of the three axes, transforming the 27 floating points into 27 vectors of dimensions 1x3, or 81 floating points. The improved dependency representation, however, does not come without a computational cost. The SegMamba is a much larger and computationally heavy model compared to the U-Mamba encoder. In addition to the ToM, the SegMamba architecture has a gated spatial convolution (GSC) immediately prior. The idea behind the gated spatial convolution is to extract the features that will enable the ToM to build the most apt dependency representation. The GSC learns to summarize the most important features from the feature maps such that the ToM can focus all its processing power on the most important features and hence efficiently capture long-range dependencies. This adds another layer of complexity, especially in the training process. The ToM alone makes the network difficult to train due to its complex nature and the need for pristine dataset that ideally are quite large or at least have very specific data augmentation. The GSC increases the difficulty of tuning, making overfitting a larger risk. Having GSC and

ToM makes SegMamba quite sensitive to hyperparameter tuning and the network's performance can vary greatly in training. The SegMamba architecture used was taken from the open source repository which was introduced with the SegMamba [6]. In contrast to the two other architectures, the training script was not optimized and we had to write it from scratch. Implementing sampling, augmentation, loss and optimizer strategies based on what we thought would benefit the training process. These different implementations and their results are discussed further in section 3.3.

4 Results and Discussion

4.1 nn-UNet

The nn-UNet, defined as a baseline model, performed very well. The open source pipeline used had tuned initial settings that had been optimized during the automatic preprocessing of our data, hence the immediately good results. The initial configuration proved to be very successful getting a validation dice of above 0.76. Some slight architecture testing and increasing the foreground oversampling rate slightly resulted in the best nn-UNet model performance. With a mean validation dice of 0.7844. This is in fact the highest performance out of all three models. See Figure 3 for training graph

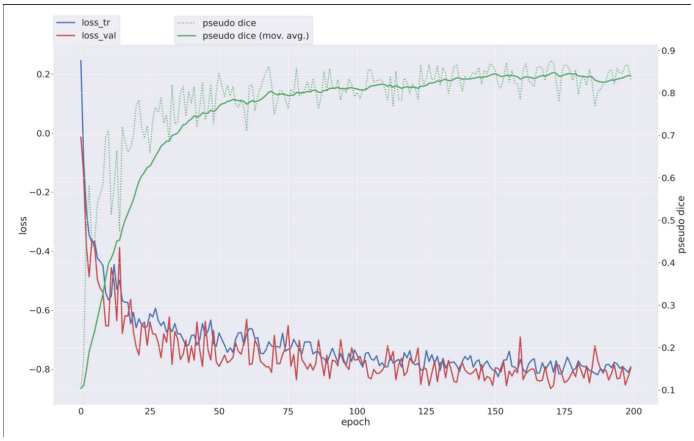


Fig. 3: Table showing the dice coefficient for each sample (train and validation) at each threshold.

4.2 U-Mamba Encoder

The earlier part of this paper outlines each architecture and why specifically we thought each architecture would be well suited for performing coronary artery segmentation. The U-Mamba encoder architecture was discussed earlier as being a significant improvement over the nn-UNet due to its long-range dependency abilities. Our results, however, do not reflect the theorized hypothesis. The mean dice coefficient from the validation set, the Mamba Encoder had an average dice coefficient of 0.7734, comparatively the nn-UNet had an average dice coefficient of 0.7844 on the same validation set. This goes to show the difficulty behind the tuning of the Mamba Encoder. A lot more hyperparameter tuning and network tuning was done on the Mamba Encoder architecture; the results, still, would not surpass the baseline model. The slightly worse results from the U-Mamba encoder is to some degree shocking, as the model did employ similar preprocessing and self-configurative architecture as the nn-UNet and simply from the automatically configured model we expected the U-Mamba to outperform the nn-UNet.

Trained on exactly the same split as the UNet and validation on the same dataset. The Mean Validation Dice: 0.7734. See Figure 4 for training graph.

