BABEŞ BOLYAI UNIVERSITY, CLUJ NAPOCA, ROMÂNIA
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE

AI ASSISTANT FOR HEART MEDICAL IMAGERY

Team members
Cotuțiu Ioana, Software Engineering, 258
Drimba Alexandru, Software Engineering, 258
Duma Iulia Ana-Maria, Software Engineering, 258
Filipciuc Andreea, Software Engineering, 258
Gherghel Denisa-Maria, Software Engineering, 258

2019

Contents

1. Introduction

In the learning process carried out by a medical student, an application would be useful to visually represent relevant information about the studied organs and diseases. The purpose of this project is to design a program that will address this need, using artificial intelligence components.

Therefore, we will develop an application that will allow the visualization of an organ or its possible defects. Using an intelligent algorithm, the program will generate the desired output based on information taken from medical imaging.

The interpretations of medical data are being mostly done by medical experts. Since there is a rapid growth in medical images, the interpretations require even more extensive and tedious efforts by medical experts. Furthermore, the interpretation by a human expert is quite limited due to its subjectivity, the complexity of the image, extensive variations that exist across different interpreters, and fatigue.

Machine learning promises the potential to deal with big medical image data for accurate and efficient diagnosis. Artificial intelligence will not only help to select and extract features but also construct new ones.

2. The scientific problem

Computer-aided diagnostic systems have long been used by radiologists, mainly in chest x-ray and mammography applications. However, in recent years, the research on the application of AI in the medical field has been expanding. Therefore, the use of artificial intelligence in the field of radiology aims to reduce the rate of occurrence of errors due to fatigue, inattention, medical judgment. Due to its state as an assistant and not as a replacement for the specialist, AI can also be introduced in the academic field. Students can use AI to practice the theoretical notions learned in the courses.

From a formal point of view, the scientific purpose of this project is the analysis of medical images, namely segmentation, classification, and detection of defects in magnetic resonance imaging and computed tomographies. Our application must detect defects marked by different textures or hues of the pixels.

3. Related work

3.1. Before AI

An attempt was made to introduce semi-automatic methods for the segmentation of medical images. These can be divided into several categories [2]:

1) atlas-based methods - this method requires the creation of masks by a human expert, masks that will be used for the segmentation of new images. This is usually done by looking for the correct alignment of the image with the mask, a process called image registration;

2) statistical models - this method obtains a parameterized model by using training data to learn how multiple organ-specific structures can vary. The correctness of this method depends on the accuracy of the data initialization and the presence of noise in training data;

3) deformable models - given an initial contour of the structure of interest, this method can evolve to fit the targeted structure as accurately as possible. This model does not require training data, nor prior knowledge, although it needs contour initialization and stopping criteria definition.

These methods are semi-automatic and require initialization by a human expert, which leads to subjective errors. At the same time, they are dependent on several factors such as the size or the acquisition angle of the image. Those are prone to long periods of segmentation, overfitting and biased results.

To solve these shortcomings, automatic solutions were introduced, based on Artificial intelligence.

3.2. Machine Learning

In recent years, the focus of the researchers has shifted towards the application of artificial intelligence in the field of medical imaging. Deep learning algorithms, in particular, convolutional networks, have rapidly become a methodology of choice for analyzing medical images. These algorithms are extensively used to solve challenging tasks such as classification, segmentation and object detection.

3.2.1. Convolutional Neural Networks

Convolutional Neural Networks (CNN) is one of the variants of neural networks used heavily in the field of Computer Vision. It derives its name from the type of hidden layers it consists of. The hidden layers of a CNN typically consist of convolutional layers, pooling layers, fully connected layers (also known as *Fully Convolutional Neural Networks - FCN*), and normalization layers.

3.2.1.1 Algorithm

Most of the papers that use CNN for medical image analysis propose a cascade structure. This structure involves the overlap of several neural networks, each of which will deal with

a binary segmentation problem. This determines a piping mechanism through which the output of one network will be the input of the next. For example, such an approach is presented in the article [8] in which three networks are used to hierarchically and sequentially segment substructures of a brain tumor. The cascade helps to reduce false positives.

In [8] the problem of high memory consumption in the case of 3D training images is also addressed. The proposed solution consists of an Anisotropic Convolutional Neural Networks, which basically uses an anisotropic kernel. These networks take a stack of slices as input with a large receptive field in 2D and a relatively small receptive field in the out-plane direction that is orthogonal to the 2D slices.

3.2.1.2. Data

Multiple research centers provide datasets containing medical images available for the general public. These may be combined, for several organs, or might be specific for a particular organ. Depending on the problem studied, a certain type of dataset may be more useful. For example, two organ-specific datasets are:

1) *for the brain* - BraTS (Multimodal Brain Tumor Segmentation Challenge) data sets were used in several scientific articles, e.g. [8], [4]. They are made available by the University of Pennsylvania and are updated annually with new medical images. The BraTS validation and testing sets contain images from patients with brain tumors of unknown grade. Each patient was scanned with four sequences: T1, T1C, T2, and FLAIR. All the images were skull-striped and re-sampled to an isotropic 1mm3 resolution, and the four sequences of the same patient had been co-registered. The ground truth was obtained by manual segmentation results given by experts;

2) *for the liver* - the dataset 3DIRCAD is used in the article [1]. This dataset is composed of 3D CT-scans of women and men with hepatic tumors in 75% of cases. For each patient, there are several images in DICOM format, the labeled image corresponding to the various zones of interest segmented in DICOM format, a new set of images corresponding to the names of the various segmented zones of interest containing the DICOM image of each mask, and finally, all the files corresponding to surface meshes of the various zones of interest in VTK format.

3.2.1.3. Results

To evaluate the performance of the algorithms, the researchers choose to use the Dice Similarity Coefficient. This is a statistical method used to determine the similarity between two samples. Namely, in the case of neural networks, it will quantify how closely the results of the algorithm matched the training dataset's hand-annotated ground truth segmentation.

In [8], the coefficient was calculated for three different levels of medical imaging, with enhanced tumor core, whole tumor, and tumor core. The proposed method achieves an average Dice score of 82% . A similar Dice score was achieved in [4].

At [1] an average of 93.1% is achieved for the Dice score. This score is compared with the results of the reference network UNet [6], underlining the significant improvements in performance, with an increase of 20.2% .

3.2.1.4. Tools

Some used tools are **Tensorflow2** and **NiftyNet** (in [8]) and **Pylearn2** library (in [4]).

3.2.2. Cellular Automata

Cellular automata (CA) were introduced to provide a formal framework for investigating the behavior of a dynamic complex system in which time and space are discrete. They include an array of cells, where each cell can be in one of a finite number of possible states, which is updated synchronously in discrete time steps according to local transition rules. The state of a cell at the next time step is determined by its neighboring cell's current state.

3.2.2.1. Algorithm

The algorithm presented in [7] is based on a deterministic cellular automaton that is a dynamic model represented by an array of cells that evolve through a succession of states t, in the space of an N-dimensional image. At each evolution step, the function that determines the next state of the current cell (based on the states of the neighboring cells) is applied simultaneously in all the cells of the automaton. After a set of seed cells (a set of cells with label and strength) have been defined in a supervised or unsupervised manner, the automaton begins its evolution.

The used segmentation seeds are chosen from the ground truth with a uniform probability distribution. Seed indices, for each dataset, are the same in every automaton variation.

3.2.2.2. Data

The data used in the article [7] for the test are BraTS (Multimodal Brain Tumor Segmentation Challenge) data sets (previously presented in 3.2.1). The difference is, however, the fact that the data was modified before the actual use by adding a new class representing healthy brain tissue. The purpose of these changes was to avoid false-positive results.

3.2.2.3. Results

For the evaluation of the results, the same Dice coefficient was used (also described in 3.2.1). The proposed method achieves an average Dice score of 94.97% for simulation data and 92.57% for clinical data. The method proposed in this paper achieves 5-10% better results than the standard methods.

3.2.2.4. Tools

Parallel Computing and Image Processing toolboxes were used to develop the algorithm in [7]. The algorithm is running on GPU and was implemented in Matlab using CUDA and CUDA kernels.

4. Application Requirements

The main functionalities of the developed application are described below:

1) The user should be able to upload the medical image of a heart, which the system will then display;

2) The system should display a 2D plot for each of the image's axis (axial, coronal, and saggital), and the user should be able to scroll along any of the axis;

3) The system should be able to generate labels for an uploaded medical image; the generated labels will depict the ventricular myocardium, blood pool, and the background;

4) The user should be able to view the original medical image with the generated labels overlayed, having control over the opacity of the labels as a whole;

5) The user should be able to view a rendered 3D view of the generated labels with the possibility of interacting with it;

6) The user should be able to save both the generated labels and the 3D rendered view to a path of his choice.

5. The U-Net Architecture

U-Net is a neural network built upon a fully convolutional network. The FCN architecture is modified and expanded in such a way that the resulting network will be superior to the previous ones. The main idea behind U-Net is that, unlike a simple CNN, after converting an image into a vector, a segmented image is later "constructed" from the vector, based on the feature mappings already learned [5]. This is the network type that we are going to use.
The name of the network is very descriptive considering its architecture. The U-Net consists of two main parts: a **contracting** and an **expansive** one, the latter one being somewhat symmetric to the contracting one (hence the U-shaped architecture) [6]. This architecture also has a cascading structure, which we mentioned in section 3.2.1, meaning that the output of one layer will be input for the next layer.
Below we briefly describe the two parts of the U-Net:

1) The first part of the architecture acts as an **encoder** and it encodes the input image into feature representations at multiple levels. This contracting part is composed of several **contraction blocks**. Every block applies to the input two 3x3 regular convolutions, each followed by a rectified linear unit (ReLU) and a 2x2 max pooling operation with stride 2 for downsampling [6]. The number of feature channels is doubled with each contraction, while the size of the image is halved. This means that the network learns the “what” information in the image, although it loses the “where” information [3];

2) The second part of the network can be considered a **decoder** that, starting from the feature map, builds the segmented image. This part is also made up of several blocks, this time called **expansion blocks**. For each block, the following are applied: an upsampling of the feature map followed by a 2x2 convolution (“up-convolution”), a concatenation with the cropped feature map from the corresponding contraction block, and two 3x3 convolutions, each followed by a ReLU [6]. This time, with each block processed, the number of feature channels is halved, and the image size is doubled. By doing so, the decoder recovers the “where” information [3].

