Segmentation of the Common Carotid Artery from Ultrasound Images using UNet

Oskar Friberg

2020

Master's Thesis in
Biomedical Engineering

Supervisors: Tobias Erlöv, Magnus Cinthio and Shinichiro Omachi

LUND UNIVERSITY

LTH

FACULTY OF ENGINEERING

Department of Biomedical Engineering, Lund University & Department of Communications Engineering, Tohoku University

Abstract

The common carotid artery (CCA), the artery that supplies our brains with oxygen, is of great importance in stroke research. The usual way to monitor the artery is by using ultrasound (US) imaging. An automated way of segmenting out the arteries from the US images is desired to optimize research. With an upswing in development of convolutional neural networks (CNN), this is now possible. CNN uses a training dataset to tweak a set of trainable parameters so that it can successfully classify images that is similar to the training data, and can then successfully segment out the CCA in new images.

However, the US images tend to have a large variation from patient to patient, since the artery and, the tissue and veins around the artery are not identical from patient to patient. A normal way that convolutional neural networks tackle this is by using a large training dataset to learn the variation, sadly a large dataset of out-segmented CCAs does not exist. This presents some challenges to using CNN to segment CCAs.

In this thesis, a case study has been performed with the fully convolutional neural network architecture called UNet, trained with less than 200 US images of the CCA to study if the network can segment the CCA in new images. With training data of around 200 US images, the network's output is compared to an expert's segmentation as the ground truth. The conclusion is that the network show promising result with an average of a 0.871 DICE similarity coefficient. Two key limitations were identified, first that images with a large difference in greyscale from the training data need to be preprocessed before, and secondly that artifacts need to be reduced to get a good segmentation.

The study does promote the use of UNet for segmenting the CCA in US images. With further development in postprocessing, a reliable way to segment the CCA in US images is possible.

Keywords: MSc, Deep Learning, CNN, Computer Vision, Ultrasound, Carotid Artery

Acknowledgements

First and foremost I would like to thank my supervisors both at LTH, Tobias Erlöv and Magnus Cinthio, and at Tohoku University, Shinichiro Omachi, for their guidance in the biomedical engineering respectively the machine learning aspects for my research. But also in their guidance and feedback regarding the content of the thesis.

Further thanks to the people at the Omachi Laboratory at Tohoku University, for welcoming me and sharing their knowledge in machine learning. Extended thanks to Tomo Miyasaki for his expertise in the area and the tip to use UNet for my project. Also to my fellow student at the laboratory Nguyen Manh Huy, for making me realize the strength of neural networks and helping me to adapt to the Japanese culture.

Contents

Chapter 1
Introduction

Medical imaging is crucial for medical diagnostics and research. Among the popular imaging methods, ultrasound (US) imaging is one of the best options for tissue, organ and vascular system. It is a method that utilizes the properties of sound and the acoustics of the body. US is a non-invasive imaging tool, relatively cheap and gives out no radiation, making it completely safe, widely accessible and easily tolerable compared to other medical imaging tools.

With the help of US imaging, the main artery that supply the brain with oxygenated blood, the common carotid artery (CCA), can be monitored and captured in images. Rupture of plaque in the CCA often results in stroke, which was the second leading cause of death 2016 [1], and is therefore of substantial focus in biomedical research. At Lund University, research on the CCA is being conducted and the researchers use US to understand how the movement of the CCA is related to different diseases that increase risks of stroke [2]. However, studying a large amount of US images of CCA from test patients under a certain time is very time consuming and more efficient approaches are needed.

One way that the researchers wish to study the CCA, is by taking multiple US images of the circular cross-section of the CCA and create a 3D model for further research. The current way this is done is using either completely manual segmentation or using semi-automated functions which requires the researcher to still give the region of the CCA in the images. This is very time consuming since a lot of pictures are needed to give an accurate analysis, and an automated segmentation is desired to more efficiently research the CCA. This is where Deep Learning (DL) can be a possible solution.

DL is a set of machine learning methods that are set to perform a specific task without explicit instructions, instead only relying on patterns and an interface. DL uses artificial neurons, which is a computerized version of our neurons in the brain. These neurons' purpose is through representational learning, to solve future tasks based on this learning. In this thesis, a convolutional neuronal network (CNN) is used. In many segmentation and com-

puter vision tasks, CNN has had a big upswing in popularity in the last years. CNN uses multiple trainable neurons in the form of filters to extract features from the image, which is then classified using another set of neurons.

However, in US image analysis CNN has not been as successful since they need large datasets for training, sometimes more than a thousand images. This thesis focuses on to determine the possibility of using CNN to automatically segment out the CCA from US images. The proposed network framework is a network called UNet, which is well suited for biomedical image segmentation [3]. Since the network is focused on using training samples more efficiently it fits the CCA segmentation problem well since not much training data is available.

Chapter 2
Background

2.1 Ultrasound Imaging of the CCA

US imaging uses the principle that sound echos back when the sound waves hit an object. By sending out a high-frequency sound, the sound penetrates the skin, goes into the tissue then into other parts like the artery. The acoustic impedance, the measurement of how much the sound is reflected after an acoustic pressure as applied on the object, is different for different objects in the body. When a sound travels from one acoustic impedance to another, some of the sound is reflected depending on the angle the sound traveled. This causes scattering of sound when the US goes through the body, and this sound can be recorded after being sent out. By using the time recorded, the depth can be calculated and using the intensity of the sound as greyscale, the US device can display an image representing where the sound has scattered.

The further the sound travels in the body, more scattering will occur in the tissue which causes the continuous traveling sound to lose energy. This can cause loss in quality in the images since the sound reflected back will have lower energy making it harder to pick up. However, the CCA is located around 2 centimeters beneath the skin, which makes it possible to get good quality in the images, and can be seen in fig 2.1. Since the acoustic impedance in blood is different from the surrounding tissue, the artery's outer arterial wall reflects back sound and gives a circular figure in the resulting image. Contrary to veins, the artery remains mostly circular during the cardiac cycle, which makes it easier to segment since it has consistency in its shape.

2.1.1 The Ultrasound Device

The US device consists of roughly five parts. A transducer probe that sends and receives ultrasound using piezoelectric elements, which are elements that can convert electricity to

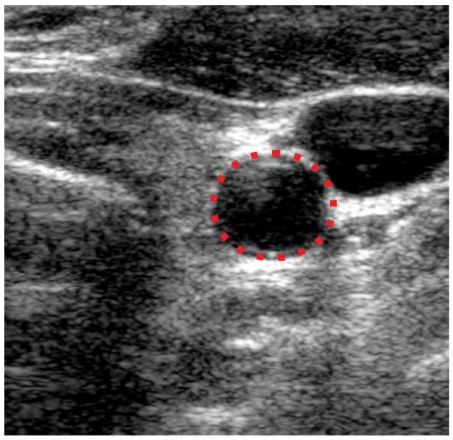


Figure 2.1: US image of the CCA, with the CCA encircled with red dots

sound and vice verse. A generator that generates the electrical signals used by the transducer. A computer that does the calculations to create an image from the recorded echoes. A beamformer which focuses the transmitted and received signals to create better resolution. And finally, a monitor to show the operator what is being captured.

The process of taking a US image is as follows. The transmitter produces a signal with a frequency at around 10 MHz, goes through the beamformer which introduces time delays to the transducers piezoelectric elements sending out the signal in a beam. These piezoelectric elements receive the echoes from the body, which are then sent through the beamformer which focuses the signal from a number of the elements to create just one line in the US image. This process is repeated multiple times to scan the targeted area, and creates multiple signals and sends it to the computer. The intensity of the echoes is presented in a greyscale image where the y-axis represents the time they were received while the x-axis represents the beam number.

2.1.2 Artifacts

When false or misleading information appears in US images from incorrect assumptions such as the propagation of the sound in the tissue, it is called an artifact. If the following assumptions are not fulfilled artifacts appear:

1. The sound propagates straight ahead along the sound beam axis
2. The ultrasound beam is thin
3. The speed of sound is constant
4. The attenuation in tissue is constant
5. The pulse travels only to targets that are on the beam axis and back to the transducer
6. A received echo always arises from the latest transmitted ultrasound pulse

Because of the circular property of the artery, the sound that does not hit the top or bottom part will not echo the sound back directly to the transducer. The worst case is on the sides of the artery, here the sound is echoed back perpendicular to the transducer and is never received. This causes the artery to be less defined at the sides compared to the top and bottom. This artifact is known as a refraction artifact since the sound is spread in a different direction than what is expected.

Another artifact that occurs is called "Posterior Enhancement", which is when the local attenuation is lesser than in the surrounding tissue. This causes the reflected sound to have a higher intensity and becomes brighter in the resulting image compare to similar tissue. Seen in figure 2.1, the tissue below the CCA is brighter compared to similar tissue since the difference in sound attenuation is large between blood and the body tissue.

Speckle

Speckle is another type of artifact and describes the scattering of sound in the tissue and objects in the body. Because of granular interference that inherently exists in the tissue, e.g. small cells, the sound scatters and the quality of the image decreases. Speckle can be seen in US imaging as either bright or dark dots.

Speckle will make the image look noisy, but it is also very much needed in ultrasound imaging. Speckle is the reason we can see the surrounding tissue in figure 2.1. However, speckle also introduces variation in the images since the speckle is different for image to image, something that is considered bad for machine learning methods.

Therefore, speckle reduction is needed to get a clearer image for analysis. The methods that are used to eliminate speckle usually smooths out the image, but ultimately lowering the sharpness of the image. The method used to reduce the US speckle in the CCA images in this project's experiments is called "Speckle Reducing Anisotropic Diffusion"[4] (SRAD). This method uses the pixel's pixel neighbors and calculates the energy based on differential equations using the pixel value and the neighbor pixels' values. SRAD works well for preserving edges, which is crucial when segmenting out circles.

2.2 Artificial Neural Network and the Perceptron

An artificial neural network (ANN), is composed of multiple artificial neurons in different layers, normally called input, hidden or output layer. These neurons are connection in a larger network, and can store and pass information similar to the neurons in our brain.

The simplest ANN-model is called the perceptron, seen in figure 2.2, and consist of an input and an output layer [5]. The perceptron's task is to classify the input to a certain class. Given

an input vector $\mathbf{x} = \{x_1, ..., x_n\}$ from the input nodes, the perceptron compute the output y:

$$y(\mathbf{x}, \omega) = \phi(\sum_{i=0}^{n} \omega_i x_i + b) = \phi(\omega^T \mathbf{x} + b). \tag{2.1}$$

Where ω_i is the weight between the input node x_i and the output node y, b is called bias, and ϕ is the activation function which determines the type of mapping the perceptron does. The perceptron uses a linear activation function:

$$Linear: \quad \phi(x) = x \tag{2.2}$$

The perceptron learns by finding suitable weights ω and bias b, using a training dataset with input patterns $\{\mathbf{x_p}\}_{p=1,...,P}$ and the input patterns' corresponding targets d_p. The most suitable parameters are when $y(\mathbf{x}_p) = d_p, \ \forall p$. A learning algorithm is used to find these parameters from a dataset with input vectors and a true output for each vector, and the most common algorithm is the gradient descent. The algorithm can be broken down into two steps:

1. Initiate the weight vector ω (include the bias in the vector) with small random numbers.
2. Repeat the following until all $y(\mathbf{x}_p) = d_p$:
   (a) Take an input pattern μ from the training set.
   (b) Compute the output $y(\mathbf{x}_\mu)$
   (c) If $y(\mathbf{x}_\mu) \neq d_n$ update ω to

$$\omega(t + 1) = \omega(t) + \eta(d_\mu - y(\mathbf{x}_\mu))\mathbf{x}_\mu, \tag{2.3}$$

   where η is a constant called the learning rate and t is the current iteration.

By training the perceptron with training data, simple linear classification tasks can be handled with ease. However with more non-linear classifications and more complex input patterns, like images, a more advanced model is needed since the perceptron lacks dimensionality.

2.2.1 Non-linear activation functions

The perceptron can only solve linear problems because of the linearity of its activation function:

$$y(\mathbf{x}, \omega) = \phi(\sum_{i=0}^{n} \omega_i x_i + b) = \omega^T \mathbf{x} + b. \tag{2.4}$$

This final function $y = \omega^T \mathbf{x} + b$ is used to find an linear approximation $y(\mathbf{x})$ of the function. This limits the network to finding linear approximations since the final approximation will always be linear.

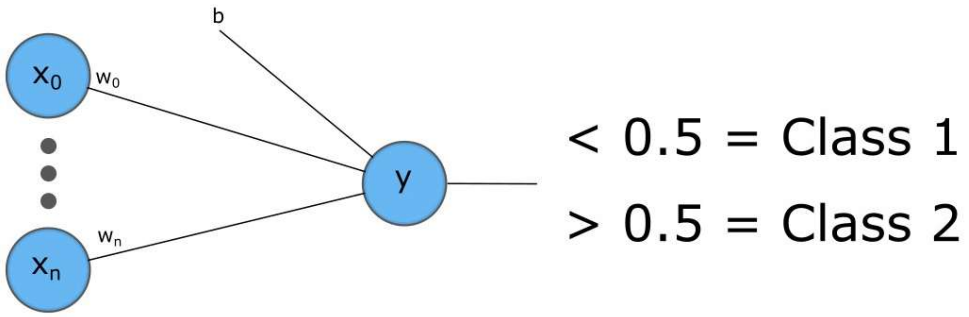


Figure 2.2: The perceptron, the simplest Artificial Neural Network. Classifying the input features $x_0 ... x_n$ into two classes using weights $w_0 ... w_n$ and a bias b.

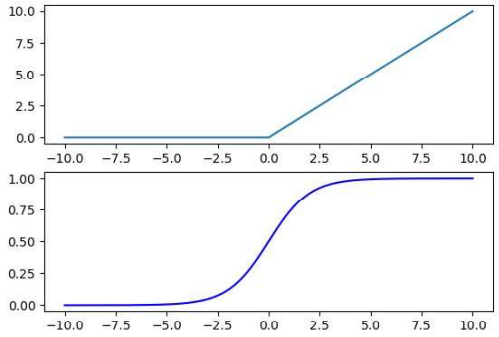


Figure 2.3: Top graph shows the Rectified Linear Unit (ReLU), and the bottom shows the Sigmoid function.

A way to solve this is by using non-linear activation functions to introduce non-linearity to the approximation. Two types are used in this thesis, the Rectified Linear Unit (ReLU) and the Sigmoid function. These functions are non-linear:

$$ReLU: \quad \phi(x) = max(0, x), \qquad Sigmoid: \quad \phi(x) = \frac{1}{1 + e^{-x}}. \tag{2.5}$$

In figure 2.3, their graphs are shown between the value of -10 and 10. The bias is used to shift the graph's center in the x-axis, and the weights are used to determine the different input features value.

ReLU is the linear activation function for values above zero, and the main purpose is to add non-linearity to the network. The Sigmoid function's value is for the main part either 0 or 1, making it good for classifying binary classification problems.

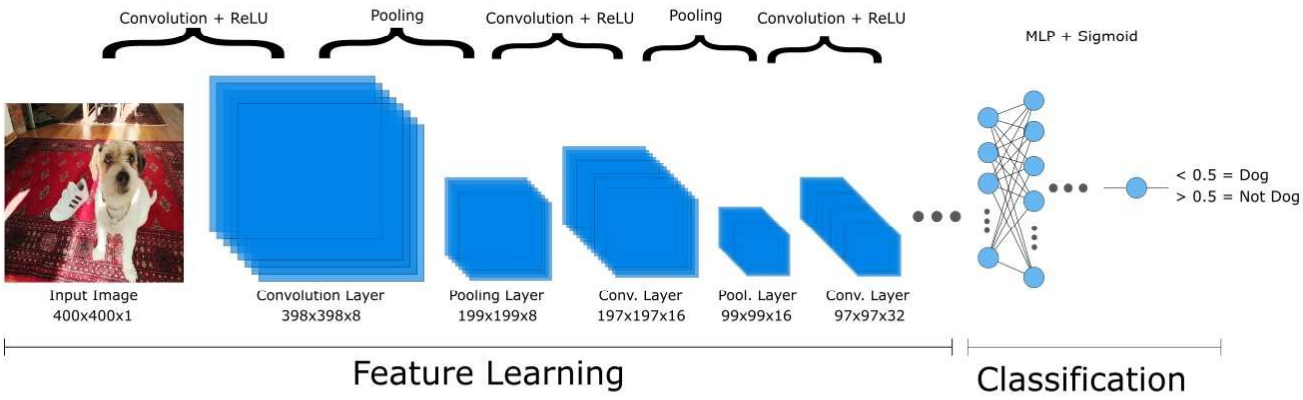


Figure 2.4: Example of a CNN with the task of image classifying if a dog is present in an image. Shows the two main steps of a CNN, the Feature Learning step and the Classification step.

2.3 Convolutional Neural Network

A CNN is usually composed of three steps, an input step, a feature learning step and finally a classification step. The first step is simply the input on what should be classified which in image analysis is images. Images are usually inputted as tensors, in the form of $WxHxC$, with W and H being the width and height of the tensors and C being the feature channels, for example, the color scheme in the images. These parameters determine the input tensor for the network, where W and H determine the size of the tensor and C the number of feature channels. And the bigger these maps are and the more maps exists the more computation time is needed for the algorithm. Since images tend to be quite big the classification task becomes much more complex and multi-dimensional, which a simple ANN would not manage.

CNN is commonly used for image classification. In figure 2.4 a CNN architecture is shown with the purpose to classify if images contain a dog or not. The CNN architecture's main parts are the feature learning part and the classification part, which contains all the weights and biases. These parameters are trained by using an algorithm called backpropagation with an error minimization method, further called optimizer.

2.3.1 Feature learning

CNN's feature learning step consists most commonly of two different layers, the convolution layers, and the pooling layers. The convolution layers use a convolution operation with multiple adaptive filters called kernels, to find features in the input. Seen in figure 2.5, the convolution operation slides its kernel over the input producing a feature map. In the figure, a convolution operation is done on a matrix of size 6x6 with a 3x3 kernel, and the red squares represent the first step of the convolution operation. Here the top right 3x3 part of the matrix's cross-product with the kernel matrix is the first element in the output matrix. The kernel's elements are used as weights in the network and are updated after each training

step, much like the perception's neurons. These elements determine what kind of filters are applied in the network.

The kernels' slide is determined by two parameters called padding and stride. There are two common types of padding, "same padding" and "valid padding". "Same padding" takes care of the edges in the convolution by putting zeros around the matrix, seen in the figure 2.5. This causes the output to have the same width and height as the input. "Valid padding" skips this and makes the output lose spatial size. E.g. if a kernel with the size of 3x3 is applied on a 5x5 input matrix the resulting matrix is going to be a 3x3 matrix. The stride parameter controls how many steps the kernel takes while it slides. This means for a stride of one the spatial size will be the same, while for a stride of two the size will be halved.

The input to the convolution layers is usually tensors with multiple feature channels, multiple layers of matrices. Thus the filters in these layers need to have multiple kernels to match the number of feature channels that the input has. For example, given an input color image with three feature channels representing red, green and blue color, the filter will have one kernel for each of the three colors. This means that the convolution operation is done and features are extracted from each channel separately. The features matrices from each input channel are summarized together, and a final feature matrix is outputted. This means that with more input channels more operations are needed.

The goal of the convolution layer is to extract features for classification, and normally the more features the better. Therefore, multiple filters are used in each layer creating a tensor. The final resulting tensor will have the same number of feature channels as the number of filters. This map is then put through an activation function (normally ReLU) to increase the non-linearity in the output. The function is applied to every element in the tensor's feature maps and therefore not changing any dimensions. Without the activation function, the model would be linear and thus unable to solve non-linear problems.

The tensor is then sent through a pooling layer. The pooling layer's purpose is to reduce the spatial size of the convolved tensor. This is to reduce the computation time but also to extract the dominant features. Simplified, the pooling layer increases the "WHAT" information but reduces the "WHERE" information in the tensor. The pooling function used in this project is called max pooling, see figure 2.6. The max-pooling function combines clusters in the feature map, usually 2x2 clusters, and returns the maximum element of the cluster. This function effectively reduces the feature map spatial size by two. Therefore, after a pooling layer, the number of filters for future convolutions can be increased since the input is smaller and requires fewer convolution operations, and more features are extracted. The pooling slide always uses a stride of the width of the cluster, and normally uses "valid padding".

The repeated process of the convolution followed by pooling, causes the final output to be considerably smaller than the input and can be seen in figure 2.4. By continuously updating the parameters of the kernel to fit the training data, the network learns to extract the best features for the classification. The training is explained in section 3.2.3.

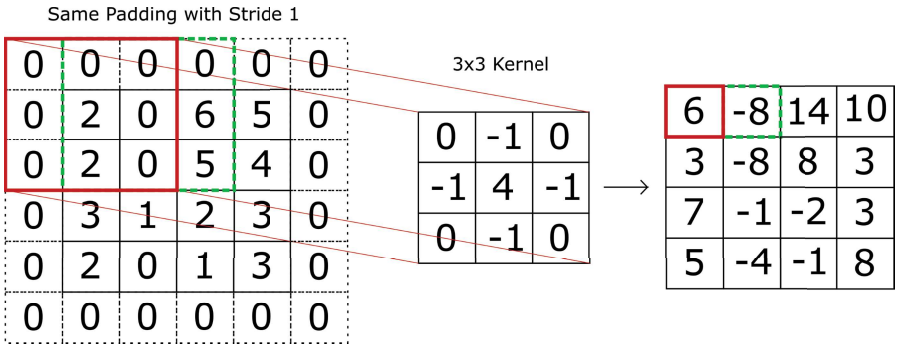


Figure 2.5: The principal of the convolution in the convolution layers. Here convolution is done with a 3x3 kernel with a stride of 1 and "same padding". The red squares represent the first step of the convolution operation, and the green the second.

