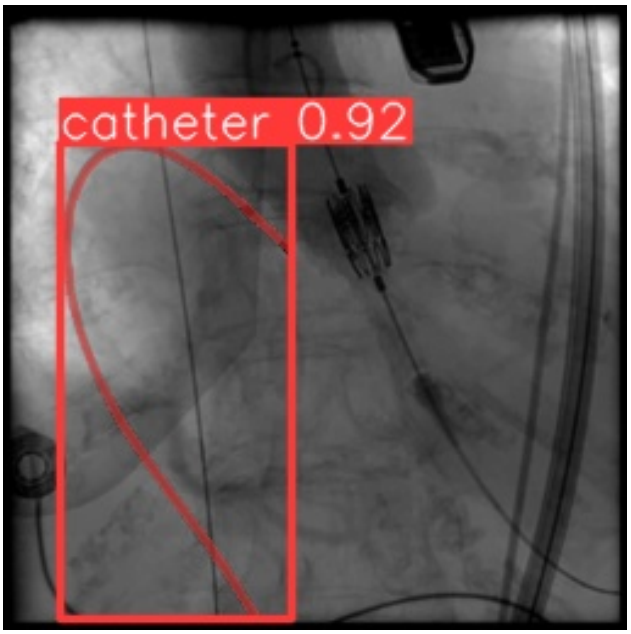
CHALMERS
UNIVERSITY OF TECHNOLOGY



Segmentation in X-ray Fluoroscopy Utilizing Virtual Simulations of Cardiovascular Procedures

Assessing and improving the quality of simulated images as training data for segmentation models using style transfer

Master's thesis in Biomedical Engineering

RASMUS ANDERSSON
MARTIN EKERSTEDT

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2024
www.chalmers.se

Master's thesis 2024

Segmentation in X-ray Fluoroscopy Utilizing Virtual Simulations of Cardiovascular Procedures

Assessing and improving the quality of simulated images as training data for segmentation models using style transfer

Rasmus Andersson
Martin Ekerstedt

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Signal processing and Biomedical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2024

Segmentation in X-ray Fluoroscopy Utilizing Virtual Simulations of Cardiovascular Procedures
Assessing and improving the quality of simulated images as training data for segmentation models using style transfer
Rasmus Andersson
Martin Ekerstedt

Supervisor: Henrik Storm, Mentice AB
Supervisor: Ida Häggström, Department of Electrical Engineering
Examiner: Ida Häggström, Department of Electrical Engineering

Master's Thesis 2024
Department of Electrical Engineering
Division of Signal processing and Biomedical engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Visualization of catheter segmentation using YOLOv8, image from *in vivo* TAVI procedure.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2024

Segmentation in X-ray Fluoroscopy Utilizing Virtual Simulations of Cardiovascular Procedures
Assessing and improving the quality of simulated images as training data for segmentation models using style transfer
Rasmus Andersson
Martin Ekerstedt
Department of Electrical Engineering
Chalmers University of Technology

Abstract

This project aims to assess the quality of simulated fluoroscopy images from the company Mentice and their surgical simulator as training data for neural networks. To do this, a simulated dataset was collected by extracting images and corresponding information from the simulation to create labels for different objects in the image automatically. The objects segmented in the images are catheters. The evaluation uses a dataset from diagnostic and Trans-catheter Aortic Valve Implantation (TAVI) *in vivo* procedures. A model called Neural Neighbor Style Transfer was used to adapt the domain of the images to make them look more like real images to improve the performance. To perform segmentation two models U-Net and YOLOv8 were used. The major finding was that a Dice score of 0.8803 was achieved using pretraining on style-transfer images using YOLOv8. It was also found that by pre-training using simulated images performance was on average increased by 0.3%/0.6% (Dice/IoU) for simulated images and 0.9%/1.2% for style-transfered images. A model trained purely on simulated images could achieve a Dice score of 0.5182. Overall it can be concluded that the simulated images help improve performance and style-transfer also shows promise for improving the metrics.

Keywords: Fluoroscopy, X-ray, Mentice, Style Transfer, Segmentation, U-Net, YOLO, TAVI, Simulation, Catheter.

Acknowledgements

Firstly, we would like to thank Mentice AB for providing this opportunity to work on this Master's thesis. Also, we would like to thank our supervisor at Mentice Henrik Storm. Finally, we would like to thank our supervisor and examiner at Chalmers University of Technology Ida Häggström for providing us with feedback and helping along the way.

Rasmus Andersson & Martin Ekerstedt, Gothenburg, June 2024

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

AP	Average Precision
BCE	Binary Cross Entropy
CNN	Convolutional Neural Network
COCO	Common Objects in Context
FN	False Negative
FP	False Positive
GUI	Graphical User Interface
IoU	Intersection over Union
ML	Machine Learning
NNST	Neural Neighbor Style Transfer
PCI	Percutaneous Coronary Intervention
ReLU	Rectified Linear Unit
SiLU	Sigmoid Linear Unit
TAVI	Trans-catheter Aortic Valve Implantation
TN	True Neagative
TP	True Positive
YOLO	You Only Look Once

Contents

List of Figures

List of Tables

1
Introduction

This chapter gives a general background and a basic description of the project. It acknowledges similar work and describes the collaborative partner of the project, Mentice. Mentice is a company that develops simulators for a certain family of medical procedures called endovascular procedures. More about that in Section 1.6.

1.1 Project Description

Fluoroscopy is a medical imaging method used to get a live stream of X-ray images, allowing the surgeon to see what is happening inside the patient in real time. X-ray fluoroscopy is explained in more detail in section 2.1.

This project used simulated fluoroscopy images to train two different computer vision models. The aim was to evaluate the feasibility of using the simulated fluoroscopy images as training data. More specifically, the models that were trained were segmentation models (models that detect objects in an image and identify which pixels represent which objects). The segmentation models were trained to detect catheters in the simulated fluoroscopy images. A catheter is a thin tube that is inserted into the patient to perform a medical procedure.

The performance of the models was then evaluated by using them to detect and mark catheters in a set of real fluoroscopy images from a publicly available dataset. Furthermore, the project used several different methods to explore the usefulness of the simulated fluoroscopy images as training data. The different methods and models used resulted in many combinations of experiments that were carried out, evaluated, and compared. The experiments are later defined in Section 3.1.4.

1.2 Motivation

The main task in this project was to perform segmentation in fluoroscopy images. One reason to segment catheters in fluoroscopy images is that it can help visualize the catheters in situations where the catheters are hard to see. However, a common problem with using segmentation models for medical images is the lack of annotated training data. This is mainly due to patient privacy protection laws. Several previous studies have mentioned the lack of training data as one of the main limiting factors [8, 9]. One workaround to this issue is to use simulated images, which can be accessed in larger quantities, to train segmentation models.

1.3 Objective

Broadly, the problem that this thesis will try to address is the problem of lack of data for AI applications in healthcare as described in section 1.2. This is done by investigating the feasibility of using simulated images from Mentice as training data instead. This project evaluates how well this approach works for fluoroscopic X-ray images using images generated with the Mentice system.
To address this issue, the project explores the following research questions:

1. To what extent can the segmentation capabilities of a model trained on simulated images from Mentice generalize to real fluoroscopic X-ray images?
2. To what extent can the simulated images increase the performance of the segmentation model?
3. How do the results compare to previous studies performing segmentation on real fluoroscopic X-ray images?
4. How do different model architectures and training strategies such as style-transfer and fine-tuning on real images affect the performance of the models?

1.4 Scope

This section introduces some specific limitations placed on the project to create a well-defined problem and focus the work. This section also clarifies how this thesis differs from other works.

1.4.1 Limitations

In short, the limitations explicitly placed on the project are:

- The use of raw simulated data is limited to that generated by the Mentice system.
- The real data used only contains annotations for one class, catheters.
- All numerical results will be generated from the segmentation of catheters.
- All models used will be neural networks.
- Novel architectures will not be developed.
- The image domain will be limited to fluoroscopic X-ray images.
- Optimizing inference time to enable live applications will not be considered.
- All training of models will be done locally.
- Models size/training memory usage will be limited by local hardware.

Some of the limitations are further detailed, motivated, and discussed in Chapter 5.

1.4.2 Novelty

Despite certain commonalities with existing research, this thesis introduces novel elements that distinguish it from prior works. Similar to some of the papers discussed in section 1.5, simulated data will be used to train the models. However, this specific method of using simulated procedures to generate the simulated data has not been found for this imaging modality. Finally, we will use a domain adaptation method not found in any similar work in medical image analysis.

1.5 Similar Work

This section reviews relevant literature to position the current project within the existing body of work, focusing on catheter segmentation in medical imaging. Also, this section helps establish a general idea of what performance can be expected from a catheter segmentation model.

Two papers that had a similar idea as the intended method for this project are described in the following two paragraphs [6, 7]. Ambrosini *et.al* produced a segmentation model for catheter segmentation in X-ray fluoroscopy images. They used a CNN-based approach for segmentation and reported results of a median tip distance error of 0.9 mm and a median centerline distance error of 0.2 mm [6].

The other study by Gherardini *et.al* [7] utilized a similar CNN-based approach as Ambrosini *et.al*. What primarily differentiates them is that Gherardini et.al utilized synthetic data, phantom data (data obtained from fluoroscopy videos with experiments carried out on a silicone aorta phantom which is a physical model of the aorta made out of silicone), and data from in-vivo operations using transfer learning. This paper also used a significantly smaller model but achieved better results with a Dice coefficient of 0.58 compared to the previous 0.53 [7]. However, they made this comparison using their dataset and not the results of the original paper, it is also unclear if they included the post-processing methods used by Ambrosini *et.al*.

Another two papers looked at catheter segmentation in X-ray [25, 26], however, these were chest X-ray images and not fluoroscopy images. The older of the two papers by Suramanian *et.al* [26] performed segmentation of the catheters and then classified what type of catheter it was. They achieved an accuracy of 85.2%. The newer paper by Boccardi *et.al* [25] performed segmentation of the catheters and also differentiated between different instances of different catheters and their overlapping regions. Boccardi *et.al* achieved a dice score of 0.739.

There are also some papers using synthetic data to train segmentation models for other medical imaging modalities. Kreitner *et.al* [8] used a segmentation model that was trained on synthetic data. The segmentation model was meant to segment vessels in optical coherence tomography angiography images. The synthetic data was generated using a generative model and they used a CNN for segmentation. They showed that they were able to get a segmentation model that performed well, a Dice score of up to 0.912, without using annotated real images [8].

Danilov *et.al* [12] used semi-synthetic data for catheter segmentation. The semi-synthetic data was generated by inserting randomly shaped catheters in the heart cavity in 3D echocardiography. They reported a dice coefficient of 92.6% when using synthetic and real data for training compared to 86.5% when using real images [12].

Wu *et.al* [13] used methods not based on deep learning methods to perform segmentation in X-ray fluoroscopy images. They reported results of an error of 1.79mm for

the catheter median center line distance [13]. Comparing this to Ambrosini *et.al*, the CNN model performs significantly better.

1.6 Mentice

This project was done in collaboration with Mentice, a company known for its endovascular procedure simulators. They describe their simulator as a "flight simulator" for medical doctors and students. Endovascular procedures are a family of medical procedures where an instrument is inserted into the vessel and navigated to where the medical issue is e.g. myocardial infarction, and then some procedure is performed to fix it. In Mentice's simulator, you instead insert the medical instrument, for example, a catheter into a box that has sensors and feedback actuators in it. The sensory data is then sent to a computer with the simulation running which models the catheter and relevant blood vessels in 3D. This model is projected to a 2D image as if it were seen through a fluoroscope. This happens several times per second to produce a video stream of images.

Mentice gave us access to their simulation system and its simulated fluoroscopic images. Full access to the source code was granted which allowed for modifications such that the system will output images well-suited for use as training data. They also provided guidance for operating their system in addition to input and expertise on the project overall.

2
Theory

This chapter presents the most relevant theoretical aspects that will be used in the method. This starts with a brief introduction to the relevant medical- devices, imaging methods, and procedures is included. Additionally, it includes technical information about relevant Machine Learning (ML) models and the theory behind their implementation, use, and evaluation.

2.1 X-Ray Fluoroscopy

Fluoroscopy is a medical imaging method that uses the X-ray imaging modality to produce a continuous time sequence of images. This imaging method can be found within catheterization laboratories in hospitals where different coronary procedures can be performed.

X-ray imaging differentiates tissues and objects based on their differential absorption of X-ray photons, which are recorded to create grayscale-intensity images. The main types of noise in X-ray are Poisson-, salt and pepper- and speckle-noise [29] and it can generally be reduced by increasing the radiation.

Several medical devices are essential for procedures performed under fluoroscopy. One such device is the introducer tube that is located at the groin or arm, this tool is the interface between the cardiovascular system and the outside; all other tools enter the cardiovascular system through the introducer tube. The guide wire is a tool used to navigate through the cardiovascular system to help other tools, primarily the catheter, reach their desired destination. A catheter is used as a tube for contrast fluid and for helping with the deployment of other tools such as stents and balloons. Stents and balloons are tools intended to deal with certain medical problems such as myocardial infarction [33].

For medical purposes, different cardiovascular procedures can be performed in the cauterization laboratories depending on the need. One such procedure done for diagnostic purposes is a coronary angiogram which is done to look for narrowed or blocked vessels in the heart [32]. Another procedure is Trans-catheter Aortic Valve Implantation (TAVI) done to implant an artificial heart valve [31]. Another procedure is Percutaneous Coronary Intervention (PCI) which is done to treat narrowed heart vessels [30].

2.2 Segmentation

Image segmentation using AI primarily consists of three different types of segmentation. These are panoptic, semantic, and instance segmentation. Semantic segmentation is where each pixel in an image is assigned a class segmenting the image into different regions that belong to a certain class. Instance segmentation is where regions of the image that are considered separate entities are segmented without directly classifying what class that part of the image belongs to. Panoptic segmentation is similar to instance segmentation but also takes the class of the segment into consideration.

Many models that can perform segmentation exist. Two of these are YOLOv8 and U-Net [1, 20]. YOLOv8 is a flexible model that can perform classification, object detection, instance segmentation, and pose estimation; it is pre-trained on the COCO dataset [11]. YOLOv8 uses different but similar architectures for different tasks, the primary difference is that the heads are different by necessity. However, all the models contained in YOLOv8 are convolutional neural networks (CNNs) with CSPDarknet53 [28] as the backbone [1, 2]. Also, the segmentation model produces an object detection bounding box with a class and an object confidence level. The object confidence is akin to the probability of a true positive detection. An example of this is shown in Figure 2.1.

Figure 2.1: Inference using YOLOv8x-seg.pt using an image of a street [10] and displaying detected objects with an object confidence threshold of 0.25

For the segmentation models in YOLO, there are five sizes with different numbers of parameters and they achieve the following results on the COCO [11] dataset [1].

Model	Size (pixels)	mAP^{box}(%) ↑	mAP^{mask}(%) ↑	Params (M)
YOLOv8n-seg	640	36.7	30.5	3.4
YOLOv8s-seg	640	44.6	36.8	11.8
YOLOv8m-seg	640	49.9	40.8	27.3
YOLOv8l-seg	640	52.3	42.6	46.0
YOLOv8x-seg	640	**53.4**	**43.4**	71.8

Table 2.1: Results comparing different sizes of YOLOv8-seg on the COCO dataset taken from the YOLOv8 docs [1]. mAP^{box}(%) ↑ is the mean average precision for the object detection bounding box and mAP^{mask}(%) ↑ is the mean average precision for the segmentation mask. These metrics are futher described in 2.3.

Moreover, the other model U-Net [20] is commonly used for segmentation tasks. U-Net is an encoder-decoder CNN with an architecture shown in Figure 2.2. The simplicity and effectiveness of the model make it a common model to use as a baseline for performance when comparing results for segmentation tasks, specifically semantic segmentation.

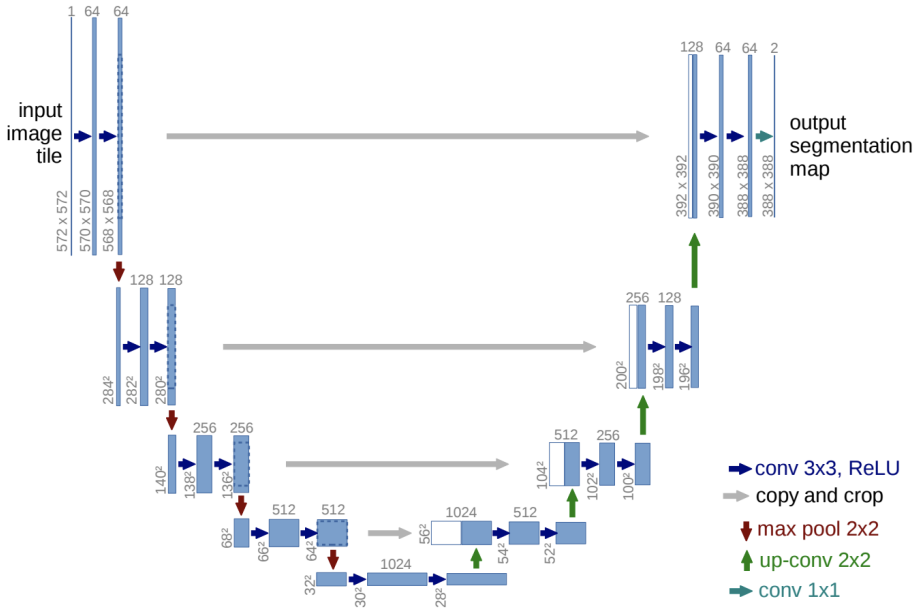


Figure 2.2: The architecture of U-Net from the original paper [20].

Furthermore, as seen in Figure 2.2 the U-Net model contains a decoder using convolutional layers and downsampling by using the max pooling operation. The decoder part uses convolutional layers with skip connections and concatenations with the results outputs from the encoder. Also, the up-sampling is done with convolutions. Other more recent models such as Trans U-net [34] and Mamba U-net [35] use a similar design.

2.3 Metrics

For each image, the model will predict segments that belong to a certain class. Using the intersection and union of this area with the ground truth label we can calculate the intersection over union (IoU) as follows:

$$IoU_{class} = \frac{label \cap prediction}{label \cup prediction} \tag{2.1}$$

Also, we can generate metrics on a per-pixel instead of a per-segment basis. We can do this by recording the true positive (TP), true negative (TN), false positive (FP), and false negative (FN) for each pixel. Using this we can calculate accuracy, precision, specificity, and sensitivity/recall as follows:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.2}$$

$$precision = \frac{TP}{TP + FP} \tag{2.3}$$

$$specificity = \frac{TN}{TN + FP} \tag{2.4}$$

$$sensitivity = recall = \frac{TP}{TP + FN} \tag{2.5}$$

When evaluating ML models it is common to use precision and recall, however, for tests in the medical field, it is more common to use sensitivity and specificity. Also, using this we can compute the total intersection over the union for the entire image as:

$$IoU_{image} = \frac{TP}{TP + FP + FN} \tag{2.6}$$

Then by taking the mean of all the obtained IoUs, we can get the mean IoU (mIoU). Furthermore, YOLO uses metrics like mAP50 which is the mean average precision for detection when $IoU \geq 50\%$. To clarify, a segmentation is classified as a TP if $IoU \geq 50\%$ meaning that it is defined per segmentation instance and not on a per-pixel basis (for the mAPXX metric). Also, higher or lower numbers could be used e.g. mAP75 which would be a higher threshold for considering a segmentation as a TP meaning that the precision will be lower: mAP75 < mAP50. Specifically, YOLOv8 uses precision, recall, mAP50, and mAP50-95; mAP50-95 is calculated at varying IoU thresholds from 0.5 to 0.95, in steps of 0.05 [1].