Similarly, an U-Net model that works with 3D data instead of 2D can be described by simply replacing the 2D convolutions and max pooling operations with their 3D equivalents.

6. Algorithm Implementation

For the implementation, we used the *Keras* library, a high-level neural networks API, with *TensorFlow* backend. The architecture of the neural network implemented in the algorithm is in accordance with the one described in the previous section. Our implementation was designed to offer some level of configurability, meaning that the input/output sizes and the segmentation classes are to be passed as arguments to the model's constructor.
We have developed and compared two different models, both based on the U-Net architecture: a 2D U-Net model, which uses 2D convolutional layers, and a 3D U-Net model, which uses 3D convolutional layers. The input and output of the 2D model are 2D medical images, or "slices" of 3D volumetric medical images. On the other hand, the 3D model uses full volumetric images as input and output. These models are described in sections 6.2. and 6.3., while in 6.1. we explain some general aspects of the implementation, which are valid for both models.
In *Appendix 2* can be seen the 3D rendered labels for the ground truth, result from the 2D U-Net model and two results from two different implementations of the 3D U-Net model, all for the same initial medical imaging of the heart.

6.1. General Implementation Details

The training of the neural network is done in epochs, the whole dataset being used in each epoch. The default number of epochs in our implementation is 50 unless otherwise specified by the method caller. After each epoch, a list of callbacks is applied. The callbacks used are **ModelCheckpoint** (saves the model after each epoch), **EarlyStopping** (stops the training when a monitored quantity has stopped improving) and, optionally, **TensorBoard** (saves various data about the model, and all the monitored metrics after each training epoch, which can be later visualized on a locally hosted server).
In the training process, we have specified some metrics that will be evaluated by the model during training and testing. We use the default implementations of the *loss* function (*categorical_crossentropy*) and of the *accuracy*, provided by Keras.
The other metrics monitored by our implementation are:

1) *Dice similarity Coefficient* - will quantify how closely the results of the algorithm matched the training dataset's hand-annotated ground truth segmentation;

2) *Intersection over Union* - will divide the area of overlap (between the predicted bounding box and the ground-truth bounding box) by the area of union (the area encompassed by both the predicted bounding box and the ground-truth bounding box). In the literature this metric is also known as Jaccard index.

The metric followed by us during model training and validation is the IoU coefficient, because it is a metric that rewards predicted areas for heavily overlapping with the ground-truth. Because Keras aims to minimize a loss function during training, but an IoU value is better as it gets closer to 1, we defined our IoU loss function as being "1 - IoU" , so that it will converge towards 0.
Finally, we implemented a method that will be used to generate masks based on a newly introduced input. This method takes as input a 3D image, does all the preprocessing needed (resizing and color-depth changes), feeds it into the model, and then processes

the model's output to generate a labeled 3D image of the same size as the input. The model's output is, for each input element (pixel/voxel), a tuple of probabilities - one for each segmentation class; the class with the highest probability sets the output element's label.

6.2. The 2D U-Net model

6.2.1. Training data acquisition

The medical images that our application will support, and the medical data our dataset consists of, are stored in the NIfTI (*Neuroimaging Informatics Technology Initiative*) format. NIfTI images have 7 dimensions, of which the first 3 are reserved to define the three spatial dimensions, x, y, and z.
Because this model uses 2D images as input and output, we need to be able to take out 2D "slices" out of the volumetric data, and combine 2D images into a single 3D one. We can do this by fixing one of the three axis of the NIfTI image, which yields a 2D image called "slice", or by stitching all the consecutive 2D slices along one axis, to generate a 3D image.
Because each learning, verification, and validation image is used multiple times during the network's training, and because extracting slices from a volumetric image and resizing them to fit the input/output dimensions takes some noticeable computational effort, we implemented a class (*NIfTI2DPreprocessor*) specialized in batch preprocessing of NIfTI images. This class offers a function that can read more NIfTI images, and then export 2D images that can be used directly as inputs or outputs for the model. The resulting images will be saved on the disk, to speed up the training process by eliminating the need for preprocessing of the entire training data set every time the network is trained.

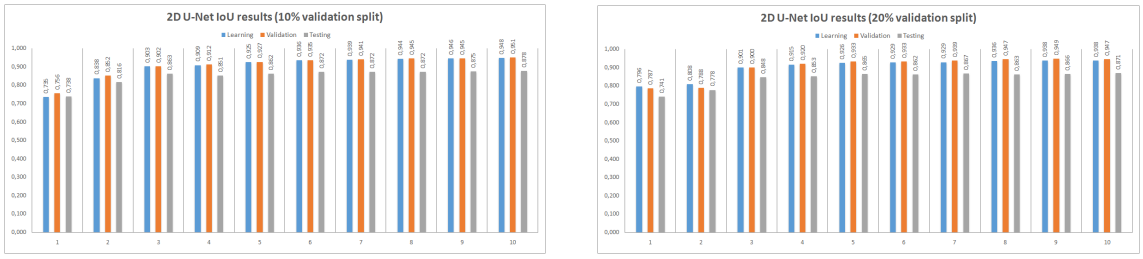
6.2.2. Model architecture

The general details about this implementation are the same as the ones described in section 6.1. The number of features per convolutional layer is [16, 32, 64, 128, 256, 128, 64, 32, 16], and the input/output image size is 128x128 RGB pixels.

6.2.3. Results

The used dataset consists of 10 NIfTI images of varying dimensions, from which 9 were used for training, and 1 for testing. Slices from all the three axis (axial, coronal, and saggital) were fed into the same model during training and testing, resulting in a preprocessed dataset of 4106 training images and 435 testing images.
To assess the performance of the image segmentation algorithm, we use the results generated by the evaluation of the metrics specified in the previous section. We compared two different validation split values: 90% -10% , and 80% -20% , while using the same training and testing dataset. The first validation split value resulted in 3695 learning images and 411 validation images. By analyzing the outcomes of these metrics, we can see that the IoU coefficient reached, after 10 epochs, a value of **0.94-0.95 for the learning and validation** data. However, the IoU coefficient on the **testing data only reached around 0.87.** The described results and their evolution through each epoch can be seen in *Fig. 6.2.1. (a)*.
For a better understanding of the impact of the data splitting on the learning process,

we also analysed the output of the previous metrics on a distinct data distribution. The 80-20 validation split value resulted in 3284 learning images and 822 validation images. In this case, the IoU coefficient reached, after 10 epochs, a value of **0.93-0.94 for the learning and validation** data, which is ~ 0.01 less than the other split. However, the IoU coefficient on the **testing data reached the same value as the 90-10 split: 0.87.** The described results and their evolution through each epoch can be seen in *Fig. 6.2.1. (b)*. The charts corresponding to the Dice coefficient can be seen in *Appendix 1.*



1) *(a) with 90-10 splitting* *(b) with 80-20 splitting*

Fig. 6.2.1. IoU value evolution over 10 epochs, on the 2D model

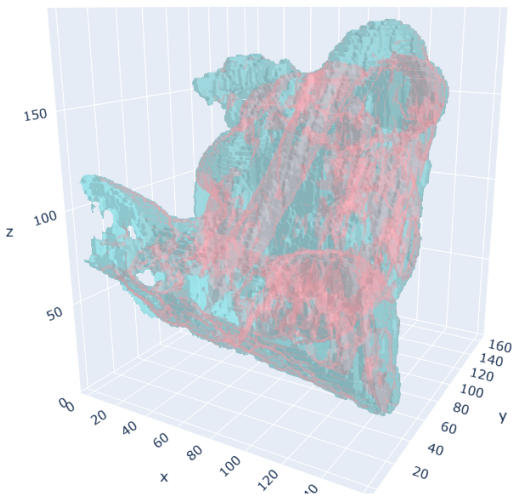


Fig. 6.2.2. 3D render of a labeled image generated by the 2D model

6.3. The 3D U-Net model

6.3.1. Training data acquisition

This model uses 3D images as input and output; we don't need to do any preprocessing besides resizing the data to the neural network's input/output size. Just as for the 2D model, preprocessing the learning, verification, and validation data each time they are needed in the training process eats up a lot of computational resources. We implemented a class (NIfTI3DPreprocessor) specialized in batch preprocessing of NIfTI images. This

function can read more NIfTI images, and then export resized NIfTI images that can be used directly as inputs or outputs for the model. The resulting images will be saved on the disk, to speed up the training process by eliminating the need for preprocessing of the entire training data set every time the network is trained.

6.3.2. Model architecture

The general details about this implementation are the same as the ones described in section 6.1. The number of features per convolutional layer is [16, 32, 64, 128, 256, 128, 64, 32, 16], and the input/output image size is 64x64x64 grayscale pixels. The total number of trainable parameters is 5,639,435.