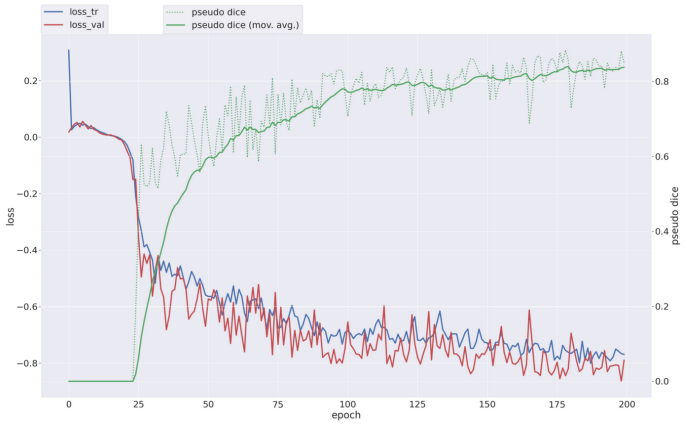


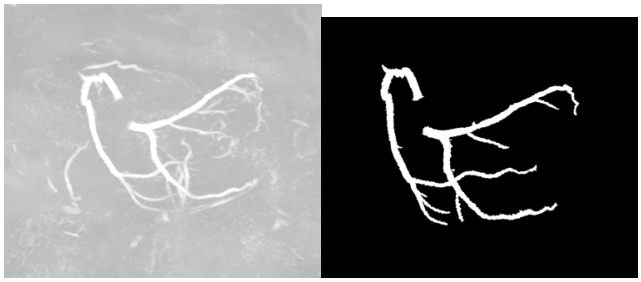
Fig. 4: Table showing the dice coefficient for each sample (train and validation) at each threshold.

4.3 SegMmaba

Multiple methods were tested for training the SegMamba architecture. As the dataset files are too large to load in an A100 80gb as the files are up to 512x512x244 in size, random cropping was utilized as primary data transformation. We quickly, without much else than random crop, trained models that gave (randomized and cropped) validation Dice Coefficient of up to 0.7. During training this technique resulted in large changes in dice coefficient and upon further inspection of the dice coefficient for each random crop there were crops (parts of the image) that had near zero coefficient and other with much higher. This showed that the model was essentially only modeling the background well as the foreground was proportionally small, this was further confirmed with inference and visualizations. Further implementations solved this to some degree with foreground oversampling based on a low resolution density-based probability map and sampling eight patches/crops per image instead of a single one. Patching at (224,224,92), each patch covered 8% of the volume and for each image for each batch, we covered 64% of the image in each batch with an oversampling of foreground; we trained each batch in eight stages due to computational limitations. Now the model showed great modeling of the arteries and both manually inspecting the dice for each crop and performing inference for visual confirmation we realized we needed to find a way to model the background slightly better as these parts were at times calculated to 0.1-0.2 dice while other parts were at 0.9+ (this is all training dice, validation dice was graphed but not inspected to guide specific changes as this would impact the validity of them as test data

[1]). The improvement of this poor background modeling performance was achieved by a mix of slightly reducing the foreground sampling in tandem with weighing the dice loss more than the focal loss. Another different method we implemented was dividing the image into a set of 16 smaller images of size 4x2x2, having four splits in depth, and two in both height and width. These individual patches would then be trained on sequentially, getting a uniform model that can predict each section to some degree, with the innate limitation of having background over-represented of course. The idea was to pre-train on the entire image before tuning the model by training on foreground oversampled patches. The results showed this idea to be of no real gain and the model performed poorly, not really transferring much learning. As somewhat detailed earlier, the best model was trained by computing a low resolution probability map from the labels to oversample the regions of interest. The patches were of sizes (224, 224, 96) covering roughly 8.2% of the entire region, using 8 random samples per image. The model was trained for 120 epochs, resulting in almost 8 of GPU hours on an RTX 4090. The entire script can be found on the github repository that is cited in the Appendix A.