Finally, a metric similar to IoU is the Dice coefficient (mathematically equivalent to F1-score) and it can be calculated as follows:

$$Dice = \frac{2 * (label \cap prediction)}{label + prediction} = \frac{2TP}{2TP + FN + FP} \tag{2.7}$$

To summarize, multiple metrics were introduced, and some of the metrics were introduced based on the result metrics produced by the models previously described. Also, some of the metrics were introduced since they are used in the papers discussed in Section 1.5.

2.4 Loss Functions

Loss functions are utilized when training neural networks to update the weights using the backpropagation algorithm. This is done using gradient descent where the gradient of the loss function with respect to the weights of the network is used.

One simple loss function for segmentation tasks is dice loss which is simply defined as $1 - Dice$. So if this is minimized, the model parameters that produce a high dice score will likely be found. Similarly, IoU loss can be calculated as $1 - IoU$.

Another loss function that can be used for binary classification tasks is Binary Cross Entropy (BCE). This function is defined as shown in Equation 2.8

$$BCE = -\frac{1}{N}\sum_{i=1}^{N} y_i * log(\hat{y}_i) + (1 - y_i) * log(1 - \hat{y}_i) \tag{2.8}$$

where $\hat{y}_i$ is the predicted value for prediction with index i and y_i is the corresponding ground truth value. The averaging is done over all pixels N typically all values in a batch. In this case, i denotes the pixel number, $y_i \in 0, 1$ denotes background/foreground in the ground truth label, and $\hat{y}_i$ is the predicted pixel value.

2.5 Activation Functions

For binary classification tasks, a sigmoid activation function is commonly used. Mathematically the sigmoid function, also called the logistic function is defined as shown in Equation 2.9. There is also a piecewise linear approximation of the sigmoid function called hard sigmoid, both can be seen in Figure 2.3.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.9}$$

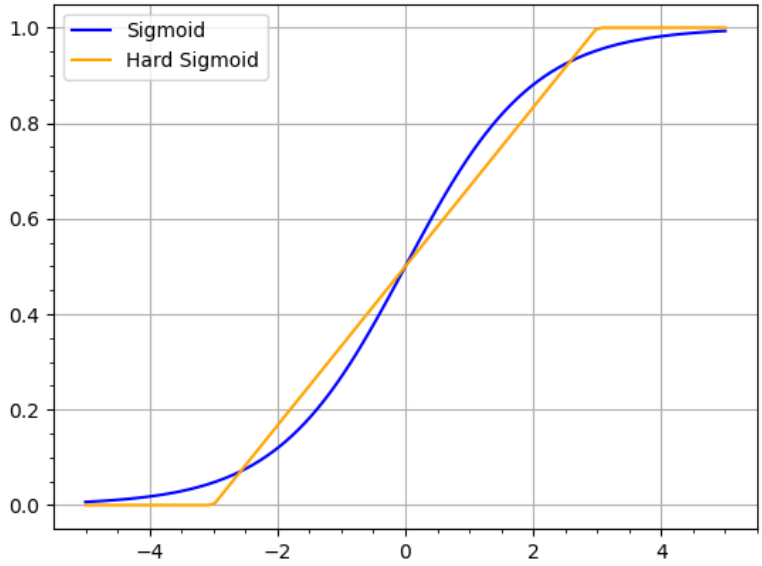


Figure 2.3: The sigmoid and hard sigmoid activation functions plotted for input values in the range from -5 to 5

One of the most used activation functions historically for neural networks is the Rectified Linear Unit (ReLU) which is defined as shown in Equation 2.10. A smoother similar function is the Sigmoid Linear Unit (SiLU) [17] defined as shown in Equation 2.11. Both functions are illustrated in Figure 2.4. SiLU can also be seen as a special case of the Swish activation function [18] with $\beta = 1$ as shown in Equation 2.12. However, in Swish β is a trainable parameter.

$$ReLU(x) = max(0, x) \tag{2.10}$$

$$SiLU(x) = x\sigma(x) \tag{2.11}$$

$$Swish(x) = x\sigma(\beta x) = \frac{x}{1 + e^{-\beta x}} \tag{2.12}$$

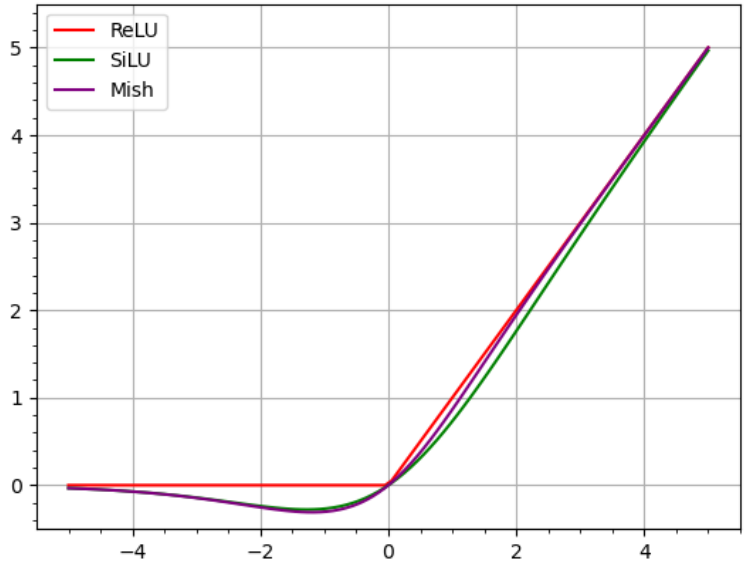


Figure 2.4: The ReLU, SiLU, and Mish activation functions plotted for input values in the range from -5 to 5

Another activation function is the Mish activation function [19] defined as shown in Equation 2.13. It is also illustrated in Figure 2.4. Where there is a high similarity in the shape of the Mish and SiLU activation functions. The Mish activation function is the primary activation function used in YOLOv8.

$$Mish(x) = x * tanh(ln(1 + e^x)) \tag{2.13}$$

2.6 Data Augmentation

One common way to increase the performance of neural networks for image analysis is data augmentation. Some of the relevant methods are listed below:

- Rotation, rotating the image
- Flipping, flipping the image up/down or left/right
- Scaling, zooming in/out on the image, up or downscaling
- Shifting, shifting the image using different translations up/down/left/right
- Contrast, increasing or decreasing the contrast in the image
- Brightness, increasing or decreasing the brightness of the image
- Noise, applying noise from some distribution, typically Gaussian, to an image
- Mosaic, combines multiple training images into one composite image.

These augmentation methods are typically done during training usually using a random range. This means that during training each batch will get augmented randomly within the range meaning that the augmentation values will likely differ between the batches and for the same image will likely differ for each epoch. Also, some data augmentation can be done as a pre-processing step.

2.7 Domain adaptation

A method that is relevant when it comes to training a model on a synthetic dataset with the purpose of working on real images is domain adaptation. Domain adaptation seeks to learn a model trained on a source domain (e.g. synthetic images) to be generalized to a target domain (e.g. real images) by minimizing the difference between domain distributions [15].

There are some alternatives to how it could be implemented. One is domain adaptation by feature alignment where the model is modified to work better on the target domain by introducing real unannotated data that aligns the features of encoded data in a latent space with the data from the target domain. Another alternative is domain translation with CycleGAN [21], this allows for images to be generated that look like the images in the target domain [16]. Both of these methods are unsupervised methods since in both cases annotated images from the target domain are not needed.

Also, style-transfer is a method similar to domain translation with CycleGAN. Here you use the images from the target domain to make images from the training data set look more like the target domain. For example, make simulated images look like real ones. At inference using a model called Neural Neighbor Style-Transfer (NNST) [14] only one image from the target domain is used. This could lead to a biased output if all augmented images using this method are augmented with the same real images. This can be mitigated by having a set of real images and at inference choosing a random image from this set as the target "style". An example of this method using both real and simulated images from Mentice is shown in Figure 2.5.

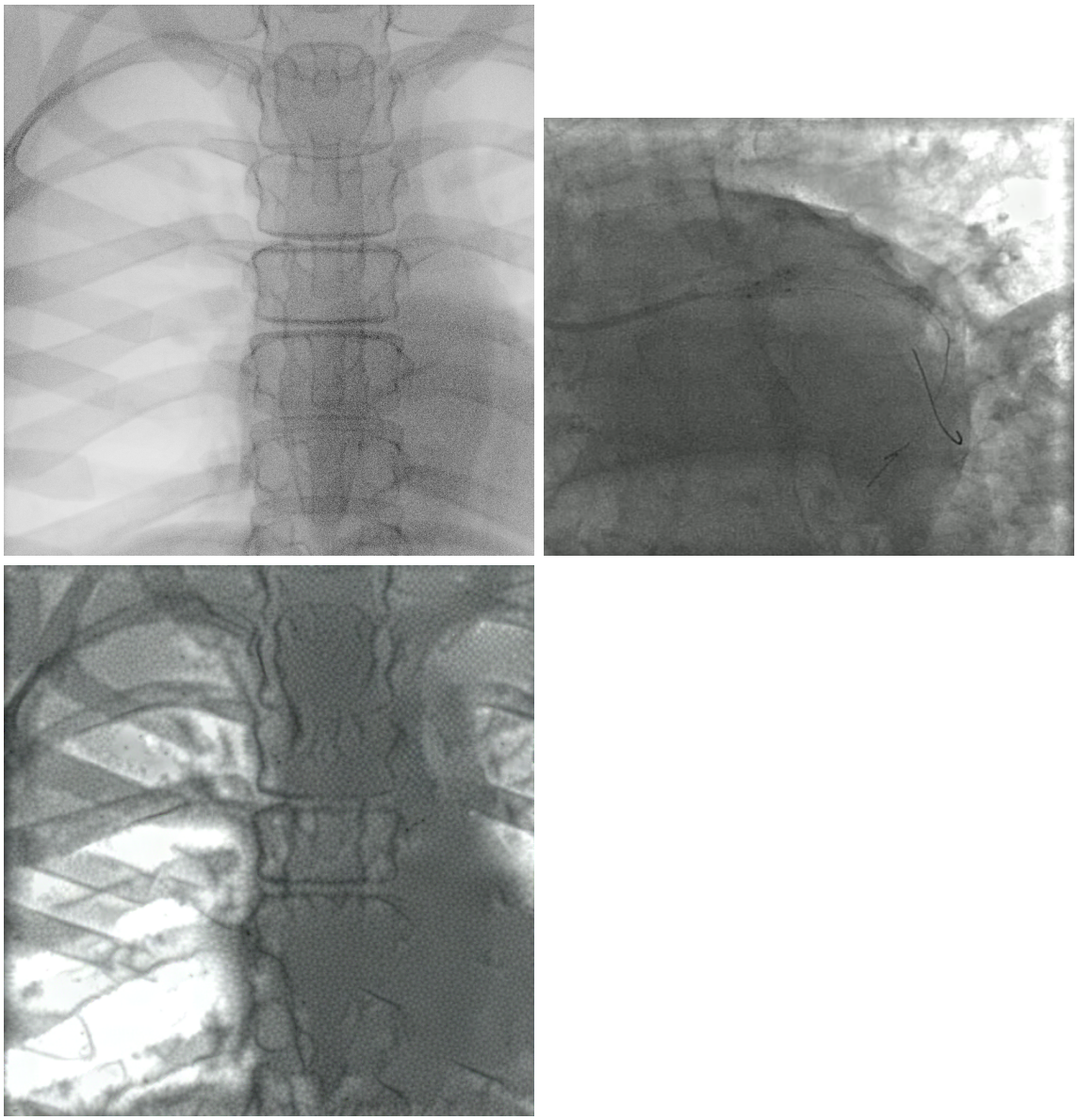


Figure 2.5: Inference using NNST. The upper left is the simulated image, the upper right is a real "style" image and the lower image is the output. The model used by NNST is available to test at `https://replicate.com/nkolkin13/neuralneighborstyletransfer`

NNST works by using VGG16 [22] pre-trained for image classification as a feature extractor. For rotational in-variance the style features are extracted from the feature image rotated at 0, 90, 180, and 270 degrees. Then the cosine distance loss between feature vectors for the stylized and content image is minimized with a number of iterations of parameter training using the Adam [23] optimizer. The feature vector is computed for each layer of VGG16 resulting in the feature vector containing features for different spatial resolutions of the image. The input image is processed in different spatial resolutions by downsampling and feature vectors are produced for each resolution. The output image is generated by using a linear combination

of up-sampled lower-resolution images with the original resolution image where the weight is referred to as the stylization level α.

3
Methods

In this chapter, the method for the thesis is described. This includes what models were used and how they were modified and set up. Also, how the data collection was set up and turned into the final dataset used for training is detailed. Finally, the details of how the training was set up and how the resulting models were evaluated are included.

3.1 Definitions of Models, Datasets, and Experiments

In this thesis, a total of 16 experiments were conducted using 2 models on 3 datasets.

3.1.1 Hardware Setups

Experiments were carried out using two different setups defined as seen below:

- S0: CPU: AMD Ryzen 9 5900X, GPU: AMD RX 6750XT 12GB
- S1: CPU: Intel Core i5-10600K, GPU: Nvidia GeForce RTX 2060 6GB

3.1.2 Models

Different models were used and a short description of each is given below:

- U-Net: One of the models used for catheter segmentation was U-Net [20] implemented with some simple changes using the PyTorch framework.
- YOLOv8m-seg: The other model used for catheter segmentation was YOLOv8m-seg [1].
- NNST: To generate a dataset with domain adaptation by style-transfer NNST [14] was used.

3.1.3 Datasets

Three different datasets were used and they are described below:

- D0: 6391 simulated images collected from the Mentice system.
- D1: 6391 augmented images using the images from D0 where each image was transformed into a style-transferred version using NNST. This used five hand-picked images from the real dataset (D2); then one of these five was randomly selected for each image during inference.
- D2: 1207 annotated real images from a publicly available dataset [7].

For the training using synthetic data (D0 and D1), the datasets were divided into train/validation/test using a 0.7/0.15/0.15 split. For training using real data (D2), we used two different splits; both a 0.7/0.15/0.15 and a 0.15/0.15/0.7 split were used for different tests.

3.1.4 Experiments

The experiments designed to answer the research questions were defined as seen below:

- E0: Train U-net on 70% of D2 and evaluate it on D2.
- E1: Train U-net on 15% of D2 and evaluate it on D2.
- E2: Train U-net on 70% of D0 and evaluate it on D0 and D2.
- E3: Train U-net on 70% of D1 and evaluate it on D1 and D2.
- E4: Fine-tune U-net from E2 on 70% of D2 and evaluate it on D2.
- E5: Fine-tune U-net from E2 on 15% of D2 and evaluate it on D2.
- E6: Fine-tune U-net from E3 on 70% of D2 and evaluate it on D2.
- E7: Fine-tune U-net from E3 on 15% of D2 and evaluate it on D2.
- E8: Train YOLOv8 on 70% of D2 and evaluate it on D2.
- E9: Train YOLOv8 on 15% of D2 and evaluate it on D2.
- E10: Train YOLOv8 on 70% of D0 and evaluate it on D0 and D2.
- E11: Train YOLOv8 on 70% of D1 and evaluate it on D1 and D2.
- E12: Fine-tune YOLOv8 from E10 on 70% of D2 and evaluate it on D2.
- E13: Fine-tune YOLOv8 from E10 on 15% of D2 and evaluate it on D2.
- E14: Fine-tune YOLOv8 from E11 on 70% of D2 and evaluate it on D2.
- E15: Fine-tune YOLOv8 from E11 on 15% of D2 and evaluate it on D2.

Experiments were conducted on different hardware setups, E0-E7 were conducted on S1, and E8-E15 on S0.

3.2 Collection of Simulated Dataset

The simulated fluoroscopic images that were used in this project came from the Mentice simulation system. These images were used to create the D0 and D1 training datasets.

3.2.1 Modifications to Mentice System

The system already supported live recording of the fluoroscopic images that were displayed while the simulation was running. However, the images were saved in a custom, non-standard, data format and the data saved did not include annotations of what were seen in the images. Modifications to the Mentice system were therefore needed in order to make the system output images that are suited for training.

The main modifications were made in the rendering pipeline. Code that saves the active draw buffer if the current frame should be captured was added in various places in the pipeline. This was done in places where objects of interest were being

rendered. Figure 3.1 shows pseudo code from such a code addition where the tools are rendered.

```
if shouldCaptureDataThisFrame:
    name = renderingObject->toolName()
    if name contains "catheter":
        saveCurrentPixelsAs(CATHETER)
    elif name contains "guide_wire":
        saveCurrentPixelsAs(GUIDE_WIRE)
```

Figure 3.1: Pseudo-code of code insertion.

In the function saveCurrentPixelsAs(), the current draw buffer is read and saved in a map that links the image to the corresponding class. Pseudo-code for this is shown in Figure 3.2.

```
def saveCurrentPixelsAs(objClass):
    image = allocateImage()
    glReadPixels(image)
    objClassToImageMap.insert(objClass, image)
```

Figure 3.2: Pseudo-code of saveCurrentPixelsAs().

Similar code as in Figure 3.1 was added in several places throughout the rendering pipeline. This allowed for the capture of the actual fluoro image as well as corresponding masks for the skeleton, kidney, liver, lung, heart, vessel, catheter, and guide wire.

Whether a frame should be captured or not is decided in a function, dataCollectionStep(), that was added at the start of the rendering pipeline. This function also saves all images from the last frame, if any were captured. See pseudo-code in Figure 3.3.

```
def dataCollectionStep():
    if shouldCaptureDataThisFrame:
        saveImagesToDisk(objClassToImageMap)
        objClassToImageMap.clear()
        contrastCounter -= 1

    isRecordingData = guiButtonState()
    isTimeToCaptureData = secondsSinceLastCapture() > 1

    if contrastInjectionVolumeChanged():
        contrastCounter = 3

    shouldCaptureDataThisFrame =
      isRecordingData and
      isTimeToCaptureData and (
        someToolHasMoved() or
        cameraHasMoved() or
        (contrastCounter > 0))
```

Figure 3.3: Pseudo-code of dataCollectionStep().

The functions someToolHasMoved() and cameraHasMoved() determine if something has moved enough to capture the frame. This is done with predefined thresholds.

Changes in the contrast injection volume is a bit different since it will change what's seen in the frame over time. Therefore, if the injection volume changes above a certain threshold, a frame will be captured each second for the next three seconds.

As described, the decision to capture a frame is based on several factors. This will be referred to as trigger-based collection and it ensures that images in the dataset are varied. Figure 3.4 shows an example of images collected in a frame capture.