6.3.3. Initial results

The same dataset as for the 2D model was used: 9 training images, and one test images. The training images were split into 8 learning images and 1 validation image. We were expecting better results than those of the 2D model, not necessarily as in metrics improvements, but as in the quality of the generated 3D labeled image. The improvements were expected due to the 3D model taking into account data from all three axis, and not only two (as in the case of the 2D model).
Training on the described dataset resulted in some disappointing results. The IoU value reached and capped, after 50 epochs, at **0.738 for the learning, 0.784 for validation**, and **0.825 for testing data.** The described results and their evolution through each epoch can be seen in *Fig. 6.3.*1. The charts corresponding to the Dice coefficient can be seen in *Appendix 1.*

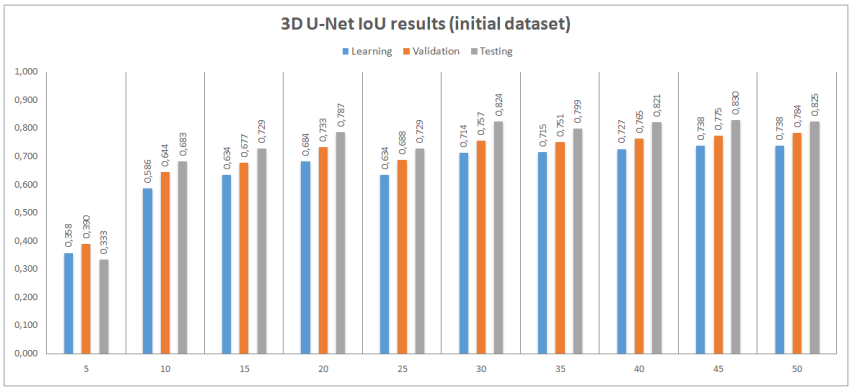


Fig. 6.3.1. IoU value evolution over 50 epochs, on the 3D model, with the initial dataset

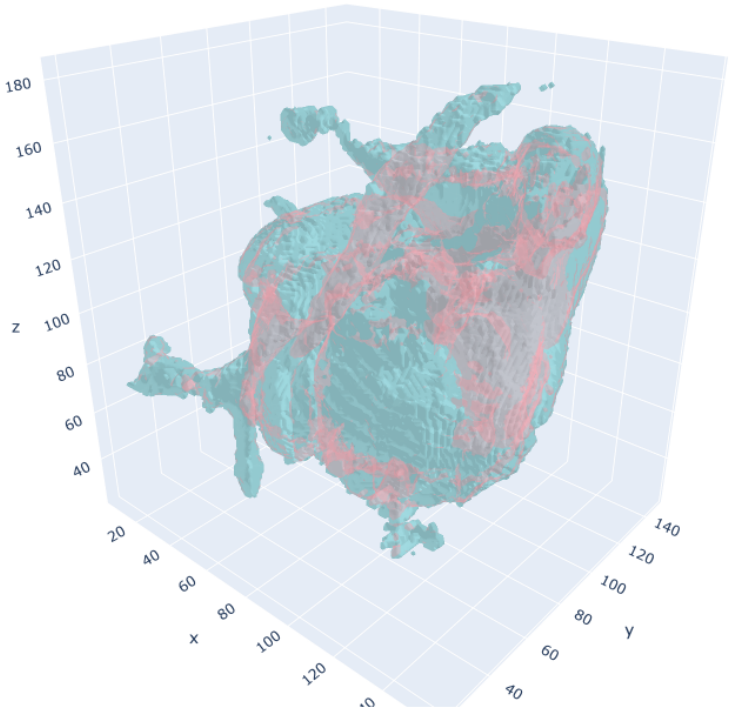


Fig. 6.3.2. 3D render of a labeled image generated by the initial 3D model; Generated for the same medical imaging as in Fig. 6.2.3.

6.3.4. Improvements and final results

The poor results were thought to be caused by the very small dataset, so data augmentation was taken into consideration. The initial dataset was based on images cropped out of 10 MRI images of the upper-half of the torso. Using these larger images, we were able to generate 55 NIfTI images, which were divided as follows: 49 in the training dataset, and 6 in the testing dataset. A validation split of 0.125 was used during training.

Because the new training set was much larger, it could not be used for training because it did not fit in the memory. This led to the creation of a **Data Generator** class, which is used to load and provide the training and testing data in batches rather than as a whole. This way, we need far less memory, and the training can run even on potato-powered computers.

Training went on for 100 epochs trying to minimize the IoU loss function, and the metrics resulted during the training can be seen in Fig. 6.3.2. The evolution of the Dice coefficient can be followed in *Appendix 1.* It can be clearly seen that the metrics for learning, validation and testing are greatly superior, reaching **0.94 for learning and over 0.90 for testing.**

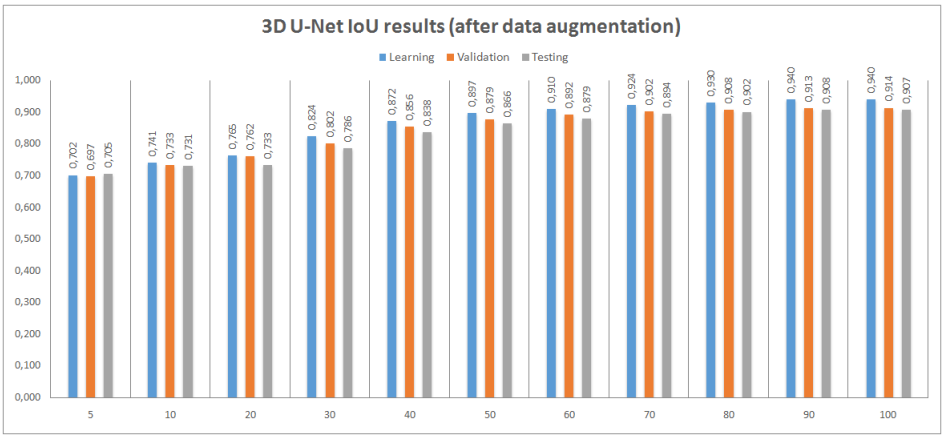


Fig. 6.3.2. IoU value evolution over 100 epochs, on the 3D model, with the augmented dataset

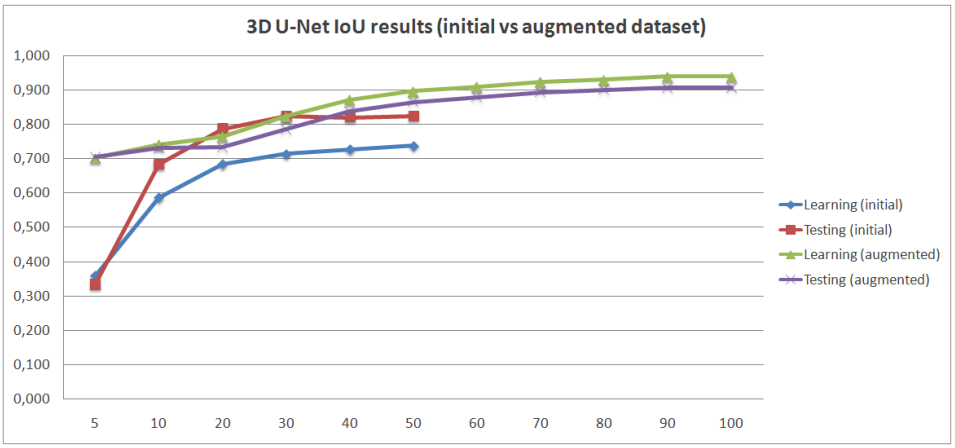


Fig. 6.3.3. Comparison between the IoU metric on the initial vs augmented dataset

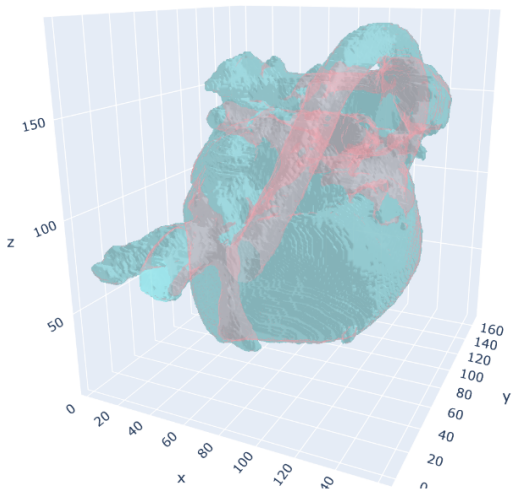


Fig. 6.3.4. 3D render of a labeled image generated by the improved 3D model; Generated for the same medical imaging as in Fig. 6.2.3.

6.4. Methods comparison and conclusions

At first glance, one would say that the 2D model performed the best, given its good metrics on both training and testing data. But the best indicator here is not the metrics, but the rendered 3D images of the generated segmentations. By comparing Figures 6.2.2, 6.3.2 and 6.3.4 with the original ground truth (Fig. 6.4.1), it is clear that the 3D model trained with augmented data performs the best and closest to the desired output.

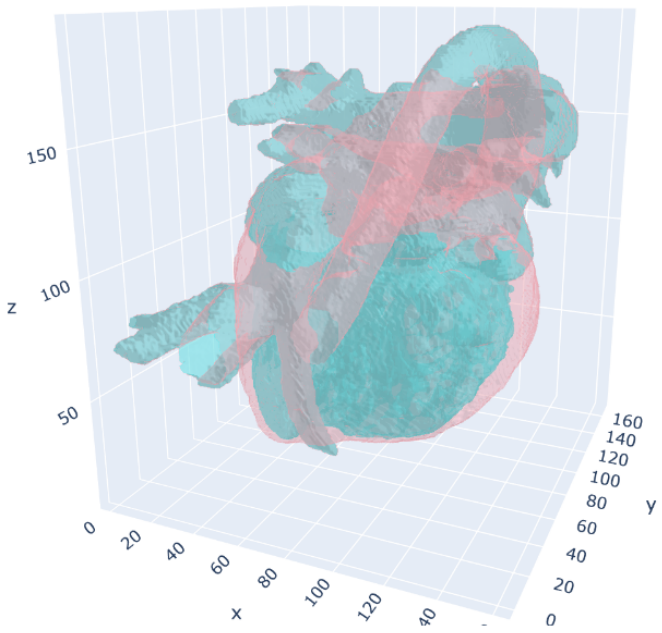


Fig. 6.4.1. 3D render of the ground truth of the previous images.

7. Application

7.1. Data

7.1.1. Data acquisition

The training and testing data were taken from an online database provided by the HVSMR 2016: MICCAI Workshop on Whole-Heart and Great Vessel Segmentation from 3D Cardiovascular MRI in Congenital Heart Disease. The 3D cardiovascular magnetic resonance (CMR) images were acquired during clinical practice at Boston Children's Hospital, Boston, MA, USA. Cases include a variety of congenital heart defects and even some post-surgery images.
Imaging was done in an axial view on a 1.5T scanner (Phillips Achieva) without contrast agent using a steady-state free precession (SSFP) pulse sequence. Subjects breathed freely during the scan, and ECG and respiratory-navigator gating were used to remove cardiac and respiratory motion (TR = 3.4 ms, TE = 1.7 ms, $\alpha = 60$). Image dimension and image spacing varied across subjects, and average 390 x 390 x 165 and 0.9 x 0.9 x 0.85 mm, respectively, in the full-volume training dataset.