The average validation coefficient for the SegMamba model was 0.7673. It is important to note that one dice coefficient is a strong outlier, image 40 (see Figure 6). The dice coefficient of image 40 is 0.654, this is 2.44 times the MAD value deviation from the mean and shows that it was an abnormally difficult scan. See Figure 5 for the inference and the label for this scan.



(a) Inference for CTCA 40 (b) Labels for CTCA 40

Fig. 5: Shows the inference and the label for the validation CTCA scan on which the SegMamba had its lowest dice coefficient.

Looking at the inference one can notice that the segmentation is in fact of decent quality, but this is visually. When evaluating the models visually we automatically choose the best threshold by looking at contrasts, but for the purpose of rigor the dice coefficient is calculated at the same threshold for the entire validation dataset. It was calculated for a threshold of 0.5; this same threshold was used for all three models. For a threshold of 0.7, scan 40 has a dice coefficient of .0722, a considerable 0.068 higher than at the 0.5 threshold. This same applies for the rest of the dataset and visually they often look much better than their dice coefficient would allude to. Considering that most imaging tools used by professionals allow you to optimize all of these aspects, e.g., changing the threshold and contrasts, the performances of each segmentation is more fairly represented visually or by using its highest dice coefficient, i.e., choosing the optimal threshold for each segmentation but this can of course not be done in practice as labels are not known.

1. Due to a misunderstanding of what was perceived to be a given test dataset of 20 CTCA scans in addition to the 40 CTCA scans, it was in fact not labeled, and hence validation data was used as the basis of comparison, this is a notably error and weakness in the rigor of this investigation.

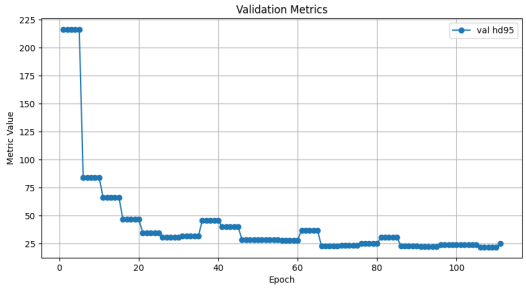
Filename	Thresh_0.1	Thresh_0.2	Thresh_0.3	Thresh_0.4	Thresh_0.5	Thresh_0.6	Thresh_0.7	Thresh_0.8	Thresh_0.9
1	0.239	0.427	0.568	0.654	0.720	0.734	0.730	0.685	0.526
2	0.406	0.825	0.902	0.942	0.936	0.910	0.852	0.758	0.585
3	0.504	0.735	0.839	0.867	0.895	0.874	0.830	0.730	0.535
4	0.527	0.828	0.872	0.853	0.828	0.788	0.741	0.671	0.556
5	0.439	0.633	0.731	0.796	0.807	0.793	0.763	0.696	0.559
6	0.507	0.686	0.830	0.916	0.925	0.883	0.823	0.741	0.555
7	0.469	0.686	0.757	0.767	0.774	0.761	0.727	0.724	0.606
8	0.545	0.726	0.847	0.899	0.875	0.838	0.791	0.717	0.564
9	0.597	0.771	0.879	0.923	0.933	0.908	0.852	0.763	0.617
10	0.550	0.768	0.822	0.832	0.798	0.734	0.662	0.561	0.413
11	0.518	0.789	0.869	0.899	0.898	0.875	0.820	0.726	0.530
12	0.490	0.770	0.867	0.888	0.889	0.858	0.809	0.741	0.596
13	0.488	0.653	0.761	0.856	0.880	0.839	0.771	0.683	0.558
14	0.492	0.617	0.687	0.758	0.816	0.814	0.778	0.721	0.582
15	0.441	0.692	0.874	0.918	0.905	0.872	0.799	0.705	0.523
16	0.439	0.624	0.771	0.864	0.897	0.903	0.858	0.777	0.618
17	0.580	0.829	0.917	0.927	0.906	0.862	0.801	0.717	0.563
18	0.573	0.778	0.898	0.937	0.917	0.882	0.832	0.760	0.605
19	0.665	0.758	0.802	0.832	0.807	0.775	0.738	0.686	0.585
20	0.499	0.712	0.820	0.834	0.835	0.805	0.759	0.687	0.565
21	0.584	0.748	0.850	0.893	0.911	0.883	0.829	0.751	0.594
22	0.542	0.695	0.772	0.810	0.820	0.826	0.784	0.685	0.539
23	0.618	0.804	0.900	0.908	0.899	0.872	0.842	0.786	0.679
24	0.530	0.675	0.751	0.826	0.860	0.854	0.806	0.727	0.566
25	0.677	0.786	0.833	0.830	0.783	0.734	0.667	0.606	0.507
26	0.490	0.643	0.794	0.845	0.903	0.889	0.842	0.792	0.700
27	0.658	0.762	0.838	0.859	0.863	0.852	0.814	0.749	0.628
28	0.546	0.755	0.863	0.901	0.889	0.852	0.809	0.737	0.589
29	0.539	0.713	0.796	0.865	0.923	0.948	0.923	0.839	0.685
30	0.623	0.798	0.896	0.918	0.908	0.870	0.822	0.734	0.571
31	0.663	0.831	0.919	0.927	0.897	0.850	0.797	0.738	0.633
32	0.552	0.747	0.855	0.916	0.926	0.911	0.868	0.800	0.658
33	0.682	0.801	0.831	0.864	0.845	0.808	0.762	0.699	0.600
34	0.666	0.845	0.884	0.872	0.844	0.820	0.772	0.709	0.591
35	0.586	0.736	0.819	0.904	0.919	0.900	0.871	0.813	0.702
36	0.597	0.717	0.853	0.926	0.919	0.897	0.842	0.780	0.673
37	0.437	0.721	0.830	0.893	0.928	0.932	0.887	0.814	0.679
38	0.574	0.609	0.693	0.786	0.777	0.741	0.698	0.633	0.548
39	0.570	0.808	0.888	0.918	0.912	0.857	0.793	0.710	0.563
40	0.455	0.538	0.554	0.615	0.654	0.688	0.722	0.696	0.622
Averages	0.539	0.726	0.818	0.861	0.866	0.842	0.797	0.726	0.589