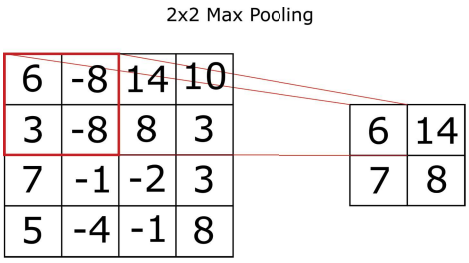


Figure 2.6: The principal of max pooling in the pooling layers. Here the network pool the feature map by a 2x2 cluster and output the maximum of the 4 features, resulting in halving the feature map size.

2.3.2 Classification

When the features from the feature learning step is extracted, they are presented to a classification step where they are run through a fully connected multi-layered perceptron (MLP). The MLP is a perceptron with extra layers called hidden layers between the usual input and output layers. In this MLP, the features works as inputs and are classified into the most probable class.

The MLP consists of an input layer, multiple hidden layers and finally an output layer in which its output is classified using an activation function, normally the Sigmoid function for binary classification. The MLP is a fully connected neural network, which means all the nodes are connected from the prior layer to the next, seen in the figure 2.7.

The input layer gets the final tensor from the feature learning step, flattens it out to a single array and each element in the array is considered a node in the input layer, seen in figure 2.4. The feature from each input node is multiplied with a weight and used as an input to each node in the first hidden layer, represented as ω_{ij}^{L} in figure 2.7. This is summarized with a bias and put through an activation function, and this function's output is the output of the the hidden layer's node. The output is then put through the same procedure and used as input to the second hidden layer's nodes, as seen in figure 2.7. The final hidden layer's outputs are inputs to the output layer's nodes. For a binary classification one node is only needed, while for multiple classes multiple nodes are needed. Usually in the output layer, the Sigmoid activation function is used to calculate the probability of the features being a certain class or not.

The output function for the output layer with one node is the same as for the perceptron in section 2.2 if only the last hidden layer's nodes is in consideration. For an MLP with M number of hidden layers, the output layer's function is given by:

$$y(\mathbf{h}^M, \omega^M) = \phi^o(\sum_i \omega_i^M h_i^M + b_M). \tag{2.6}$$

Which is the same as equation 2.1. The activation function is defined as ϕ^o since it is the output layer's activation function. In this case of binary classification ϕ_o is normally the Sigmoid function. Further instead of the input nodes $\mathbf{x}$ the input to the output layer is the last hidden layer's nodes $\mathbf{h}^M$. Each layer L in the MLP has their own weights ω^L and bias b_L, to make them independent of each other for the training.

The hidden layer's nodes' output is dependent on the previous layers' nodes. After each layer the nodes value is forwarded to the next layer, till eventually the output layer and the classification is done. To understand the relationship between the input values and the output value, the first hidden layer is needed to be explained. Given an input layer with values $\mathbf{x} = [x_1 ... x_n]$, each input node is connected to every node in the first hidden layer. Each node h_i^0 in this layer has the same function but with different weights:

$$h_i^0(\mathbf{x}, \omega_i^0) = \phi^h(\sum_i \omega_{ij}^0 x_i + b_0). \tag{2.7}$$

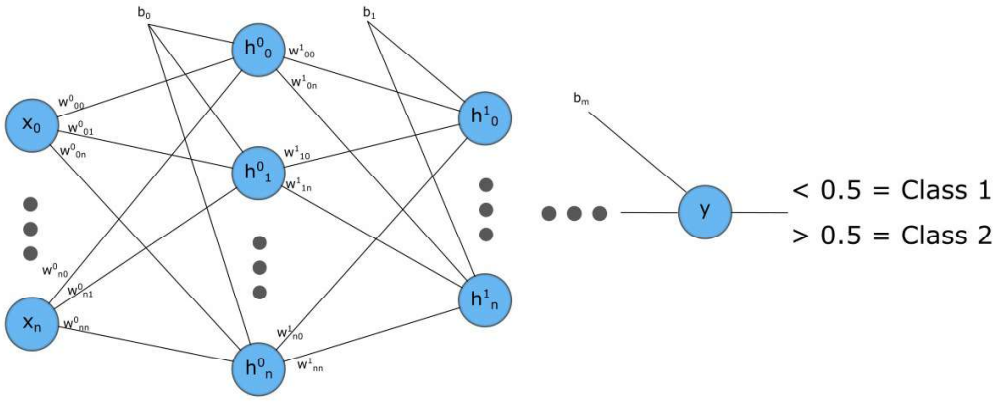


Figure 2.7: A generic multi-layered perceptron (MLP) for binary classification. x_i denotes the input nodes and h_i^L denotes the hidden nodes for layer L, and y as the output node. The MLP is fully connected, where node i in layer L is connected to node j in the layer $L+1$ with its weight w_{ij}^L and a layer bias b^L for all j, as in all nodes in the next layer.

Where the hidden layers' activation function ϕ^h is ReLU, this is to introduce non-linearity. Following the first hidden layer, the second hidden layer's relationship with the input layer is dependent on the first hidden layer:

$$h_i^1(\mathbf{h}^0, \omega_i^1) = \phi^h(\sum_j \omega_{ij}^1 h_i^0 + b_1) \rightarrow \tag{2.8}$$

$$h_i^1(\mathbf{x}, \omega_i^1) = \phi^h(\sum_i \omega_{ij}^1 * \ \phi^h(\sum_i \omega_{ij}^0 x_i + b_0) \ + b_1). \tag{2.9}$$

This continues for each hidden layer and the finally the relationship between the input layer and the output layer is:

$$y(\mathbf{x}, \omega) = \phi^o(\sum_i \omega_i h_i^{M-1}(\mathbf{h^{M-2}}(\mathbf{h^{M-3}}(...), \omega^{M-3}), \omega^{M-2}) \ + b_M). \tag{2.10}$$

Where "(...)" represent all the layers dependencies. This makes it possible to get the gradients of every node with regards to the weights, thus making it possible for gradient descent optimization.

2.3.3 Training

Backpropagation

The network's training is done by using an algorithm called backpropagation to minimize the error given by an error function. Backpropagation is short for backward propagation of

errors.

By running the network on a training set the network can update all its weights and biases similar to the perceptron. However with multiple layers, it becomes more complex and simply using the difference between the output and target does not cut it, as in equation 2.3. Hence, an error function E is introduced and a minimization problem can now be formulated as:

$$\min_{\omega}(E(\mathbf{d}, y(\mathbf{x}, \omega)) \tag{2.11}$$

Here ω is denoted as all the weights and biases, $\mathbf{d}$ as the targets, $\mathbf{x}$ as the input and $\mathbf{y}$ as output function. The parameters that minimizes the error function is considered the best parameters. There are many different error functions suited for different tasks, but for binary classifications the binary cross entropy error function works well. Given a training dataset $D = \{\mathbf{x}_i, \mathbf{d}_i\}_{i=1...N}$ with the targets being either a 1 or a 0, representing two classes, the cross entropy error function is formulated as:

$$E(\omega) = -\frac{1}{N}\sum_{i=1}^{N}\Big(d_i \; log(y(\mathbf{x}_i)) + (1 - d_i) \; log(1 - y(\mathbf{x}_i)) \Big). \tag{2.12}$$

The function empathizes either the first term if $d_i = 1$ or the second term if $d_i = 0$, therefore forcing the network to output as close to the same as the target. The training is done by putting an input through the network calculating the error and updating the parameters, much like the perceptron. However with multiple layers it becomes more complex, and the gradient descent used for the perceptron would take a lot longer to converge to a small error. The most common solution to this is using backpropagation with a smarter gradient descent learning algorithm.

The way network updates the parameters with gradient descent can be formulated by this function:

$$\Delta\omega_{ij}^{m} = -\eta * \frac{\delta E}{\delta\omega_{ij}^{m}(t)} \tag{2.13}$$

$$\omega_{ij}^{m}(t+1) = \omega_{ij}^{m}(t) + \Delta\omega_{ij}^{m} \tag{2.14}$$

Here $\omega_{ij}^{m}(t)$ represent the parameter's current solution at iteration t, in layer m, between node i and j, and η is the learning rate set by the user. The gradient is calculated from the last layer's parameters, moving backwards through all the layers.

The updating is normally done in multiple cycles of the training data called epochs. The training is considered finished after a maximum of epochs set by the user, or when the error no longer decreases.

Error Minimization

In this project, three optimizers were used, the Gradient Descent shown in (2.13), in addition to the Adaptive Gradient Algorithm (ADAGRAD)[6] and the Adaptive Moment Estimation (Adam)[7] which are both gradient descent enhancements.

ADAGRAD and Adam both introduce a dynamic learning rate to the weight updating. Since the learning rate is set by the user, it can be hard to find a good value. If it is too small the network will take to long to converge to find an acceptable error and if it is too big the network may take too big steps and move all over the function consequently never finding an acceptable error. One big problem with having only one learning rate is that due to the high dimensional complexity of CNNs, the learning rate is differently sensitive depending on each dimension. ADAGRAD solves this by adaptively scaling the learning rate by using the squared gradient received at each iteration in each dimension.

Adam is another popular optimizer and is shown to perform well compared to other popular methods. Adam combines the adaptive scaling of ADAGRAD with keeping a running average of the squared gradient which is added. This average is used to increase the learning rate if two successive points are in the same direction otherwise decrease it. The idea is to increase the speed of the optimization by using a momentum of the function.

2.4 UNet

The network architecture chosen for this research project is the UNet[3], and is illustrated in fig 2.8. The network consists of one contracting path, which is very similar to an ordinary CNN architecture, and one expansive path, that makes several up-samplings to a similar size of the input. These two paths create the characteristic U-shape.

The network is well fitted for image segmentation, which not only says "what" is in the image (like image classification) but also "where". The network classifies all pixels in the image to a certain class, and returns a segmentation mask, in this case, the CCA.

UNet is a fully-connected convolutional network (FCN), which means that the network does not use an MLP at the end for classification but uses solely local spatial input. This means that the features are always spatially dependent on each other, so for UNet, the classification part is done by a convolutional layer.

The desired output dimensions are the same as the input dimensions since the goal is to classify where in the image the classes are. This means that the need for "WHERE" information is more important than the need for "WHAT" information. Thankfully, an FCN is better at using this information than a normal CNN, since an MLP does only use the "WHAT information.

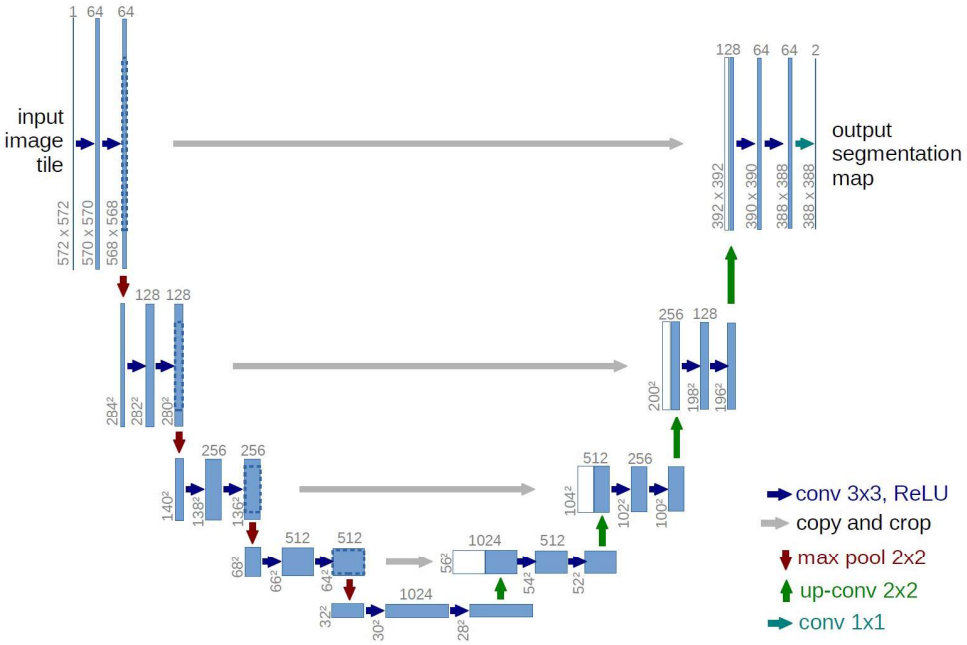


Figure 2.8: Structure of UNet, picture from [3]. The dark blue arrows represent convolution operations with 3x3 kernels and ReLU and the light blue represent a pixel by pixel convolution step to classify the pixels. Further, the grey represents a copy of the tensor from the contracting path to the expansive path, the red represents max pooling with a size of 2x2 and finally, the green arrows represent up-sampling using transposed convolution using a 2x2 kernel.

2.4.1 The Architecture

Contracting Path

The contracting path, the left side in figure 2.8, consists of continuously having two consecutive convolution layers (with ReLU) followed by a max-pooling. The convolution layers in the example use 3x3 kernels and valid padding, which leads to the loss in height and width in the tensor, while the max-pooling uses a 2x2 cluster with stride 2. The example shown in the figure takes in 572x572 images with one feature channel (greyscale) and after ten convolution layers and four pooling layers, the tensor is 28x28 with 1024 feature channels. After each of the double convolution layers, the feature map is saved for use in the expansive path. The now considerably smaller feature map proceeds to the expansive path to be up-sampled to a similar size of the original image.

Expansive Path

The expansive path, right side in figure 2.8, consists of repeated steps of up-sampling layers instead of pooling layers. The function used for up-sampling is called transposed convolution, however it is not at all a convolution. The reason it is called "transposed convolution" is that it works similar to convolution but backwards. The convolution operation works in a many-to-one relationship, where a 3x3 matrix becomes one value, a 9-to-1 relationship. While transposed convolution works in a one-to-many relationship, where one value can give a 3x3 matrix, a 1-to-9 relationship.

The way it works is that given an input feature map, the map is up-sampled through a matrix to give the one-to-many relationship, and this matrix uses learnable parameters similar to the convolution kernels. The reason it is called a transposed convolution is because this learnable matrix works like a transposed convolution matrix. Given a 3x3 kernel W, a 4x16 convolution matrix W_{conv} can be created:

$$W^{[3x3]} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \longrightarrow W_{conv}^{[4x16]} = \begin{bmatrix} a & b & c & 0 & d & e & f & 0 & h & h & j & 0 & 0 & 0 & 0 & 0 \\ 0 & a & b & c & 0 & d & e & f & 0 & g & h & i & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & a & b & c & 0 & d & e & f & 0 & g & h & i & 0 \\ 0 & 0 & 0 & 0 & 0 & a & b & c & 0 & d & e & f & 0 & g & h & i \end{bmatrix} \tag{2.15}$$

This 4x16 convolution matrix can be used to convolve a 4x4 matrix. If the 4x4 input matrix X is flatten to the 16x1 matrix X_{flat} and a matrix multiplication is made with the convolution matrix W_{conv}, the output is the flatted 4x1 matrix Y_{flat} which is the flatted output matrix of

Y. Shown here:

$$X^{[4x4]} = \begin{bmatrix} y_{11} & y_{12} & y_{13} & y_{14} \\ y_{21} & y_{22} & y_{23} & y_{24} \\ y_{31} & y_{32} & y_{33} & y_{34} \\ y_{41} & y_{42} & y_{43} & y_{44} \end{bmatrix} \longrightarrow X_{flat}^{[16x1]} = \begin{bmatrix} x_{11} \\ x_{12} \\ x_{13} \\ x_{14} \\ x_{21} \\ x_{22} \\ x_{23} \\ x_{24} \\ x_{31} \\ x_{32} \\ x_{33} \\ x_{34} \\ x_{41} \\ x_{42} \\ x_{43} \\ x_{44} \end{bmatrix} \tag{2.16}$$

$$W_{conv}^{[4x16]} * X_{flat}^{[16x1]} = Y_{flat}^{[4x1]}, \; where \; Y_{flat} = \begin{bmatrix} y_{11} \\ y_{12} \\ y_{21} \\ x_{22} \end{bmatrix} \rightarrow \; Y^{[2x2]} = \begin{bmatrix} y_{11} & y_{12} \\ y_{21} & y_{22} \end{bmatrix} \tag{2.17}$$

However, the reversed way is also possible. Imagine an nxn input matrix X and a kernel W of size 3x3, an output matrix Y of twice the size, 2nx2n, can be produced using the transpose of the convolution matrix achieved from the kernel W. The convolution matrix W_{conv}'s transposed matrix multiplied with the input matrix X_{flat} will output the flatten up-sample Y. Shown here:

$$X^{[nxn]} \; \rightarrow \; X_{flat}^{[2nx1]}, \; transpose(W_{conv}^{[2nxn^2]})^{[n^2x2n]} * X_{flat}^{[n^2x1]} = Y_{flat}^{[(2n)^2x1]} \; \rightarrow \; Y^{[2nx2n]} \tag{2.18}$$

This means that we can use kernels with trainable weights to up-sample our tensor to twice the size. Thus making it possible to further optimize the feature learning step in the network.

In the expansive path, the architecture is as follows. After the contracting path, the width and the height of the tensor are very small while the number of feature channels is large. The first step is an up-sampling, in form of a transposed convolutions, which will double the width and height of the maps. The number of filters used is half the number of feature channels since the spatial size is increased a lower amount of feature channels is desired to lower the computation time.

Here the previous tensor from the expansive path are concatenated to the current feature map. They then go through two convolution layers with 64 filters to find and incorporate the important features while not increasing the feature channels for computation time purposes. This is repeated three more times, and the output will be the same size if "same padding" is used, but in the figure, "valid padding" is used causing it to have a smaller size.

Finally, the tensor go through a final convolution layer with only one filter, here the kernel size is 1x1 and the activation function is the Sigmoid function. This final layer works as a pixel-wise classification, where each pixel's features is taken in consideration from the tensor. And in total, 23 learnable layers were used (19 convolution layers and 4 transposed convolution layers) in figure 2.8's network [3].

2.5 Related work

Neural networks have a long history, with the perceptron being introduced already in 1958 [5]. However, it was not until the 1990s that modern CNN was introduced [8]. Yann LeCun found that using a learnable CNN it could solve handwritten digits classification. With sufficient training data and with learning rate decay they manage to avoid over-fitting the network resulting in a below than 1 percent error. Better performance than the classical machine learning algorithms and the MLP used before. Concluding that the need for handcrafted features is eliminated with the use of CNN. Further, they showed that with more training data, faster computers and more fine-tuned learning algorithms, the recognition systems will rely more on learnable networks and perform even better.

As predicted, these three things occurred and CNN had a big upswing in popularity in 2012. in particular when a state-of-the-art image classification CNN called AlexNet was introduce [9]. The network managed to label correctly 83% of 1.2 million of images into 1000 different classes, considerably better than the previous networks.

It did not take long until CNN could do image segmentation, with the introduction of FCN in 2014 [10]. With an FCN trained to do dense predictions, like image segmentation, they have the ability to solve previous tasks than regular CNN could not. Using transposed convolution as up-sampling, the network learned how to better classify the pixels into their proper class. A year later, UNet was introduced and proved to be particularly efficient for tasks with few training data and further developed the use of transposed convolutions.

In the biomedical field, DL has had a recent upswing in the past years. The growth for DL in the field began 2015 and the number of published papers has since increased by a factor of 9 in bio and medical imaging alone with CNN as the most commonly used deep neural network [11]. For US imaging of the CCA, a study showed that the use of an MLP to segment out the CCA in longitudinal US images with the purpose to evaluate the intima-media thickness, which roughly is the thickness of the artery wall [12]. The solution purposed was to use an autoencoder, which is an unsupervised network that does not use training data, to lower the dimensionality and then train an MLP with a single hidden layer to classify the pixels and managed to classify with a mean error around 5%.

Chapter 3
Research Questions and Methodology

3.1 Thesis Goals and Research Questions

The objective of this thesis is to evaluate DL and CNN as a tool to automatically segment the CCA from US images, specifically using UNet which seems to be one of the better choices for the task at hand. Furthermore, the objective is to test how the network can handle the challenges that US images bring, such as artifacts and the limited and diverse training data.

The ambition is to get a close to perfect segmentation of the CCA with the purpose to replace the need for experts to manually segment it in the future. The main goal of the thesis is to show the use of CNN in US image segmentation and that DL is a promising tool in US image analysis.

With these goals in mind, the following question is chosen for the research of this thesis:

1. Can US images of the CCA be segmented reliably with a trained CNN?
2. What are the main limitations and challenges of using a CNN for this task?

3.2 Method

3.2.1 Dataset

The dataset that is used in this paper consists of in total of 249 US images and their segmented mask was done by an expert in the biomedical field. The expert chooses to segment the CCA using circles since the images were taken on healthy patients that should not have any plaque in their CCA the circles are an accurate assumption.