7.1.2. Storing the data

The training of the model takes place before it is delivered to the customer, therefore the training dataset is stored locally on the development servers. The client receives an application comprising of the already trained model, hence there is no need to store the training data onto the client's devices. However, the data representing the input of the application, namely the medical images that need to be analyzed, is stored onto the client's devices.

7.1.3. Data visualization

The NIfTI images are displayed using ***Pyplot*** from the Python 2D plotting library, *Matplotlib*. *Pyplot* is mainly intended for interactive plots, and in the case of our application, the interaction consists of successively displaying NIfTI slices according to the user's commands. Furthermore, the plot shows the axial, coronal and sagittal planes of the NIfTI image.

The labels generated by the application are also saved as NIfTI, therefore *Pyplot* is also used for their display. The generated labels will be represented within the same plot with the NIfTI used for the generation. The opacity of the labels can also be set by the user. These two things make it very easy to identify the Ventricular Myocardium and the Blood Pool in the analyzed imaging. In the *Fig. 6.1.* can be seen a 2D plot of a heart and the generated labels.

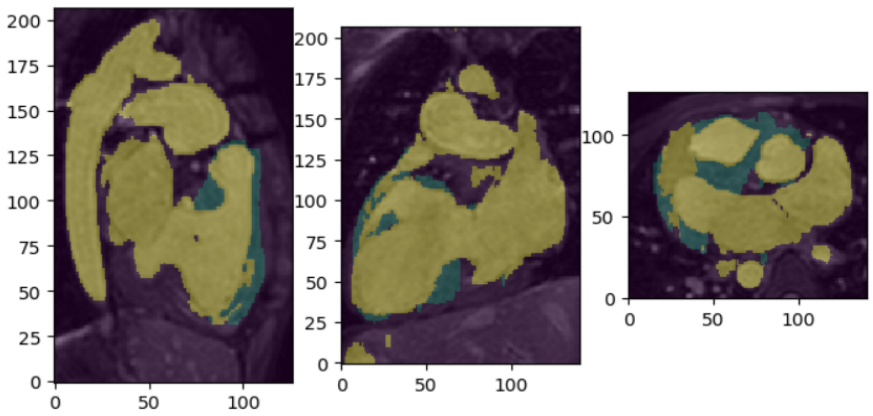


Fig. 7.1. Generated labels on original NIfTI image;
Yellow - Ventricular Myocardium; Green - Blood Pool

In addition to the labels saved as NIfTI, the application also renders the result so that a 3D representation of the analyzed image can be obtained. This is possible using the ***scikit-image*** implementation of the Lewiner marching cubes algorithm (that finds surfaces in 3D volumetric data) along with another visualization library, ***Plotly***. The result of the 3D rendering is saved as an HTML file that contains an interactive plot. In *Fig. 6.2.* can be seen the 3D plot generated using the ground truth label for the medical imaging used to generate the 3D models from figures *6.2.2 and 6.3.4.*

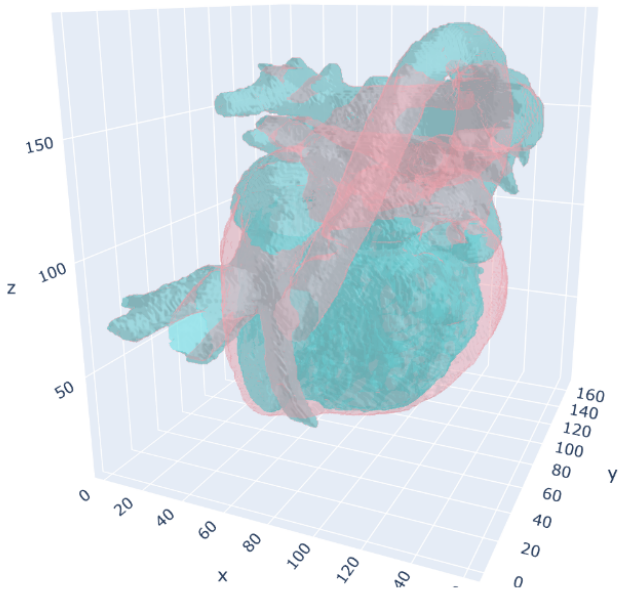


Fig. 7.2. 3D rendered labels;
Green - Ventricular Myocardium; Red - Blood Pool

7.2. Pipeline

7.2.1. Versioning tools

In terms of versioning tools, we decided to rely on the general ones, namely on GitHub. We chose this tool because it meets our project requirements, granting an easy source code management (SCM) and a distributed version control. It provides access control and several collaboration features such as bug tracking, feature requests, task management, and wikis for every project. Even if we examined multiple data configurations when training our model, we decided to keep the dataset that generated the best results, instead of retaining multiple ones. Therefore, there is no need to use a data versioning tool, employed to handle multiple data configurations, managed in the application.

7.2.2. Architecture

The system is separated into multiple independent components that collaborate to produce the desired results. Therefore, the application is highly cohesive due to its clear separation of responsibilities and grouping of related methods in highly reusable and less complex modules. Together with a high cohesion, the application boasts a low coupling between modules, which enables an easy and facile refactorization.

The architecture is a layered one that contains the following layers:

1) ***AI*** *layer* which contains the two implementations of the neural network (2D U-Net model and the 3D U-Net model)

2) ***service*** *layer* that contains all the logic that is required by the application to meet its functional requirements; this is the only layer that will directly interact with the *AI layer*; this layer also includes the 3D renderer;

3) ***presentation*** *layer* that contains all of the classes responsible for presenting the GUI (graphical user interface)

4) ***utils*** *layer* that only contains helper functions, used by all of the other layers.

The main benefit of this architecture is its *flexibility*. Due to the decoupling between the UI and the service, it is very easy to transform the application into a server-client application. This can be achieved by creating a REST API to expose the functionality of the service and creating a web client to access it.

7.2.3 SW Pipeline

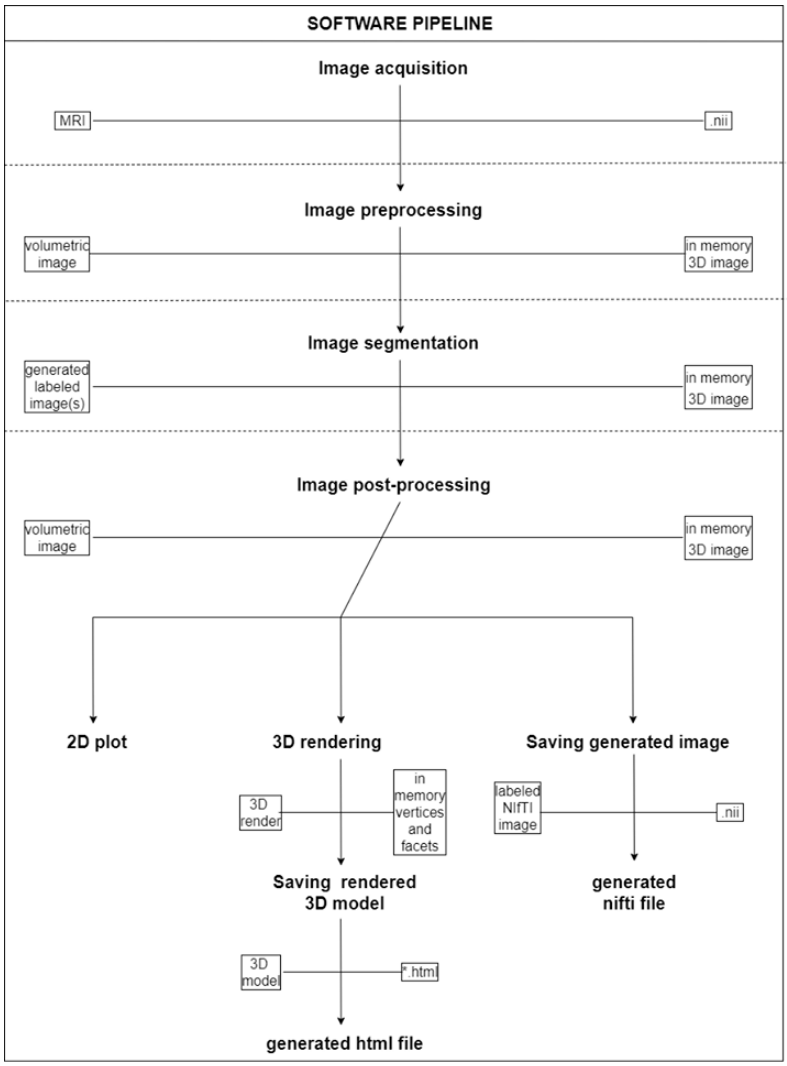


Fig. 7.3. SW Pipeline Diagram

This division of the requirements and the steps we had to follow helped us to have a clear view of the software concepts. We had to test the given model with small input data. After that, we saw how the model works and what we should improve to have better results regarding our domain's input data. We had to develop the application in small and achievable steps. The software process was not impacted by the changes from the intelligent algorithm because we used basic design principles which allowed us to decouple the intelligent system from the rest of the application.

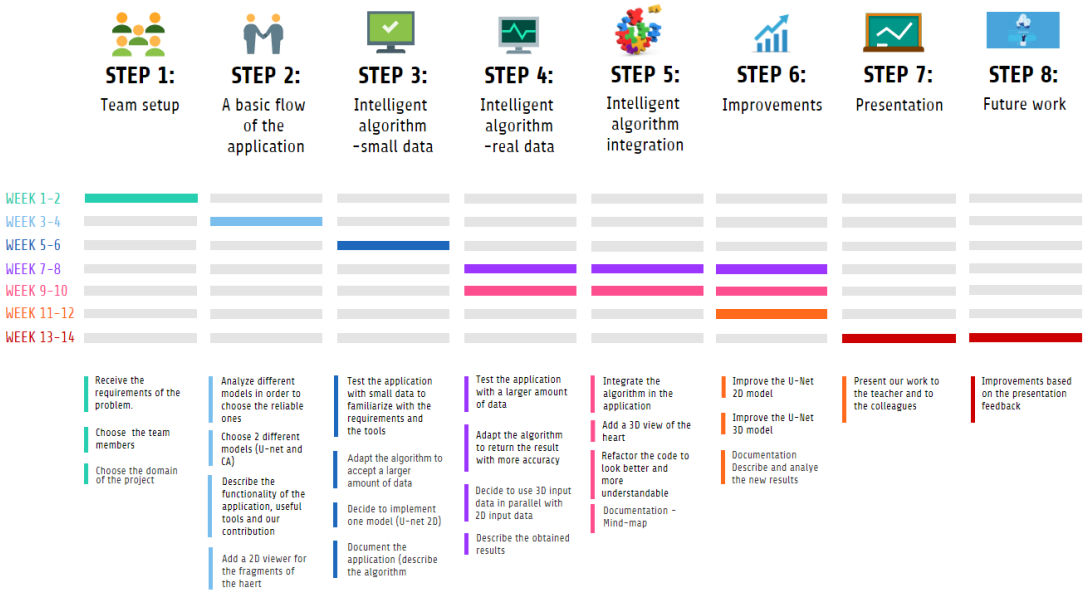


Fig. 7.4. The software development process

7.2.4. SW Code

For the implementation of the intelligent algorithm, we used the *Keras* library, a high-level neural networks API, with *TensorFlow* backend.