Fig. 6: Table showing the dice coefficient for each sample (train and validation) at each threshold.

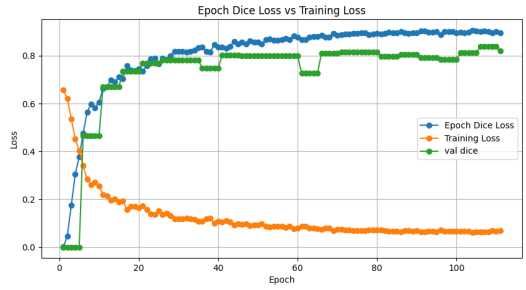
It is evident from Figure 8 that a large amount of training is necessary to fully utilize the complex architecture that is SegMamba. The validation dice was steadily increasing throughout the 110 epochs and would it not have been for an unfortunate disconnect the model might have improved even further. The 95% percentile HausDorff distance score did not decrease much after 70 epochs and was not one of immense focus for this investigation. At its lowest in validation the HD95 went to 1.4 in CTCA 17. For the HD95 values for all CTCA scans see Figure 7

Index	Value	Index	Value	Index	Value	Index	Value
1	48.416	11	1.000	21	3.742	31	1.414
2	1.000	12	1.414	22	3.606	32	1.414
3	1.000	13	1.000	23	5.030	33	4.123
4	14.965	14	3.000	24	2.000	34	1.414
5	50.339	15	1.000	25	4.243	35	1.000
6	1.000	16	1.000	26	1.000	36	1.000
7	1.414	17	1.000	27	1.414	37	1.000
8	1.000	18	1.414	28	1.000	38	24.372
9	1.000	19	18.366	29	1.000	39	1.000
10	3.742	20	16.673	30	1.000	40	54.778

Fig. 7: Shows the HD95 metric for each of the 40 inferences using the SegMamba model.