Test dataset	Test data/ Training data	Percentage
TP1	27 / 221	11% / 89%
TP2	60 / 188	24% / 76%
TP3	81 / 161	33% / 67%
TP4	81 / 161	33% / 67%

Table 3.1: Training datasets vs test datasets

The US images are collected using a Philips Epiq 7 (Philips Medical Systems, Bothell, WA, USA) equipped with a linear array transducer with a 7 MHz center frequency. The transducer is attached to a 3D stepping motor controlled by another US machine Visual Sonics Vevo 3100 (VisualSonics Inc., Toronto, Canada) to guide the transducer along the neck. The image collection is ECG triggered, meaning they are taken on the same part of the heartbeat and are taken on the test person's left CCA while laying down. The step size of the 3D motor is 250 or 300 micrometers (μm) between each image. All images have a size of 400x400 pixels and are exported using the medical imaging file format DICOM.

The images are collected from four test persons, called TP1 to TP4 for simplicity and anonymity. The dataset consists of 27 images of TP1, 60 images of TP2, and 81 images of both TP3 and TP4, leaving a dataset of 249 images in total. In figure 3.1, one image from each of the four test persons is shown with its corresponding segmentation mask. The images differ quite a lot, which depends on mainly two things, different bodies, and different settings.

While all images suffer from artifacts and speckle, TP1 is probably the one that suffers most and the circular shape is particularly less defined than in the others, especially on the sides. TP3 has a very high contrast while TP4 has very low contrast, and both suffer from speckle in the top part of the circle. TP2 suffers the least from speckle and artifacts, the images are also quite coherent making this dataset easier for segmentation in theory.

The veins have different shapes and locations in the sets because of the difference of the test persons, some even take more of a circular form seen for TP1 and TP4 in figure 3.1 which might be misclassified as an artery. Each pixel represents an area of around 0.0061 mm^2 for TP1 and TP3, and 0.0046 mm^2 for TP2 and TP4, meaning the circles will have a different pixel radius between these sets.

The complete dataset is then separated into a training dataset and a test dataset, where three test persons' images are used for training the network and the remainder's images are used for testing the network. The number of images in the train and test data is shown in table 3.1.

3.2.2 Network Architecture

The network used for this thesis uses the same amount of steps as the network introduced in section 2.4, and can be seen in 3.2. The input images are of size 400x400x1, where the third dimension represents the intensity of the pixels, and is represented in dark blue in the figure. These images go first through the contracting path, seen as light blue rectangles in the

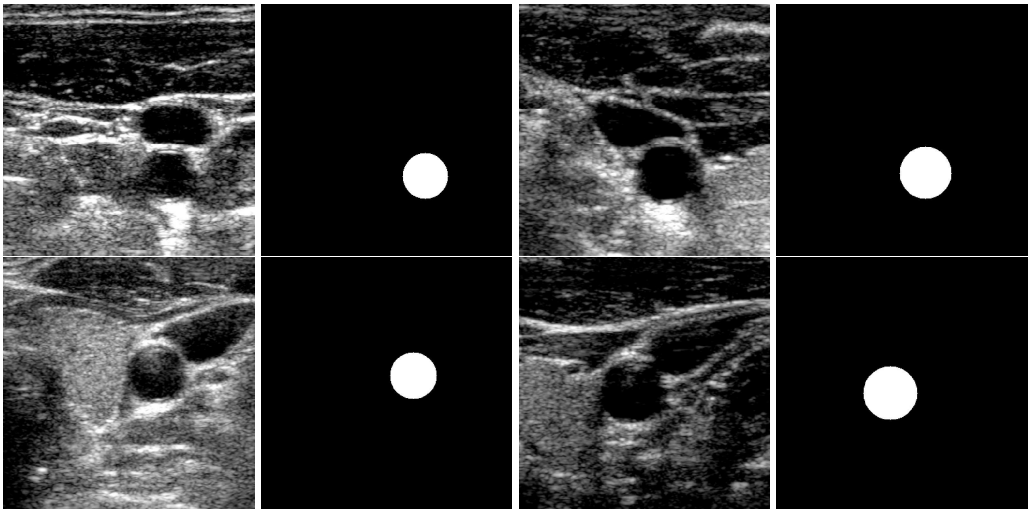


Figure 3.1: The dataset of the US images and their corresponding segment mask. (TP1, top left; TP2, top right; TP3, bottom left; TP4, bottom right)

figure. The input images are sent through two convolution layers, with stride and padding of 1 ("same padding") and 64 filters with ReLU as the activation function, seen as blue arrows in the figure. This results in a feature map of 400x400x64, which is then down-sampled through a max-pooling layer with cluster size 2x2 effectively halving the width and height, red arrows in the figure. These steps are repeated three times with the number of feature channels increasing by two after every step. Finally resulting in a 25x25x512 feature map, which goes through two final convolution layers and giving the final number of feature channels to 1024 before the expansive path begins.

These tensor are then passed through the expansive path, shown as dark green rectangles in the figure. The 25x25x1024 feature map is up-sampled with transposed convolution, shown as green arrows, to 50x50x512 feature map which is concatenated with the previous feature map of the same size from the contracting path resulting in a 50x50x1024 feature map. This larger feature map is sent through two convolution layers both with "same padding" and ReLU as previously. Instead of doubling the number of feature channels, they are halved instead to integrate the two concatenated maps giving more "WHERE" information to the feature map while keeping the complexity low. These steps are once again repeated three times resulting in a 400x400x64 feature map.

These final tensor are sent through a convolution layer with a 1x1 kernel to be pixel-wise classified using the Sigmoid as activation function, represented as light green arrows in the figure. Resulting in a 400x400x1 image, the same size as the input image, seen as the light green rectangle in the figure. The segmentation of the artery is represented with 1 in pixel value while the other pixels are 0 in the output image, thus giving the area of the artery from the input images.

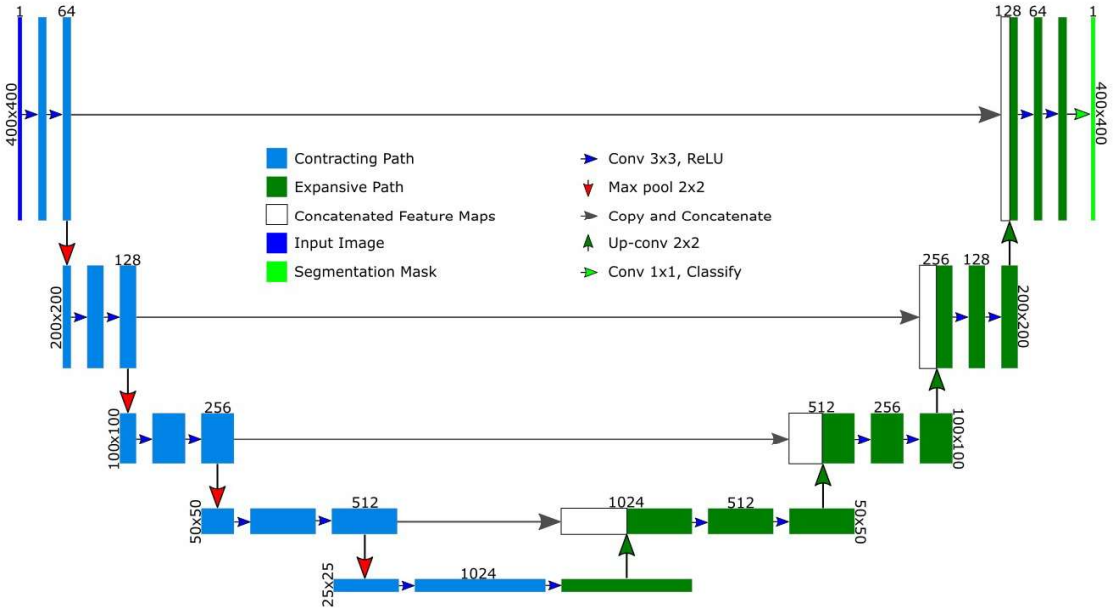


Figure 3.2: The UNet architecture used in this thesis.

3.2.3 Training

The training data and test data are split into four, one for each test person as test data, seen in table 3.1. These are then split into two main sets, first sets using the raw images as they are taken from the US imaging, and the second set using SRAD pre-processing on images before training. Leaving a total of 8 experiment datasets.

The network is implemented using the PyTorch [13] library in Python and is trained using a GTX970 Nvidia graphic card.

Since the output segmentation only has two pixel-values (1 and 0 representing white or black), only two classes are needed for classification and the binary cross-entropy is used as the error function. Three different optimizers are used, gradient descent, ADAGRAD, and ADAM. The training is done by running the training data through the network and using backpropagation with error compared to the target segmentation masks.

Since the datasets are small, only a few epochs are used to avoid overfitting the network making it less suitable to data that differs from the training data. The maximum number of epochs is set to 15, but each epoch is tested and the best is used.

3.2.4 Experiments

The experiments are run on all eight datasets and are split into two steps. The first step is to train the network and segment the test data using the trained network then evaluate how the networked performed. The second part, after the segmentation, finds the best fitting circle using Hough transforms[14] in the best segmented datasets and evaluate if they improve.

The network is trained multiple times, with different parameters to find the best suitable for each dataset. The main parameters are optimizer, learning rate and the number of epochs. The best parameters are going to be presented for each dataset used. The results will be evaluated using a DICE-coefficient [15], which gives a good score on how well the network segmentation performed. The DICE coefficient is calculated by:

$$DICE = \frac{2 * (Y \cap Y')}{Y \cup Y'} \tag{3.1}$$

where Y and Y' in this case are the set of pixels classified as the CCA for the network's mask respectively the expert's mask. A perfect DICE coefficient is 1, which means that the segmentations overlap perfectly.

The main evaluation is done using the resulting segmentation masks. The most basic requirement is that the CCA is found and that the mask is presented in the same area the CCA is located in the input data. Furthermore is that the network does not heavily misclassify other areas in the image as the CCA. If the mask has the same location as the CCA, the next evaluation is the shape of the mask. E.g. if the mask is incoherent or fractured. The DICE coefficient is mainly used for comparing the different experiments but does give a good score of how well the network manage to segment out the CCAs.

Chapter 4
Analysis

In this chapter, a more in-depth explanation of why the experiments are run and what motivates the evaluation. Section 3.2, the datasets, and the experiments are presented and not fully explained, this chapter's purpose is to give the reader a deeper understanding for what to expect and look for in the presented results of chapter 5.

4.1 Experiments

The experiments purpose is to see if the network in the first place can find the CCA in a completely new set of images from the training data. Seeing the difference in the results for the test persons datasets, will give an indication of what might be challenging in the images like artifacts, the size of the training data and variance between the images. The three parameters, optimizer, learning rate and number of epochs, is presented for each experiment mainly to help the reader to be able to replicate the experiments.

One of the biggest challenges with using US images for CNN is that the images are different from patient to patient in greyscale and the amount of speckle. With the help of SRAD, the images will have a more common greyscale and less amount of speckle. In figure 4.1, a comparison is shown for the test persons images before and after preprocessing with SRAD. After the preprocessing, the images get a more normalized greyscale and less speckle appears, but at the cost of sharpness. By comparing how the network performs using the images raw and with the preprocessing, the networks strength and weakness can be analysed.

However, the artifacts like enhancements and refraction will remain even after SRAD. The network does take the whole image in consideration when it learns, this means that it can potentially use the artifacts to its advantage. For it to learn this, it will need enough training data to understand the artifacts which the current datasets might not provide. This will be analysed after the results.

Another problem with US images and the artifacts that occurs and especially with CCA imaging, is that the circles tend to lose their shape. Since the network is not forced to produce circles, its output segmentation will be taking different shapes. By finding the best circle that fits this output, the shape will be retained and this might prove to give better results. Hough transforms for circle detection works well for this solution, since the pixel to mm^2 ratio is known and the average diameter of the CCA is around 6 mm, a good estimation can be provided for the algorithm. This postprocessing is mainly used to show that the network can be used combined with other methods, and is a lesser important part of this project.

4.2 Evaluation method

In section 3.2.4, the evaluation of the experiments is presented. The segmentation masks are the main focus for evaluation while the DICE coefficient does give a very good number for comparison. The DICE coefficient is commonly used for a similarity score for segmentation tasks since it punishes misclassifications in both cases of either a pixel being a true negative or a false positive. The DICE coefficient is used for a comparison of the experiments and will give information for the first research question in section 3.1, if a trained CNN can be used for segmenting CCA in US images.

Determining what a good DICE coefficient is can be hard. A DICE coefficient of 1 would mean a perfect circle is segmented, however since the US images very rarely can give a perfect circle because of the physical limitations in sound propagation some margin of error in the DICE score is acceptable. In addition to the limitation of the US, the expert's manual segmentation is prone to contain a small margin of error as well.

The main desire with the network is to be able to find the CCA's location in the image, which the DICE coefficient does not show. Therefore, the images need also to be analyzed to show how well the network performs in this regard. Further, it is important to consider that DICE does not consider shape either. The shape of the masks can tell much of the limitations of using a CNN for segmentation and thus answering the second research question. To understand the effects of the artifacts or other characteristics that the different images have, the shapes of the masks need to be analyzed.

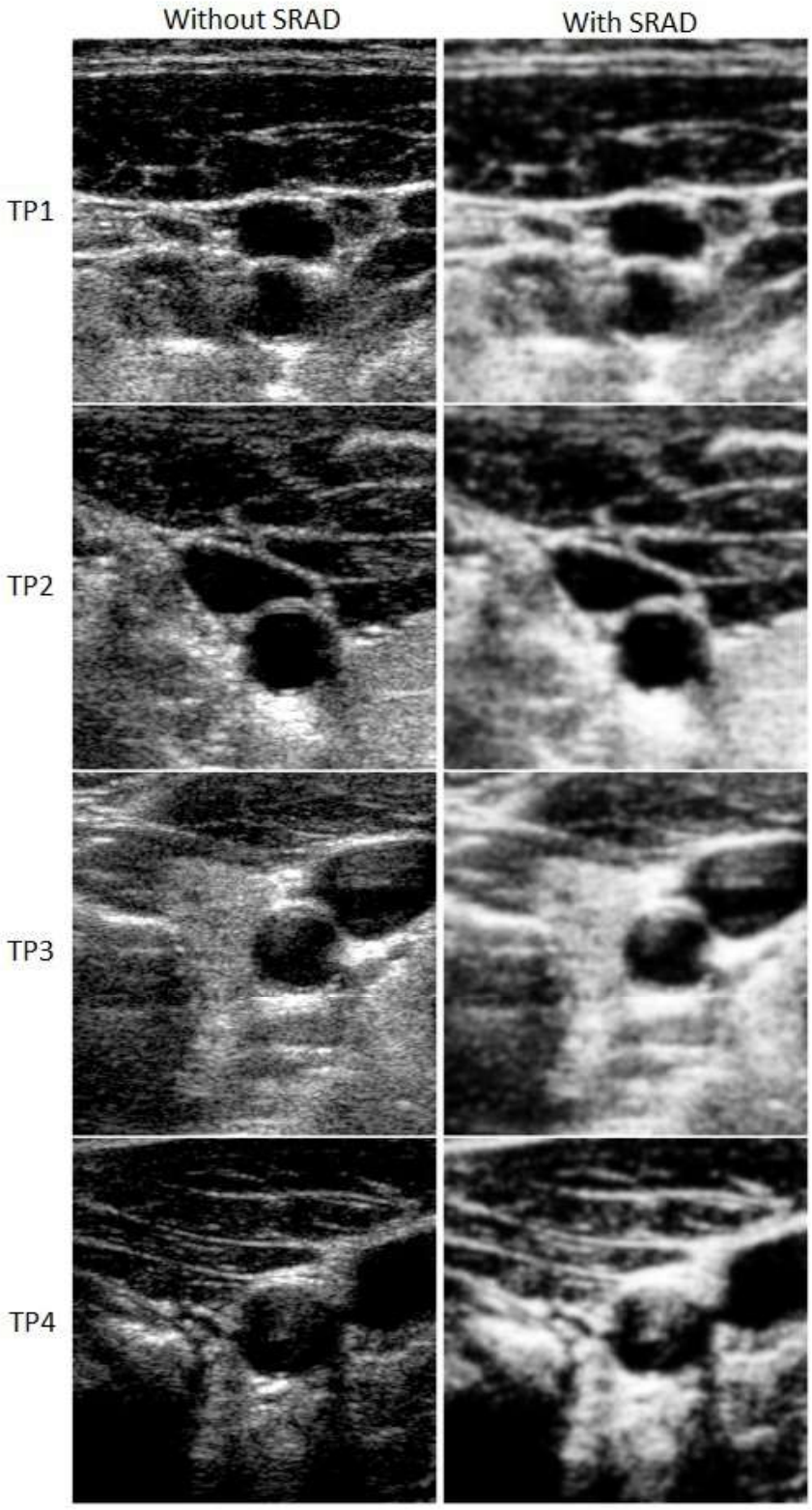


Figure 4.1: Comparison before (left) and after (right) preprocessing.

Chapter 5
Results

In this chapter, the results of the experiments are presented. The average DICE coefficient for each of the datasets and what parameters were used for the best results are presented in a table. All resulting segmentation masks are shown in Appendix A, and for each dataset, both a good and a bad example of the resulting segmentation mask are presented. The output masks are presented on top of their original image with the color red, the expert's mask as the color green and their overlapping area as blue. The more "blue" that is present the better the result is, while either green or red will give a worse result.

5.1 Raw datasets

In this section, the dataset where the raw images were used is presented. In table 5.1, the average DICE coefficient is shown for each of the datasets and an overall average for all the datasets, and the parameters used for training the best network.

A good example of outputs is presented on the right side of figure 5.1, and on the left side, a bad example is shown. For all except TP4, the network manages to find the location of the CCA in all the images. However, TP3 suffers from misclassification in the veins. Only TP1 and TP2 gets an acceptable score and finds acceptable segmentation masks.

Dataset:	TP1	TP2	TP3	TP4	Overall average
DICE coef:	0.845	0.950	0.749	0.346	0.722
Optimizer:	SGD	SGD	SGD	ADAGRAD	
Epochs:	10	10	5	3	
Learning rate:	0.05	0.05	0.01	0.01	

Table 5.1: Raw datasets' resulting DICE coefficient and best parameters

Dataset:	TP1	TP2	TP3	TP4	Overall average
DICE coef:	0.816	0.942	0.871	0.856	0.871
Optimizer:	ADAM	SGD	SGD	SGD	
Epochs:	5	5	7	7	
Learning rate:	0.005	0.05	0.01	0.05	

Table 5.2: Preprocessed datasets' resulting DICE coefficient and best parameters

TP1's segmentation gets a good DICE coefficient, they, however, suffer from being non-coherent which can be seen in the bad example. Especially on the sides of the circle, indicating that influence of the artifacts and speckle.

The best dataset's segmentation is TP2 with a DICE coefficient of 0.950. The network manages to nearly find the whole CCA and its shape. The good example in figure 5.1 has nearly no green or red is present. Showing that the network can segment out most of the CCA in some cases.

The network manages to find the CCA in the TP3 images, however, it does tend to falsely identify a vein as part of the CCA, as seen in the bad example in figure 5.1. Further, the artifact that is present in the top part of the circle is affecting the network's ability to correctly classify the CCA in that area.

The only one that the network could not segment the CCA was with the TP4 images, as seen in figure 5.1. In the first few segmentation, seen in Appendix A.1.4, the network tends to classify other parts of the image that is dark and not the CCA. However, when the images become brighter the network manages to find the CCA and segment it out, as seen in the good example in figure 5.1.

5.2 Preprocessed datasets with SRAD

For the SRAD preprocessed images, similar results are presented in table 5.2. A good example of outputted segmentation masks is shown on the left side of figure 5.2 and a bad example is shown on the right side. In appendix A.2, all the resulting segmentation masks are shown.

The network manage to find the CCA in all of the images and with an overall average DICE coefficient of 0.871, which means that the network is able to find a significant area of the CCA in all the images. Especially for datasets TP3 and TP4 the network manages to get a much better score and does not miss identify vein or other dark areas. However, the use of SRAD did give a slightly worse result for TP1 and TP2.