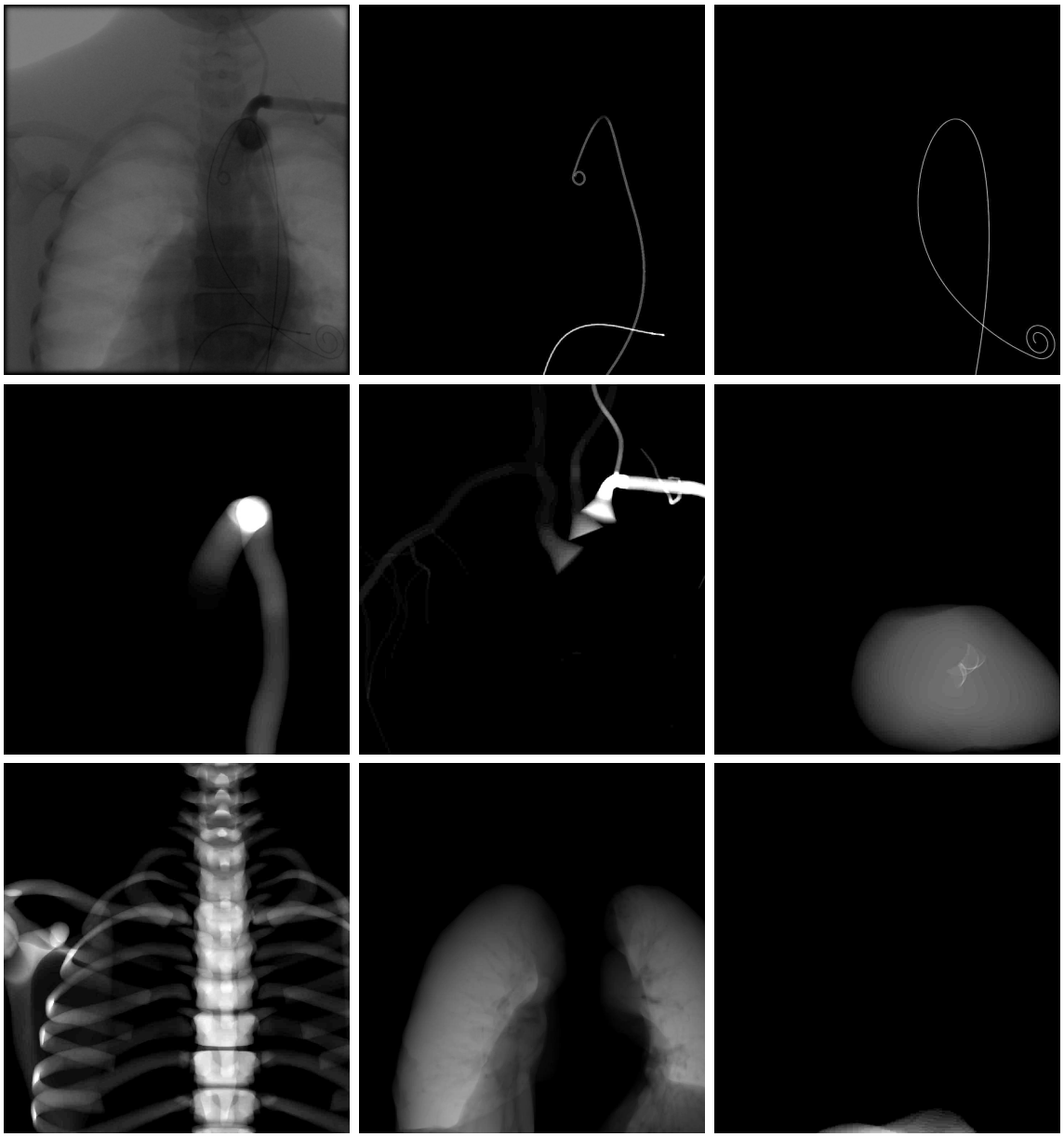


Figure 3.4: Images collected from the simulator. The images in the figure starting from the top left depict the following: complete fluoro, catheter, guide wire, aorta, vessels, heart, skeleton, lungs, and finally liver.

In 3.4 the fluoro image is displayed in the top left and the remaining images are object renders that are later used to create training labels. Note the brightness of the images is scaled here to make them more visible in the thesis. For example, the catheter is barely visible in the raw catheter render image. Hence the different pre-processing steps are later described in Figure 3.6.

Modifications were also done to the GUI of the simulator. A button was added to be able to start and stop the recording. This button toggles the value of the isRecordingData variable seen in Figure 3.3. Another change to the GUI was the addition of preset projection angles to enable efficient capture of images from differ-

ent angles. The added GUI elements are highlighted in Figure 3.5.

Figure 3.5: The arrows point to the changes made to the GUI. The upper arrow points to the projection angle preset. The lower arrow points to the recording button.

3.2.2 Data Collection

The method for collection the final dataset was to perform many different procedures in the simulation. The system has a built-in procedure guidance system which allows for performing many realistic procedures such as TAVI and Coronary angiograms.

The trigger-based collection and the additions made to the GUI enabled the execution of these procedures with little deviation from the normal operations of the simulator. Essentially the only difference in the procedure execution was that the camera was frequently moved around to get images from many different angles.

In order to have short feedback loops data was collected incrementally as the data collection system and the model training pipelines were developed. Once a complete pipeline was in place and the data collected was considered good (labels, classes, diversity), the collection of a larger dataset began.

Images were collected from 12 different cases with the help of the built-in procedure guidance system. The final number of simulated fluoroscopy images captured was 6391. The total number of images collected was 63099 when including all mask images. Since each fluoro image had up to 10 coresponding mask images captured. The D0 dataset was made by excluding all but the catheters masks images.

3.2.3 Convert Renders of Objects Into Binary Masks

The dataset of images that had been collected had to be pre-processed before they could be used as training data. The renders of individual objects or classes of objects obtained from the simulator needed to be converted to binary masks. Ideally, this will create binary masks that mark up all visible pixels of the objects in the simulated fluoroscopy images. To achieve this, first thresholding was applied to the masks, setting all pixels above the threshold to the maximum pixel value, which was 255 in this case. Then noise was removed from the masks using either Gaussian or median blur filtering. After that, thresholding was applied again to get a final binary mask. For very thin masks like catheter and guide wires the final binary masks were dilated to make them somewhat thicker. This was especially helpful when later converting the masks to polygons for use with the YOLOv8 model.

These conversion and pre-processing steps were determined by trial and error. Furthermore, the processing was different depending on mask class. For example, applying too much filtering to the catheter and guide wire masks could result in the masks becoming too thin. This problem does not exist for classes with larger masks. Additionally, renders of some objects contained more noise and artifacts than others. The reason for that is not clear but could be related to the use of different rendering methods for different objects. See the pseudo-code in Figure 3.6 for what was done for each class.

```
mask = imread(image_path, GRAYSCALE)

if image_class_name == "skeleton":
    mask_binary = threshold(mask, 4, 255)
    mask_median = medianBlur(mask_binary, 7)
    mask_smooth = dilate(mask_median, (1,1))
elif image_class_name == "catheter":
    mask_binary = threshold(mask, 1, 255)
    mask_smooth = medianBlur(mask_binary, 5)
elif image_class_name == "guide_wire":
    mask_binary = threshold(mask, 5, 255)
    mask_median = medianBlur(mask_binary, 3)
    mask_smooth = dilate(mask_median, (7,7))
else:
    mask_binary = threshold(mask, 5, 255)
    mask_blurred = GaussianBlur(mask_binary, (7,7), 0)
    mask_smooth = threshold(mask_blurred, 5, 255)
```

Figure 3.6: Pre-processing and conversion of captured renders to binary masks.

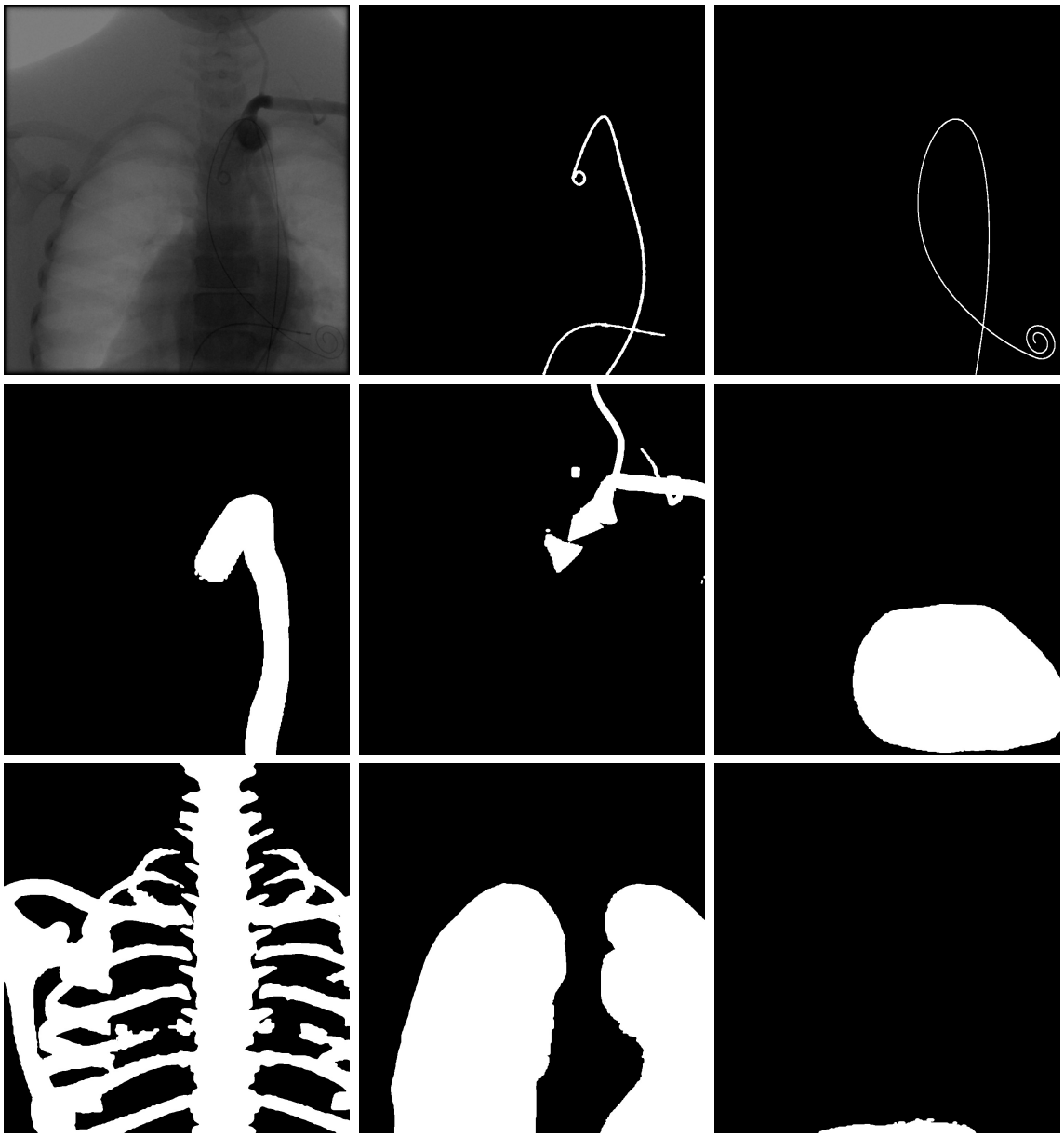


Figure 3.7: Images collected from the simulator and the masks are thresholded to create binary masks. The images contain the same content as in Figure 3.4.

For U-Net only segmentation of catheters is performed and therefore this process is simplified. For real images the mask was thresholded, and for synthetic images, a median blur filter was used after thresholding. The mask was then normalized to a binary mask so that background pixels had a value of zero and foreground a value of one in the dataloder.

3.2.4 Pre-processing of Masks for Use With YOLOv8

The YOLOv8 model doesn't use binary masks as labels. Instead, it uses a text file with lists of points that describe polygons that define the mask area. The binary masks therefore have to be converted to polygons before use with YOLOv8.

The first step was to use OpenCV's findcontours function to get an initial list of polygons. This list was then refined by filtering out polygons with areas below a certain threshold.

Furthermore, polygons that were inside another polygon were merged with its parent polygon to be able to express shapes with holes in them. For example, picture a binary mask of a donut. To express that as a list of points encircling that shape to form a polygon, you would have to create a bridge between the outer and inner contours. Effectively creating a "C"-shape where the two ends of the "C" are touching. See Figure 3.8 for an illustration of this.

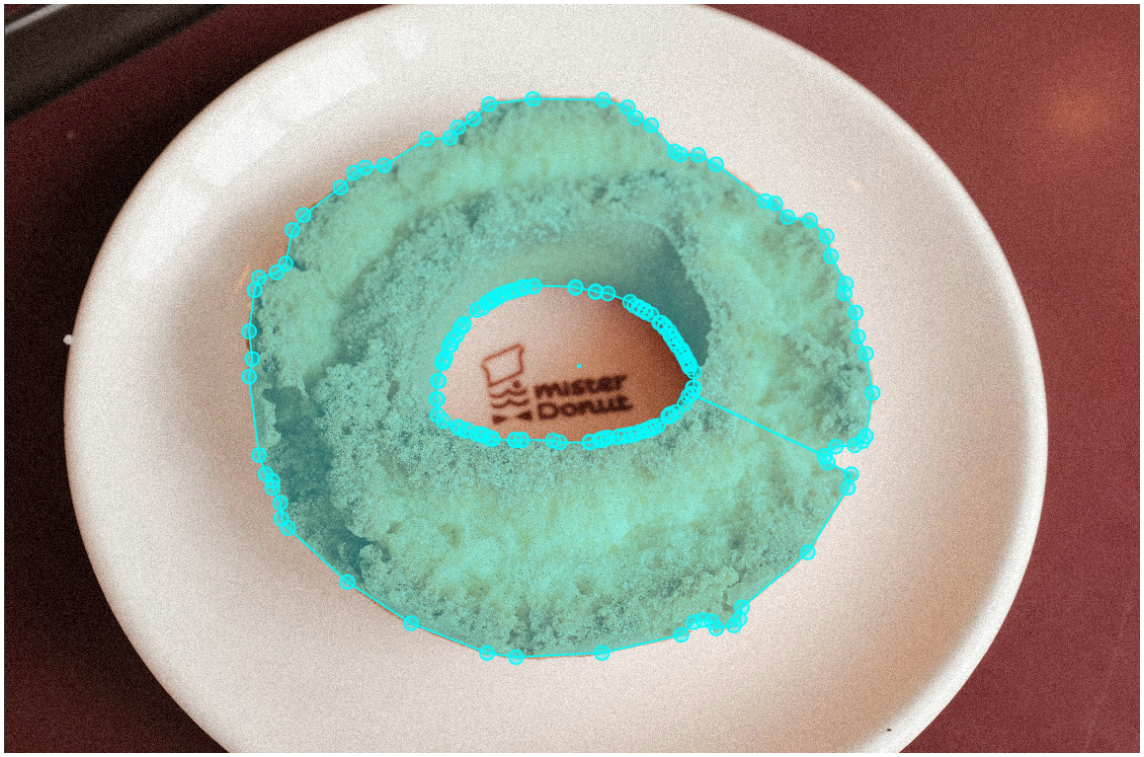


Figure 3.8: Polygon of donut contours with bridge [5].

The algorithm and code for this merging of polygons were found on Github in the repo "donut" by user "ryouchinsa"[5].

3.2.5 Style-Transfer

In experiments E3 and E11 the models were trained on dataset D1, which is a style-transferred version of D0. The method for constructing D1 was the following; for each image in D0 pick one of five style images and generate a copy of the D0 image in the style of the style image. The five style images were handpicked from the D2 dataset (real images). These images were chosen to represent all the different styles in D2. Furthermore, images with as few tools as possible were chosen. This is because having sharp dark edges in the source style image will result in a style

where the contours of structures are highlighted.

Before dataset D1 was generated both the hand-picked style images and the D0 images went through noise reduction using a bilateral and a median blur filter. Also, the inference parameters for NNST were chosen until some sample images were deemed as visually looking more like real images. The inference parameters are iterations, alpha, and max_scls. The number of iterations is how many iterations the model goes through before returning the stylized image, alpha is the stylization level as described in Section 2.7 and max_scls is the number of downsampling operations done, also described in Section 2.7. The parameters were chosen to be: $max\_iter = 10$, $\alpha = 0$, and $max\_scls = 2$. A sample D0 image, D2 style image and coresponding D1 image is shown in Figure 3.9.

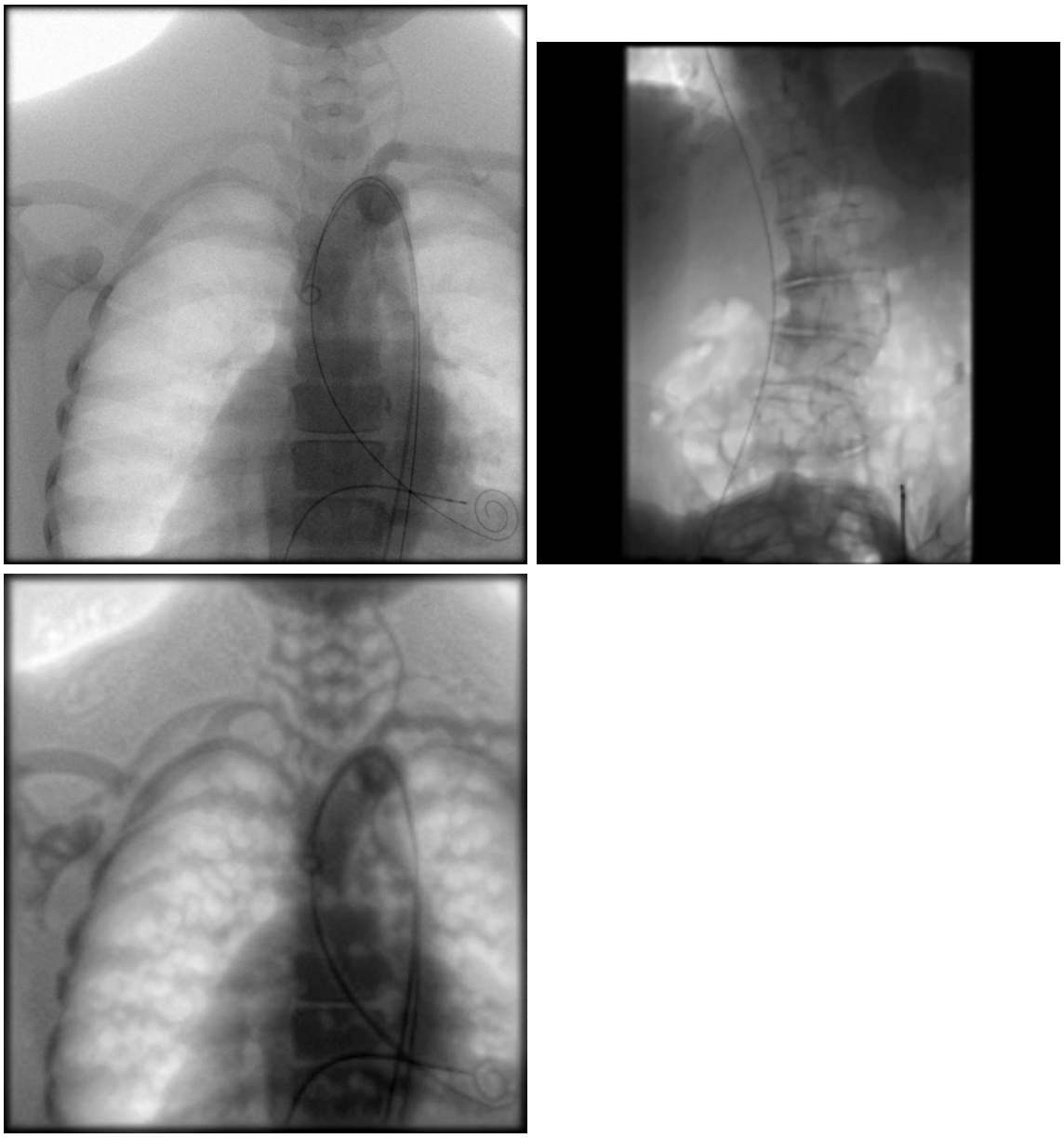


Figure 3.9: Inference using NNST. The upper left is the simulated image (from D0), the upper right is a real "style" image (one of the hand-picked images used for style-transfer in D1) and the lower image is the output (image from D1).

3.3 Dataset From Real Cases

For the real dataset, D2, we used a publicly available dataset also used and made available by Gheradini *et.al* [7]. This dataset consists of 6 different *in-vivo* surgical procedures labeled T1-T6. T1-T4 are from TAVI procedures and T5-T6 are from diagnostic procedures. T1 were human annotated and T2-T6 were automatically annotated using a semi-automated tracing method [27]. However, the precision of the labels of this dataset was not optimal. The labels were therefore modified before

use in this project. For motivation see Section 5.1.2.