(a) Graph showing the HD95 metric (blue) with each epoch.



(b) Graph showing the val (green) and train (blue) dice coefficients and training loss (orange)

Fig. 8: SegMamba training graphs

5 Conclusion

The investigation employed three models in the segmentation of coronary arteries. The nn-UNet model, which was chosen as a baseline model performed extremely well. Our hypothesis were that the architectures leveraging Mamba would produce significantly better results than the nn-UNet given their theoretical complexity and suitability for the task. The discrepancy between the hypothesis based on theory and the results can have many explanations. An evident reason is that the more complex models are not necessarily better given limited test data and limited compute. Simpler models that can perform a task well should never be deemed insufficient due to their lack of flash or for being relatively archaic (in deep learning this would mean more than a year old). Looking at the results from the original papers in applications not much different from the segmentation of coronary arteries it is highly likely that the Mamba based models would have a theoretical higher ceiling for the task of segmenting coronary arteries. Further, for all the underlying theory outlined in the methods section it is clear that the Mamba architectures are in theory better suited models for this task. Our conclusion is that the results of this investigation do not accurately reflect the performance ceiling of Mamba architectures within coronary artery segmentation. Mostly due to the task of training and tuning these models correctly being an involved one. Our dedication and effort in maximizing the results from the model is not in question, and we plan on pursuing the models in future work. The performance of the final SegMamba model is one of significance to us, as results for this model were very poor for much of the project duration and only after testing, debugging and theorizing new training implementations did we get somewhat satisfactory results.

Appendix A
SegMamba Training Code

For SegMamba training code and other further details, see the following GitHub repository: https://github.com/olivernakamoto/MambaCoronaryArterySegmentation.

Appendix B
LightM-UNet repository

For LightM-UNet GitHub repository that has both the nn-UNet and U-Mamba encoder pipelines used: https://github.com/MrBlankness/LightM-UNet

Acknowledgments

We would like to thank the Computer Science Department of NTNU for giving us access to the Cybele Lab throughout our project and for allowing a Mechanical Engineering bachelor student into a Deep Learning focused Master's course.

References

[1] O. Ronneberger, P. Fischer, and T. Brox, "U-NET: Convolutional Networks for Biomedical Image Segmentation," *arXiv.org*, May 18, 2015. Available: https://arxiv.org/abs/1505.04597

[2] F. Isensee, J. Petersen, A. Klein, D. Zimmerer, P. F. Jaeger, S. Kohl, J. Wasserthal, G. Koehler, T. Norajitra, S. Wirkert, and K. H. Maier-Hein, "NNU-NET: Self-adapting Framework for U-Net-Based Medical Image Segmentation," *arXiv.org*, September 27, 2018. Available: https://arxiv.org/abs/1809.10486

[3] W. Liao, Y. Zhu, X. Wang, C. Pan, Y. Wang, and L. Ma, "LightM-UNET: Mamba assists in lightweight UNET for medical image segmentation," *arXiv.org*, March 8, 2024. Available: https://arxiv.org/abs/2403.05246

[4] A. Gu, and T. Dao, "Mamba: Linear-Time Sequence Modeling with Selective State Spaces," *arXiv.org*, December 1, 2023. Available: https://arxiv.org/abs/2312.00752

[5] J. Ma, F. Li, and B. Wang, "U-MambA: Enhancing long-range dependency for biomedical image segmentation," *arXiv.org*, January 9, 2024. Available: https://arxiv.org/abs/2401.04722

[6] Z. Xing, T. Ye, Y. Yang, G. Liu, and L. Zhu, "SEGMAMBA: Long-range Sequential Modeling MAMBA for 3D medical image segmentation," *arXiv.org*, January 24, 2024. Available: https://arxiv.org/abs/2401.13560