Similar to the experiments with the raw images as input, the preprocessed dataset for TP1 suffers from artifacts which result in that the segmentations tend to take weird shapes, seen especially in the bad segmentation in figure 5.2. The preprocessed image become a little

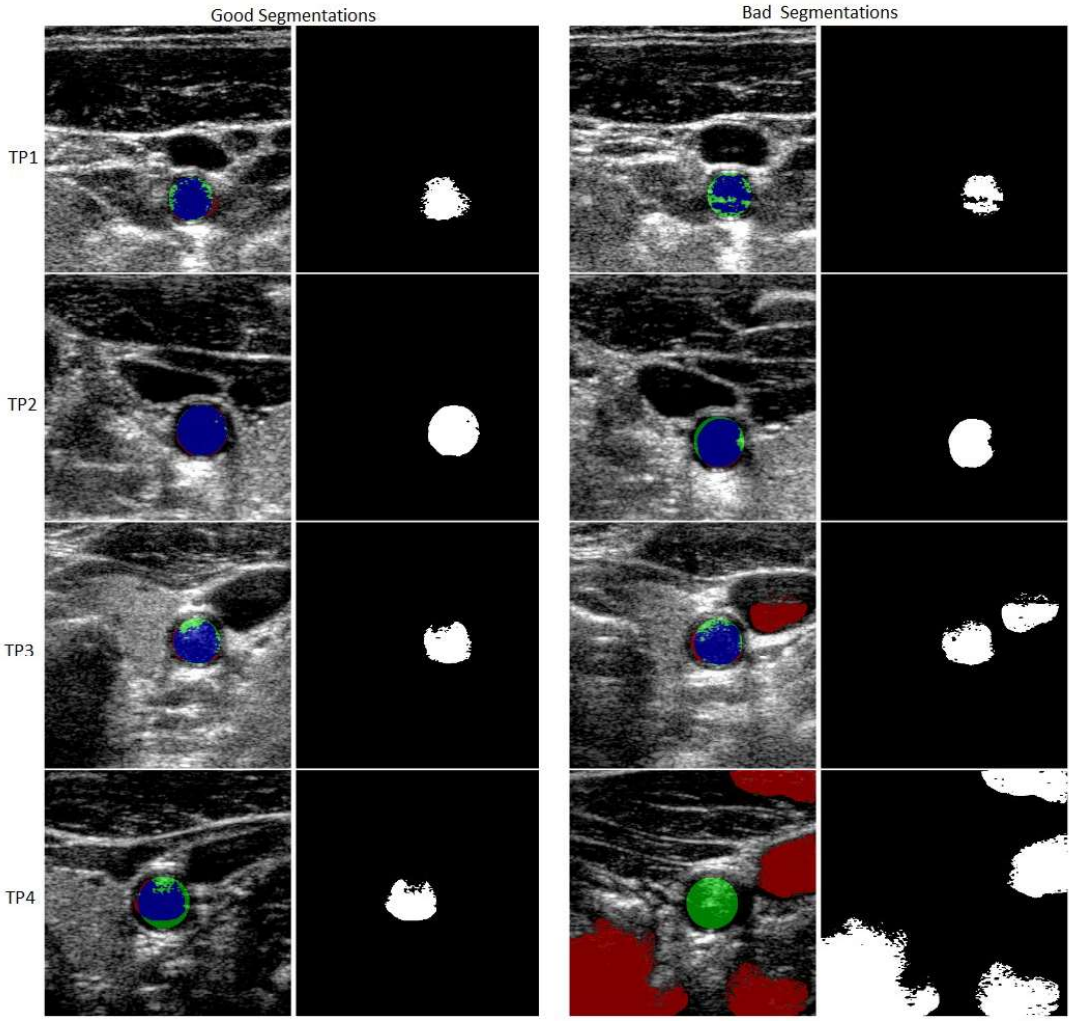


Figure 5.1: Examples of good and bad resulting segmentation masks for the raw datasets (in order, TP1, TP2, TP3 and last TP4).

Dataset:	TP1	TP2	TP3	TP4	Overall average
DICE coef:	0.863	0.876	0.886	0.886	0.873

Table 5.3: Circle detection's resulting DICE coefficient and best parameters

more coherent, and does suffer more from the sides being affected by artifacts in the images. Overall the DICE coefficient is 0.029 worse than in the previous experiments, which points that the preprocessing did not help to improve the result for this dataset but not necessarily worsen it either.

TP2 does get a worse score, but with a very small difference of 0.008. Overall the segmentations look similar to that of the previous dataset, except that they tend to lose some area in the top of the circle. However, a score of 0.942 is a satisfying score.

The preprocessing does help the TP3 and TP4, which suffered a lot from misclassifications. The help of smoothing out the images and normalizing the greyscale cause the overall training to be more effective and the test data to be more similar to the training data. This can definitely be seen for TP4's segmentations, in the previous experiments the network could not find the CCA in a quarter of the images while now the network find it in all of the images, as seen in Appendix A.2. However, some cases still suffer from the artifacts and speckle, as seen in the bad segmentation in figure 5.2, where only a small part of the CCA is segmented.

5.3 Postprocessed with circle detection

The circle detection is applied to the preprocessed dataset outputs since it scored a better DICE coefficient and does find the CCA in all images. The purpose of the postprocessing with circle detection was to force the output of the network to fit the shape that is desired and to help the segments that were incomplete and incoherent, which datasets TP1, TP3 and TP4 did suffer from. Shown in the table 5.3, these datasets do get a better DICE coefficient than they did without the circle detection. The TP2, which did not suffer as much from this, score a much lower score than before.

The bad segmentation masks in figure 5.2 that gave a bad score was improved with the circle detection in figure 5.3 for TP1, TP3 and TP4. So the circle detection seems to fulfill its purpose well. However, the circles detected can sometimes be too large or be miss placed as seen in the bad segmentation on the right side of figure 5.3. It does show that if the network struggles with finding most of the CCA, circle detection can prove to be useful but if the network can find most of the CCA it can worsen the result as seen for TP2.

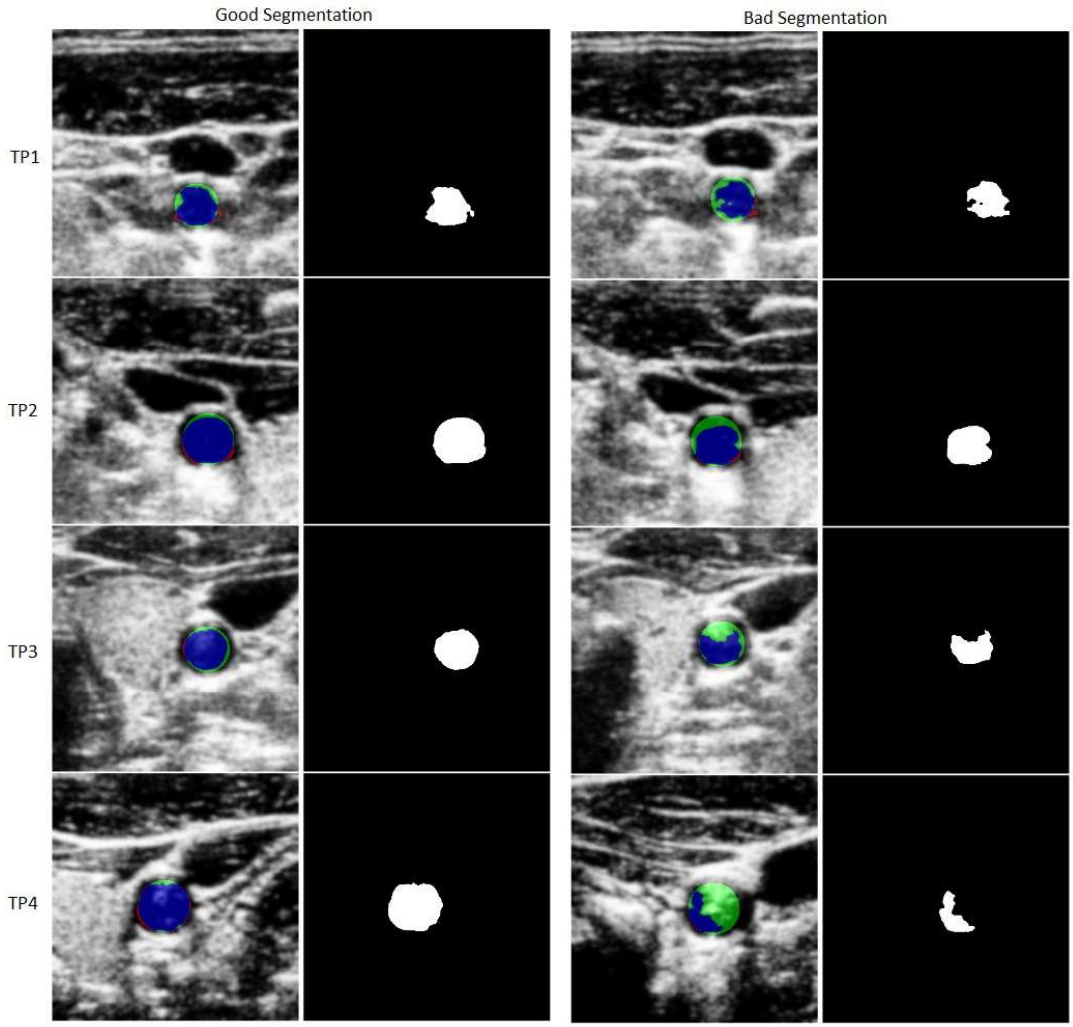


Figure 5.2: Examples of good and bad resulting segmentation masks for the preprocessed datasets (in order, TP1, TP2, TP3 and last TP4).

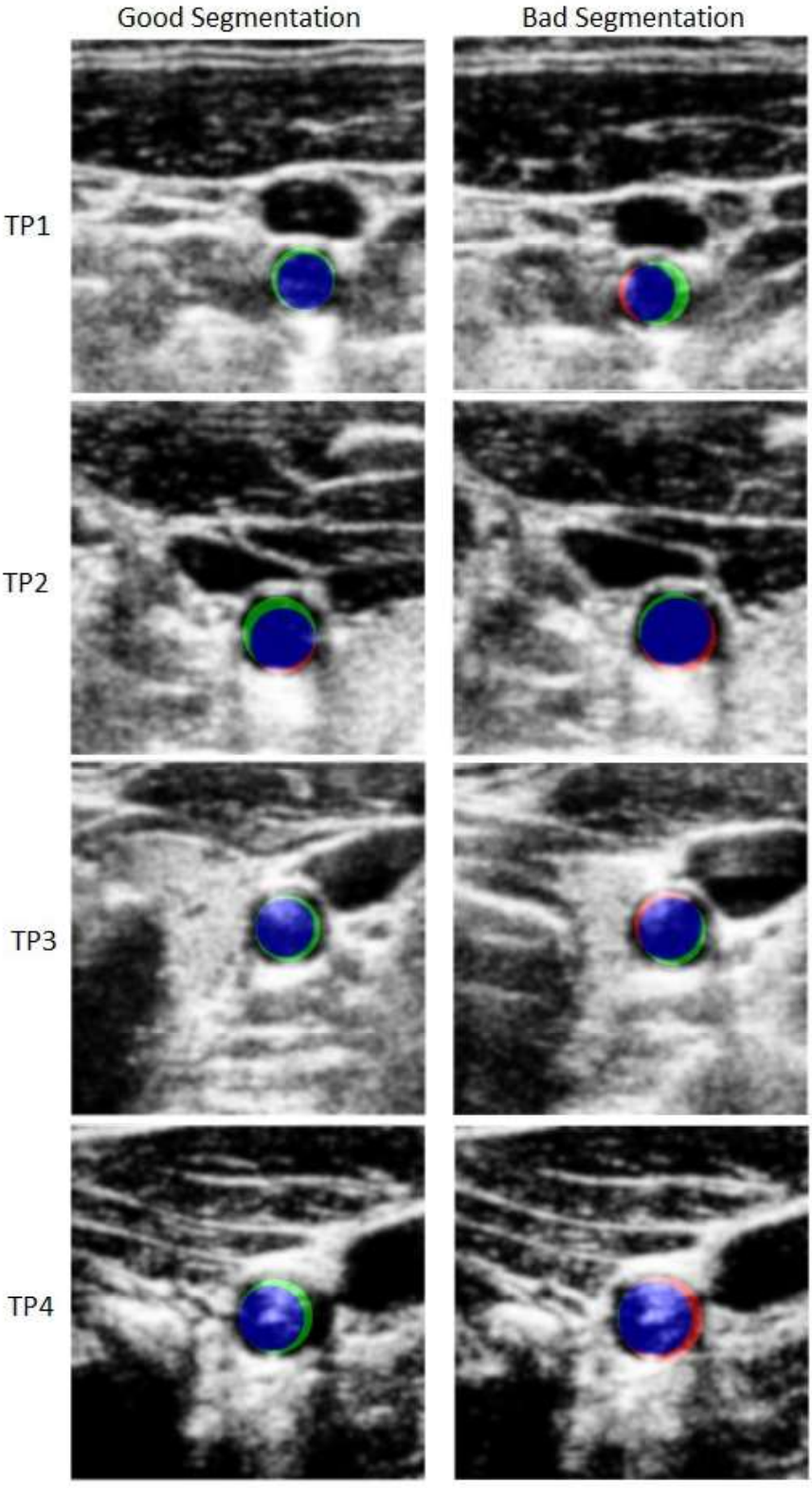


Figure 5.3: A good and bad example Circle detection from the segmentation masks received from the preprocessed test dataset. From top left to bottom right, TP1, TP2, TP3 and finally TP4.

Chapter 6

Discussion

In this chapter, a more in-depth explanation is given for the results considering the thesis goals and research questions. Further, the evaluation of the network's usage for real-world applications and what further research and work can be done to further improve the usability of the network is discussed.

6.1 Thesis Goals and Research Questions

The general goal of the thesis was to evaluate the usage of DL and CNN in US images, specifically for segmenting the CCA, and to explore how the network manages with the challenges that US images contain. In section 3.1, two research questions was defined:

1. Can the CCA in US image be segmented with a trained CNN?
2. What are the limitations and challenges of using a CNN for this task.

These questions were outlined with the goals in mind, and the main goal was to find the usage of CNN and show how well it can handle the arguable hard task to segment US images. The way to show that is to understand the result presented in chapter 5, and what the different experiments tested. Using the raw images shows how the network handles large variances in images and which characteristics are suitable for using the network for segmentation. The preprocessed dataset experiments show how the network can perform generally with the help of standard preprocessing and gives a better answer of how the network's usage in real-world scenarios.

With a DICE coefficient of 0.950 and 0.942 for the TP2 dataset, the network proves that it is definitely possible to segment the CCA in US images with reliability. Looking at the overall average, a DICE coefficient of 0.871 on the preprocessed images shows that it can segment the CCA somewhat consistently for a range of different images, but it is not reliable

and sometimes the segmentation masks are not very desirable. Especially seen in Appendix A.2.4, the masks for TP4 has a big variance from being a good segmentation to a bad segmentation.

Considering that without preprocessing the network struggles with a big variance in images, a big challenge is to manage to train the network to handle the variance that US images contain. Preprocessing does help the network with some of these variances, as seen for TP4 where the network could not find the CCA for images without the preprocessing. However, artifacts proved to be hard to handle for the network, which might be the reason why TP2 scores a really good score since it suffers less from the artifacts. The network seems to be able to handle the artifacts well and at some points poorly, and this can be seen in the masks in Appendix A.2. Especially for TP3 and TP4 which suffer from enhancements in the top of the circle, sometimes the outputted mask is a very circular shape and covers most of the CCA and sometimes it loses most of its shape and only the dark area is segmented.

While preprocessing proved to be essential, postprocessing with circle detection also proved to be useful in some cases but overall not that impressive. Only in the cases when the output masks are bad it helped using circle detection, but for the already good segmentation, it could damage the output and making it worse.

The network does show good potential and the ambitious goal of getting near-perfect segmentation was achieved for one of the datasets, but overall it does show some unreliability. Considering the low training data and its variance it was a hard challenge and with a larger training dataset, the network most probably will give better results. The main goal of the thesis can be considered fulfilled and the first research question is confirmed.

The second research question is more open to interpretation. The results are that the network can manage to surpass the inherent challenges that US image segmentation has but does struggle when the network is trained on images with less variance or when input data is very different from the training data. A training dataset of around 200 images is considered a very small dataset in DL which tends to be around thousands of images. Given this small dataset and the result that the network gives, the network shows that it does use the training data well and does tackle that challenge in an acceptable way, but that there still is room for improvement.

6.2 Future Work

6.2.1 UNet

In this project, the UNet architecture was used for the network and it shows great potential. With further training data, the network will perform even better, and it is recommended that more training data is introduced when using the network. The network was chosen for it is the most prominent architecture for medical imaging, however, in US imaging the network has not been tested fully comparing to other medical imaging algorithms. With further development in the field, more trained networks for US images will be available.

In this project, the network was trained from scratch using only the small dataset available. A common way to tackle this problem in DL is using an existing network that has already been trained with a big dataset, then fine-tune it to fit a specific task that has a small dataset. In October, a published a paper in how a UNet trained with 1000 natural images can be fine-tuned to fit US segmentation [16]. They found that fine-tuning the contracting path proved to give better results compared to fine-tuning the expansive path, further that starting fine-tuning from the first layers in the contracting path and moving up to the later ones in the expansive path proved to be a better workflow than the opposite. Fine-tuning the network's layers can help it from overfitting and make it use the limited training data even better and should be considered in the future.

Since the network was introduced in 2015, better versions of the UNet has been developed. In February this year, an enhanced UNet called MultiResUNet was introduced [17]. Instead of directly concatenate from the contracting path to the expansive path, MultiResunet takes the tensor through a convolution layer with residual connectors to increase the learnable parameters and to introduce more "WHERE" information that can be lost in the contracting path. Trying this network out on US images is a good idea and was also left for future work.

6.2.2 Preprocessing

The results showed that the purposed SRAD preprocessing proved to be useful to normalize the data and increased the success rate for the segmentation. Preprocessing is highly recommended when using CNN on US images since the variation in the images is a big challenge, and normalizing the images is a good way of reducing this variation. While SRAD gave good results, the preprocessed images did still suffer from certain artifacts. The SRAD can be more refined, however, the algorithm took a considerable long time with the computers used for this project, making refining very time consuming and since it was not the main focus of this thesis it was left for future work.

Another way to use preprocessing besides normalizing is to create more training data using data augmentation on the existing images. Augmentation is common in DL when using a small dataset, and could be a good idea to use for this task. By rotating, cropping and horizontally flipping the training data, more variance is introduced to the network's training making it better at segmenting diverse images. In April this year, research was done using simulated US images to increase the training data for the UNet [18]. The simulated images showed a positive impact on the network and could be a good way to improve the network for segmenting the CCA.

6.2.3 Postprocessing

The circle detection used in this project is very specialized for this particular task since the segmentation mask was circles. However, there are many more powerful algorithms available that can be used to better fit the outputted segmentation masks. Since the network

manages to find the location of the CCA reliably, the mask can be used to make previous semi-automated algorithms fully automated.

Further postprocessing that can be done, is creating a 3D reconstructions from the outputted segmentation masks. The research of the CCA at the Lund University includes segmentation of the CCA to create a 3D reconstruction, which is currently done manually, and use this for further research. To implement this step with the network, and create an automatic 3D reconstruction from only cross-section US images would be a very useful tool in their research.

Chapter 7
Conclusion

The main purpose of this thesis was to see if UNet is able to solve segmenting CCA from US images. And with the help of SRAD pre-processing, the UNet architecture shows promising solutions to the segmentation problem and can be a good tool for future biomedical research. With a 0.871 DICE coefficient on different sets of train and test datasets, the network is suitable for CCA segmentation. The resulting segment masks show that the network is affected by the big variation in images and the artifacts that make the CCA unclear in the images. Without a normalization prior to training and testing the network has trouble when the difference between the train and test data is big, but when the training data has a big variance the network works best. Overall, the best use for this network is on clear images of the CCA or at least images with less variance and artifacts.

Bibliography

[1] W. Johnson, O. Onuma, M. Owolabi, and S. Sachdev. Stroke: a global response is needed.

[2] M. Cinthio, J. Albinsson, T. Erlöv, N. Bjarnegård, T. Länne, and Å. R. Ahlgren. Longitudinal movement of the common carotid artery wall: New information on cardiovascular aging. *Ultrasound in Medicine and Biology*, 44 No. 11:2283 – 2295, 2018.

[3] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015.

[4] Y. Yu and S. T. Acton. Speckle reducing anisotropic diffusion. *IEEE Transactions on Image Processing*, 11(11):1260–1270, Nov 2002.

[5] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386, 1958.

[6] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[7] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

[8] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998.

[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, May 2017.

[10] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. *CoRR*, abs/1411.4038, 2014.

[11] R. Zemouri, N. Zerhouni, and D. Racoceanu. Deep learning in the biomedical applications: Recent and future status. *Applied Sciences*, 9:1526, 04 2019.

[12] R. Menchón-Lara and J. Sancho-Gómez. Fully automatic segmentation of ultrasound common carotid artery images based on machine learning. *Neurocomputing*, 151:161 – 167, 2015.

[13] PyTorch, 2019. [Online; accessed 23-july-2019].

[14] Wikipedia. Circle hough transform — Wikipedia, the free encyclopedia, 2019. [Online; accessed 23-july-2019].

[15] L. R. Dice. Measures of the amount of ecologic association between species. *Ecology*, 26(3):297–302, 1945.

[16] M. Amiri, R. Brooks, and H. Rivaz. Fine tuning u-net for ultrasound image segmentation: Which layers? In Q. Wang, F. Milletari, H. V. Nguyen, S. Albarqouni, M. J. Cardoso, N. Rieke, Z. Xu, K. Kamnitsas, V. Patel, B. Roysam, S. Jiang, K. Zhou, K. Luu, and N. Le, editors, *Domain Adaptation and Representation Transfer and Medical Image Learning with Less Labels and Imperfect Data*, Cham, 2019. Springer International Publishing.

[17] N. Ibtehaz and M. S. Rahman. Multiresunet : Rethinking the u-net architecture for multimodal biomedical image segmentation. *CoRR*, abs/1902.04049, 2019.

[18] B. Behboodi and H. Rivaz. Ultrasound segmentation using u-net: learning from simulated data and testing on real data, 2019.