The labels for the T4 were manually relabeled in Windows Photos as the original labels only covered a fraction of the catheter. The labels of T5 and T6 were programmatically aligned to better fit the actual location of the catheter. This was done by first using OpenCV's edge detection to get a binary image of the edges. The parameters of the edge detection was manually tuned with trial and error for the T5 and T6 datasets respectively. Then, the fitness of a mask to a given image is defined as how well it overlaps with the edges. Finally, for each image in T5/T6 calculate fitness for each label in the same dataset, for a range of offsets from the original position of the label. For each image, the mask and offset which had the best fitness was chosen as the new mask for that image. See Figure 3.10 for the pseudo-code of the algorithm.

```
for dataset in [T5, T6]:
  for image in dataset:
    edge_image = egde_detection(image)
    for mask in dataset:
      for x_ofs in range(-X_OFS, X_OFS):
        for y_ofs in range(-Y_OFS, Y_OFS):
          moved_mask = move(mask, x_ofs, y_ofs)
          fitness = sum(moved_mask*edge_image)/sum(moved_mask)
          if fitness > best_fitness:
              best_fitness = fitness
              best_mask = moved_mask

    update_mask(image, best_mask)
```

Figure 3.10: Pseudo-code for aligning the masks of T5 and T6.

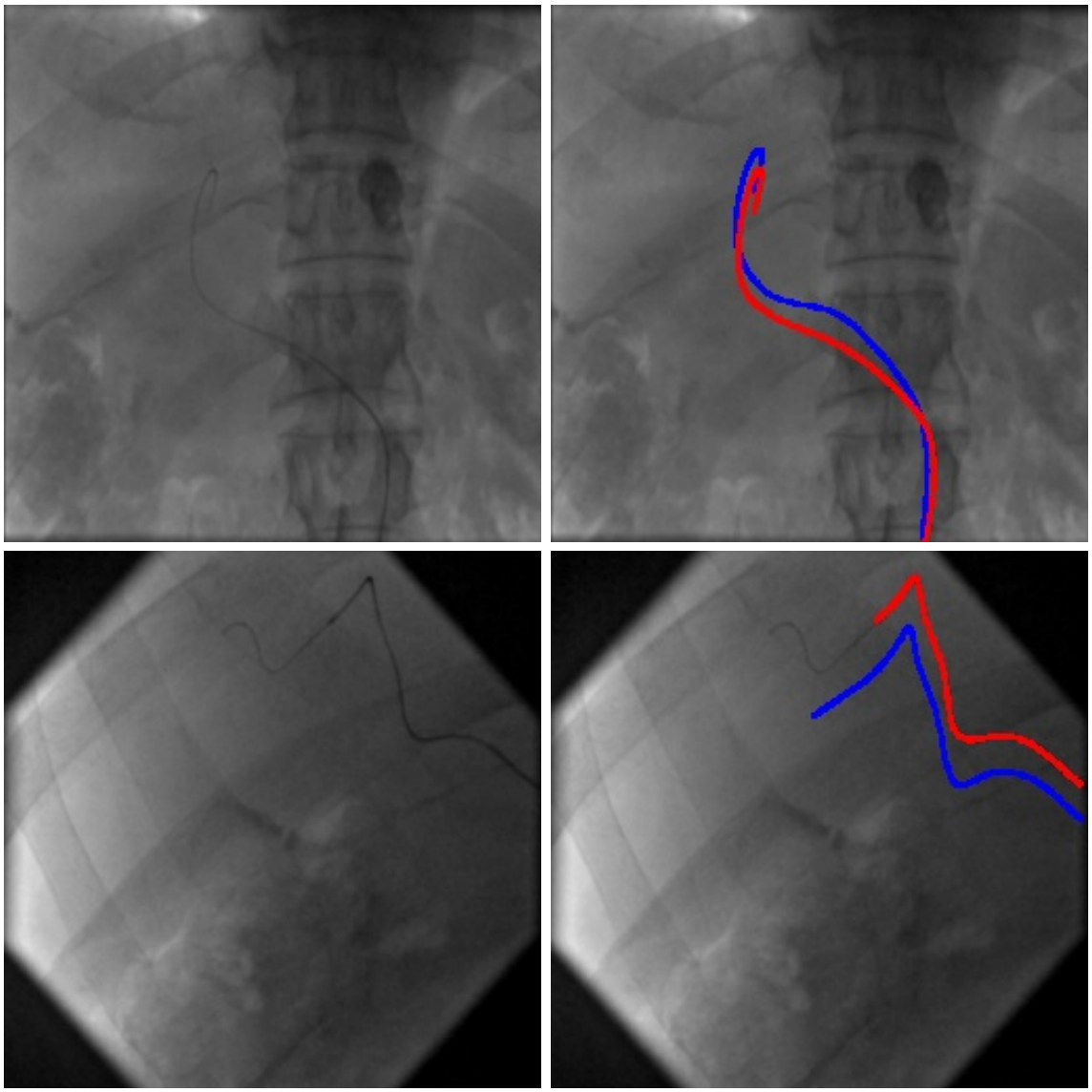


Figure 3.11: In this figure the original images are on the left side. The images on the right show the old and new labels overlayed on the image; the blue label is the original label and the red is the updated label used in D2.

3.4 Modifications to Models

For U-net it was implemented in pytorch similar to how it was done in the original paper [20] with some changes. For the convolutional layers padding was used to maintain dimensions. The activation function used was changed from ReLU to SiLU and a sigmoid was added as an activation function after the final layer. Also, batch norm was added to every layer, and dropout was added to the last two blocks of the decoder part of the network. This resulting network had 31M parameters.

For YOLOv8, no modifications were made to the model architecture. However, a function was added to enable the freezing of the backbone layers. This was done

by conditionally running a function at the start of training using YOLOv8's callback interface. Figure 3.12 shows the Python code of the function and the condition of whether to run it or not.

```
def on_train_start(trainer):
    model = trainer.model
    max_backbone_index = len(list(model.yaml["backbone"])) - 1

    for module in model.modules():
        if hasattr(module, "i"):
            if module.i <= max_backbone_index:
                for name, param in module.named_parameters():
                    param.requires_grad = False

if FREEZE_BACKBONE_LAYERS:
    model.add_callback("on_train_start", on_train_start)
```

Figure 3.12: Callback function used to freeze the backbone layers of YOLOv8.

3.5 Training

In this section details of how the models were trained are described.

3.5.1 U-Net

During initial testing different values for the data augmentation were used, arriving at the values outlined in the parentheses. The different methods of data augmentation that were implemented were:

- Flip (±20%)
- Contrast (±30%)
- Noise ($\lambda, A = 2, 5$)

All the augmentation methods were done during training by randomly augmenting each image in each batch within the constraints set by the values in the parentheses. Flip is the probability that an image is flipped and it is implemented for both left/right and up/down flipping with the same independent probability. Contrast is applied using a contrast augmenting function that decreases or increases the contrast by some % and selects the amount randomly within the range. Finally, noise is added by sampling random 2D noise from a Poisson distribution with mean and standard deviation λ and amplitude A and then shifting the mean to zero and adding the resulting noise to the image.

The loss function used for U-net in this project is a weighted average of dice loss and BCE loss weighting the dice loss higher at 0.8/0.2. For all training, a cosine annealing with warm restarts learning rate scheduler was used. For regular training as well as fine-tuning with a frozen encoder a maximal learn rate of 10^{-3} and a minimal learn rate of $5 * 10^{-5}$ were used. For final fine-tuning without freezing, both learning rates were halved. Also, a batch size of 16 and an image size of 256 by

256 were used for all tests. The learning rate, epochs, and cosine annealing reset constant T_0 for each test is outlined in Table 3.1.

Parameter	E0	E1	E2	E3	E4	E5	E6	E7
Epochs	80	200	40	40	100/100	200/200	100/100	300/200
T_0	20	20	10	10	20	20	20	20

Table 3.1: Epochs and T_0 used for different experiments. For experiments with fine tuning the / differentiates between with freezing / without freezing the encoder.

3.5.2 YOLOv8

For E8-E11 the yolov8-seg.pt model configuration was used which used weights pretrained on the COCO dataset [4]. No layers were frozen however so this only affects the initial starting point of the training and potentially speeds up the training.

For E12-E15 the backbone layers were frozen. The backbone in the YOLOv8 architecture refers to the initial feature extracting layers which is a CSPDarknet53 [28, 3]. Moreover, the backbone consists of 11,855,856 weights out of a total of 27,240,227. This means that in E12-15, 40% of the YOLOv8 model is frozen, and only 60% is actually trained on D2 (real images).

The YOLOv8 model has a lot of different parameters that can be tuned to obtain optimal performance. Some of the parameters used in this thesis were chosen with an educated guess. Parameters that are harder to reason about were selected by trying a couple of values and then picking the one with the best performance. All of the YOLOv8 models trained in E8-E15 had the same values for all parameters, except for the number of epochs. The parameters of the YOLOv8 model not discussed in this section were left at their default values. See Table A.1 in the Appendix for a complete list of the YOLO arguments used during training.

The performance of a model on its training dataset is determined by the fitness value. The fitness of the model is calculated after each epoch by running validation on the val split of the dataset. It is defined as

$$\begin{aligned} fitness = 0.1 * mAP_{50}(B) + 0.9 * mAP_{50-95}(B) + \\ 0.1 * mAP_{50}(M) + 0.9 * mAP_{50-95}(M) \end{aligned} \tag{3.1}$$

where $mAP(B)$ and $mAP(M)$ are the mean average precision (see Section 2.2) for the bounding box and segmentation mask respectively. The YOLOv8 train function compares the fitness of the last epoch with the best fitness seen so far and saves the weights if the fitness is better. In other words, after each epoch, there always exists saved weight of the version of the model with the best fitness yet.

For all experiments E8-E15 except E10 and E11, training was done three times with the number of epochs set to 100, 150, and 200 respectively. The model with the best fitness across those training runs was then used for evaluation for the respective experiment. For E10 and E11, the number of epochs was set to 400. The

dropout ratio was slowly increased during initial training tests. Performance increases were observed up to a drop-out ratio of 0.5 for the YOLOv8 model and therefore that ratio was used in all subsequent training. The optimizer used was SGDM with initial learning rate, final learning rate ratio, and momentum set to the default values of 0.01, 0.01, and 0.937 respectively. The default mask ratio in YOLOv8 is 4, meaning that the masks are down-scaled by a factor of 4. This was changed to 1 to keep the labels at full resolution since the catheters are just thin lines.

YOLOv8 implements runtime data augmentation where certain properties of the input images are modified at random at every training batch. The interval for allowed rotation augmentation was set to (0, 180) degrees to increase the diversity of the dataset. This is especially useful when training on catheters because they usually appear in images at a limited set of angles. The translation augmentation range was set to (0.0, 0.2) since most catheters in the dataset are along the center of the image. In addition to the runtime data augmentation, contrast augmentation was applied as a pre-processing step when preparing the dataset. Each image had its contrast adjusted with a random factor in the interval [0.7, 1.3].

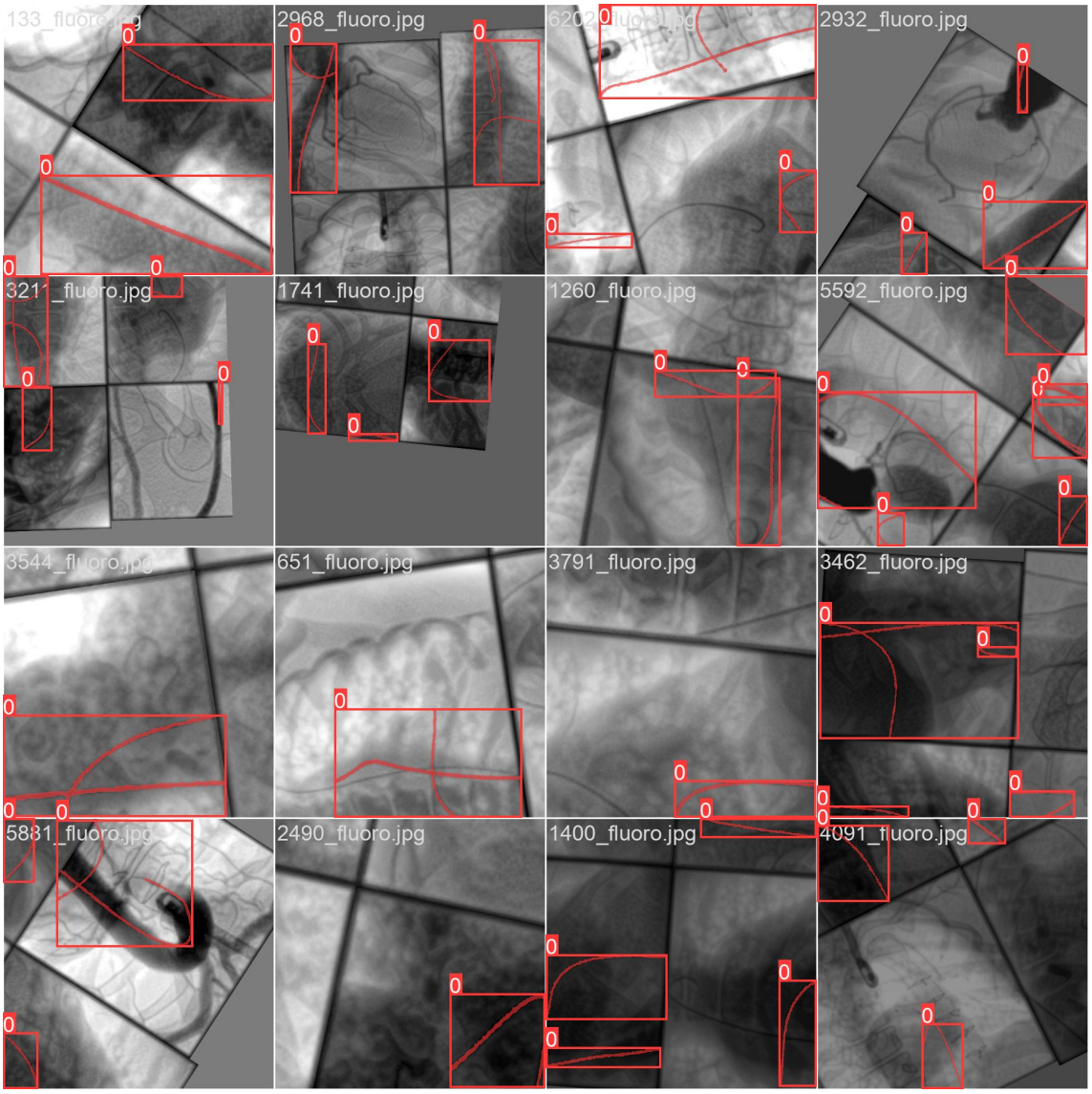


Figure 3.13: YOLOv8 training batch from training on D1, with ground truth labels. Note that the zeros on the bounding boxes represent the class of the object and not confidence.

3.6 Validation and Evaluation

Since deep neural networks will be trained to perform segmentation we will have to evaluate how well the resulting model performs. For this, the following metrics will be used:

- Dice coefficient
- Precision
- Recall
- IoU

The results will be obtained by evaluating the models on the test part of the datasets as outlined in Section 3.1.4. For the evaluation, a confidence value is needed to

generate the metrics. The confidence will be chosen by generating a dice-confidence curve and then using the confidence that yielded the maximal dice score.

4
Results

In this chapter, the results of this project are presented. Starting with the curves presenting the different metrics over the confidence level. Then, tables containing the numerical values of the results obtained from each experiment are presented. Finally, some qualitative demonstrations of the segmentation performance of the models are included.

4.1 Metrics Over Confidence Curves

In this section plots from all experiments of the IoU and Dice score using different confidence thresholds are shown. In other words, the confidence can be seen as the cutoff for the detection probability output by the model. This is done so that all models are evaluated at the confidence threshold that is optimal for performance. The confidence that maximizes Dice is used to determine the optimal confidence level. The plots will start with the results from the U-net experiments in the order of the amount of the real dataset D2 that was used for training. Figure 4.1 shows the results of U-net trained only using synthetic data (datasets D0 and D1).

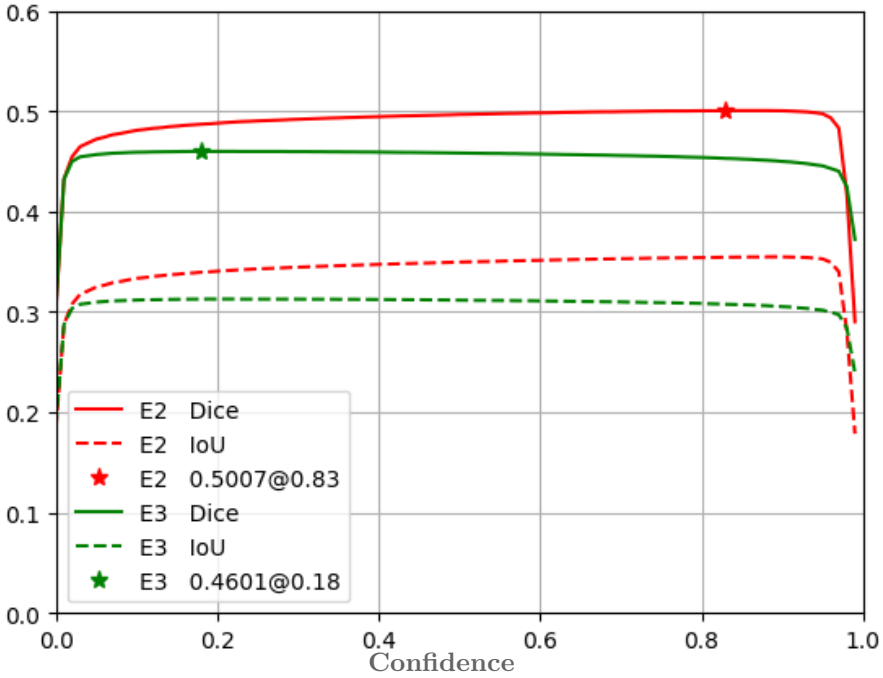


Figure 4.1: Dice and IoU over confidence for models from E2 and E3, evaluated on D0 and D1, respectively. (E2: Train U-net on 70% of D0. E3: Train U-net on 70% of D1).

If Figure 4.1 the E2 model (the model trained on simulated dataset D0) outperforms the E3 model (trained on style transfer dataset D1) this difference is more apparent at higher confidence. Furthermore, Figure 4.2 shows the results of U-net trained using 70% of the real dataset.

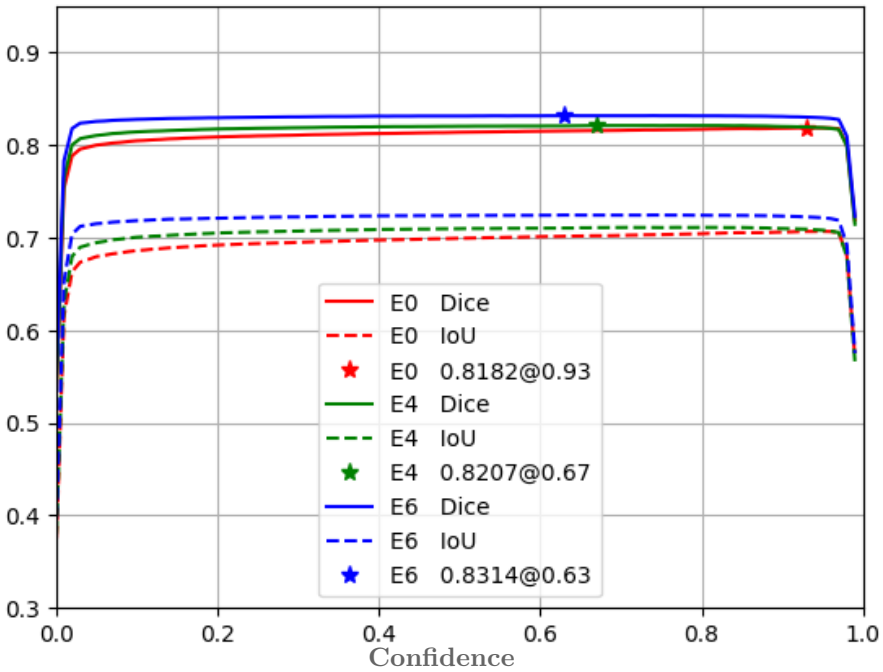


Figure 4.2: Dice and IoU over confidence for evaluations on D2 of models from E0, E4 and E6. (E0: Train U-net on 70% of D2. E4: Pre-train U-net on 70% of D0, then fine-tune on 70% of D2. E6: Pre-train U-net on 70% of D1, then fine-tune on 70% of D2.)