For debugging the application, we use the debugger provided by the IDE. Therefore, to identify the issues, we would execute the code in debugging mode, namely running the code step by step and analyzing the values of the variables. Furthermore, the messages printed in the console start with *INFO*, *WARN* and *ERROR*. As further improvements, at least for *WARN* and *ERROR*, special dialogs can be added to inform the user of the occurrence of these events.

The testing was done manually by each one of us. We kept a running list of known issues/bugs of all the problems discovered. Although not all of these were addressed, this helped us keep track of the current state of the application.

7.2.5. Model deployment

The current application does not offer any means of deployment. The application and the intelligent model are delivered to the user as-it-is, and further updates are to be done manually by the user by replacing the old application with its new version, provided by the developer.

8. Philosophical aspects

8.1. Social impact

The application can be seen as having a double purpose: to help medical students learn how to identify and distinguish the heart and its blood pool from the background and also to assist the radiologists to read the x-rays and MRIs.

In the academic field, the application and its results can assist the students and resident doctors, improving the learning process by providing relevant visual information, extracted from the RMI images. Due to the possibility of 3D data visualization, users can explore the organ from various angles and depths, therefore engaging in active and interactive training. This method supports the interest of the students and enhances the visual learning.

As the demand for time-consuming imaging services keeps on growing, radiology departments are increasingly vulnerable to staff shortages. Therefore, in the practical field, namely in radiology, this application can help improve the accuracy of imaging reads and extend the time providers have to spend with patients.

False-positive and false-negative image reads can significantly impact patients, both emotionally, physically and financially. Repeated and unnecessary tests can be emotionally draining and medical bills from repeated examinations can stack up, creating substantial cost burdens for patients. Artificial intelligence has repeatedly demonstrated lower false positive and false negative read rates compared to radiologists. Therefore, the complementary use of image processing in the radiology interpretations can improve the accuracy and efficiency of the examination process by providing the specialist with a detailed 3D visualization of the organ.

8.2. Ethics in medical image processing

The ethical aspects related to the use of artificial intelligence in radiology can be divided into several categories, namely: ethics of data, of algorithm and trained models, and ethics of practice.

The ethics of data are fundamental to AI in radiology and reflect trust in acquiring, handling and evaluating data. Key areas of data ethics include informed consent, privacy and data protection, ownership, objectivity, and transparency [gbwc2019]. The training and testing data for our application were taken from an online database provided by the HVSMR 2016: MICCAI Workshop on Whole-Heart and Great Vessel Segmentation from 3D Cardiovascular MRI in Congenital Heart Disease. The data used in the project consisted of human medical imaging data and their corresponding meta-information. All data used in the application are anonymized and comply with the ethical standards presented above.

Human decision making is influenced by a person's knowledge, values, preferences, and beliefs. For supervised learning, the algorithm chooses alternatives based on prior training to match labels to data features. Within these labels, human values and preferences may be transferred to the model. This is where human bias may manifest. To increase the specialists and patients level of trust in the accuracy and precision of the algorithm, the

decision process must be highly transparent. They must understand the steps that the application takes in order to compute the results. However, increased transparency comes with several potential disadvantages, such as security threats and intellectual property theft. The trained model must be evaluated before it is used in practice, in order to ensure compliance with the effectiveness and security standards [gbwc2019]. We tested our application by both numerical and visual means, namely, by the use of various metrics (presented in sections 5.4 and 5.5) and by visually comparing the results with the ground truth.

The integration of artificial intelligence into the medical field generated the automation bias phenomenon, which is the tendency for humans to prefer machine-generated decisions, ignoring contradictory data or opposing human judgments [gbwc2019]. The target area of our application is the academic field, however, it can be extended to the practical field, namely to radiology. Its main goal is to be used as a complementary and automated tool in the medical image interpretation, not as a substitute for the specialists.

8.3. Fairness of the intelligent algorithm

Fairness in data and machine learning algorithms is crucial to developing reliable, effective and efficient artificial intelligence systems. As accuracy is a metric for measuring the correctness and precision of a machine learning model, the fairness analyzes the practical implications of using the model in real-life situations. One of the main problems that occurs in machine learning is the bias or preference phenomenon. Fairness is, therefore, the process of understanding and identifying bias introduced by data, and ensuring that the model provides equitable and impartial predictions. This method is particularly important when AI is used in critical processes, like medical diagnosis, that affect a broad spectrum of users.

As stated in the previous section, the human's decision-making process is influenced by a person's knowledge, experience, values, preferences, and beliefs. These features can be easily transferred to the artificial intelligence model as data bias. For example, if the training data provided to the supervised learning algorithm is based on the evaluation and interpretation of a single specialist or a single doctor, then his experience and personal knowledge will influence or even will be transferred to the model. In the development process of the model, there are multiple stages where a fairness analysis can be performed. Often, dataset imbalances, such as single doctor diagnosis, narrow range of ages, typical instead of various cases, can favour the occurrence of biased predictions.

During the training and the evaluation of the model, we need to make sure that it is training all the groups fairly, without manifesting certain preferences for specific groups. For example, when identifying the chambers of the heart and the blood pool, the model must not perform better in identifying the chambers in comparison with the blood pool or the other way around.

As stated in the previous section, the integration of artificial intelligence into the medical field known to generate the automation bias phenomenon, in which humans prefer machine-generated decisions without considering possible prediction errors. Therefore, the main advice is to use AI as a complimentary evaluation and interpretation system, without relying solely on model-based decisions.

References

[1] Christ, P. F., Elshaer, M. E. A., Ettlinger, F., Tatavarty, S., Bickel, M., Bilic, P., Remper, M., Armbruster, M., Hofmann, F., D'Anastasi, M.: *Automatic liver and lesion segmentation in CT using cascaded fully convolutional neural networks and 3D conditional random fields.*In: MICCAI, Vol. 9901, pp. 415–423 (2016)

[2] Dolz, J., Desrosiers, C., Ben Ayed, I. *3D fully convolutional networks for subcortical segmentation in MRI: A large-scale study.*In: NeuroImage. 170 (2017)

[3] Harshall Lamba, *Understanding Semantic Segmentation with UNET*, 2019, https://towardsdatascience.com/understanding-semantic-segmentation-with-unet-6be4f42d4b47

[4] Havaei, M.; Davy, A.; Warde-Farley, D.; Biard, A.; Courville, A.; Bengio, Y.; Larochelle, H.: Brain tumor segmentation with deep neural networks.In: MIA, 35, pp. 18–31 (2017)

[5] Heet Sankesara, *UNet - Introducing Symmetry in Segmentation*, 2019, https://towardsdatascience.com/u-net-b229b32b4a71

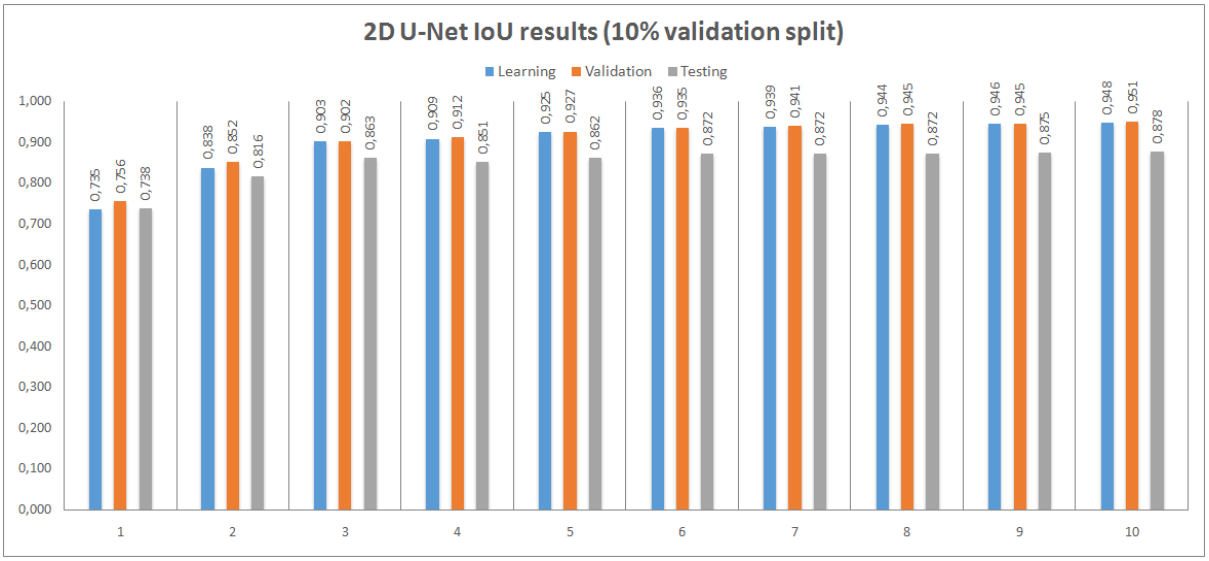
[6] Ronneberger, O., Fischer, P., Brox, T.: U-net: Convolutional networks for biomedical image segmentation.In: MICCAI, vol. 9351, pp. 234–241 (2015)