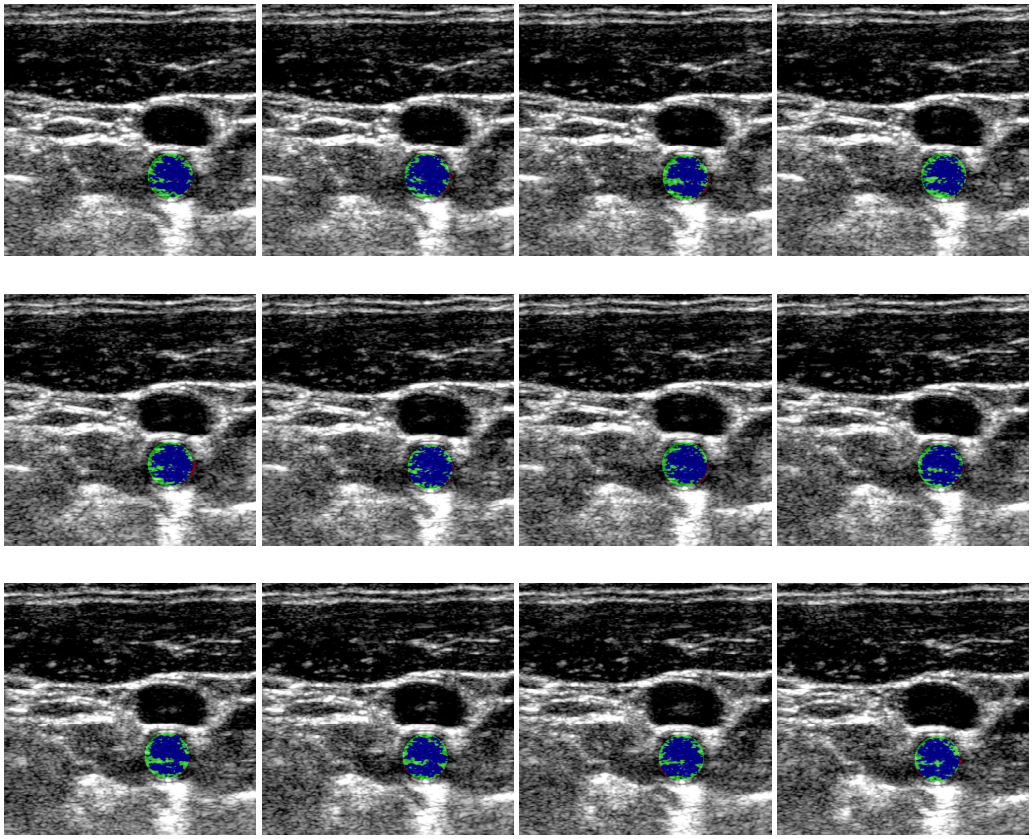
Appendices

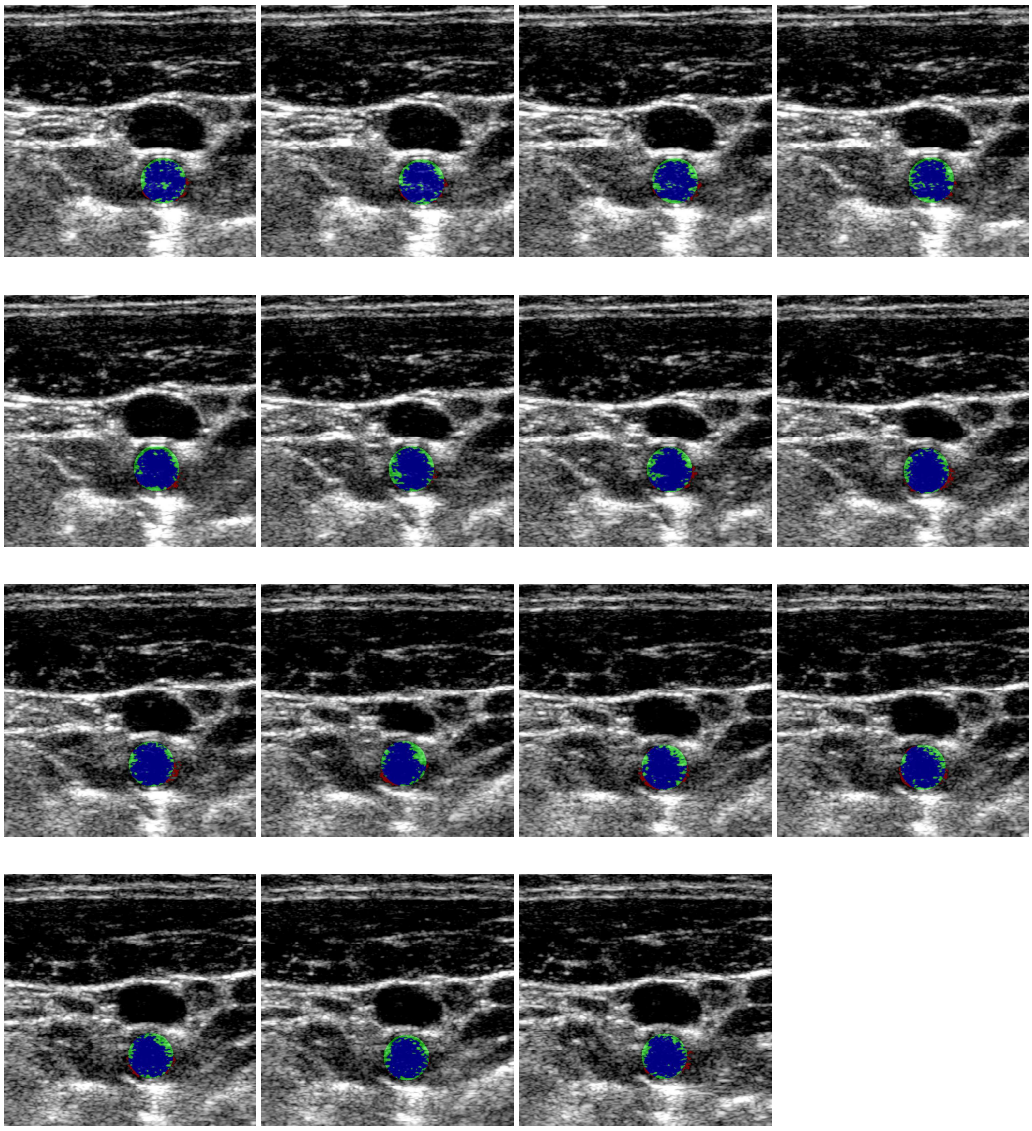
Appendix A

Outsegmentation masks

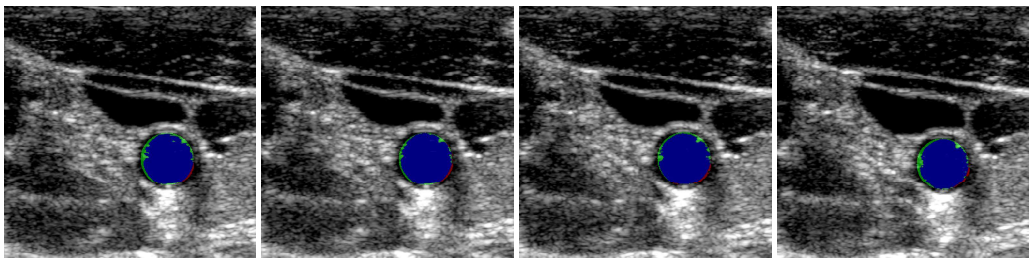
A.1 Raw datasets

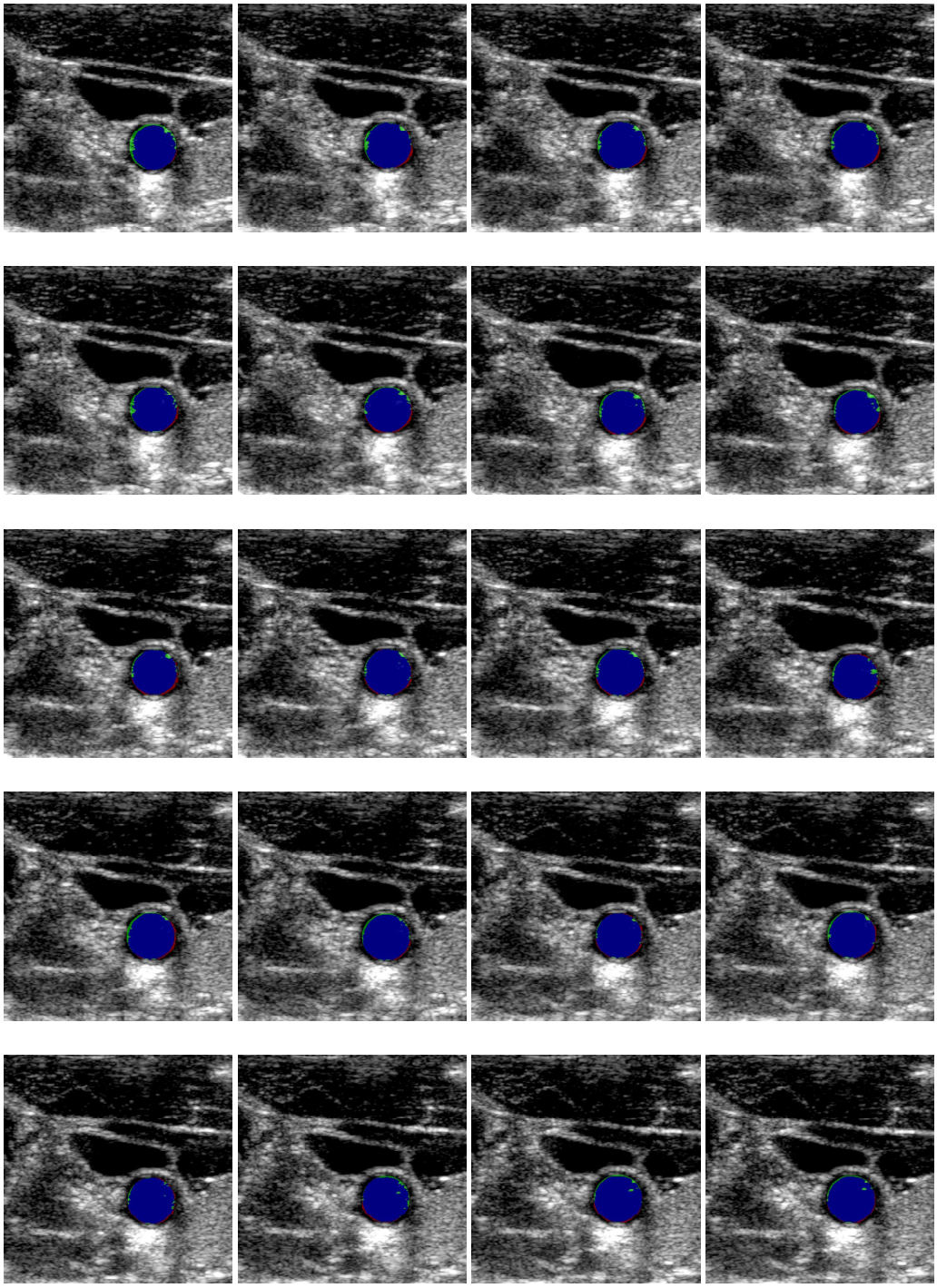
A.1.1 TP1

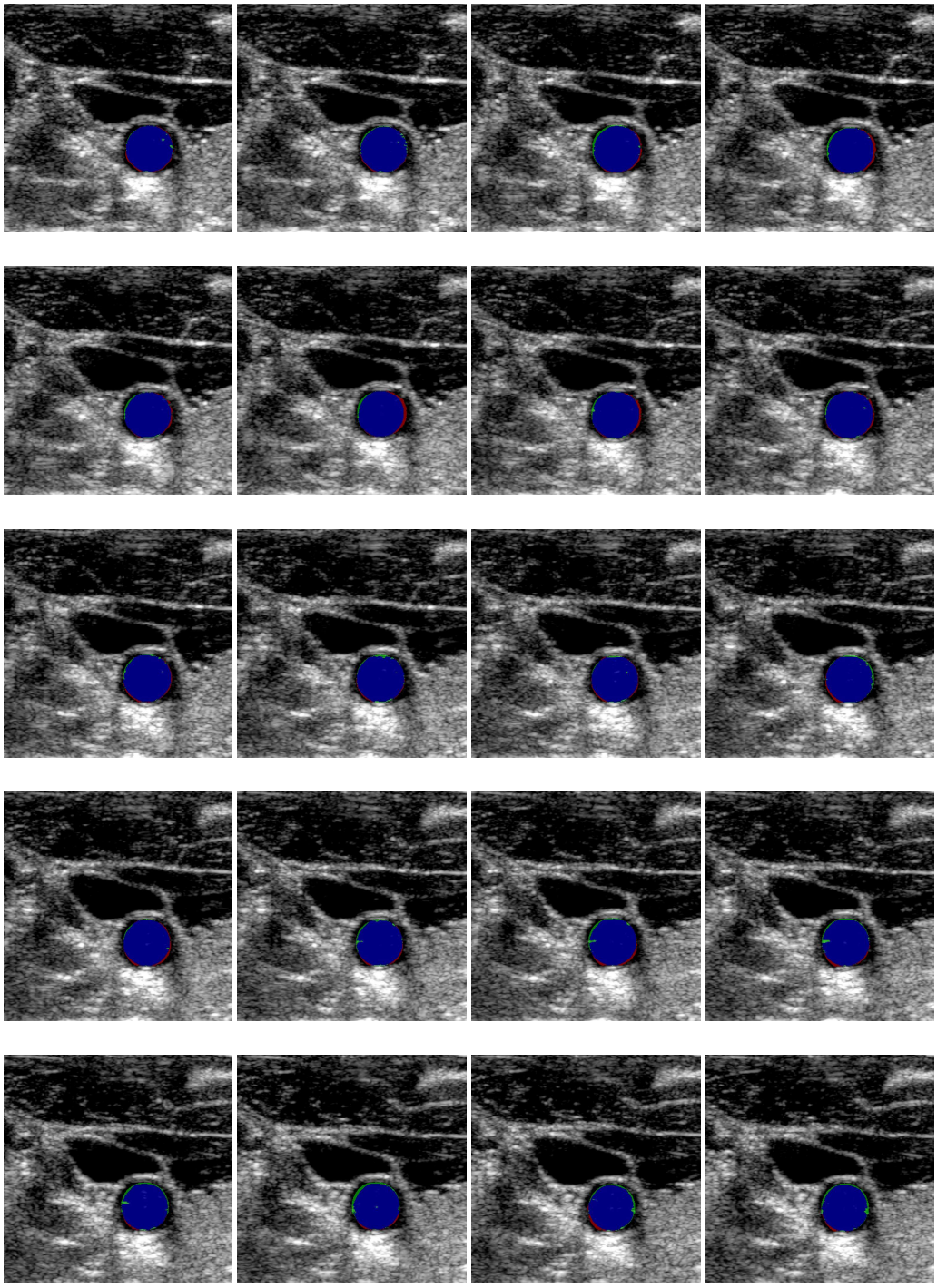


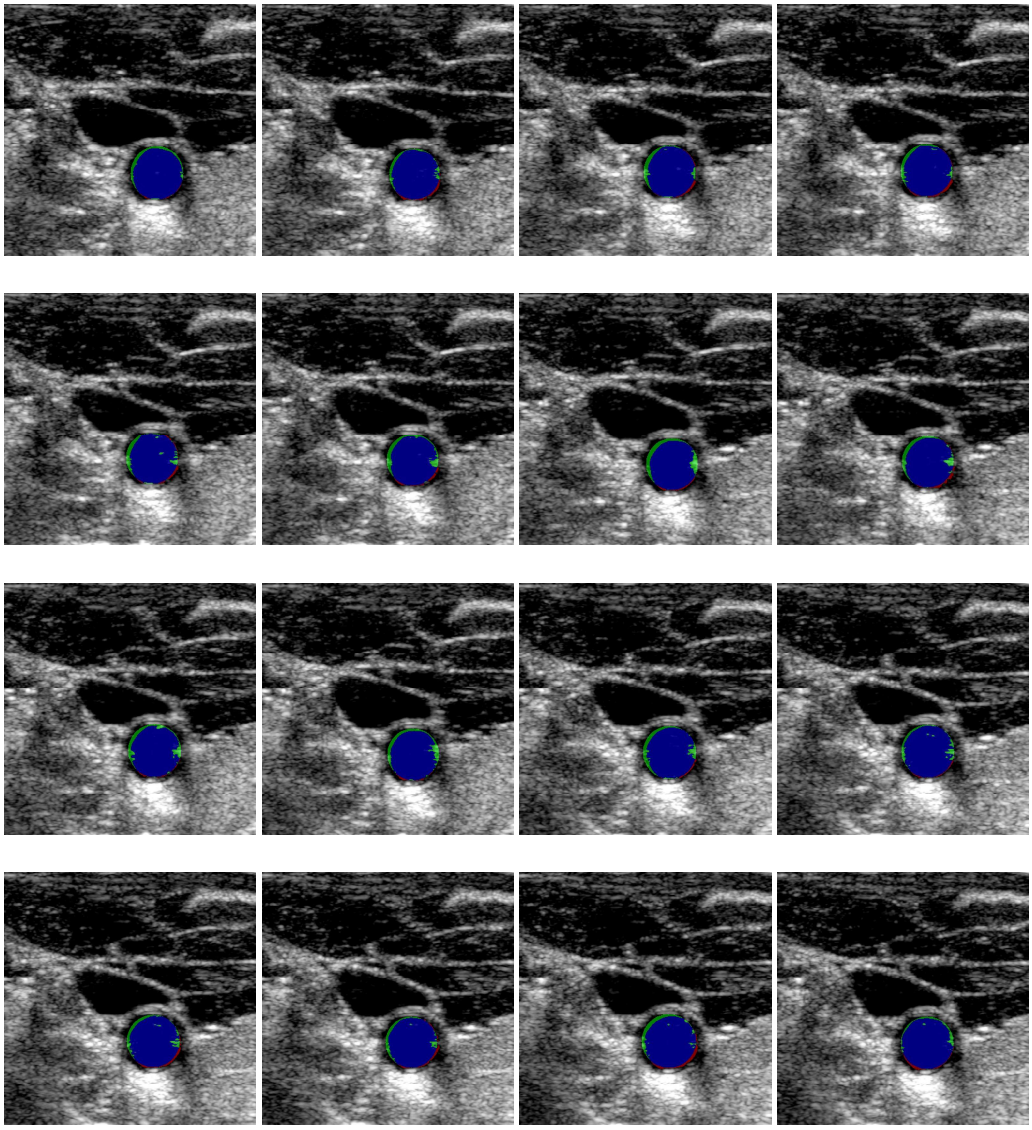