In Figure 4.2 the models with pretraining, E4 and E6, outperform the model trained only on the real dataset from E1. Also, the model that was trained using the style transfer dataset, from E6, outperforms the model trained on the simulated images, from E4, and this is apparent for all confidence levels. Figure 4.3 shows the results of U-net trained using 15% of the real dataset.

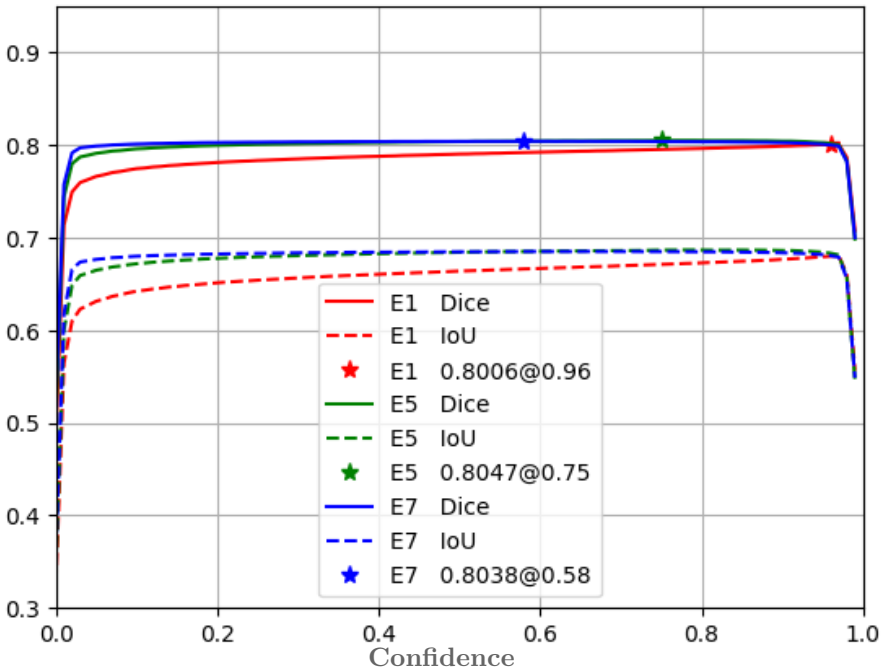


Figure 4.3: Dice and IoU over confidence for evaluations on D2 of models from E1, E5 and E7. (E1: Train U-net on 15% of D2. E5: Pre-train U-net on 70% of D0, then fine-tune on 15% of D2. E7: Pre-train U-net on 70% of D1, then fine-tune on 15% of D2.)

In Figure 4.3 the result of E1 (no pretraining trained on 15% of D2), E5 (trained using the simulated dataset D0 and fine-tuned on 15% of D2) and E5 (trained using the style-transfer dataset D1 and fine-tuned on 15% of D2). The best result is from E5 which outperforms E7 slightly and only at high confidence levels. Both experiments that did pretraining on D0 or D1 outperformed the one without (E1).

Figure 4.4 shows the Dice and IoU scores over confidence for predictions on D2. The predictions are made by the YOLOv8 models that are only trained on D0 and D1, respectively.

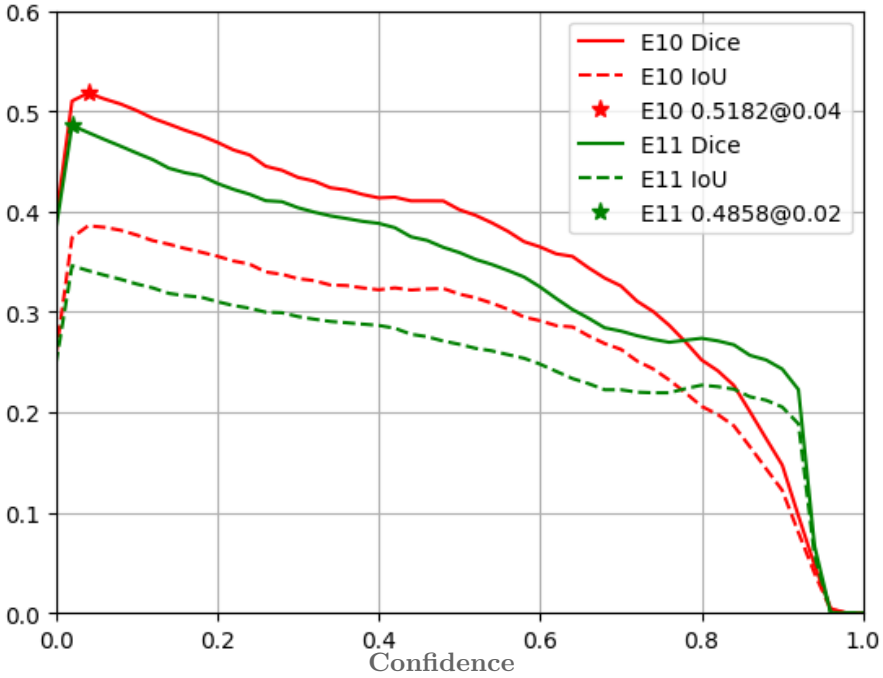


Figure 4.4: Dice and IoU over confidence for models from E10 and E11, evaluated on D0 and D1, respectively. (E10: Train YOLOv8 on 70% of D0. E11: Train YOLOv8 on 70% of D1).

Figure 4.4 shows similar results as the coresponding U-net experiment. In experiment E10 where YOLOv8 is trained on simulated images, a higher Dice score was observed compared to the model in experiment E11, where the model was trained on style-transferred images.

The results from E8, E12, and E14 are shown in Figure 4.5. These experiments include all YOLOv8 models that have seen 70% of D2 (real images). The models in E12 and E14 were finetuned on 70% of D2 and pre-trained on D0 (simulated images) and D1 (style transferred image), respectively. The E8 model is trained directly on 70% of D2.

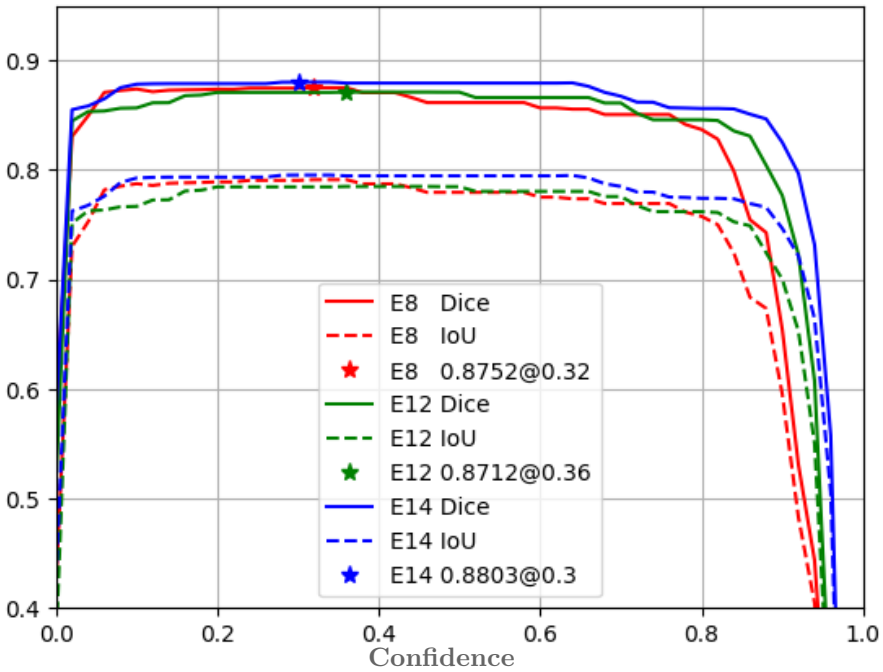


Figure 4.5: Dice and IoU over confidence for evaluations on D2 of models from E8, E12 and E14. (E8: Train YOLOv8 on 70% of D2. E12: Pre-train YOLOv8 on 70% of D0, then fine-tune on 70% of D2. E14: Pre-train YOLOv8 on 70% of D1, then fine-tune on 70% of D2.)

Figure 4.5 shows that the model pre-trained on the simulated images, E12, has a slightly lower max dice score than the non-pretrained model E8. However, the E12 model has a higher dice score than the E8 model at almost all confidences above 0.4. The E14 model achieved the highest max dice score of all models in this project with 0.8803 at 0.3 confidence. This model was pre-trained on D1 (style transferred images). Figure 4.5 also shows that the pre-trained models in E12 and E14 have higher confidence levels than the non-pre-trained model E8.

Figure 4.6 shows dice and IoU over confidence for the final experiments E9, E13, and E15. These experiments are the same as the ones shown in Figure 4.5 (E8, E12, E14), except that the models here have only ever seen 15% of D2 (real images).

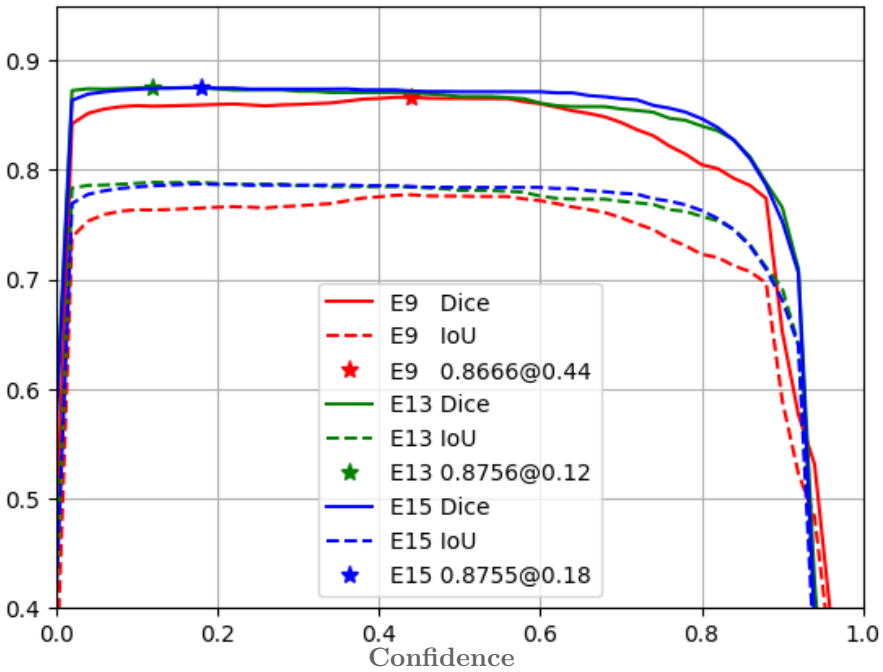


Figure 4.6: Dice and IoU over confidence for evaluations on D2 of models from E9, E13 and E15. (E9: Train YOLOv8 on 15% of D2. E13: Pre-train YOLOv8 on 70% of D0, then fine-tune on 15% of D2. E15: Pre-train YOLOv8 on 70% of D1, then fine-tune on 15% of D2.)

Like previous experiments, Figure 4.6 shows that the pre-trained models from E13 and E15 achieve a higher Dice score than the non-pre-trained model from E8. However, here there's almost no difference in the curves of the pre-trained models as they seem to perform the same. Furthermore, the difference between the pre-trained and non-pre-trained models is larger here compared to the experiments where the models trained or finetuned on 70% of D2.

4.2 Numerical Results From Experiments

This section illustrates the numerical results obtained from the different experiments. Table 4.1 shows the results obtained from the different experiments that use U-net.

Experiment (@conf.)	Trained on	Fine-tuned	Dice ↑	Precision ↑	Recall ↑	IoU ↑
E0 (@0.93)	70% of D2	No	0.8182	0.8056	0.8414	0.7062
E1 (@0.96)	15% of D2	No	0.8006	0.8025	0.8175	0.6798
E2 (@0.83)	70% of D0	No	0.5007	0.5218	0.5255	0.3546
E3 (@0.18)	70% of D1	No	0.4601	0.4966	0.5085	0.3128
E4 (@0.67)	70% of D0	70% of D2	0.8207	0.8029	0.8477	0.7106
E5 (@0.75)	70% of D0	15% of D2	0.8047	0.8027	0.8207	0.6863
E6 (@0.63)	70% of D1	70% of D2	**0.8314**	**0.8159**	**0.8512**	**0.7241**
E7 (@0.58)	70% of D1	15% of D2	0.8038	0.8011	0.8165	0.6848

Table 4.1: Results from experiments E0 to E7 (U-Net) evaluated on the test set of D2. The best result for each metric is in bold font.

In Table 4.1 the model obtained from experiment E6 utilizing pretraining on style transfer data and then fine-tuning using 70% of the real data obtains the best results for all metrics. Table 4.2 shows the results from experiments E8-15 that came from using YOLOv8.

Experiment (@conf.)	Trained on	Fine-tuned	Dice ↑	Precision ↑	Recall ↑	IoU ↑
E8 (@0.32)	70% of D2	No	0.8752	0.8786	0.8760	0.7912
E9 (@0.44)	15% of D2	No	0.8666	0.8788	0.8625	0.7774
E10 (@0.04)	70% of D0	No	0.5182	0.5261	0.6018	0.3859
E11 (@0.02)	70% of D1	No	0.4858	0.5185	0.5569	0.3461
E12 (@0.36)	70% of D0	70% of D2	0.8712	0.8898	0.8573	0.7850
E13 (@0.12)	70% of D0	15% of D2	0.8756	0.8742	0.8846	0.7888
E14 (@0.30)	70% of D1	70% of D2	**0.8803**	**0.8841**	0.8784	**0.7954**
E15 (@0.36)	70% of D1	15% of D2	0.8731	0.8646	**0.8892**	0.7847

Table 4.2: Results from experiments E8 to E15 (YOLOv8) evaluated on the test set of D2. The best result for each metric is in bold font.

In Table 4.2 the model obtained from experiment E14 utilizing pretraining on style transfer data and then fine-tuning using 70% of the real data obtains the best results for all metrics except for recall which is higher for the similar model obtained from E15 where the difference is that E15 is trained using 15% of the real data. Moreover, Table 4.3 shows the results segmentation for the experiments only trained on synthetic data (D0 and D1) evaluated on the respective dataset they were trained on.

Experiment (@conf.)	Trained on	Dice ↑	Precision ↑	Recall ↑	IoU ↑
E2 (@0.28)	D0	0.5282	0.5437	0.5623	0.4039
E10 (@0.12)	D0	0.7229	0.6658	0.7934	0.5788
E3 (@0.54)	D1	0.6282	0.5943	0.7085	0.5062
E11 (@0.44)	D1	0.6935	0.6893	0.7076	0.5555

Table 4.3: Results from experiments E2, E3 (U-Net), E10, and E11 (YOLOv8) evaluated on the test set of the respective datasets they were trained on.

In Table 4.3 the results from the dataset the models were trained on can be seen. Also, by comparing the result with the results from Table 4.1 and 4.2 the U-net models that performed worse on the dataset it was trained on performed better on the real data and the opposite was true for the YOLOv8 models. Finally, Table 4.4 shows the relative performance boost that came from using pre-trained models.

Experiments	Simulated (Dice/IoU)	Style-transferred (Dice/IoU)
U-net trained on 70%	0.3%/0.6%	1.6%/2.5%
U-net trained on 15%	0.5%/1.0%	0.4%/0.7%
YOLO trained on 70%	-0.5%/-0.8%	0.6%/0.5%
YOLO trained on 15%	1.0%/1.5%	0.8%/0.9%

Table 4.4: Performance increases achieved by using pre-training with synthetic data for different models and different proportions of real data

From Table 4.4 the maximum increase from pre-training was found to be 2.5%. Also, in one case from E12, a decrease of 0.8% was the worst relative performance of all. In the cases where 15% of the real data was used pretraining on simulated images outperformed training on the style transfer data, the opposite was true for the tests using 70% of the real data. Averaging the relative increases from the simulated data yields a 0.3%/0.6% average increase and 0.9%/1.2% average increase for the style transfer data.

4.3 Sample Segmentation Results

In this section, some real images from dataset D2 were segmented using a few of the different models, to enable visual inspection of how well the models can perform segmentation. Figure 4.7 visualizes the model results obtained from E6, the model that performed the best for U-net on real data. Furthermore, Figure 4.8 shows the U-net model from E2, trained using only simulated images from dataset D0.

Figure 4.9 visualizes the model results obtained from E14, which was the best model across all experiments. Finally, Figure 4.10 shows the best YOLOv8 model that has only trained on simulated images and has not seen any real images.

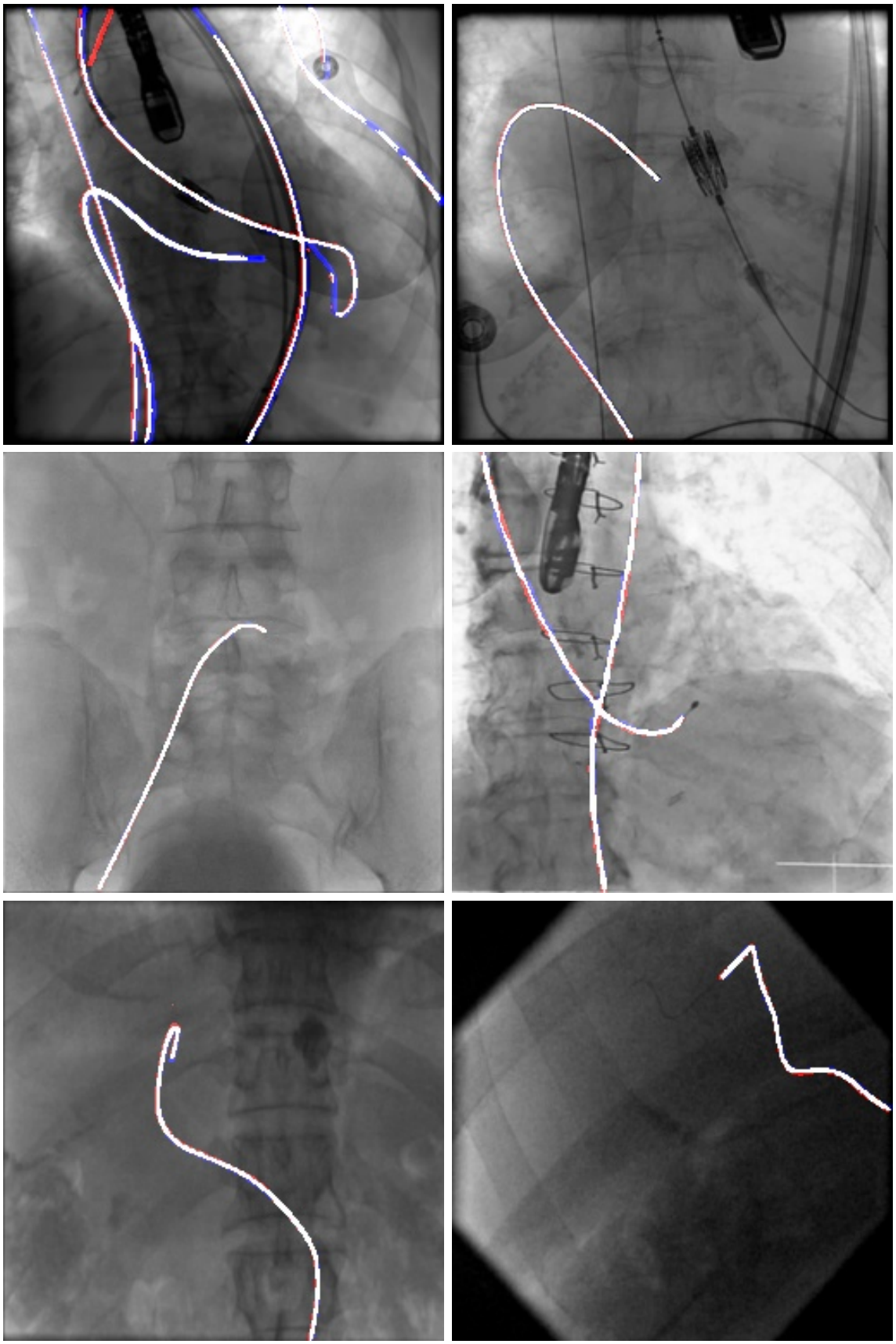


Figure 4.7: Segmentation result from E6 on the test dataset of D2. Blue is the label, red is the prediction and white is the overlap.