[7] Rueda-Toicen, A., Carmona, R., Martín-Landrove, M., Torres, W.: Evolution Rules of Deterministic Cellular Automata for Multichannel Segmentation of Brain Tumors in MRI (2014)

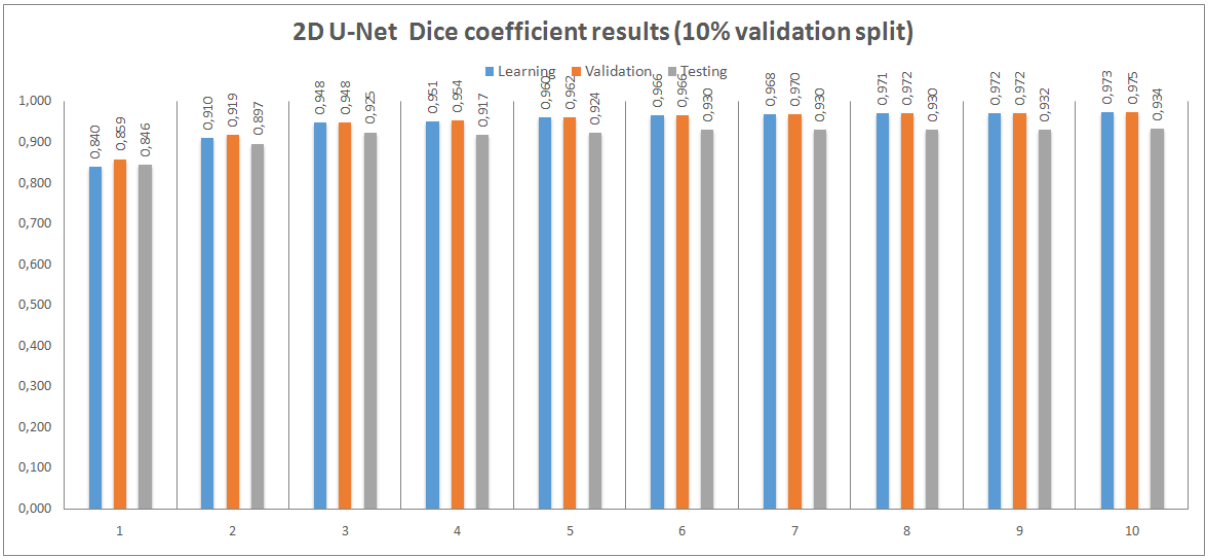
[8] Wang, G., Li W., Ourselin, S., Vercauteren, T.: Automatic Brain Tumor Segmentation using Cascaded Anisotropic Convolutional Neural Networks.9 2017

Appendix 1

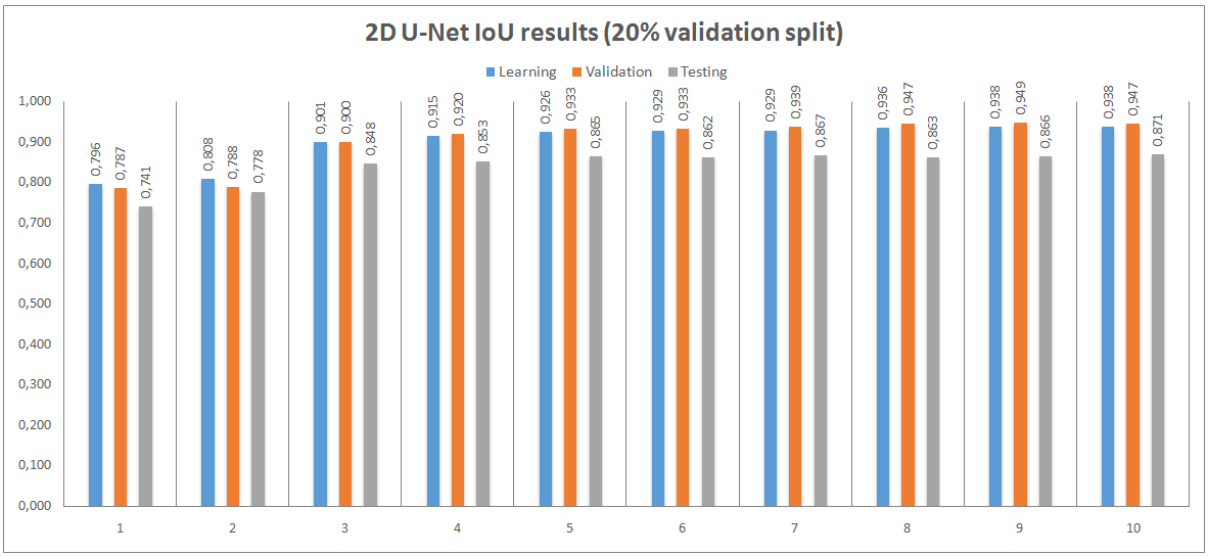
Metrics for various trainings



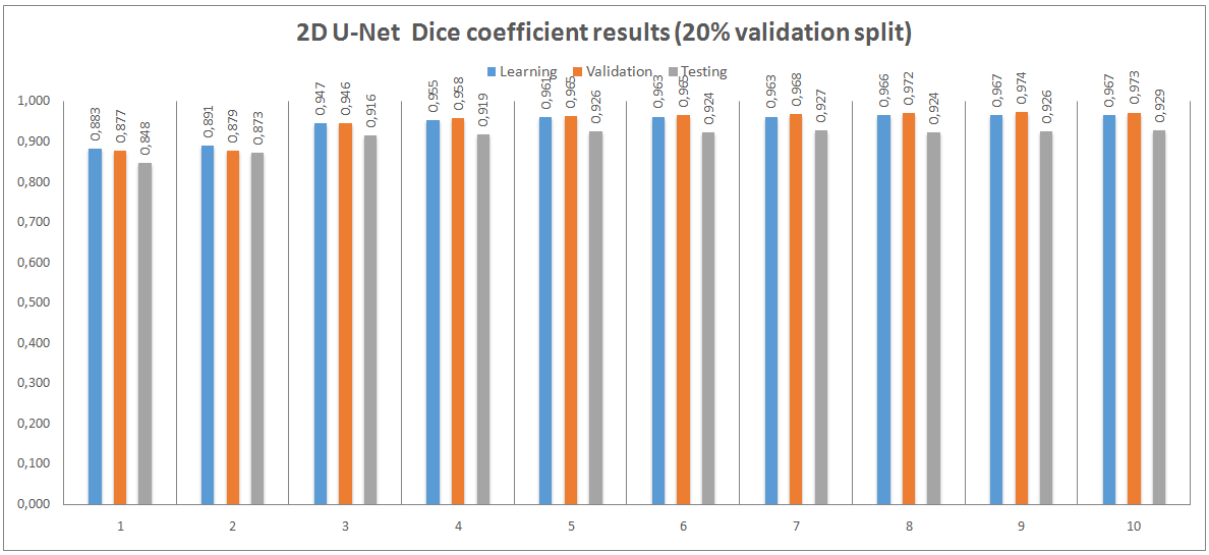
IoU evolution over ten epochs training on the 2D model (90-10 splitting)



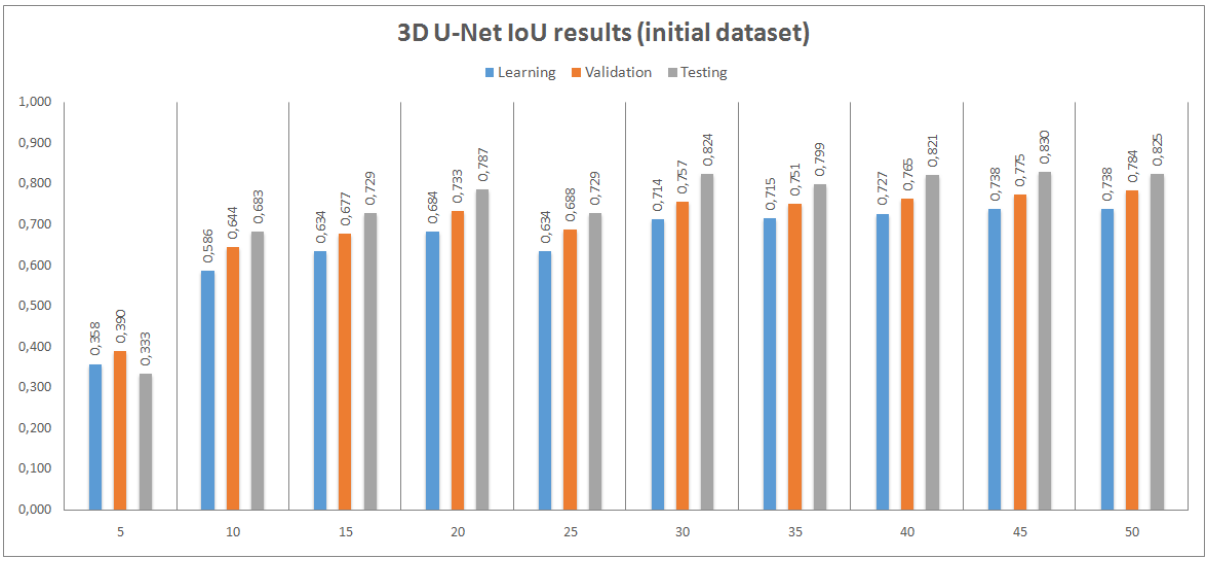
Dice coefficient evolution over ten epochs training on the 2D model (90-10 splitting)