A.1.2 TP2

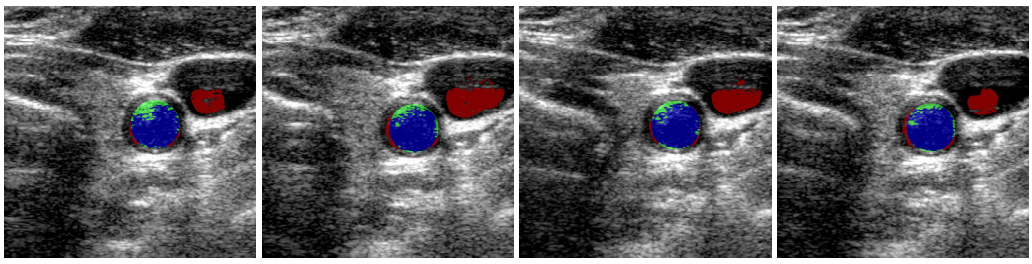


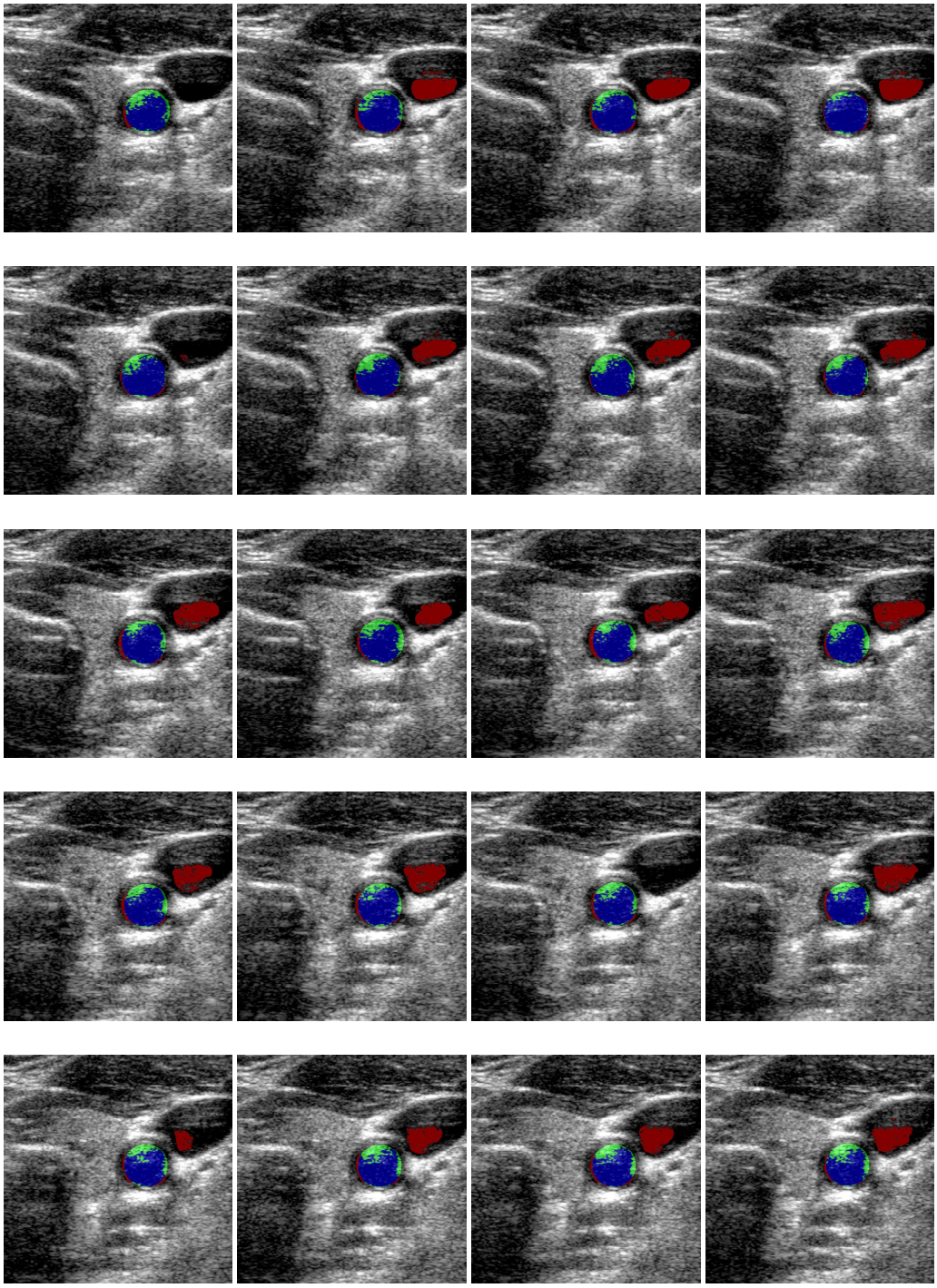


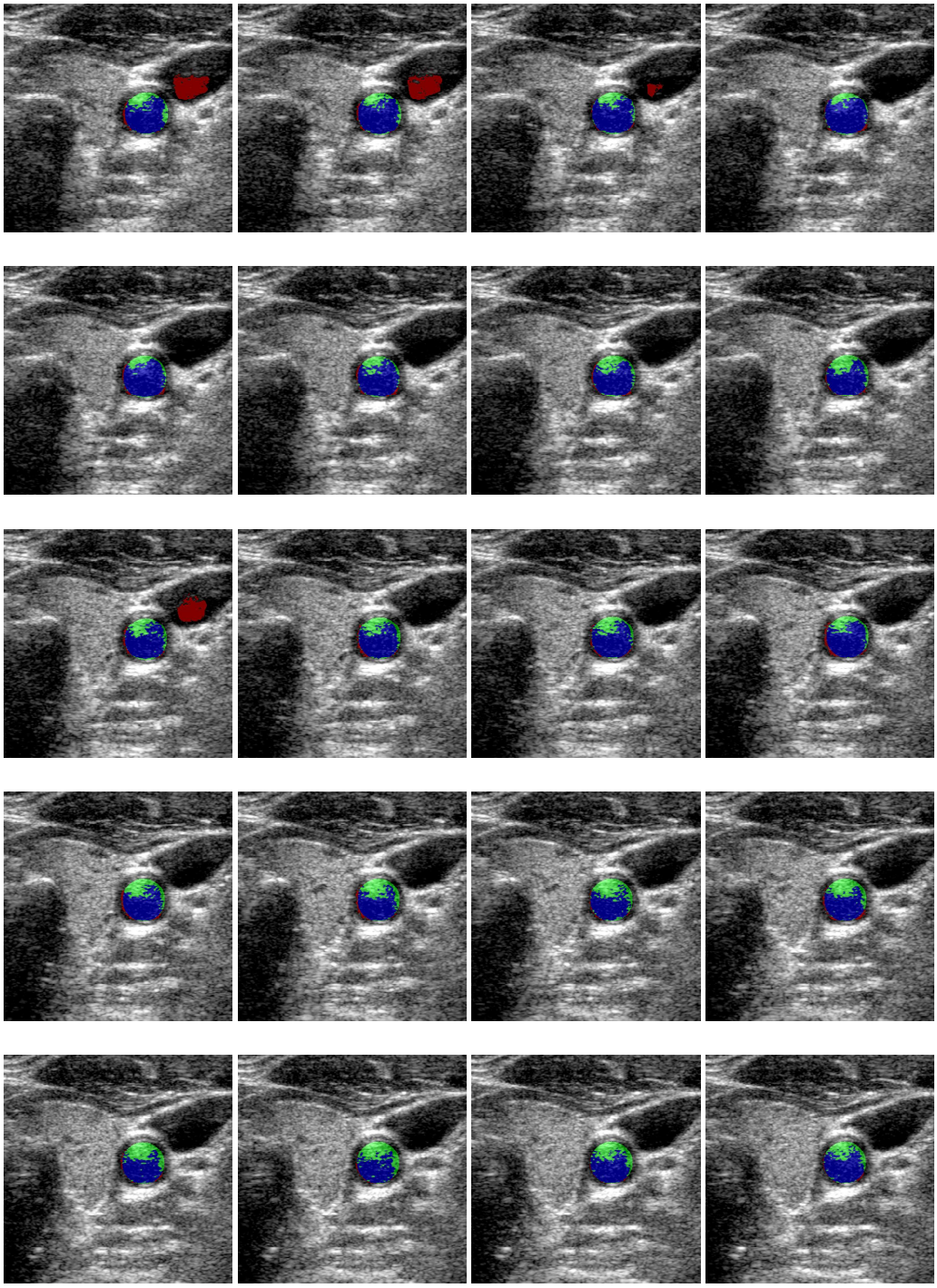


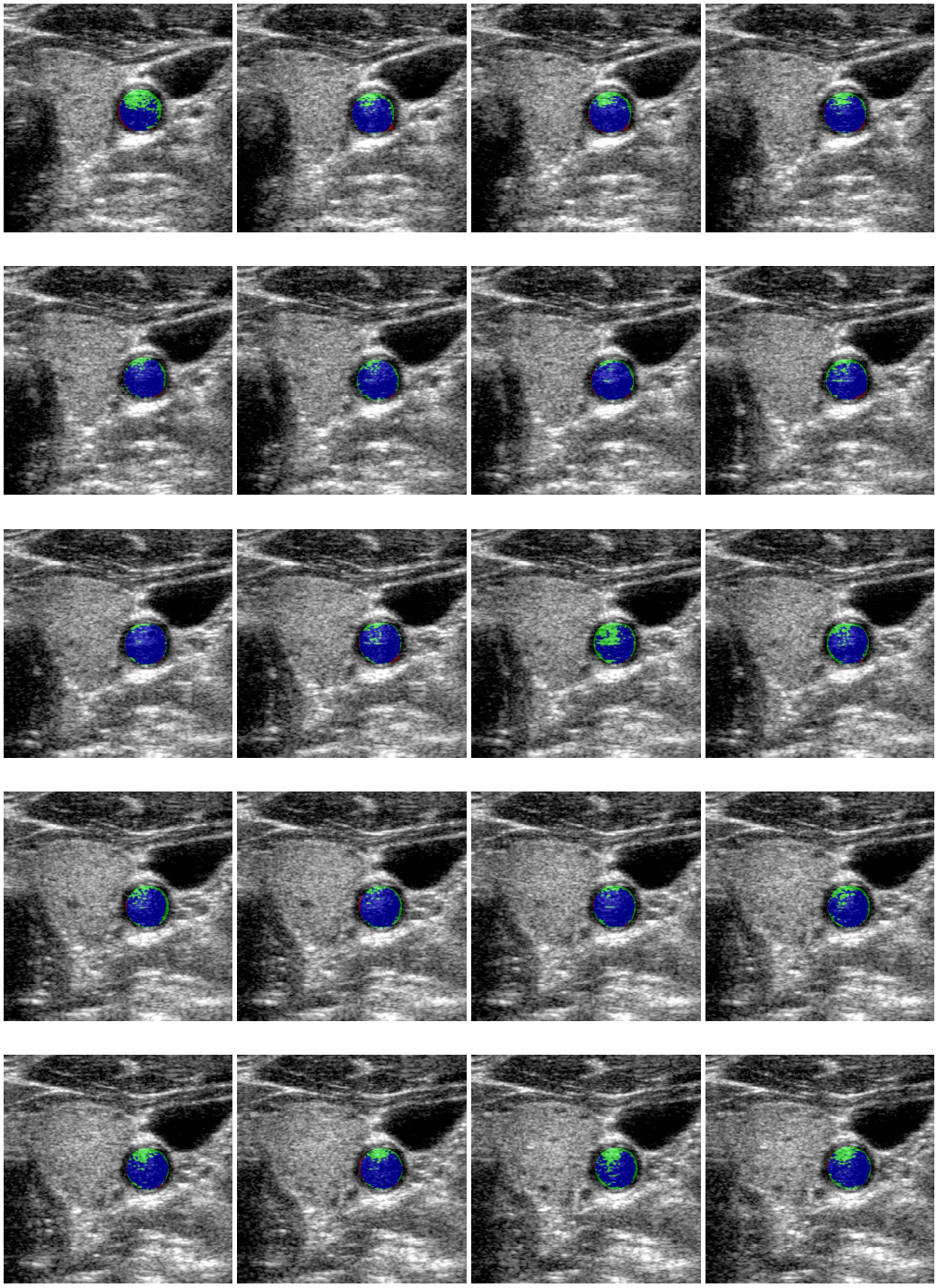


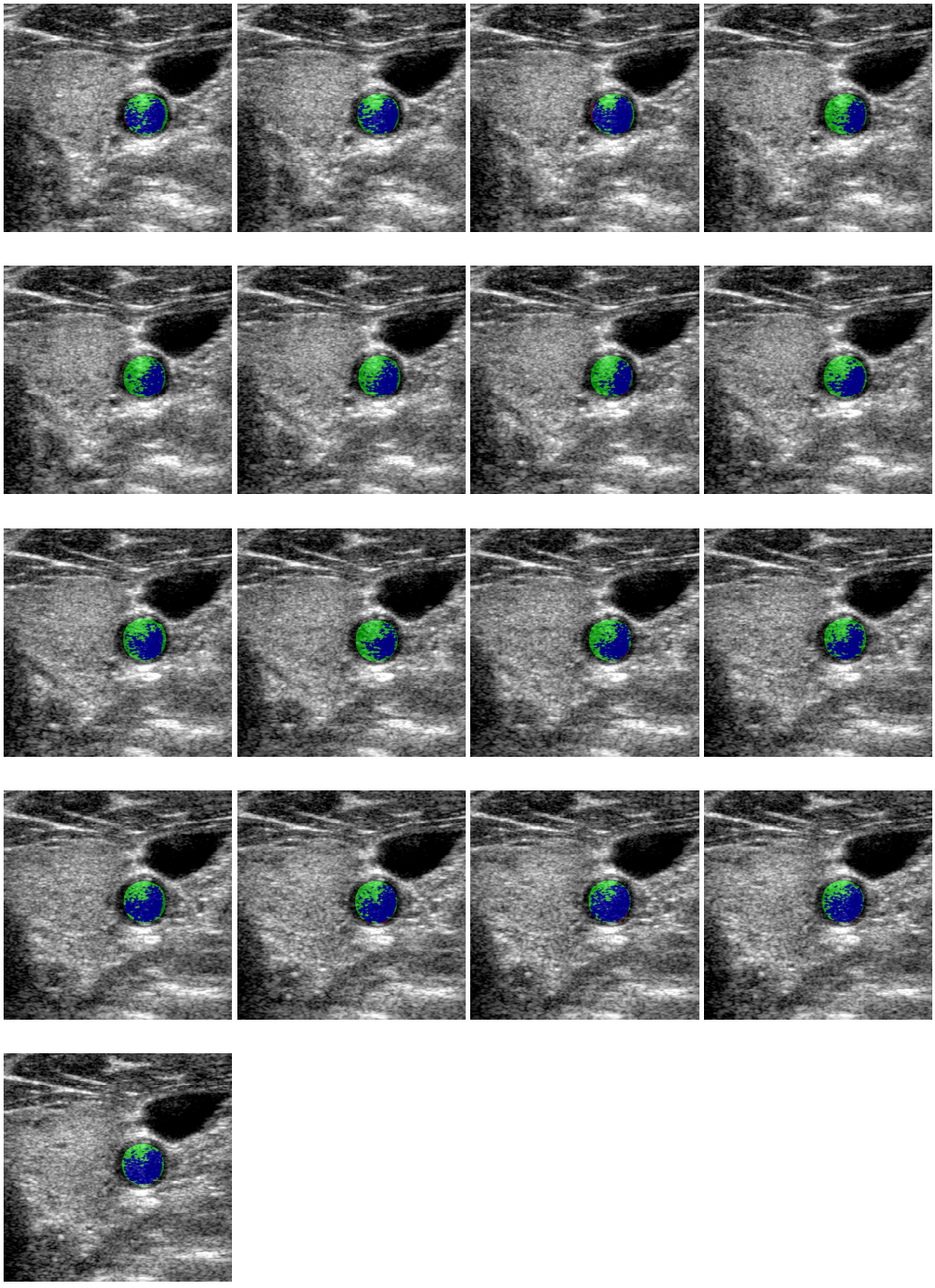
A.1.3 TP3



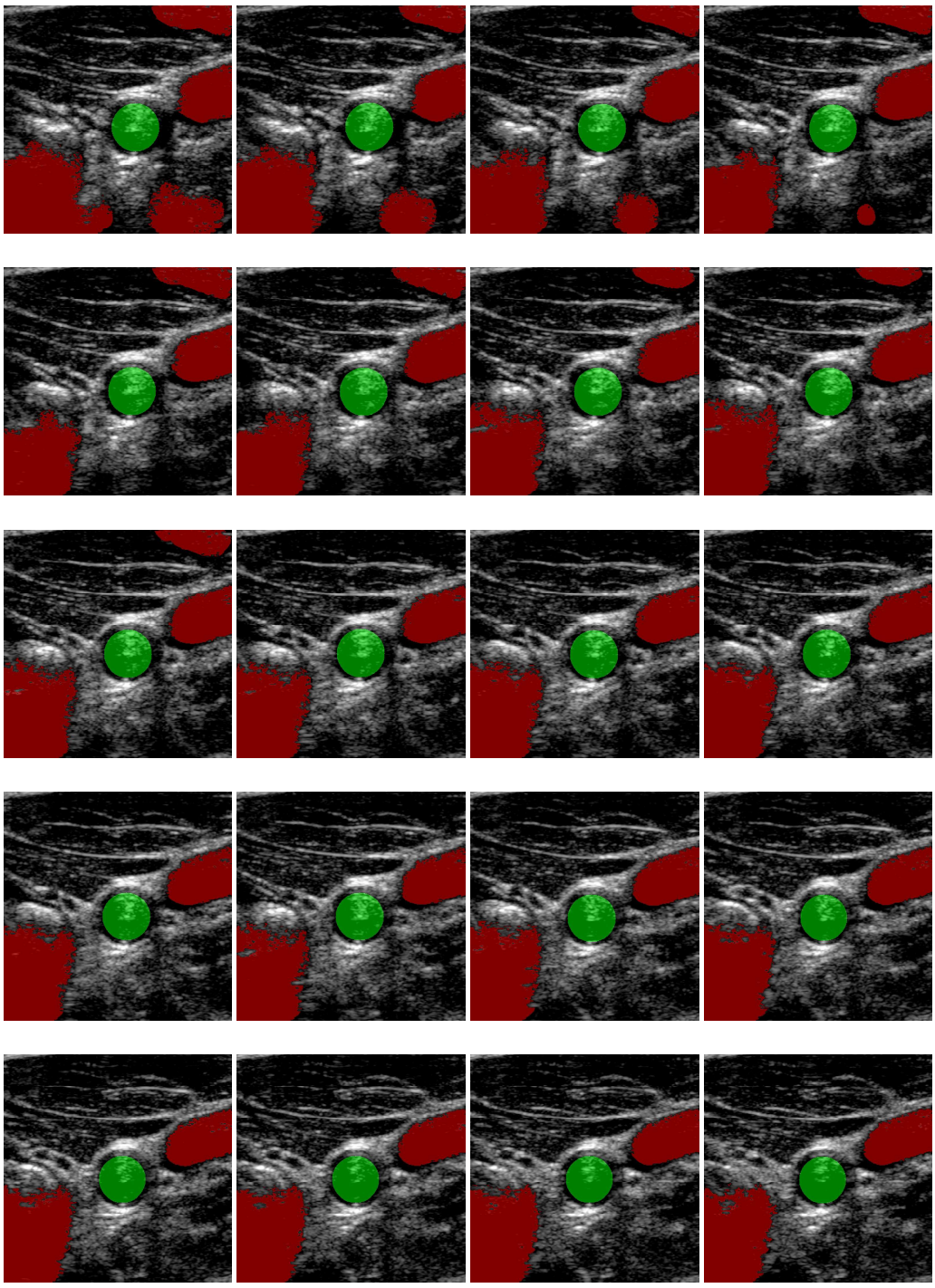


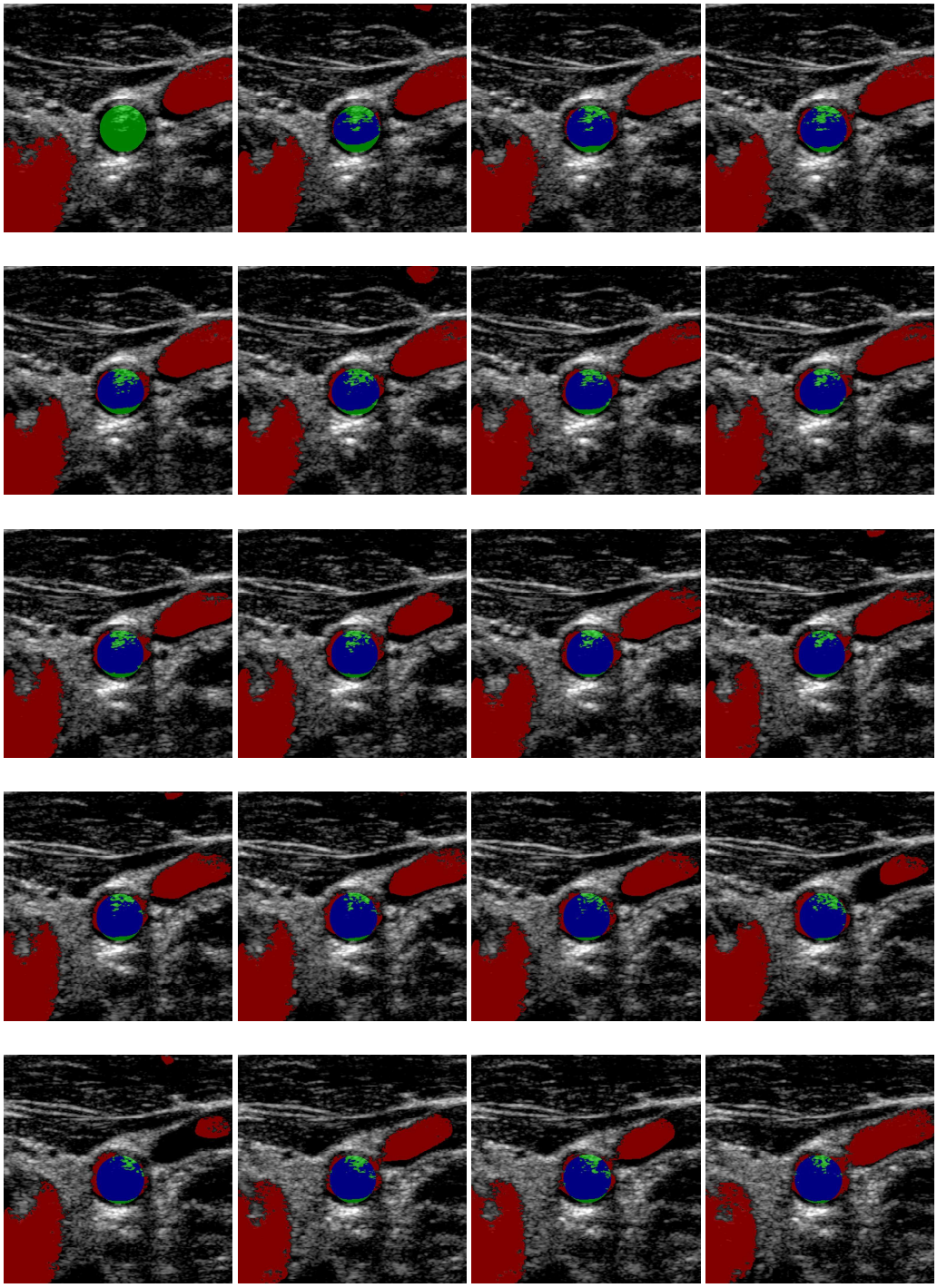