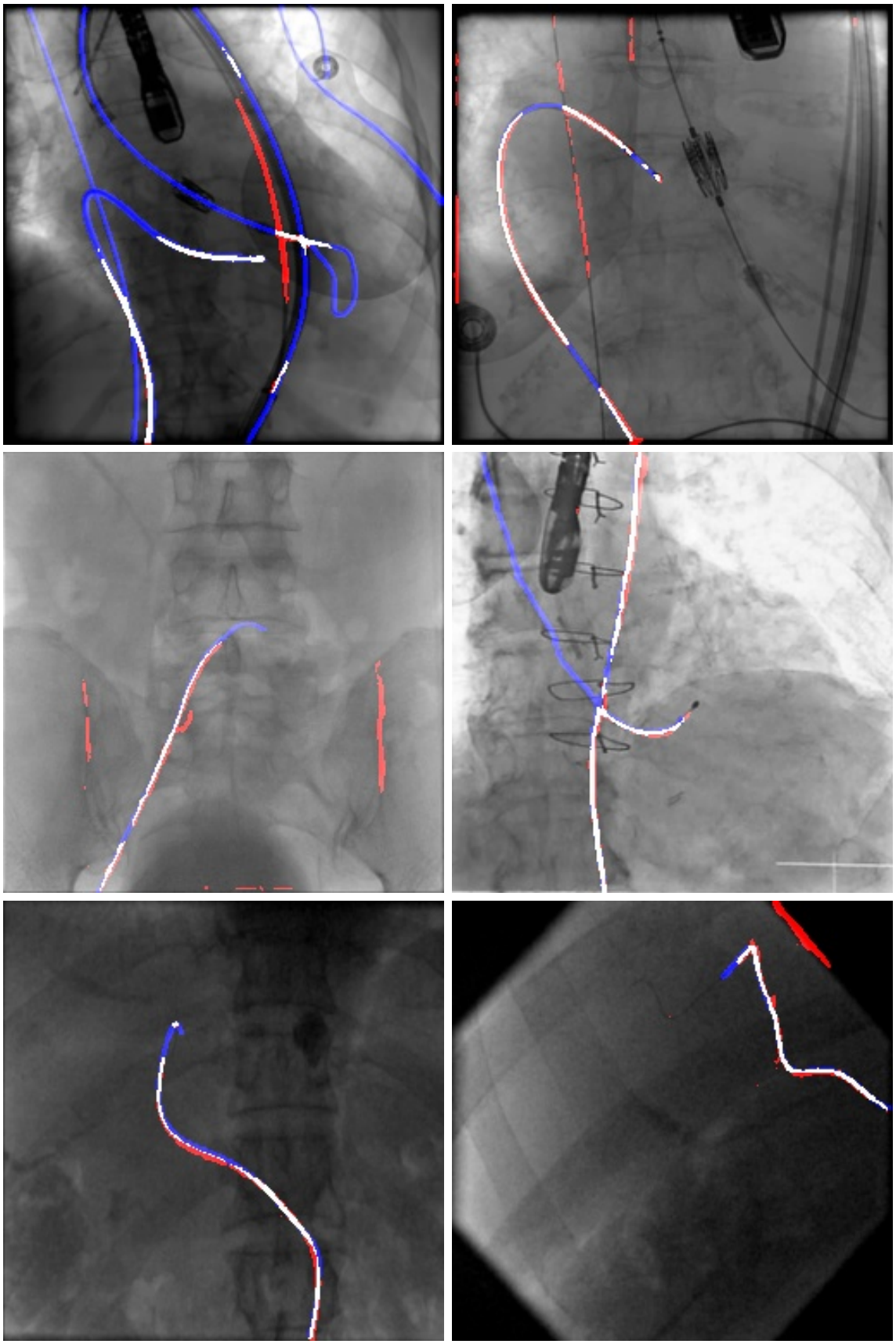


Figure 4.8: Segmentation result from E2 on the test dataset of D2. Blue is the label, red is the prediction and white is the overlap.

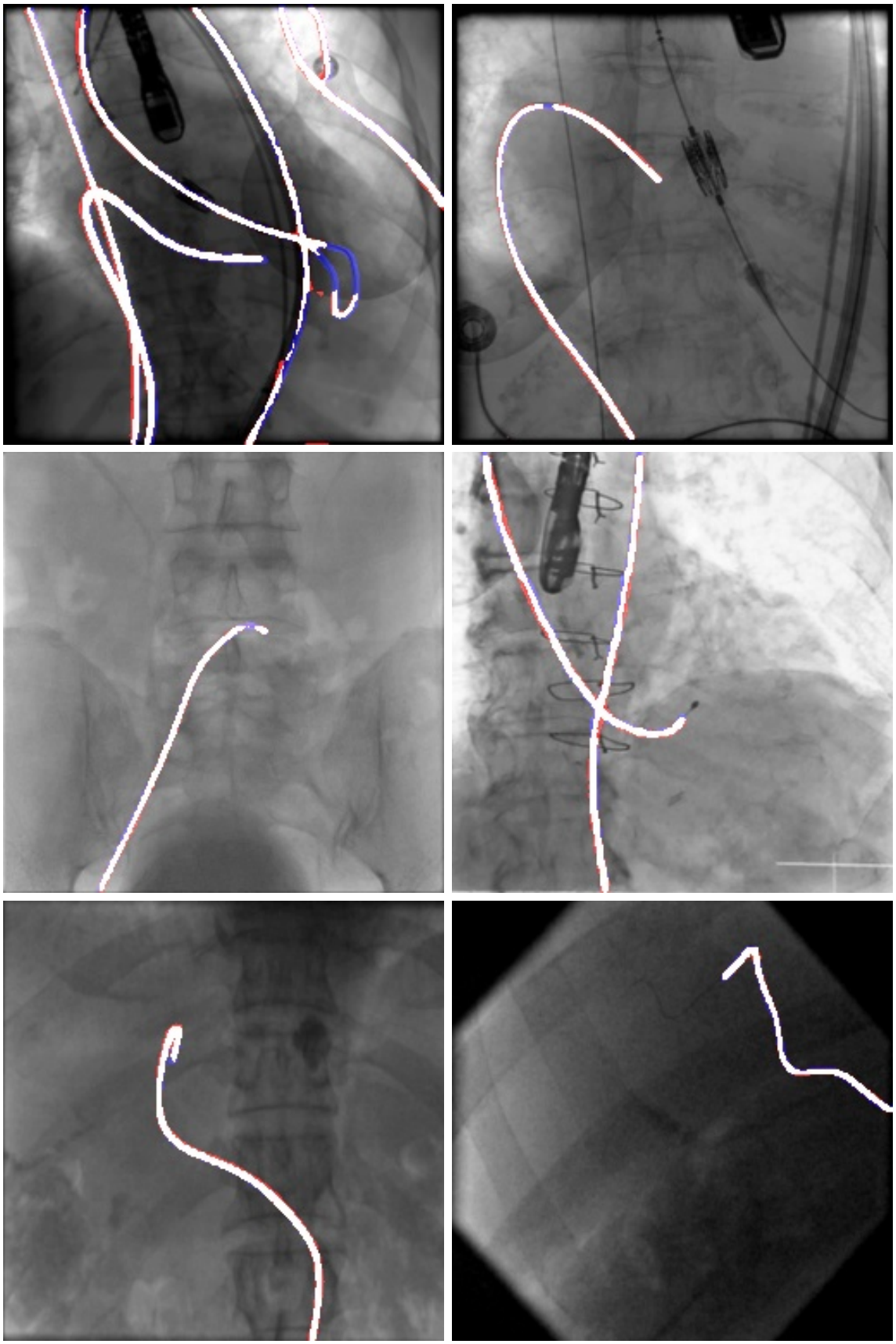


Figure 4.9: Segmentation result from E14 on the test dataset of D2. Blue is the label, red is the prediction and white is the overlap.

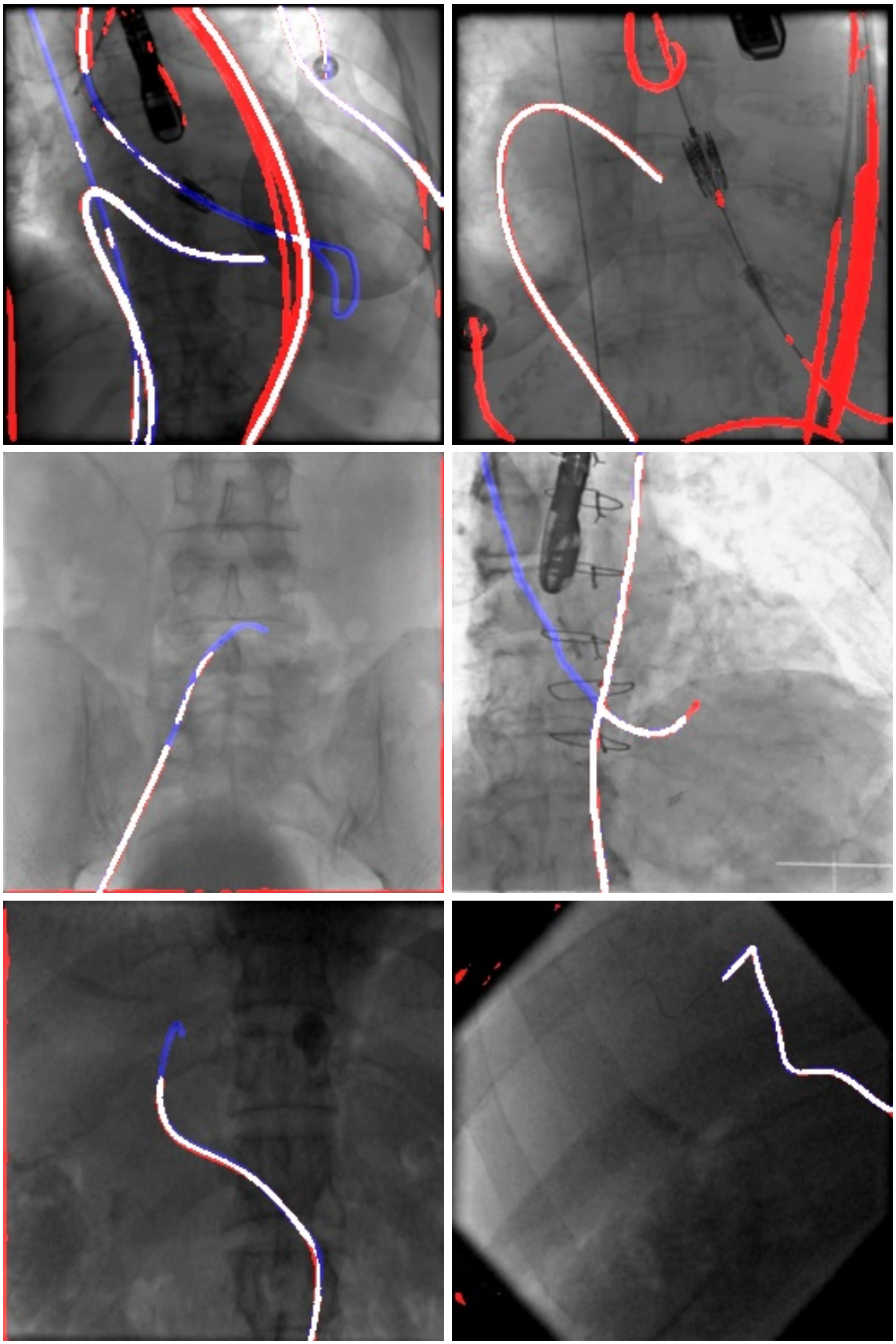


Figure 4.10: Segmentation result from E10 on the test dataset of D2. Blue is the label, red is the prediction and white is the overlap.

5
Discussion

This chapter discuss the different choices made in the project and describes the reasoning behind them. Additionally, the results presented in the previous chapter are interpreted and analyzed. Finally, different social and ethical issues regarding the project are discussed.

5.1 Discussion and Motivation of Method

This section details and motivates the method used. Specifically, the choice of models, collection of data, and design of experiments are presented and motivated.

5.1.1 Models

For this project, we used three different AI models, U-net, YOLOv8, and NNST. The models were chosen primarily based on three criteria, specificity of task, performance, and scale. The first one means that the model needs to perform the task that we want it to perform within some constraints. Also, performance and scale go a bit hand in hand since scale impacts performance and scale is limited by hardware constraints. In this subsection, the choice of models will be motivated in more detail for each model.

As previously mentioned, U-net was chosen to be one of the models we evaluated. This model was chosen for multiple reasons. It is a simple architecture that is well established and is used as a baseline for performance by many papers. It was also originally developed for use specifically within medical imaging.

However, U-net is an old architecture that does not utilize many of the modern advancements made in the field of AI. Therefore, it was decided to implement the model with some simple changes utilizing some of these advancements. With these changes, the model still holds up decently well especially considering the size of the model.

The other segmentation model used, YOLOv8 is one of the best real-time one-shot object detection models. Because it is optimized for real-time use together with its high performance on object detection it was reasoned that this model would be one of the better models for its size. It is also as of the writing of this project, a relatively new model released in 2023 meaning that it utilizes more of the recent advances in

the field of AI. Also, the m (medium) size was chosen since larger models did not show significant increases in performance, and the m model allowed for larger but still reasonable batch and image sizes within the given hardware constraints.

Regarding NNST, the reason this model was chosen was based on a couple of factors. Firstly the model needed to work by updating local features. Many other style-transfer models update the images globally potentially making big changes to structures. This would in our case mean that the labels we automatically generate would not be usable with the style-transferred images. However, when only the local features and textures are updated and when appropriate settings for NNST are used, the catheter will remain in the same position and the same label can be used.

Also, for this project, other segmentation models that were not used were still looked into. OneFormer is a segmentation model that holds state-of-the-art performance on multiple segmentation benchmarks. Another model is Trans U-net with an architecture similar to U-net, the primary difference being that it uses a transformer encoder. Furthermore, YOLOv9 was released during this project, however, when running a simple test a slight performance decrease was found when changing models, perhaps due to using settings better configured for YOLOv8. Still, even with changed parameters the change of model would likely only lead to minor changes in the result so it was decided that YOLOv8 would be the model that was used. Finally, Mamba U-net is another model made for medical imaging with promising results.

While some of these models might have achieved better performance compared to the models used in this project we are still able to investigate the research questions posed in Section 1.3. The primary advantage of using multiple models to investigate these questions is that it lowers the chance that the results are due to random factors or that it has something to do with the model architecture. Since YOLOv8 and U-net are quite different one performing instance segmentation on object detection and the other producing a segmentation mask directly from the input, the models serve as a good comparison, because the main objective is to examine the quality of the synthetic datasets.

5.1.2 Data Collection and Datasets

The method for collecting the D0 dataset of simulated images from the Mentice simulator was chosen with two things in mind, to allow for easy collection of many images and that the final dataset should be as diverse as possible. This immediately led to the decision that the collection would be automatic. The simulator would record and save images in the background while an operator would perform a procedure. Furthermore, the initial design also included that images would be saved no more than once a second to reduce the number of similar-looking images in the dataset. After a few test runs, however, it became clear that this was not enough. The operator quite often stops and thinks about what to do next while performing a procedure in the simulator. This led to the addition of using changes in the tools

and camera positions as conditionals for when to collect images.

One of the project's research questions was to investigate how different training strategies, such as style-transfer, would affect the performance of the models. The D1 dataset was therefore created from the D0 dataset with the help of the NNST model. The theory is that this would bring the domain of the D0 dataset closer to the domain of the D2 dataset and thereby increase the performance of the models. However, the style-transfer process was not optimized beyond visual inspection of a few sample images. This was due to time constraints as the style-transfer of a single image from D0 into an image in D1 could take as long as 80 seconds with the default settings of NNST. Processing all 6391 images of D0 would have taken 5.9 days. Then a model would have to be trained and evaluated before the effects of the specific settings of the style-transfer could be measured. The settings of NNST used in this project were, therefore, chosen to reduce the processing time to about 10 seconds per image while still resulting in a visual difference, an example can be seen in Figure 3.9.

The dataset of real images, D2, was chosen simply because it was the only publically available dataset of catheter segmentation found. The modifications done to the labels of D2 are best motivated by looking at Figure 3.11. The original labels were not very good and in order to get reliable results they had to be fixed. However, even after the alignment of the labels they were not perfect. In images with many tools present some tools that were not catheters were incorrectly labeled, and some that were catheters were not labeled. Fixing all these labels manually would be too labor-intensive.

5.1.3 Training and Experiments

It was decided to train all models locally since free cloud computing is unreliable, especially for longer training sessions, also, this project did not have access to other larger computing units. Moreover, no method for improving memory efficiency when training the models was used.

For U-net, some hyperparameter tuning was done since it had no default values. For augmentation rotation, scaling and shifting decreased performance when used in all tests so they were set to zero. For the other values that were changed in increments before arriving at the values described in subsection 3.5.1.

For YOLOv8, the only hyperparameter tuning done was for the number of training epochs. Except for experiments E10 and E11, all experiments were carried out three times with 100, 150, and 200 set as the number of training epochs. The choice to tune the number of epochs was made because that seemed to matter a lot when we did initial training tests. This was especially true in experiment E9 where the model trained on only 15% of D2 (real images). The chosen range of different epochs to test seemed to be appropriate since not all models trained for 200 epochs were the best. Experiments E13 and E15 got the best results with models trained for 150 epochs. However, no model train for 50 epochs gave the best result compared to the 150

and 200 epoch models in each experiment. No extensive tuning was done for other parameters of the YOLOv8 model. Tuning other parameters could potentially have given notable performance increases. However, due to the combinatory explosion that occurs with many parameters, it would take way too much time to tune each model for each experiment.

The design of the experiments was decided upon based on the research questions. Experiments E2-7 and E10-15 correspond directly to the research questions while experiments E0-1 and E8-9 are used as baselines. Furthermore, almost all experiments exist in two versions, one trained/finetuned on 15% of D2, and one which uses 70% of D2. The reasoning for this was to try to reveal any potential advantage the pre-trained models might have. Additionally, it was done in line with the objective of dealing with the problem of lack of data. To see if performance could be similar with a small amount of annotated images. However, the conclusion made from that approach is limited due to the lack of diversity between the images in the D2 dataset.

Furthermore, dataset D2 only contained data from 6 patients making it hard to split the data on a patient level. One alternative would be to divide it on a patient level anyway and have either the train set or the test set be very skewed towards a few patients. The paper that published the dataset used different approaches [7] one where they divided all images from every patient into the fine-tune/test split and two others where they did it on a patient level. They drew their primary conclusions from the model trained on images from all patients, the same as for this thesis. The problem with this is that it causes data leakage since the models have access to information from the same patient during training. This affects all experiments except for the ones that are only trained on the simulated or style-transfer datasets.