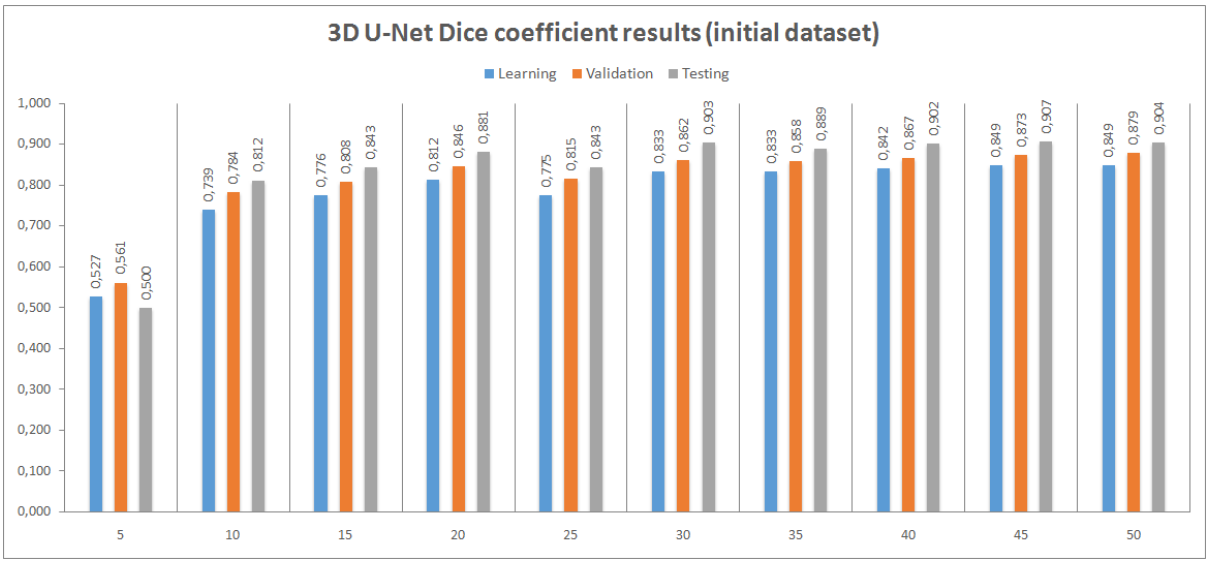


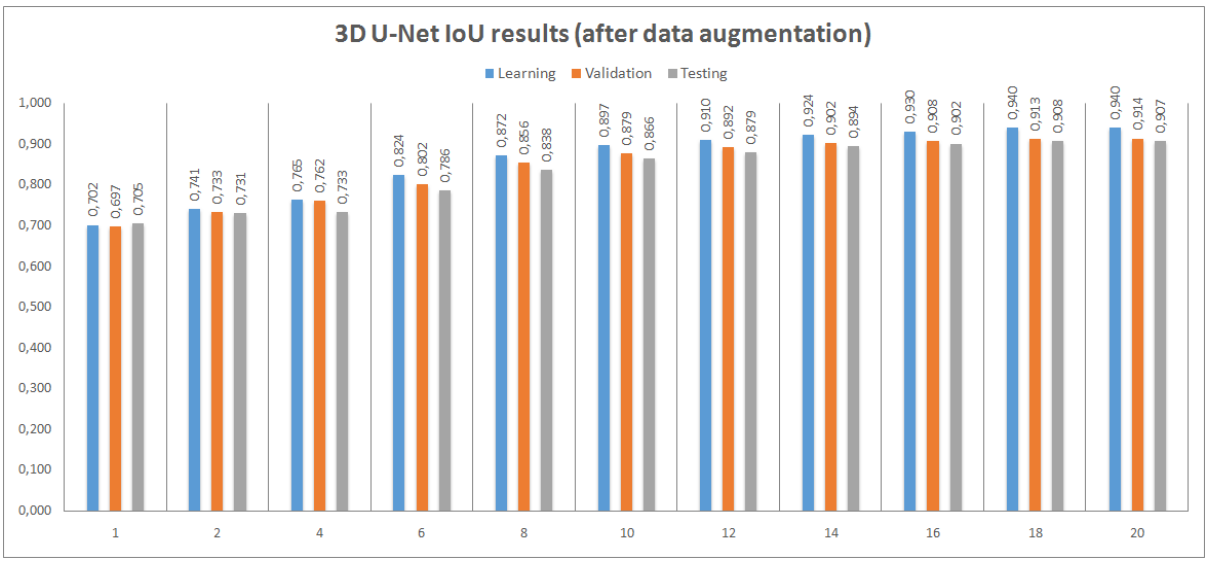
IoU evolution over ten epochs training on the 2D model (80-20 splitting)



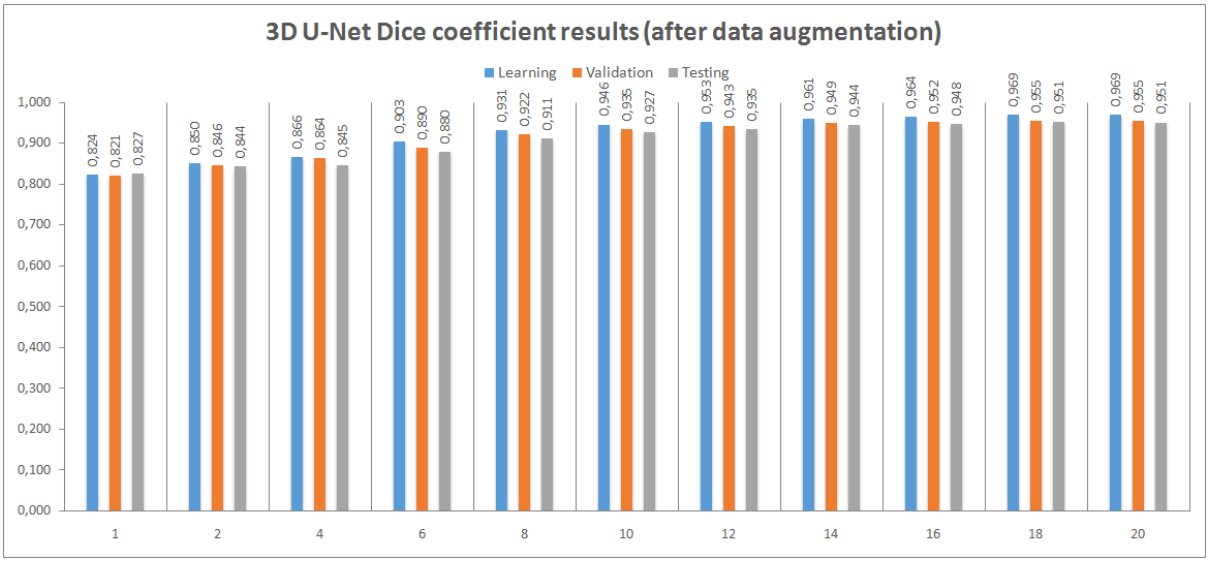
Dice coefficient evolution over ten epochs training on the 2D model (80-20 splitting)


3D U-Net IoU results (initial dataset)
Learning
Validation
Testing
1,000
0,900
0,800
0,700
0,600
0,500
0,400
0,300
0,200
0,100
0,000
0,358 0,390 0,333
0,586 0,644 0,683
0,634 0,677 0,729
0,684 0,733 0,787
0,634 0,688 0,729
0,714 0,757 0,824
0,715 0,751 0,799
0,727 0,765 0,821
0,738 0,775 0,830
0,738 0,784 0,825
5 10 15 20 25 30 35 40 45 50

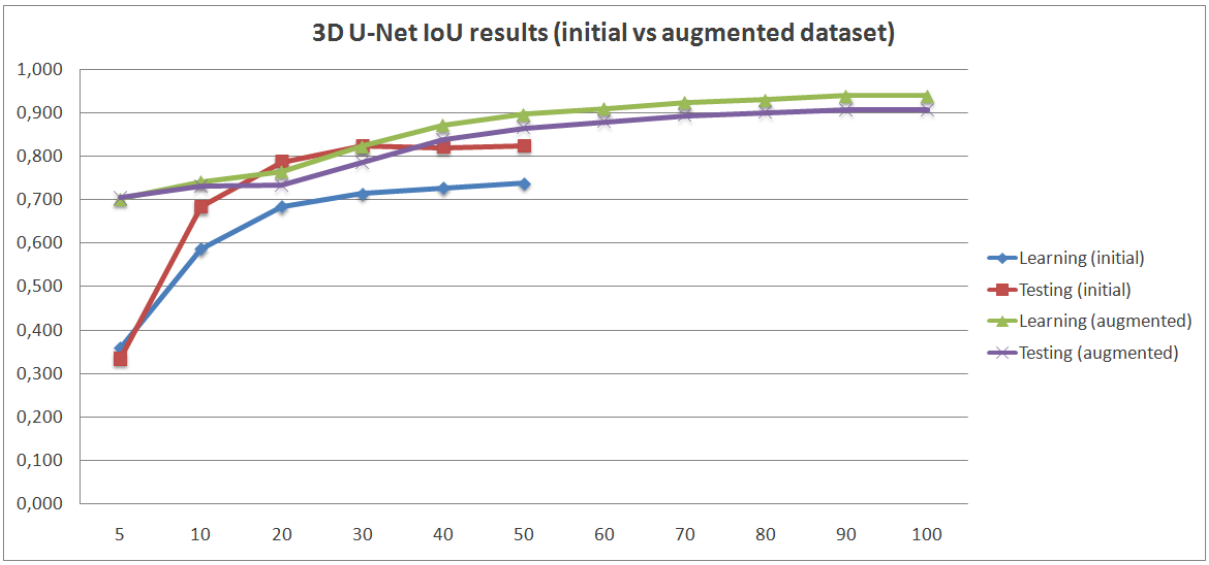

3D U-Net Dice coefficient results (initial dataset)
Learning
Validation
Testing
1,000
0,900
0,800
0,700
0,600
0,500
0,400
0,300
0,200
0,100
0,000
0,527 0,561 0,500
0,739 0,784 0,812
0,776 0,808 0,843
0,812 0,846 0,881
0,775 0,815 0,843
0,833 0,862 0,903
0,833 0,858 0,889
0,842 0,867 0,902
0,849 0,873 0,907
0,849 0,879 0,904
5 10 15 20 25 30 35 40 45 50


3D U-Net IoU results (after data augmentation)
Learning
Validation
Testing
1,000
0,900
0,800
0,700
0,600
0,500
0,400
0,300
0,200
0,100
0,000
0,702 0,697 0,705
0,741 0,733 0,731
0,765 0,762 0,733
0,824 0,802 0,786
0,872 0,856 0,838
0,897 0,879 0,866
0,910 0,892 0,879
0,924 0,902 0,894
0,930 0,908 0,902
0,940 0,913 0,908
0,940 0,914 0,907
1 2 4 6 8 10 12 14 16 18 20

IoU evolution over 20 epochs training on the 3D model (with data augmentation)



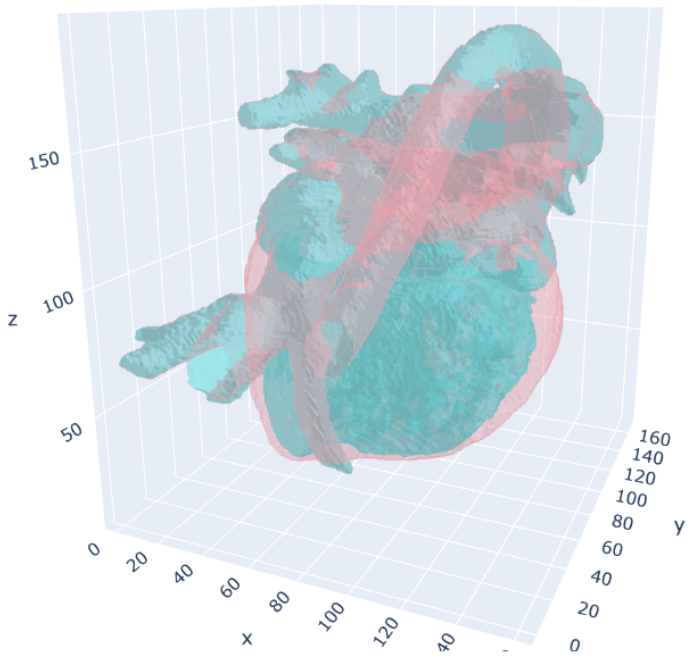
Dice coefficient evolution over 20 epochs training on the 3D model (with data augmentation)



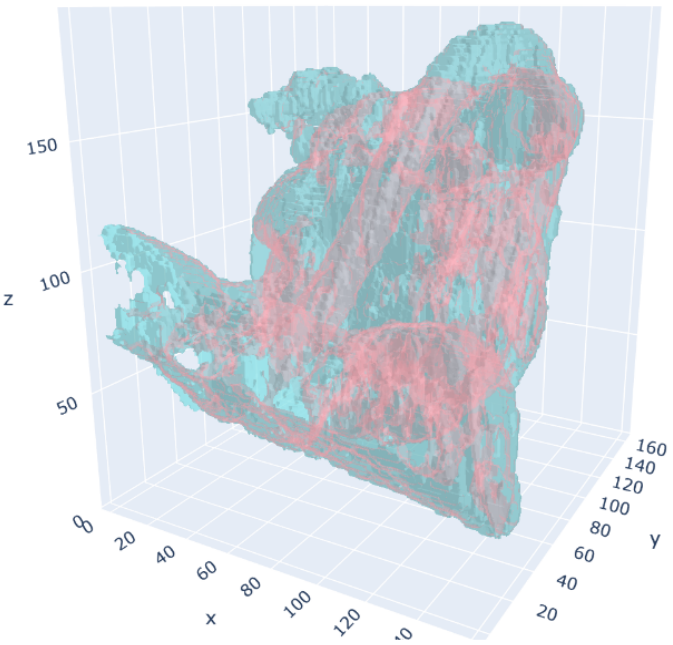
Comparison between the 3D model's IoU metric on the initial vs augmented dataset

Appendix 2

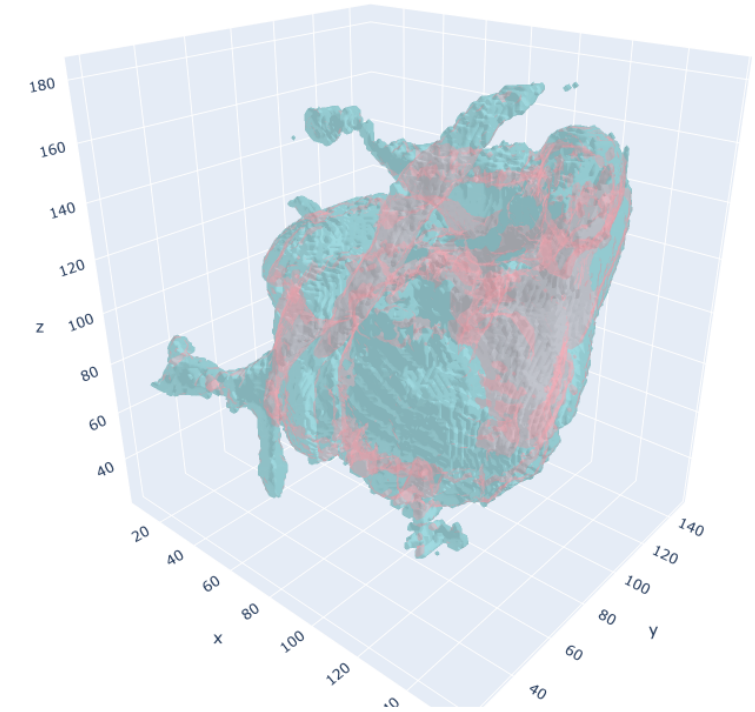
Comparison between the 3D rendered labels for the ground truth, 2D U-Net and 3D U-Net



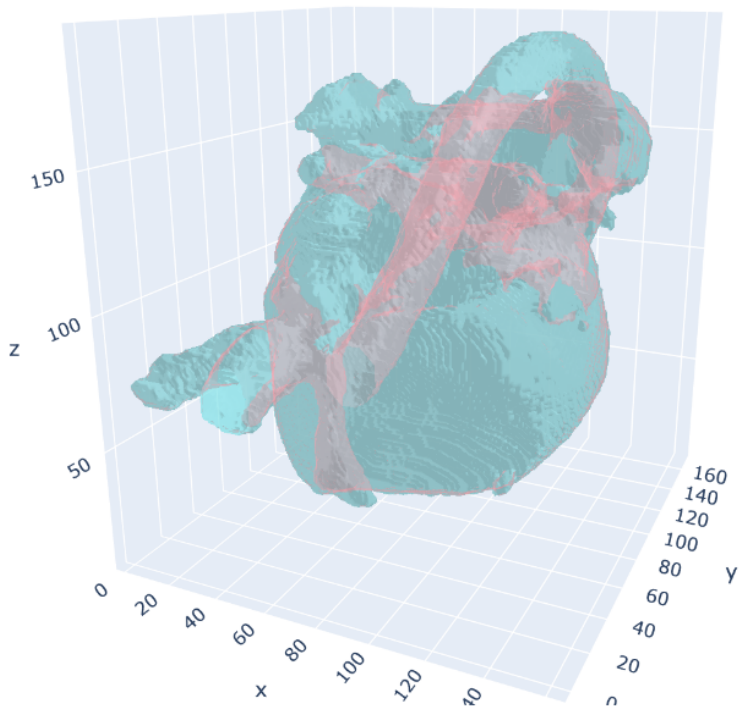
Ground truth for a medical imaging depicting a short-axis view of a heart



Generated labels for the previous heart using the 2D U-Net



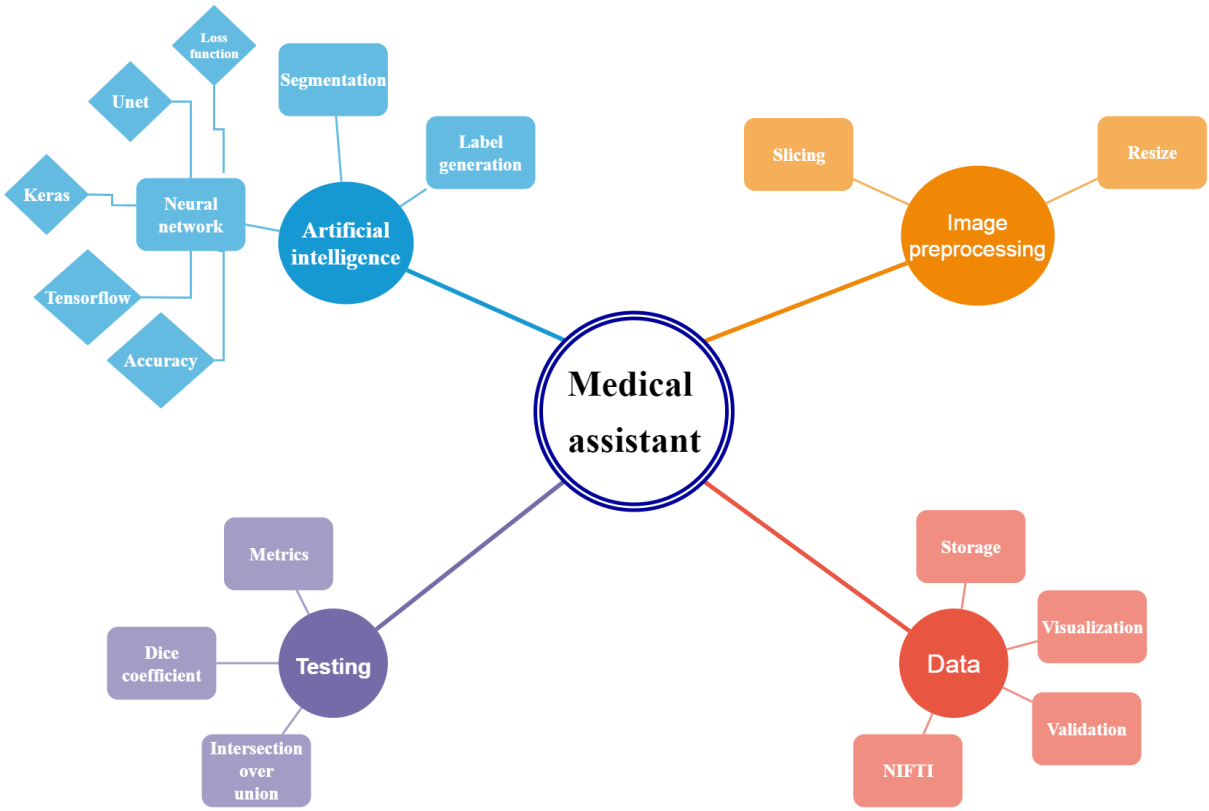
Generated labels for the previous heart using the first implementation of the 3D U-Net



Generated labels for the previous heart using the improved implementation of the 3D U-Net

Appendix 3

Mindmap



Medical assistant mindmap