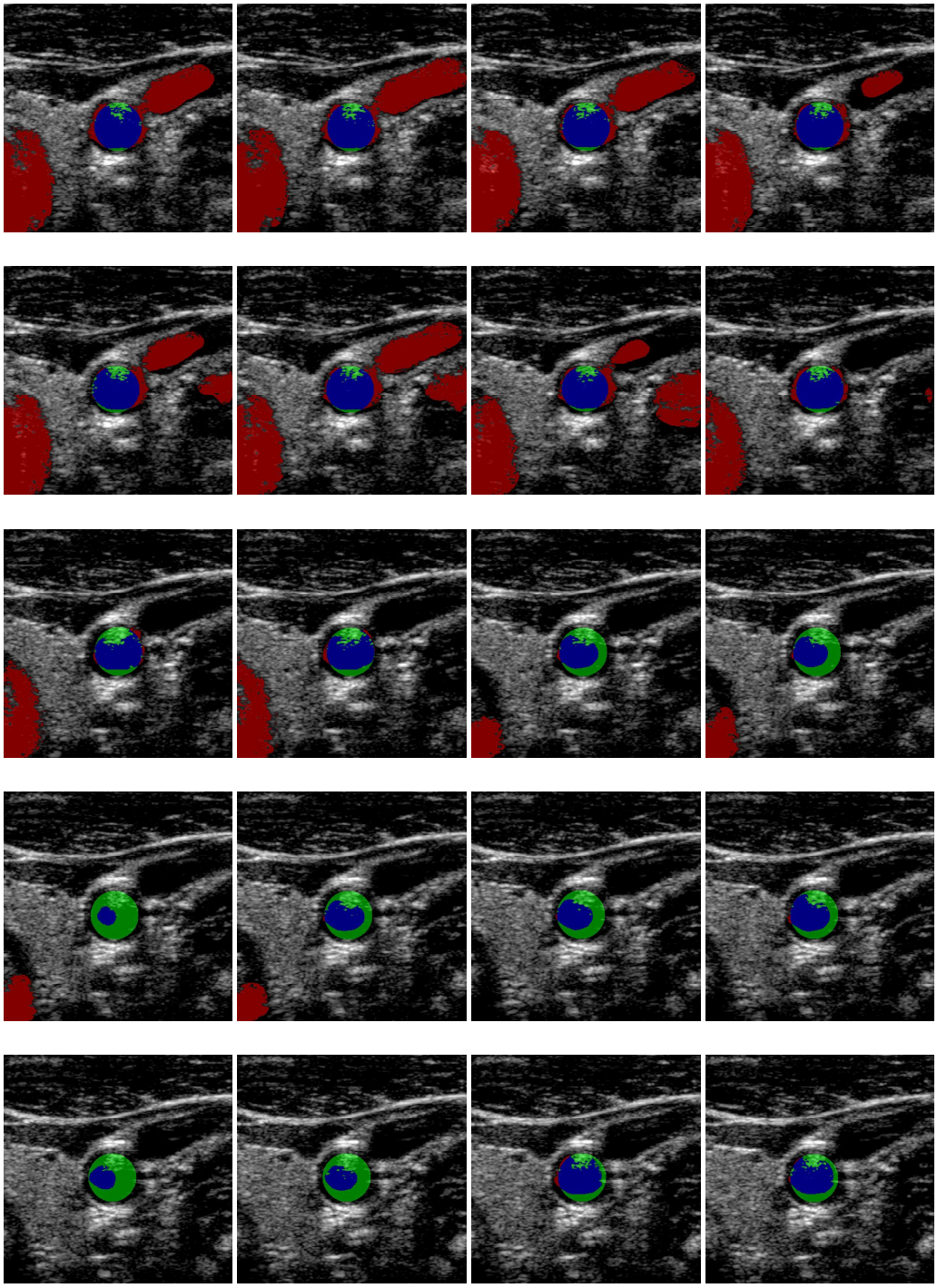


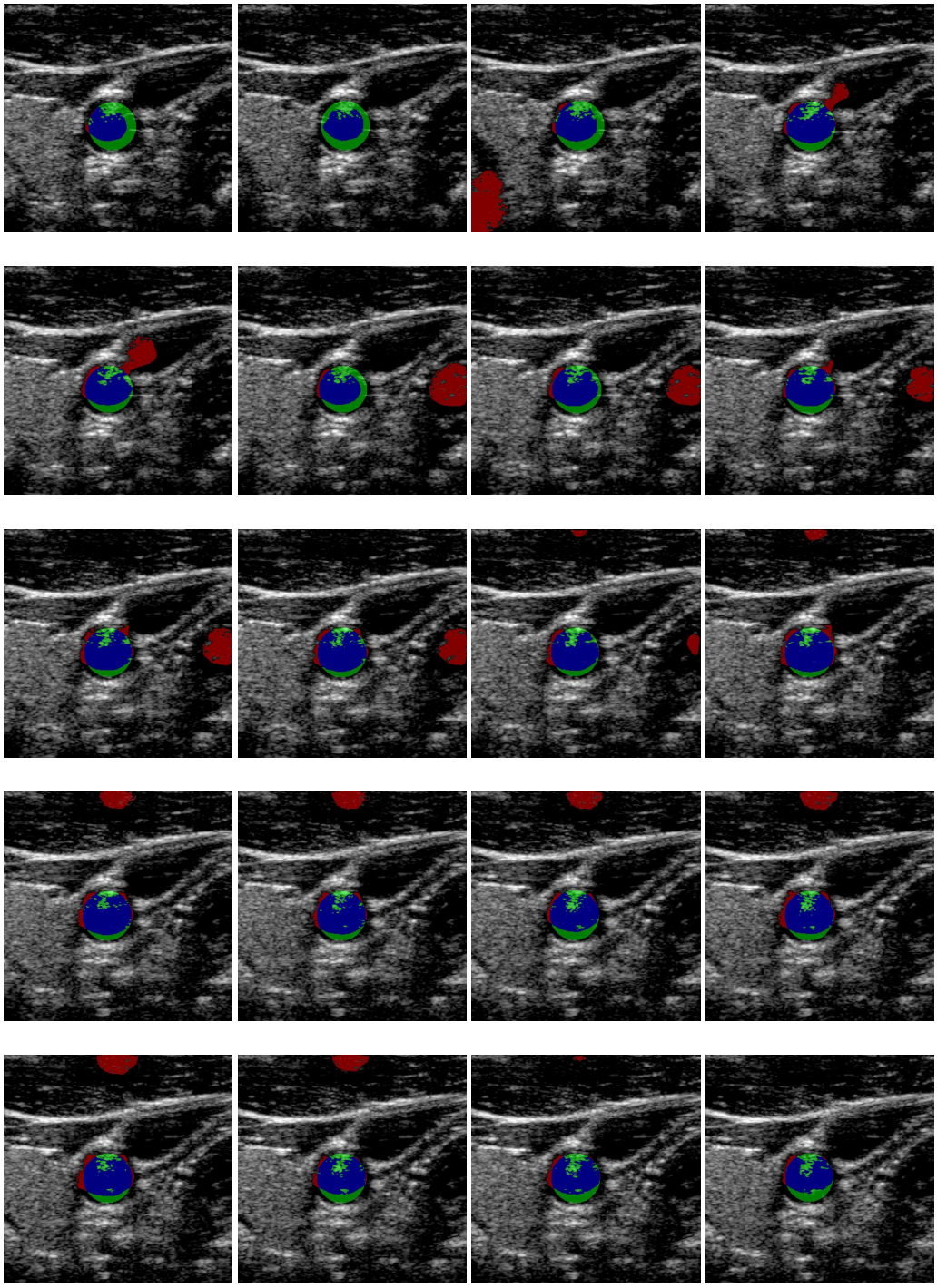


A.1.4 TP4



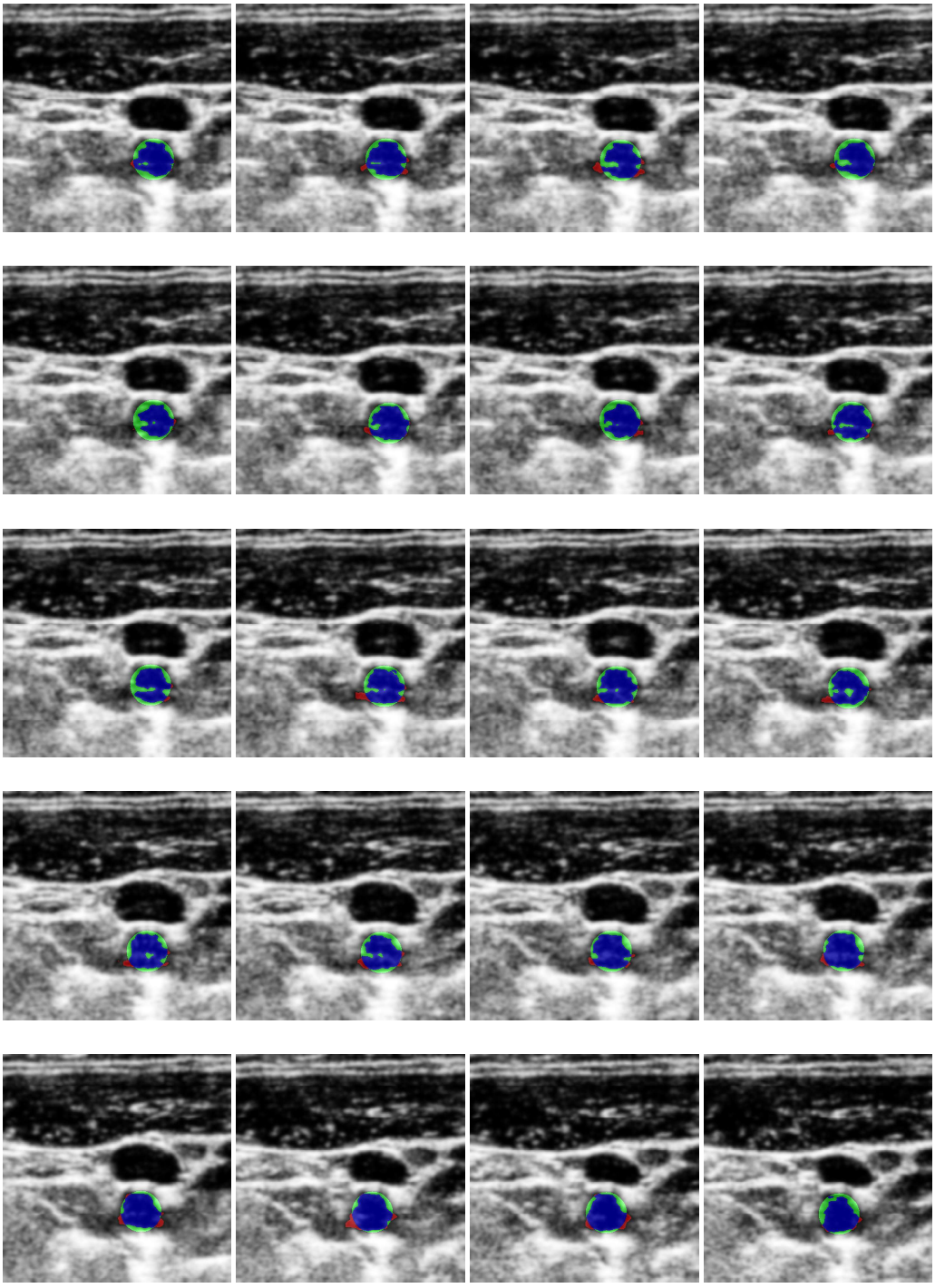


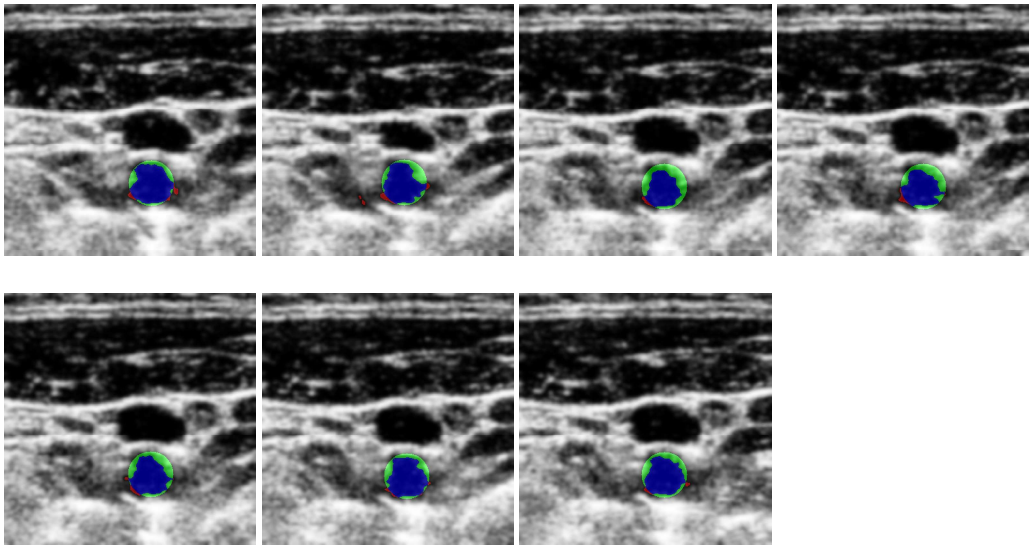




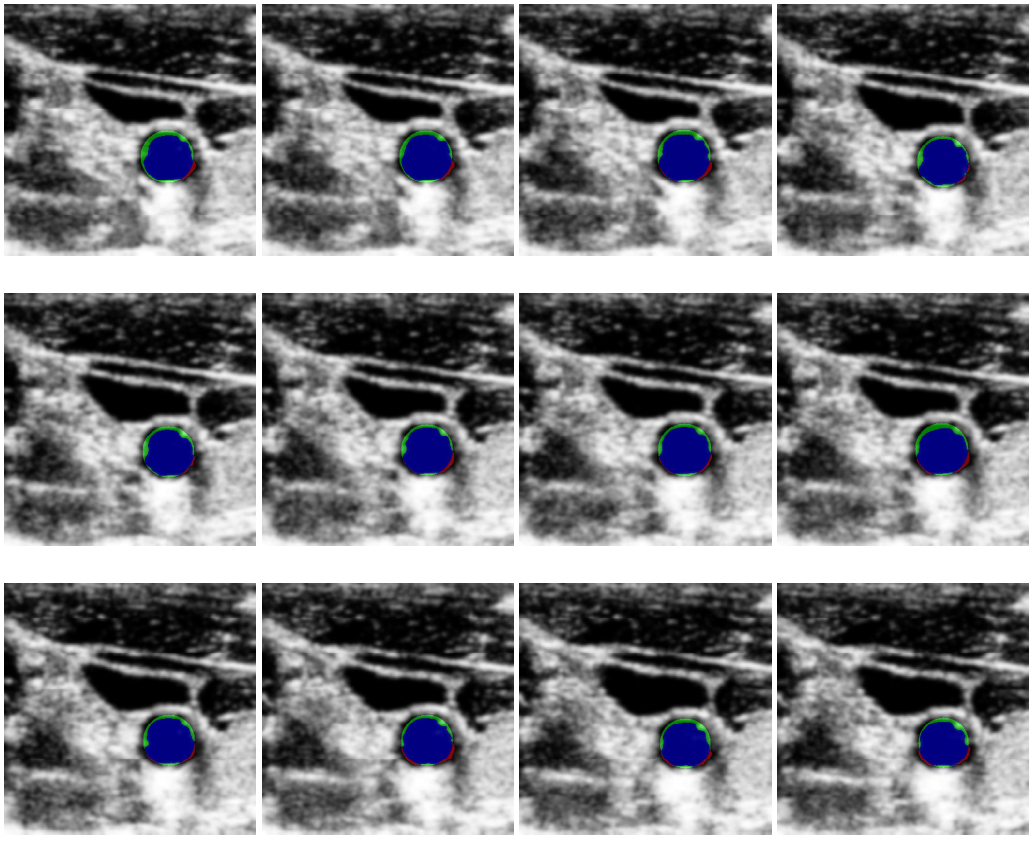
A.2 Preprocessed datasets

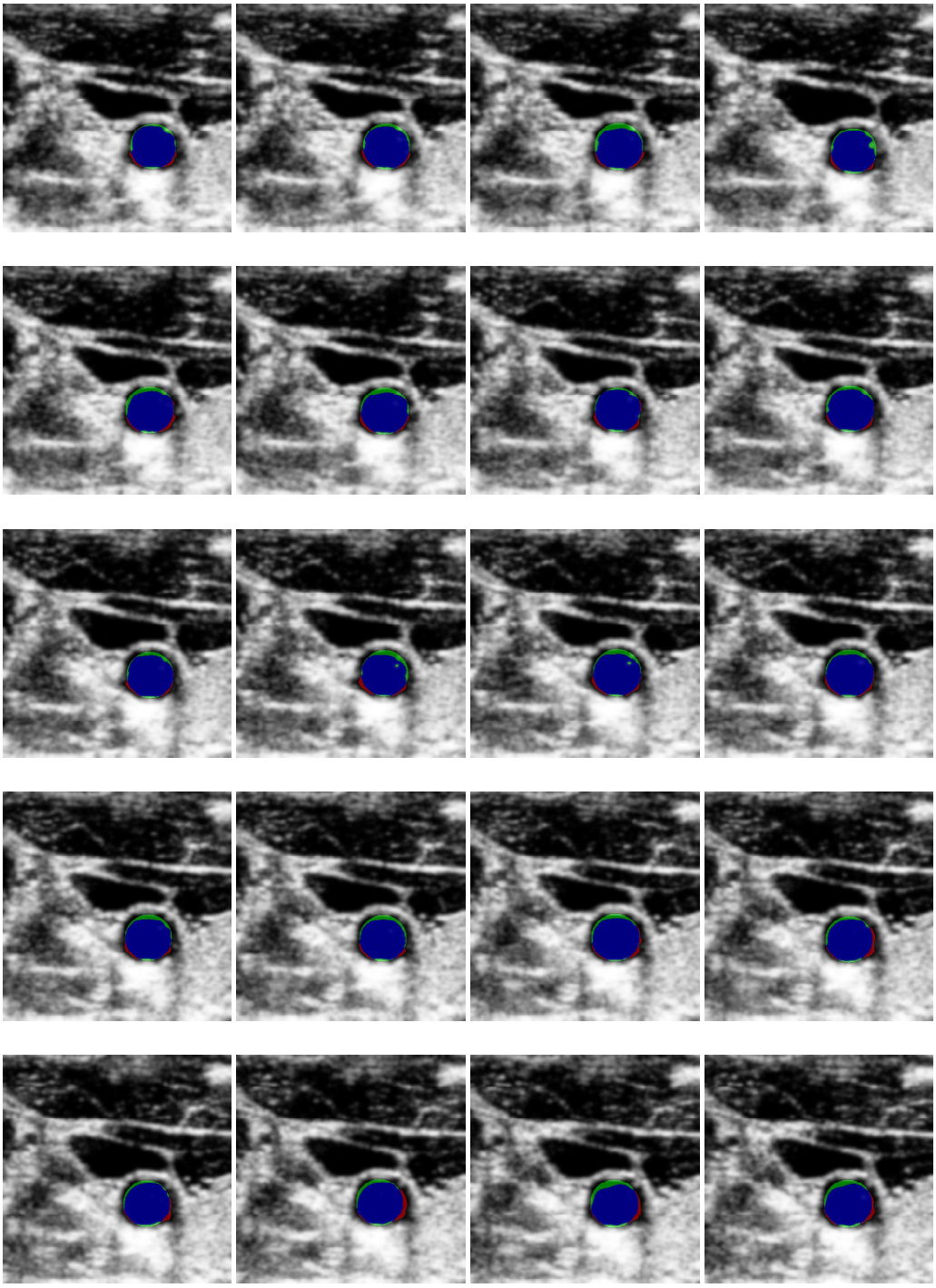
A.2.1 TP1

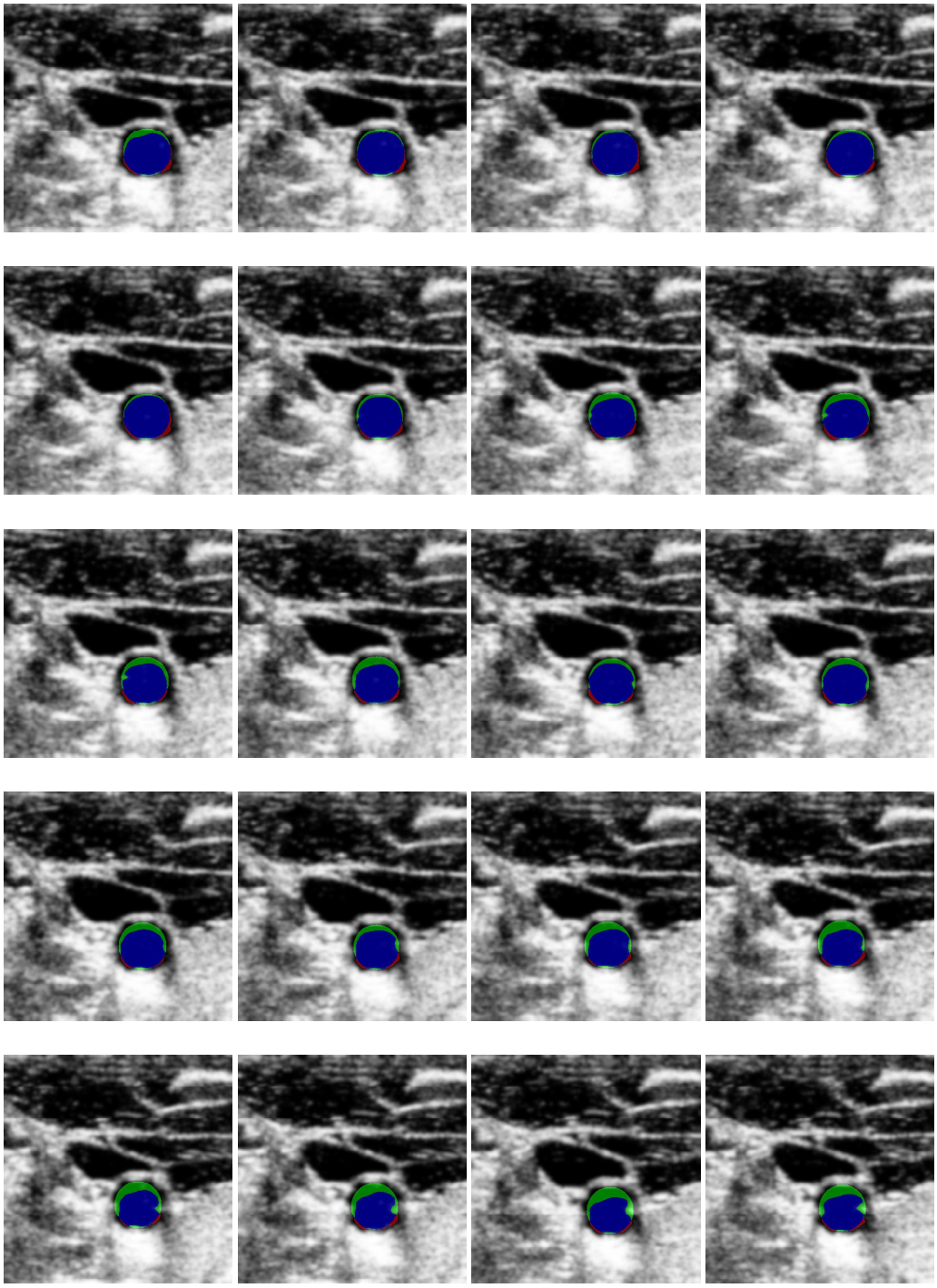


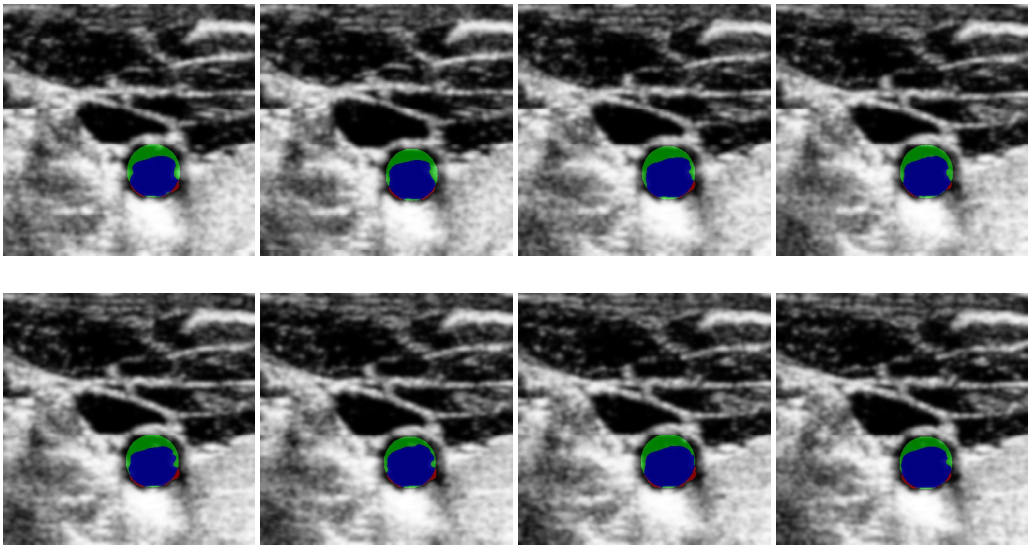


A.2.2 TP2

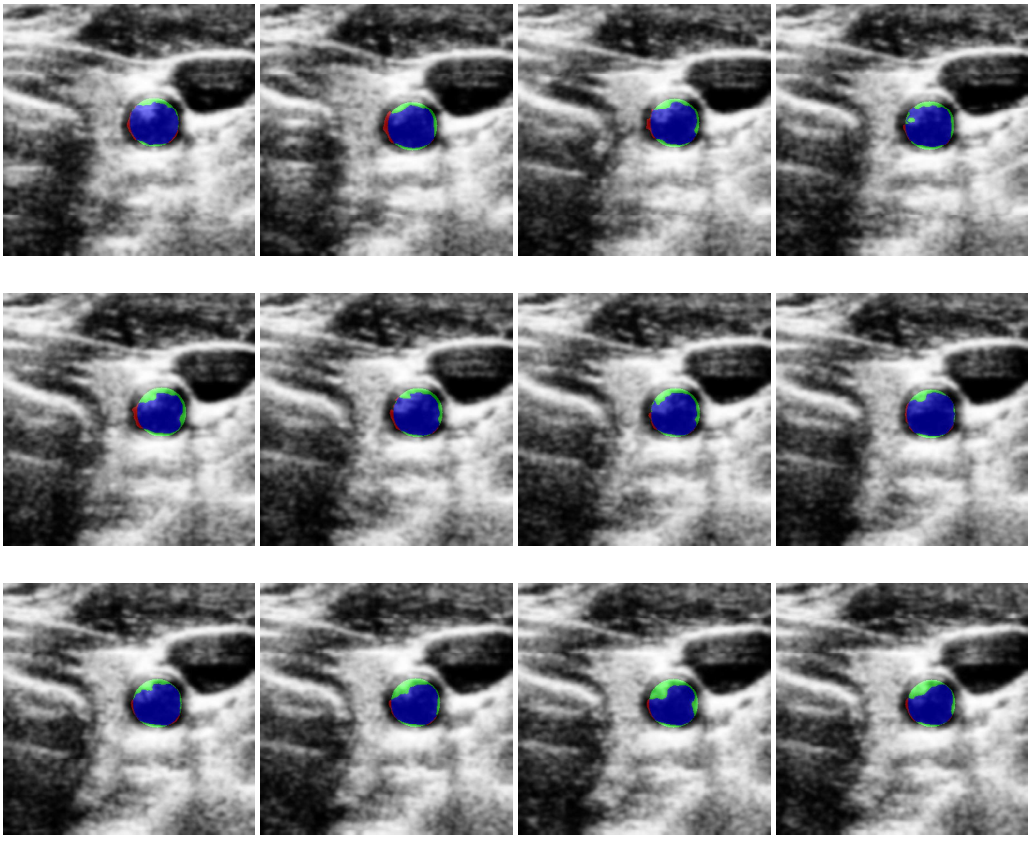


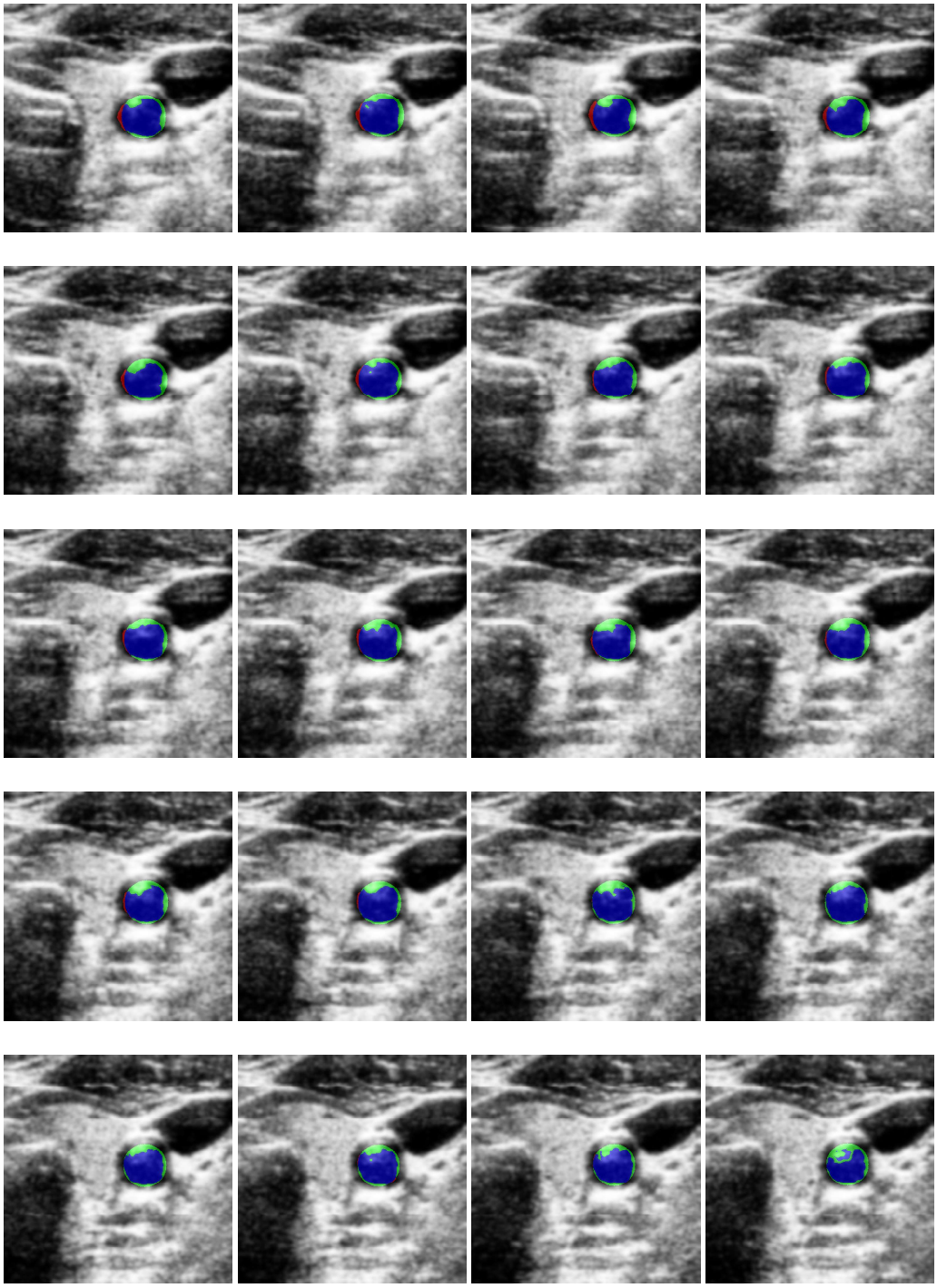


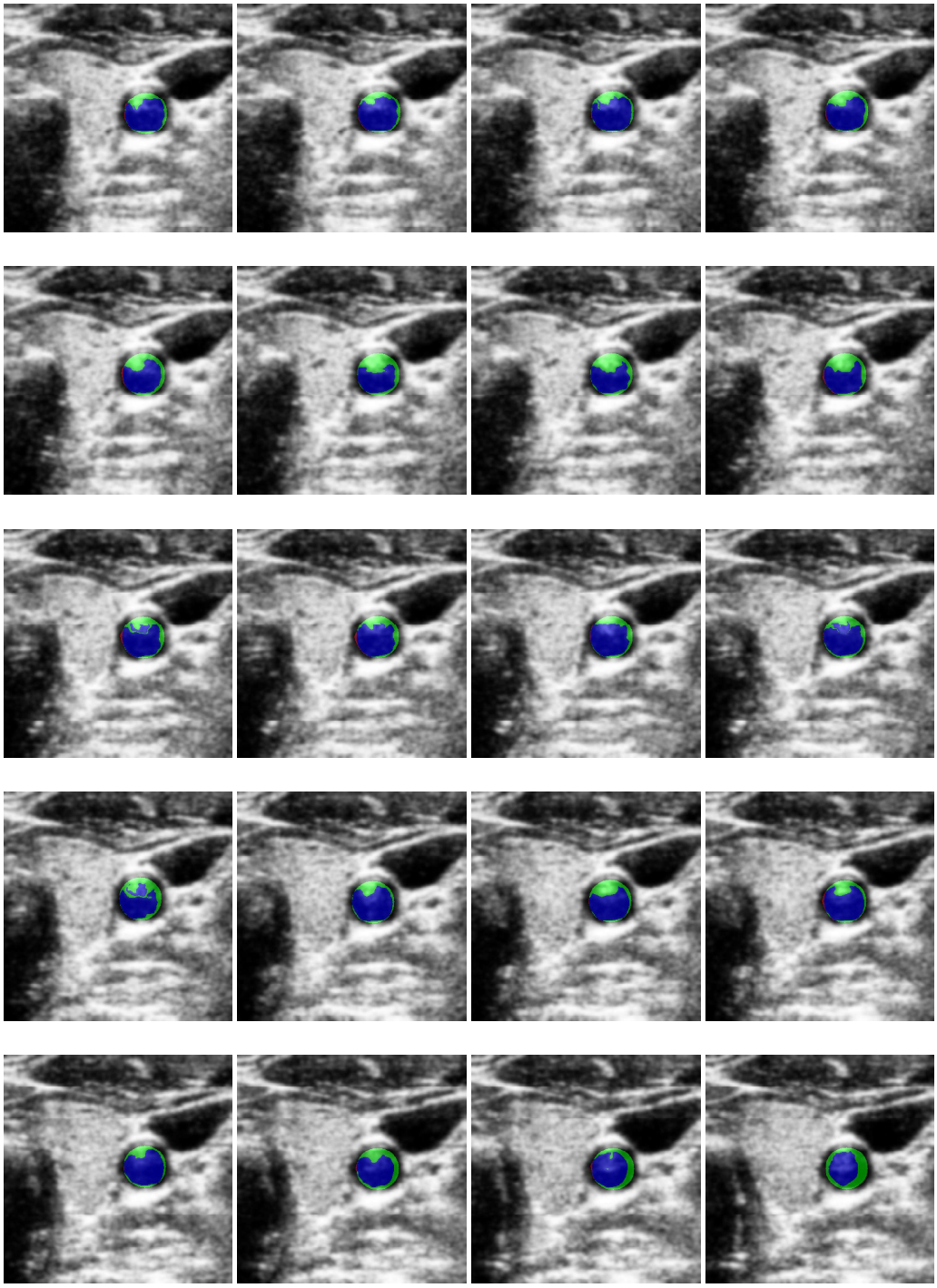


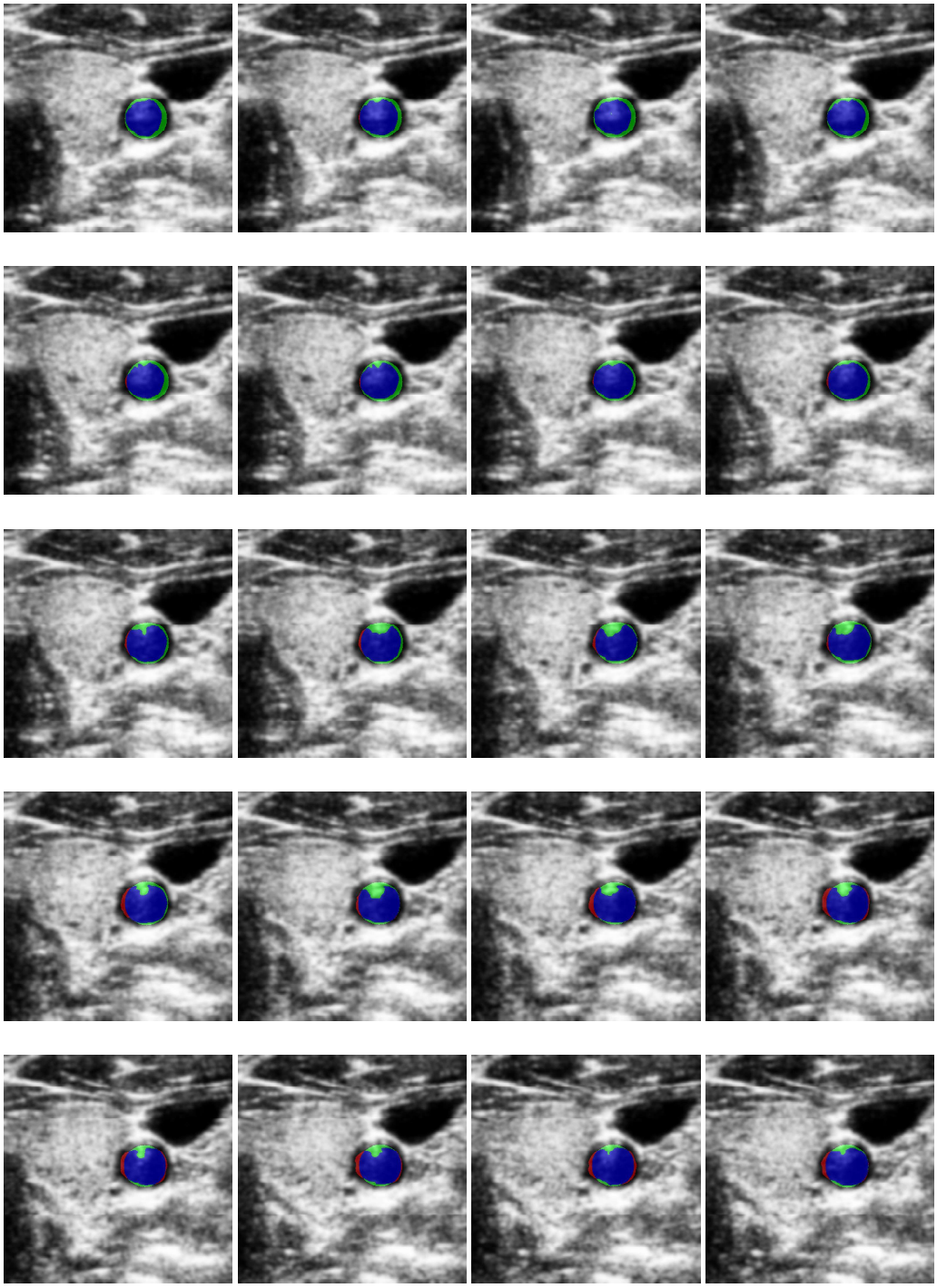


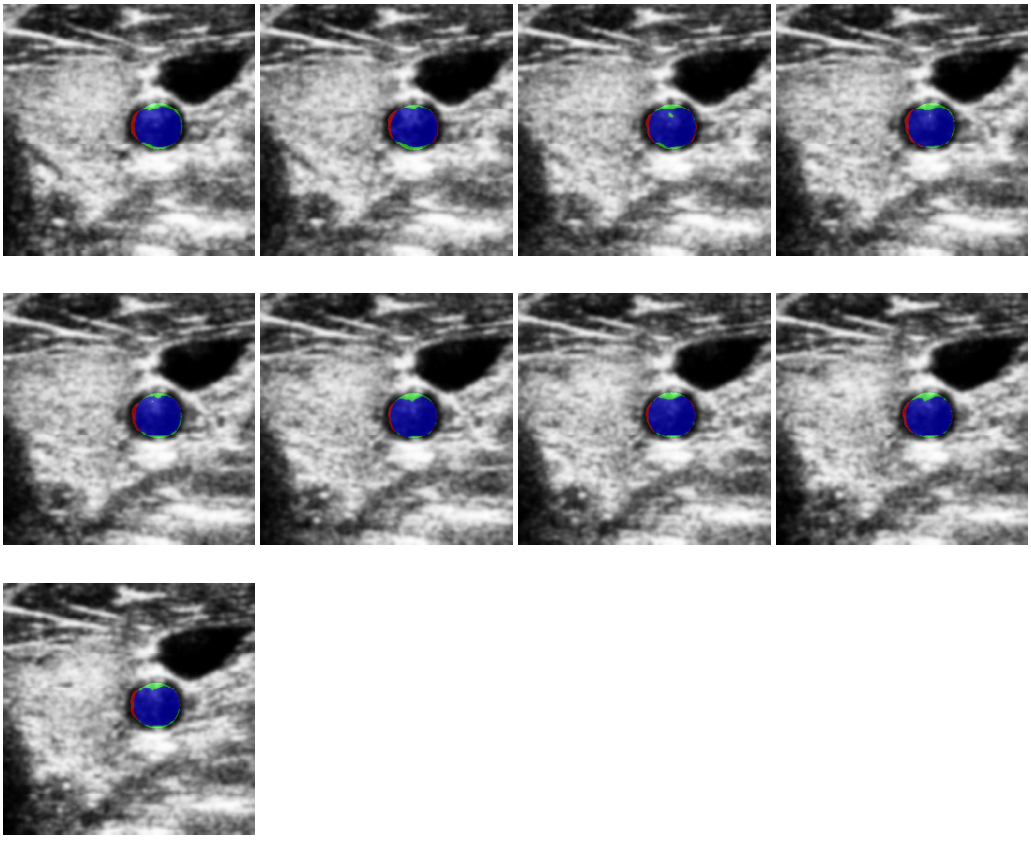
A.2.3 TP3











A.2.4 TP4