5.2 Results

Firstly, one of the research questions of this project was "To what extent can the segmentation capabilities of a model trained on simulated images from Mentice generalize to real fluoroscopic X-ray images?". The results from experiments E2, E3, E10, and E11 show some performance indicators of this. The plots showing Dice and mIoU over confidence of these experiments can be seen in Figure 4.1 and 4.4. Numerical results can also be seen in Table 4.1 and 4.2 respectively. By looking at these numbers we can see that the experiments using YOLO (E10 & E11) outperform their U-net counterparts (E2 & E3). We can also see that the experiments with synthetic data without using style-transfer (E2 & E10) outperform the models trained with the style-transfer dataset (E3 & E11).

Another interesting observation that can be made by also observing Table 4.3 is that for U-net the models seem to overfit to the training data so that the model that performs better on the synthetic data performs worse on the real data, however, this does not seem to hold for YOLO. Also for the U-net experiment that had the best performance on real data, E2, the result from the dataset it was trained on was very close to the performance on the real dataset, 0.5282 and 0.5007 Dice

respectively. Qualitative results from this model can also be seen in Figure 4.8.

Furthermore, the method used is simple. The models look at one image at a time and don't have any temporal information or other context to consider. Upon visual inspection of the segmentations, it is clear that even a simple post-processing step like aggregating information over 5-10 images or using multi-shot detection might have given a significant increase in performance. This is further supported by false positives often appearing in the corners of the images significantly impacting the performance metric even though these would be easy to remove in a post-processing step. Additionally, the labels of D2 are still not completely correct even after the modifications of them. Images from one singular time sequence contain catheters that are not labeled. See the image in the top left of Figure 4.10. Finally, a dice around 0.5 can be regarded as promising given that the models have not seen any real images and that there is clear room for improvement.

Secondly, the research question "To what extent can the simulated images increase the performance of the segmentation model?". This can be answered by comparing the results of experiments trained directly on the real data (E0-1 & E8-9) with the fine-tuning experiments (E4-7 & E12-15). This comparison is what is done in Table 4.4. The absolute performance values can again be seen in Table 4.1 and 4.2 respectively. From these experiments we can see that pretraining using the stylized data is usually better than using synthetic data straight from the simulation, the exception being for U-net with the 15% training split. This is the opposite result found for the previous research question where the stylized data performed worse and not better. However, we still see that YOLO outperforms U-net. Numerically, a performance increase of up to 2.5% was found with an average relative increase of 0.3%/0.6% (Dice/IoU) for simulated data and 0.9%/1.2% for style-transfer data.

Thirdly, the research question "How do the results compare to previous studies performing segmentation on real fluoroscopic X-ray images?". To answer this question the easiest paper to compare to is Gherardini *et.al* where they used the same real dataset and their best result was a Dice coefficient of 0.58, comparing that to our best 0.8803 which is a 51.8% increase. However, as noted previously the labels of that dataset needed updating which was not done in this paper making it not a fair apples-to-apples comparison.

Another interesting comparison is with the chest X-ray paper by Boccardi *et.al* and they achieved a 0.739 Dice score. Comparing that to our best Dice of 0.8803 which is a 19.1% increase. However, they used a different imaging modality and segmented a different type of catheter so it is also hard to make a direct comparison.

To conclude the comparisons, it would be advantageous if a benchmark existed for catheter segmentation and even better if it was catheter segmentation for X-ray fluoroscopy specifically. Also, the real dataset used did not contain very diverse data meaning that the models will have trained on images very similar to the ones it is evaluated on. This also explains why high results can be achieved using only 15%

of the dataset to train with, even with no pre-training. However, this is also a very niche research area where data is hard to come by so the lack of public datasets and benchmarks is not surprising. The aforementioned data leakage problem also makes the comparison harder.

Finally, the research question "How do different model architectures and training strategies such as style-transfer and fine-tuning on real images affect the performance of the models?". This question ties together all the results of what we investigated. The best result achieved was 0.8803 Dice using YOLOv8 (E14), comparing that to the best U-net model (E6) with 0.8314 Dice the performance increase is 5.9%. We can also see that fine-tuning increases the results, comparing the best model without fine-tuning from E10 with a 0.5182 Dice score, to the 0.8756 obtained with fine-tuning that model(E13), a 69.1% increase; similar increases are found for all fine-tuning experiments.

5.3 Social and Ethical Issues

We do not provide any of the code or datasets used in this project. This complies with patient privacy and data security requirements since Mentice has access to real images that are not publicly available. For the synthetic data the anatomy used is based on real patients, however, it is uncertain to what extent it is possible to gather real-world data from these models. In any case, they are not published either.

Not sharing the data set can be seen as lacking transparency. Also, it is harder for a third party to verify the results to hold us accountable. However, we still share information about the models we use, what evaluation metrics we use as well as outlining the training procedures. This helps mitigate this issue and increases transparency.

For the resulting product of this project, it is not developed as a medical product. Therefore, we do not consider aspects such as regulatory compliance, equitable access to healthcare, and impact on healthcare professionals. This is because an application that is not intended as a medical product will likely have little to no impact on these areas.

We try to use a large and varied dataset. However, the model might still be biased. For example, the images generated in the simulation contain anatomies that are quite uniform outside of the anomalies in the cardiovascular system. This is probably best dealt with by using real images with a wider variety of anatomies for training/fine-tuning, also certain types of data augmentation might help. We do not evaluate the bias of the models or the effect of the counter strategies on bias.

Also, when it comes to informed consent the patients that have agreed to have their data used might not know that the data will be used for developing AI. It is important that when data is collected from patients there is transparency in communication in what the data will be used for. This is less of an issue when using

simulated data.

6
Conclusion

This chapter will cover the conclusions from this thesis. That includes the major findings from the results as well as putting them in context. Finally, future work that could be interesting to investigate based on our findings is discussed.

6.1 Findings

Firstly, by answering the research question "To what extent can the segmentation capabilities of a model trained on simulated images from Mentice generalize to real fluoroscopic X-ray images?" a Dice score of up to 0.5182 was achieved when training on data from the simulation. This can be regarded as promising given that the models have not seen any real images and that there is clear room for improvement.

Secondly, the findings from answering the research question "To what extent can the simulated images increase the performance of the segmentation model?", was that performance could be increased by up to 2.5% with an average relative increase of 0.3%/0.6% (Dice/IoU) for simulated data and 0.9%/1.2% for style-transfer data. This resulted in the best model achieving a Dice score of 0.8803.

Thirdly, findings were obtained by analyzing previous work to answer the research question "How do the results compare to previous studies performing segmentation on real fluoroscopic X-ray images?". The comparison showed that our result compared favorably to other studies. However, the dataset used to obtain our results limits the conclusions that can be drawn from this.

Fourthly, the research question "How do different model architectures and training strategies such as style-transfer and fine-tuning on real images affect the performance of the models?" was investigated. The result of comparing style-transfer to not using it is seen by comparing the numbers obtained by answering the second research question, while also noting that the style-transfer data performed worse without fine-tuning. When comparing the architectures YOLOv8 outperforms U-net and by comparing the best models for each YOLO performed 5.9% better. The largest effect was found by fine-tuning, for the best model a 69.1% increase was found and similar results were found for all fine-tuning experiments.

6.2 Future Work

For future work, there are multiple areas that would be interesting to investigate. Based on the results from answering the research question of how well the synthetic data generalizes to real images, it would be interesting to investigate how post-processing techniques could improve performance. Often in segmentation masks produced by these models, edges of the image are detected as positives and the detected segments are often missing snippets. This could potentially be improved by utilizing post-processing techniques.

Another idea would be to optimize the domain adaptation method used, to find a method that is a better fit for improving segmentation performance and perhaps it would also be beneficial to use a model intended for medical applications. Moreover, other forms of synthetic images such as methods using generative AI could be investigated.

Furthermore, the models in this thesis use one-shot segmentation, and using multi-shot or temporal information could potentially be used. The use of temporal information is also logical for fluoroscopy since during procedures a short sequence of images is usually utilized for visualization.

Finally, testing on a more diverse dataset and setting up better benchmarks for segmentation models in the medical field would be good for the further development of this area. This would also make it easier to deal with the data leakage problem.

Bibliography

[1] Ayush Chaurasia and Glenn Jocher. "Ultralytics YOLOv8 Docs". ultralytics.com. Available: https://docs.ultralytics.com/ (Accessed: 2024-02-14)

[2] YOLOv8. (2023). Ultralytics. Accessed: 2024-01-25. [Online]. Available https://github.com/ultralytics/ultralytics

[3] YOLOv8. (2023). Ultralytics. Accessed: 2024-04-09. [Online]. Available https://yolov8.org/yolov8-architecture/

[4] YOLOv8. (2023). Ultralytics. Accessed: 2024-05-02. [Online]. Available https://docs.ultralytics.com/tasks/segment/

[5] ryouchinsa/donut. (2023). Github. Accessed: 2024-04-17. [Online]. Available https://github.com/ryouchinsa/donut

[6] Pierre Ambrosini and Daniel Ruijters and Wiro J. Niessen and Adriaan Moelker and Theo van Walsum, "Fully Automatic and Real-Time Catheter Segmentation in X-Ray Fluoroscopy", International Conference on Medical Image Computing and Computer-Assisted Intervention, 2017-09-04, Accessed: 2024-01-31. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-66185-8_65

[7] Marta Gherardini and Evangelos Mazomenos and Arianna Menciassi and Danail Stoyanov, "Catheter segmentation in X-ray fluoroscopy using synthetic data and transfer learning with light U-nets", Computer Methods and Programs in Biomedicine, August 2020, Accessed: 2024-01-31. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0169260719312301

[8] Linus Kreitner and Johannes C Paetzold and Nikolaus Rauch and Chen Chen and Ahmed M Hagag, "Synthetic optical coherence tomography angiographs for detailed retinal vessel segmentation without human annotations", IEEE Trans Med Imaging, January 2024, Accessed: 2024-01-31. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/38224512/

[9] Zambrano-Vizuete M and Botto-Tobar M and Huerta-Suárez C and Paredes-Parada W and Patiño Pérez D, "Segmentation of Medical Image Using Novel Dilated Ghost Deep Learning Model", Comput Intell Neurosci, August 2024, Accessed: 2024-01-31. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9391132/

[10] Misan Etchie, "Black Mirror in Real Life: Facebook Now Owns and is Developing a City", hackernoon.com, June 2021, Accessed: 2024-02-08. [Online]. Available: https://hackernoon.com/tech-giants-building-cities-there-is-a-catch-2idb35x5

[11] Tsung-Yi Lin and Michael Maire and Serge Belongie and Lubomir Bourdev and Ross Girshick and James Hays and Pietro Perona and Deva Ramanan and

C. Lawrence Zitnick and Piotr Dollár, "Microsoft COCO: Common Objects in Context", arXiv e-prints, 2014-05-01, Accessed: 2024-02-08. [Online]. Available: https://arxiv.org/abs/1405.0312

[12] Viacheslav V. Danilov et.al, "Use of semi-synthetic data for catheter segmentation improvement", Computerized Medical Imaging and Graphics, June 2023, Accessed: 2024-02-12. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S089561112300006X#ab0015

[13] X. Wu, J. Housden, Y. Ma, B. Razavi, K. Rhode and D. Rueckert, "Fast Catheter Segmentation From Echocardiographic Sequences Based on Segmentation From Corresponding X-Ray Fluoroscopy for Cardiac Catheterization Interventions," in IEEE Transactions on Medical Imaging, vol. 34, no. 4, pp. 861-876, April 2015, Accessed: 2024-02-12. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6913532

[14] Nicholas Kolkin and Michal Kucera and Sylvain Paris and Daniel Sykora and Eli Shechtman and Greg Shakhnarovich, "Neural Neighbor Style Transfer", arXiv e-prints, Accessed: 2024-05-23. [Online]. Available: https://arxiv.org/abs/2203.13215

[15] Abolfazl Farahani and Sahar Voghoei and Khaled Rasheed and Hamid R. Arabnia, "A Brief Review of Domain Adaptation", arXiv e-prints, 2022-10-07, Accessed: 2024-02-13. [Online]. Available: https://arxiv.org/abs/2010.03978v1

[16] Stanford. (2023). "Stanford CS330 Deep Multi-Task & Meta Learning - Domain Adaptation l 2022 I Lecture 13". [Online]. Available: https://www.youtube.com/watch?v=Uk6MU_PLDMs

[17] Stefan Elfwing and Eiji Uchibe and Kenji Doya, "Sigmoid-Weighted Linear Units for Neural Network Function Approximation in Reinforcement Learning", arXiv e-prints, 2017-02-10, Accessed: 2024-03-19. [Online]. Available: https://arxiv.org/abs/1702.03118

[18] Prajit Ramachandran and Barret Zoph and Quoc V. Le "Searching for Activation Functions", arXiv e-prints, 2017-10-16, Accessed: 2024-03-19. [Online]. Available: https://arxiv.org/abs/1710.05941v2

[19] Diganta Misra, "Mish: A Self Regularized Non-Monotonic Activation Function", arXiv e-prints, 2020-08-23, Accessed: 2024-03-19. [Online]. Available: MISH, paper:https://arxiv.org/vc/arxiv/papers/1908/1908.08681v1.pdf

[20] Olaf Ronneberger and Philipp Fischer and Thomas Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation", arXiv e-prints, 2015-05-18, Accessed: 2024-03-26. [Online]. Available: https://arxiv.org/abs/1505.04597

[21] Jun-Yan Zhu and Taesung Park and Phillip Isola and Alexei A. Efros, "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks", arXiv e-prints, 2017-03-30, Accessed: 2024-03-26. [Online]. Available: https://arxiv.org/abs/1703.10593

[22] Karen Simonyan and Andrew Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition", arXiv e-prints, 2014-09-04, Accessed: 2024-03-26. [Online]. Available: https://arxiv.org/abs/1409.1556

[23] Diederik P. Kingma and Jimmy Ba, "Adam: A Method for Stochastic Optimization", arXiv e-prints, 2014-12-22, Accessed: 2024-04-02. [Online]. Available: https://arxiv.org/abs/1412.6980

[24] Ilya Loshchilov and Frank Hutter, "Decoupled Weight Decay Regularization", arXiv e-prints, 2017-11-14, Accessed: 2024-04-02. [Online]. Available: https://arxiv.org/abs/1711.05101

[25] F. Boccardi *et.al*, "Bottom-Up instance segmentation of catheters for Chest X-rays", arXiv e-prints, 2023-12-06. Accessed: 2024-04-09. [Online]. Available: https://arxiv.org/html/2312.03368v1

[26] V. Sabramanian *et.al*, "Automated Detection and Type Classification of Central Venous Catheters in Chest X-rays", arXiv e-prints, 2019-07-02. Accessed: 2024-04-09. [Online]. Available: https://arxiv.org/abs/1907.01656

[27] P. Chang *et.al*, "Robust Catheter and Guidewire Tracking Using B-Spline Tube Model and Pixel-Wise Posteriors", IEEE Robotics and Automation Letters, 2016-01-13. Accessed: 2024-04-09. [Online]. Available: https://ieeexplore.ieee.org/document/7381624

[28] Chien-Yao Wang and Hong-Yuan Mark Liao and I-Hau Yeh and Yueh-Hua Wu and Ping-Yang Chen and Jun-Wei Hsieh, "CSPNet: A New Backbone that can Enhance Learning Capability of CNN", arXiv e-prints, 2019-11-27, Accessed: 2024-04-10. [Online]. Available: https://arxiv.org/abs/1911.11929

[29] E.N. Manson *et.al*, "Image Noise in Radiography and Tomography: Causes, Effects and Reduction Techniques", Curr Trends Clin Med Imaging, 2019-10-11. Accessed: 2024-04-24. [Online]. Available: https://juniperpublishers.com/ctcmi/CTCMI.MS.ID.555620.php

[30] "Percutaneous Coronary Intervention (PCI)", Yale Medicine, Accessed: 2024-04-24. [Online]. Available: https://www.yalemedicine.org/conditions/percutaneous-coronary-intervention-pci

[31] "What is TAVR? (TAVI)" American Heart Association, Accessed: 2024-04-24. [Online]. Available: https://www.heart.org/en/health-topics/heart-valve-problems-and-disease/understanding-your-heart-valve-treatment-options/what-is-tavr

[32] "Coronary Angiogram" British Heart Foundation, 2023, Accessed: 2024-04-24. [Online]. Available: https://www.bhf.org.uk/informationsupport/tests/angiogram

[33] "What Is A Cath Lab?" Medwish, Accessed: 2024-05-02. [Online]. Available: https://www.medwish.com/blog/buying-guide/what-is-a-cath-lab/

[34] Jieneng Chen and Yongyi Lu and Qihang Yu and Xiangde Luo and Ehsan Adeli and Yan Wang and Le Lu and Alan L. Yuille and Yuyin Zhou, "TransUNet: Transformers Make Strong Encoders for Medical Image Segmentation", arXiv e-prints, 2021-02-08. [Online]. Available: https://arxiv.org/abs/2102.04306

[35] Wang, Ziyang and Zheng, Jian-Qing and Zhang, Yichi and Cui, Ge and Li, Lei, "Mamba-unet: Unet-like pure visual mamba for medical image segmentation", arXiv preprint arXiv:2402.05079. [Online]. Available: https://github.com/ziyangwang007/Mamba-UNet/blob/main/pdf/MambaUNet.pdf

A

Appendix 1

Metrics over confidence for individual exepriments.

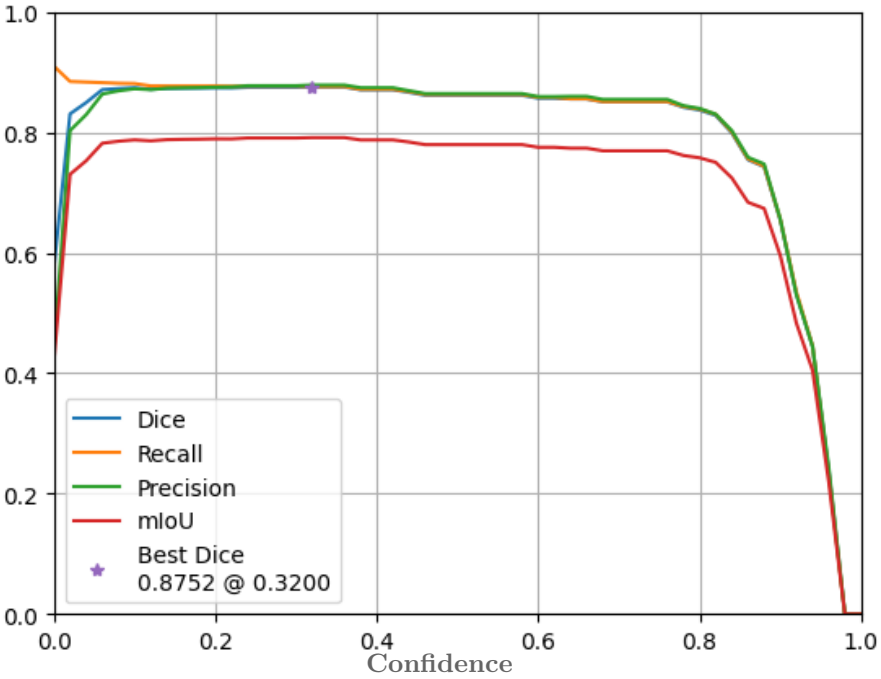


Figure A.1: Model from E8 evaluated on D2.

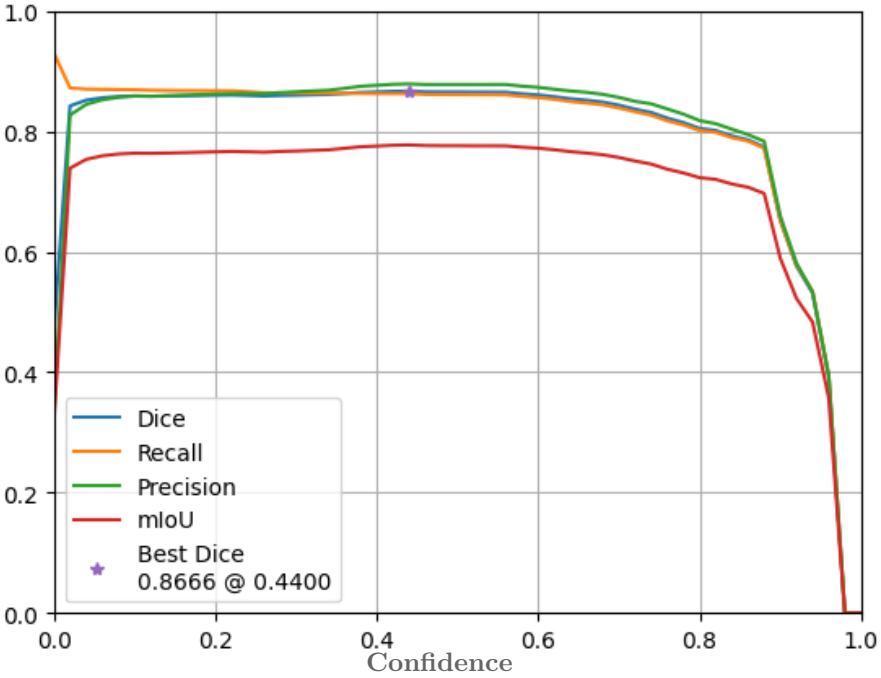


Figure A.2: Model from E9 evaluated on D2.

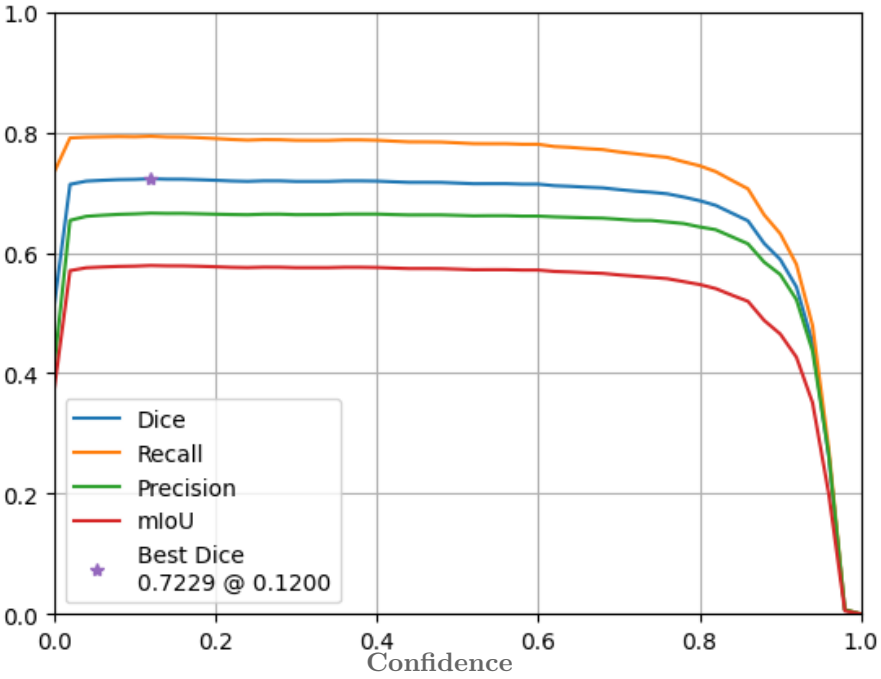


Figure A.3: Model from E10 evaluated on D0.

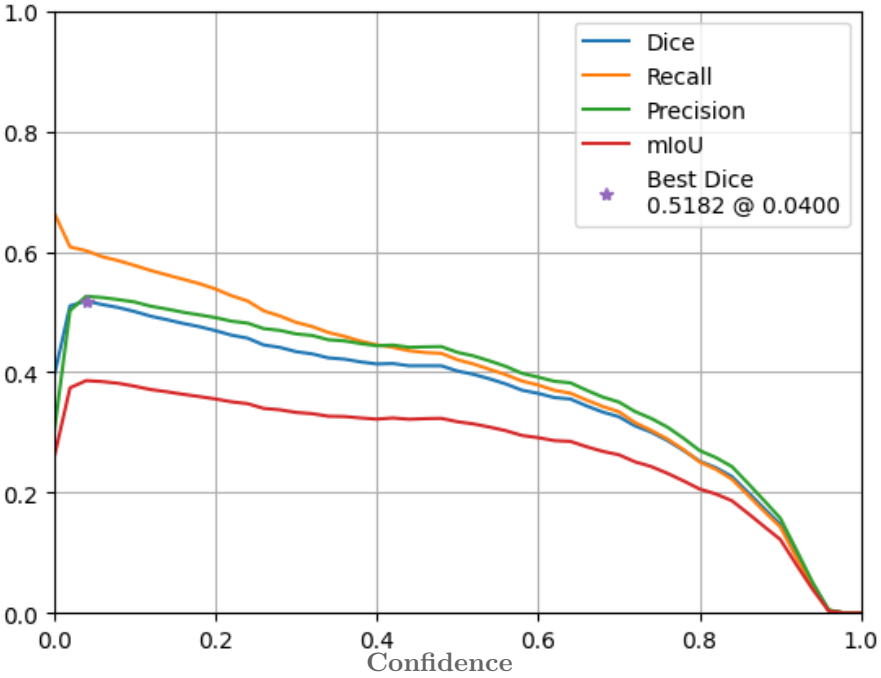


Figure A.4: Model from E10 evaluated on D2.

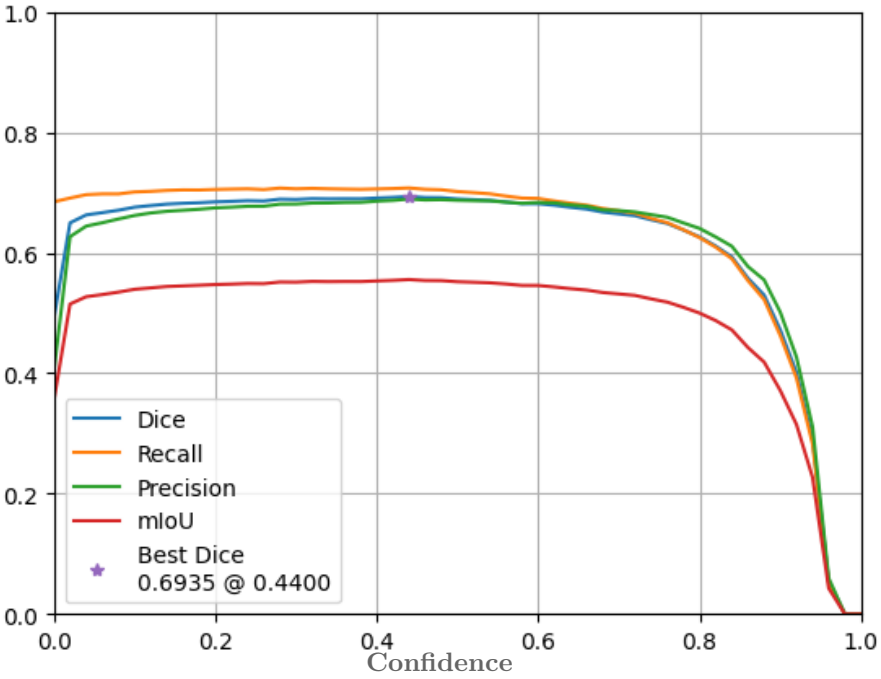


Figure A.5: Model from E11 evaluated on D1.

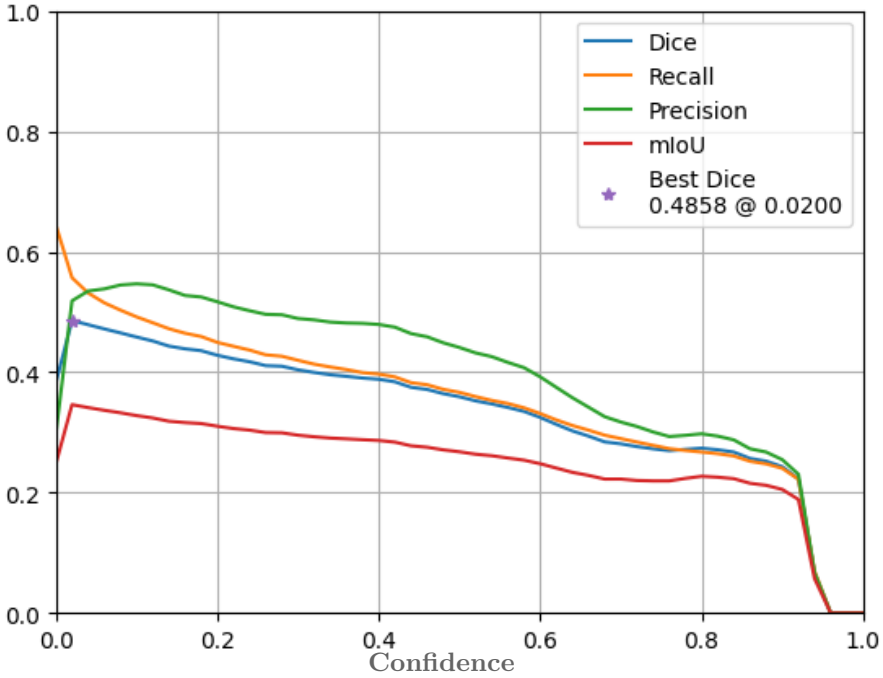


Figure A.6: Model from E11 evaluated on D2.

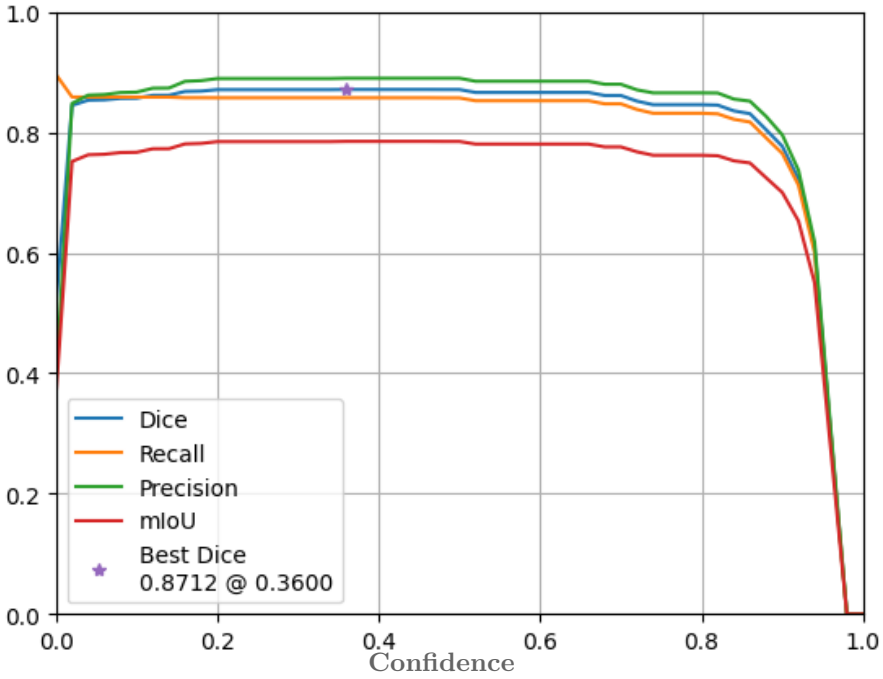


Figure A.7: Model from E12 evaluated on D2.

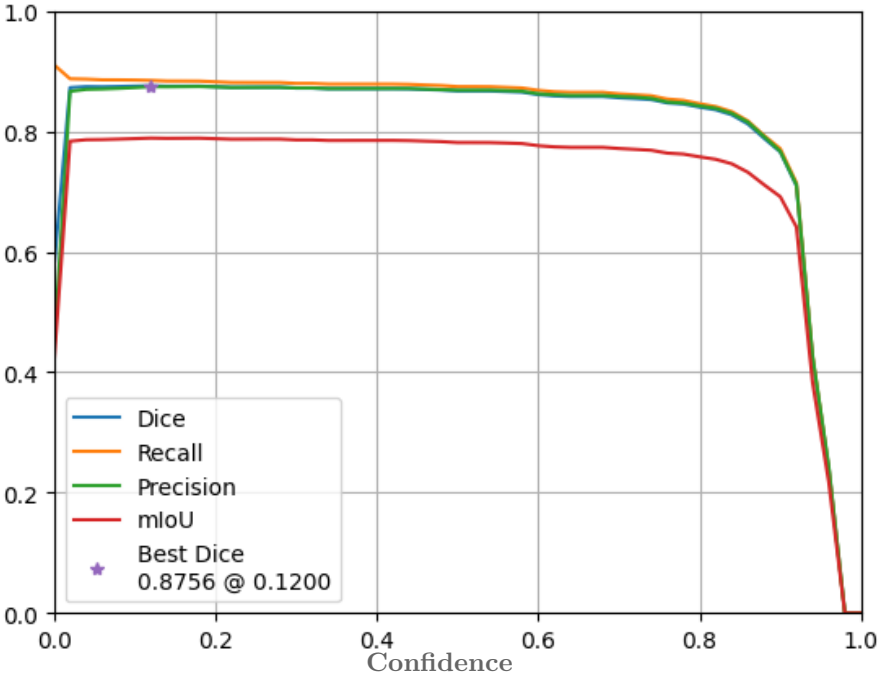


Figure A.8: Model from E13 evaluated on D2.

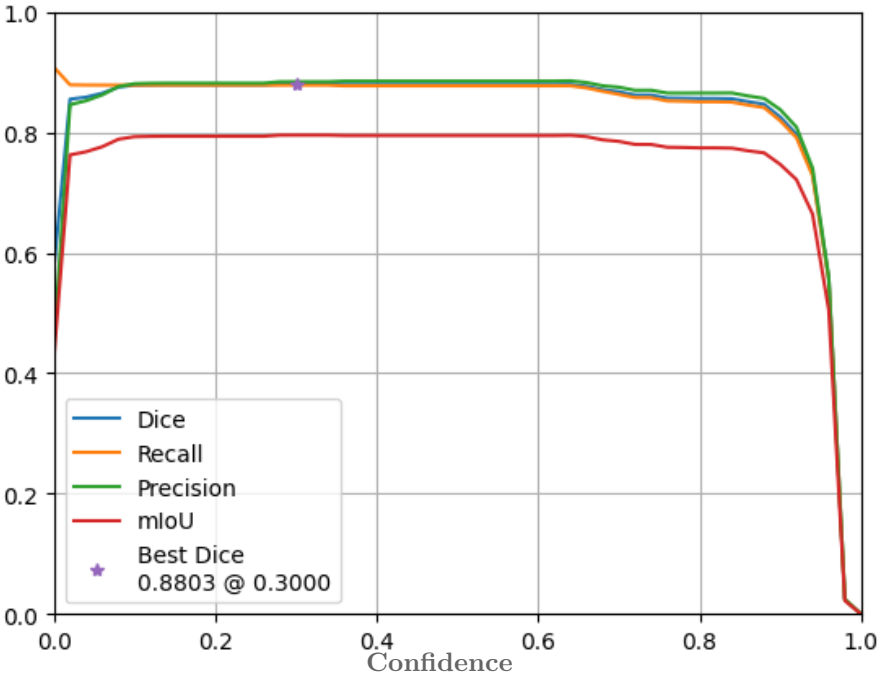


Figure A.9: Model from E14 evaluated on D2.

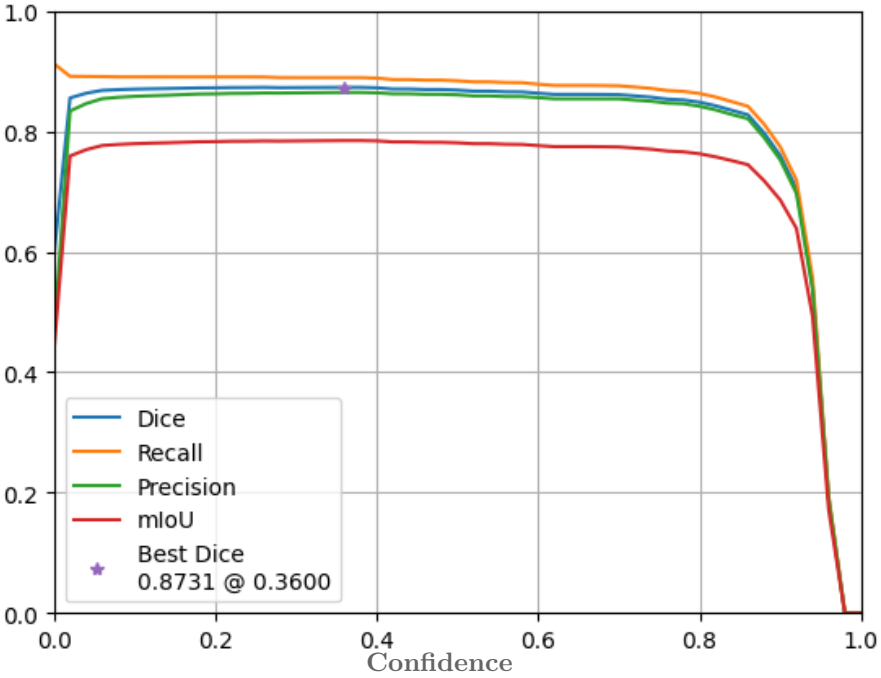


Figure A.10: Model from E15 evaluated on D2.

Argument	Value	Argument	Value
task:	segment	embed:	null
mode:	train	show:	false
model:	yolov8m-seg.pt	save_frames:	false
data:	<PROJECT_DIR>/config.yml	save_txt:	false
epochs:	<EPOCHS>	save_conf:	false
time:	null	save_crop:	false
patience:	1000	show_labels:	true
batch:	16	show_conf:	true
imgsz:	640	show_boxes:	true
save:	true	line_width:	null
save_period:	100	format:	torchscript
cache:	ram	keras:	false
device:	null	optimize:	false
workers:	8	int8:	false
project:	<PROJECT_DIR>	dynamic:	false
name:	<TRAIN_RUN_NAME>	simplify:	false
exist_ok:	false	opset:	null
pretrained:	false	workspace:	4
optimizer:	SGD	nms:	false
verbose:	true	lr0:	0.01
seed:	0	lrf:	0.01
deterministic:	true	momentum:	0.973
single_cls:	false	weight_decay:	0.0005
rect:	false	warmup_epochs:	3.0
cos_lr:	false	warmup_momentum:	0.8
close_mosaic:	0	warmup_bias_lr:	0.1
resume:	false	box:	7.5
amp:	true	seg:	7.5
fraction:	1.0	cls:	0.5
profile:	false	dfl:	1.5
freeze:	null	pose:	12.0
multi_scale:	false	kobj:	1.0
overlap_mask:	true	label_smoothing:	0.0
mask_ratio:	1	nbs:	64
dropout:	0.5	hsv_h:	0.015
val:	true	hsv_s:	0.7
split:	val	hsv_v:	0.4
save_json:	false	degrees:	180.0
save_hybrid:	false	translate:	0.2
conf:	null	scale:	0.5
iou:	0.7	shear:	0.0
max_det:	300	perspective:	0.0
half:	false	flipud:	0.0
dnn:	false	fliplr:	0.5
plots:	true	mosaic:	1.0
source:	null	mixup:	0.0
vid_stride:	1	copy_paste:	0.0
stream_buffer:	false	auto_augment:	randaugment
visualize:	false	erasing:	0.4
augment:	false	crop_fraction:	1.0
agnostic_nms:	false	cfg:	null
classes:	null	tracker:	botsort.yaml
retina_masks:	false	save_dir:	<SAVE_DIR>

Table A.1: Complete list of all YOLOv8 argument values used during training.

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY