IMA206

Project Report

Academic year 2022 - 2023

U-Net for cardiac MRI segmentation

ACDC Challenge

Antoine Andurao, Xiangyi Chen, Lauria Sun, Shangning Xia
Under the supervision of Loïc Le Folgoc

July 2023 - Oral presentation
Jury : Elsa Angelini, Loïc Le Folgoc

Public report

Contents

1 Introduction

1.1 ACDC Challenge

Cardiac function analysis is very important in clinical cardiology for diagnosing diseases, patient management and therapy decisions. Some heart problems can become life-threatening if not detected early. That's why it is crucial to identify these conditions early in order to provide the right treatment and prevent complications.

In recent years, there have been advancements in using computer-aided diagnosis (CAD) with cardiac magnetic resonance imaging (CMRI) to automatically diagnose heart problems. These advances help doctors make more accurate diagnoses and provide timely treatment.

The ACDC Challenge project aimed to create an autonomous control and decision-making system for diagnosing heart diseases in patients.

1.2 Goal of Our Project

The goal of our project is to utilize deep learning techniques, specifically U-Net architecture, to develop a model for segmenting cardiac images. We want to focus on three specific areas: the left ventricle cavity, right ventricle cavity, and the myocardium. To train our model, we will use a set of cardiac MRI images that already have the desired areas marked. Our goal is to develop a system that can automatically and precisely identify these regions without human intervention. We intend to investigate the impact of various data augmentation techniques and test-time augmentations on the performance of our model.

1.3 Composition of Dataset

We used the dataset provided by INSA Lyon : https://www.creatis.insa-lyon.fr/Challenge/acdc/

The dataset provided for our project consists of 150 subjects, each accompanied by their respective MRI images and corresponding segmentations. The images represent two distinct cardiac phases: end diastole (the end of dilation in the cardiac cycle) and end systole (the end of contraction). Each MRI image is a 3D volume encompassing the heart and neighboring structures, stored in .nii format.

The dataset has already been split into a training-validation set and a test set. The training-validation set comprises two-thirds of the total subjects (100 subjects), while the remaining third (50 subjects) forms the test set. The training-validation set includes the ground truth segmentations provided by clinicians for the purpose of training and validation, while the test set lacks segmentation.

For each image in the training-validation set, a corresponding 3D segmentation map is provided, delineating the cardiac anatomy into three substructures: the left ventricle cavity, right ventricle cavity, and myocardium. These segmentations are represented as 3D multi-label masks, where each label corresponds to a specific region:

- label '0' : the background,

- label '1' : the right ventricle cavity,
- label '2' : the myocardium,
- label '3' : the left ventricle cavity.

By utilizing this annotated dataset, we aim to train our deep learning model to accurately segment the cardiac structures in unseen MRI images, enabling automated and precise analysis of cardiac anatomy.

2 Method

In this section, we will present in detail what we have done to reach our goal. First, in order to transform the raw *nii.gz* format medical images to uniformly shaped tensor data with the same intensity distribution, we have used several pre-processing methods. We have also adopted some data augmentation methods to diversify our dataset. This would help reduce the overfitting problem and increase model's robustness. Then, we will explain the segmentation model: U-Net and the efforts made to tweak it. At last, we will talk about some post-processings to improve the model's performance.

2.1 Pre-processing

Since there exists a well-known python library *Monai* which is dedicated to medical image processing, we have decided to directly use it in corporate with *Pytorch* for the segmentation task. Monai has a handful of useful predefined transformations for converting MRI images to Pytorch tensors. Using them would greatly reduce our workload and avoid re-coding for the exact same purpose. However, the original transformations do not always meet our requirements. Thus, we have also implemented some custom ones. The list of transformations is as follows 2. We have decided to scale the images twice (in Spacing and Resize) because the margin used to crop the image is a fixed integer and different voxel sizes will result in different distances in reality and thus different scaling factors. With the same idea, we have modified the vanilla version of CropForeground to introduce a flex margin to ensure that the voxels will not be distorted in the final images (having different scaling factors for the x-axis and the y-axis). The comparison of the two versions of CropForeground is shown in Figure 1.

Transformation	Explanation
LoadImage	To load the image along with its metadata from the given path
EnsureChannelFirst	To add one dimension as the channel for Pytorch models
Orientation	To orient the images in the same format
Spacing	To unify the voxel spacing of images
CropForegroundSquare[1]	To center the image at the ground-truth mask with a flex margin to produce a squared image
ScaleIntensityRangePercentiles	To apply a linear range scaling on the image
Resize	To resize the images to the same resolution
To2DSlice[2]	To convert 3D images to 2D slices

[*] The custom version of CropForeground, with a minimum margin for each side instead of a fixed one.

[**] A custom transformation made for 2D segmentation. It accepts 3D input and will return a list of 2D slices.

Table 1: The list of pre-processing operations done in order

As we can see, the yellow bounding box has the same margin for each side, resulting in a rectangular shape. However, the final resolution of the images for training is set to 128×128. Using non-square cropping will inevitably distort the images and the distortion will be different from one image to another. But with the white bounding box implemented by our custom

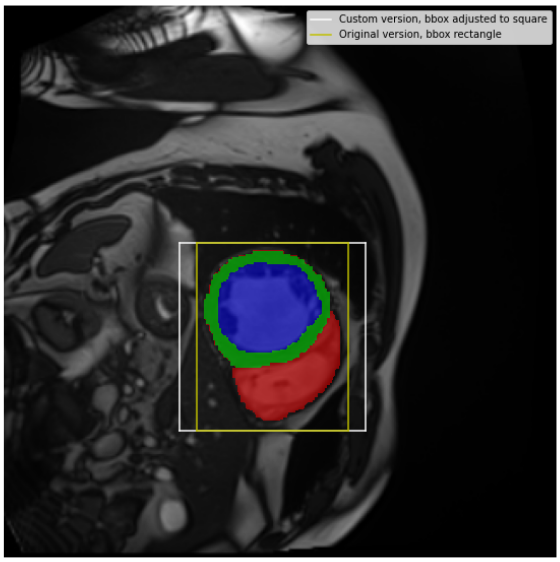


Figure 1: Illustration of custom versions of CropForeground

CropForegroundSquare, the cropped image will be a perfect square, and there will be no distortion when resizing the images.

2.2 Data augmentation

Data augmentation is a widely-used technique in the field of machine learning. One key advantage is that it allows us to generate additional data samples through transformations like rotation, scaling, and flipping. Thus, we can provide the model with more diverse training set.

Moreover, data augmentation helps in improving the model's ability to generalize to unseen data. By introducing variations to the training samples, we enable the model to learn robust features that are invariant to changes in the input. This helps the model perform well in real-world scenarios where the data may exhibit variations in lighting conditions, orientations, or other factors.

In our case, the data came from the same two machines throughout the entire experiment, as mentioned in the ACDC Challenge. These machines had consistent resolution. The data were collected over a period of six years using two MRI scanners with different magnetic strengths: 1.5 T (Siemens Area, Siemens Medical Solutions, Germany) and 3.0 T (Siemens Trio Tim, Siemens Medical Solutions, Germany). Therefore, we knew that changing the intensity of the data would not significantly improve the results. However, in general, adjusting the intensity of the data is also important. It helps the model learn to handle different levels of brightness or contrast. That is why we still included intensity augmentation in our approach. However, we were more interested in spatial data augmentation. We used RandFlipd, RandRotated, RandZoomd, and RandAdjustContrastd, applying each transformation with the same probability, *PROB*.

By applying spatial augmentations like flipping or rotating to the data, we expected to

significantly improve the results. The impact of data augmentation on the performance will be discussed in the Results section.

2.3 Segmentation network

2.3.1 Basic U-Net architecture

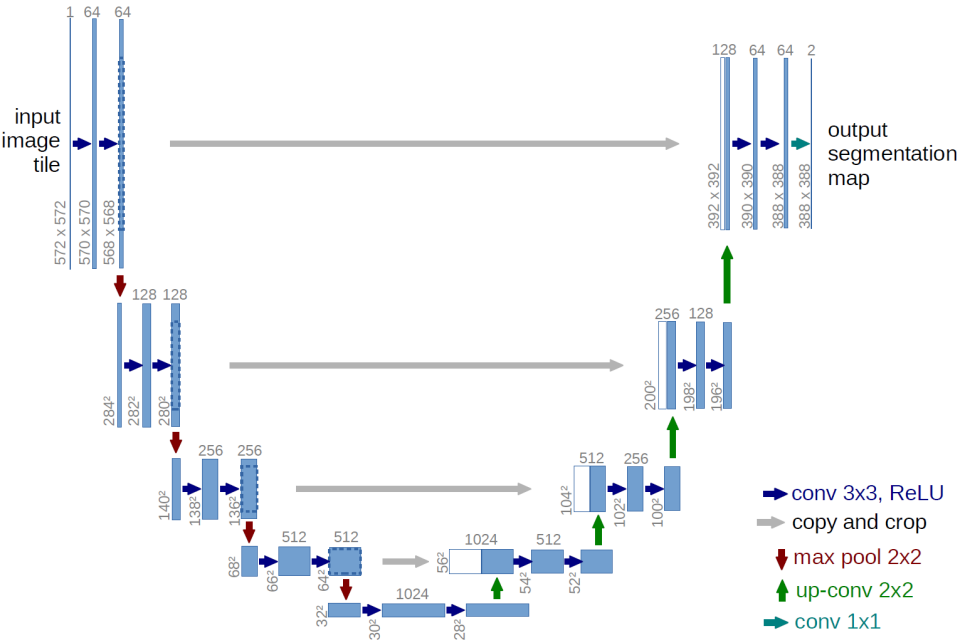


Figure 2: Architecture of U-Net

We based our implementation of the U-Net model on the code provided by the github : https//github.com/milesial/Pytorch-UNet/tree/master

The architecture of this basic U-Net model consists of an encoder-decoder structure with skip connections. In general, U-Net models are commonly used for image analysis.

The U-Net architecture can be divided into two main parts: the encoder and decoder. The contracting path, encoder, captures the context and reduces the spatial dimensions of the input, while the expanding path, decoder, recovers, though skip connections, the spatial information and generates the segmentation mask.

The U-Net architecture is composed of 4 classes:

1. DoubleConv: a building block consisting of two sets of convolutional layers followed by batch normalization and ReLU activation. It is used throughout the model to extract features and increase the model's capacity.

2. Down : performs downsampling of the spatial dimensions using max pooling followed by a DoubleConv block. It helps in capturing larger context by reducing the spatial resolution.

3. Up : performs upsampling of the spatial dimensions. It consists of an upsampling operation (either bilinear upsampling or transpose convolution) followed by a DoubleConv block. It concatenates the feature maps from the contracting path (skip connections) to preserve the

spatial information.

4. OutConv : performs a 1x1 convolution to map the final feature maps to the desired number of output classes. It produces the final segmentation logits.

Layer	Use
inc	Down modules that perform downsampling operations, gradually reducing the spatial dimensions.
up1 to up4	Up modules that perform upsampling operations and concatenate the skip connections from the corresponding down modules.
down1 to down4	Performs downsampling of the spatial dimensions using max pooling followed by a DoubleConv block. It helps in capturing larger context by reducing the spatial resolution.
outc	Convolutional layer producing the output.

Table 2: The components of the U-Net model

Overall, the U-Net architecture combines the high-level contextual information captured by the contracting path with the detailed spatial information preserved by the skip connections in the expanding path. This allows the model to effectively segment images by capturing both global and local features.

2.3.2 Receptive Field

The receptive field in a neural network refers to the effective size of the input space that influences the output of a particular neuron. It represents the spatial extent of the input that a neuron is sensitive to and can directly influence its activation.

It is important to determine and adjust the size of the receptive field. Indeed, in cases where the receptive field size is not sufficient, increasing the size of the receptive field allows for the capture of more contextual information and optimization of U-Net performance. That is why we initially wanted to tune the the receptive field size.

At the end, we realized that for our U-Net architecture with an initial input image size of (128, 128), the receptive field is sufficiently large. The initial receptive field is already large enough to capture relevant contextual information in the input image.

2.3.3 Hole convolution

Hole convolution, also known as dilated convolution, offers several benefits in the context of convolutional neural networks.

The major benefit of hole convolution is the ability to capture more information without increasing the number of parameters. This is the first reason why we chose to be interested in

the hole convolution. By introducing holes or gaps between the kernel elements, the receptive field expands. This enables the network to gather information from a broader spatial context.

We tried with a dilation rate of 2, which introduced hole convolution into our network architecture. However, despite the theoretical advantages of hole convolution, we did not observe significant improvements (See table 3).

In our specific case, the reason why hole convolution did not yield improved results could be attributed to the fact that the initial context provided by U-Net was already sufficient. The receptive field was already large enough to capture the necessary contextual information from the input data, as we mentioned previously. As a result, in our case, introducing hole convolution, which further expands the receptive field, did not provide any significant additional benefit in terms of performance.

2.4 Post-processing

The post-processing pipeline consists of several operations performed sequentially. First, a softmax function was applied to the predicted logits to obtain class probabilities. Next, the probabilities were discretized using the *AsDiscrete* transformation of *Monai*. This step involved converting the probability maps into discrete segmentation labels using an argmax operation. The resulting labels were one-hot encoded with four channels, corresponding to the background and the three cardiac structures of interest: right ventricle, myocardium, and left ventricle.

To further refine the segmentation results, we employed the *KeepLargestConnectedComponent* operation. This post-processing step retained only the largest connected component for each of the three cardiac structures, effectively removing any isolated or fragmented regions that might have been erroneously included in the segmentation output.

3 Experiment

3.1 Loss functions

During the training of our image segmentation network, we utilized a combination of Dice Loss and Cross Entropy Loss to guide the learning process.

DICE Loss is commonly used to train image segmentation networks. The Dice coefficient is defined as twice the intersection between the predicted segmentation and the ground truth mask, divided by the sum of their areas. The Dice loss equals one minus the Dice coefficient. The DICE loss measures the spatial overlap between the predicted and ground truth masks, emphasizing the spatial accuracy and alignment of the segmentation results.

$$L_{dice} = 1 - \frac{2\sum_i^N p_i g_i}{\sum_i^N p_i^2 + \sum_i^N g_i^2} \tag{1}$$

where p_i is the predicted label, g_i the ground truth label.

Cross Entropy Loss is commonly used in multi-class classification problems. While Dice Loss penalizes dissimilarity between a segmented class and its corresponding ground truth as whole (as an ensemble), the cross entropy loss focuses on pixel-wise classification accuracy and penalizes dissimilarity at per pixel scale.

$$L_{CE} = -\sum_{i=1}^{N} g_i \log(p_i) \tag{2}$$

To strike a balance between accurate classification and precise spatial localization, we opted for a weighted combination of the two loss functions. This approach allows us to leverage the strengths of both losses. By adjusting the weighting factors λ, we can control the relative importance of each loss during training, ensuring an optimal balance between classification accuracy and spatial coherence. Our final loss function is defined as follows :

$$L = L_{dice} + \lambda L_{CE} \tag{3}$$

In our experiments, we used the weighting factor $\lambda = 0.1$ which experimentally proved to give the best result.

3.2 Optimizer

For the training of the U-Net, we used one of the most common optimizer : **Adam optimizer**.

Adam is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments, commonly used in many deep learning tasks. According to Kingma et al., 2014, the method is computationally efficient, has little memory requirement, invariant to diagonal rescaling of gradients, and is well suited for problems that are large in terms

of data/parameters.

Adam notably has adaptive learning rates making it converge faster and more efficiently compared to traditional optimizers with fixed learning rates.

3.3 Hyperparameters

To ensure the reproducibility and control of our experiments, we carefully selected and set several key hyperparameters for training the image segmentation network. The following hyperparameters were utilized in our experiments:

- Number of Epochs: 150
- Batch Size: 64
- Learning Rate: 0.01
- Probability of Data Augmentation: 0.3
- Train-Validation Set Split Ratio: 70%

4 Results

The U-Net model was evaluated for its segmentation performance using the ACDC test set. This section presents the qualitative and quantitative results obtained through the U-Net segmentation approach. Furthermore, to assess the efficacy of data augmentation, comparative experiments were conducted without employing data augmentation, enabling a thorough examination of its impact on the results. The subsequent sections detail the findings of these evaluations, providing a comprehensive analysis of the U-Net model's performance and the significance of data augmentation in the study.

4.1 Analysis of training curves

Figure 3 shows the curve of training and validation loss with and without data augmentation. Without data augmentation, we observed a common trend where the training loss steadily decreased with epoch, while the validation loss stagnated since around the 80th epoch. This suggests that the model tends to overfit to the training data. This hinders the model's performance on unseen data, leading to limited improvement in the validation loss (and worst performance on test set as will be detailed in the following).

In contrast, when data augmentation was incorporated, the training loss and the validation loss followed a similar evolving pattern. This alignment indicates that the model's generalization ability improved significantly thanks to the various augmentations we employed, leading to improved performance on unseen data, as reflected by the concurrent reduction in the validation losses.

Interestingly, we also observed a noteworthy phenomenon: during the initial epochs, the validation loss decreased at an even faster rate than the training loss. This intriguing observation can be attributed to the difference in how the losses are computed. While the training loss is calculated at every iteration and then averaged over the entire epoch, the validation loss is computed only once at the end of each epoch. This phenomenon suggests that in the early stages of training, the model rapidly learns with a high degree of generalization capability.

It is worth noting that while data augmentation brings substantial benefits, it also incurs an increase in training time. This additional time arises from our method of performing data augmentation on-the-fly, i.e., generating augmented samples during the loading of training batches. This trade-off is well-justified by the enhanced model performance and better generalization.

4.2 Quantitative results of segmentation

Model	Mean	Right Ventricle	Myocardium	Left Ventricle
w. data augmentation	**0.893**	**0.882**	**0.872**	**0.923**
w.o. data augmentation	0.883	0.862	0.867	0.919
w. hole convolution	0.883	0.871	0.857	0.920

Table 3: DICE scores for image segmentation with and without data augmentation.

Table 3 shows the average Dice score for LV, RV and MYO. We have achieved an overall average DICE score of 0.893. We notice that the model achieves a relatively higher dice score for LV when compared to RV and MYO.

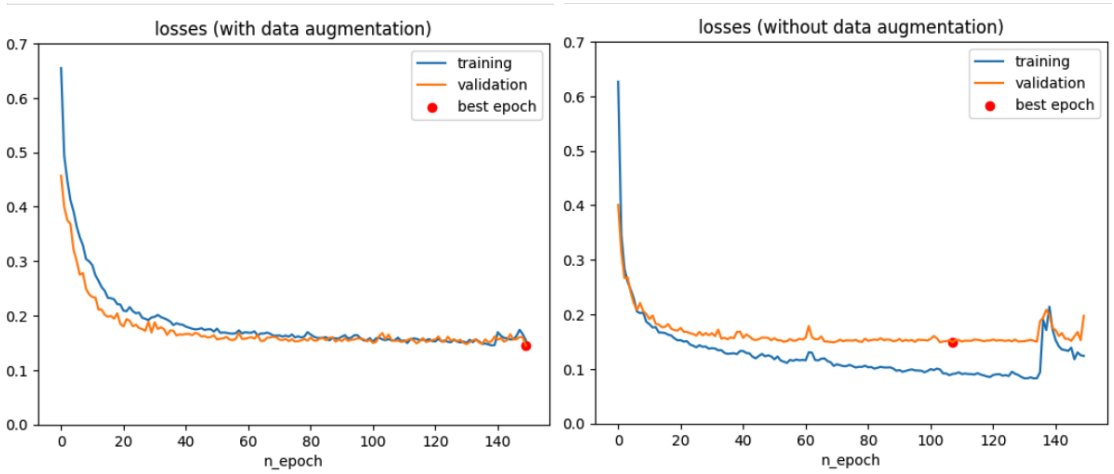


Figure 3: Left: training curve with data augmentation. Right: training curve without data augmentation

As for the effect of data augmentation, we observe that the average DICE score on the test set increased 0.04. Analyzing the results for individual classes, we found consistent improvements across the board when data augmentation was utilized, most significant for right ventricle. These findings provide evidence that our data augmentation process effectively contributes to the performance of the image segmentation model. Indeed, by introducing additional variations and diversity into the training data, the model becomes more robust and capable of capturing the subtle differences in the target classes.

4.3 Qualitative results of segmentation

Figure 4 shows respectively our best and worst performance in terms of average Dice score. We observed that the performance of our model varied depending on the location of the 2D slices within the cardiac MRI volume. Notably, the best performance was observed for slices located in the middle of the volume, while the worst performance was consistently observed for slices near the apex or the base of the heart structure.

This discrepancy can be attributed to the complex nature of cardiac structures and their spatial distribution within the heart. The slices in the middle of the volume often contain larger and more distinct sections of the structures that are more discernible in terms of shape, size, and intensity patterns, facilitating accurate segmentation. Conversely, for slices located at the extremities of the volume, the heart structures tend to be smaller, or exhibit variations in shape due to their proximity to adjacent anatomical features. Besides, the limited visibility of certain cardiac structures in the extremity slices can also result in reduced contrast and ambiguous intensity patterns.

In Figure 5 we show the worst performance for each class. Upon examining the failure cases for each cardiac structure, several patterns emerged. For the right ventricle, the model struggled specifically near the apex of the structure along the z-axis. This difficulty arises when the ground truth label indicates a small volume for the right ventricle, yet our model fails to predict any segmentation at all, leading to a low DICE score. As for the myocardium and left ventricle, the failure cases occurred in slices located at the extremities of the volume. Typically, In these regions, some of the Unet segmentation results exhibited visible errors, as they failed to adhere to the anatomical rule that the myocardium should always surround the left ventricle.

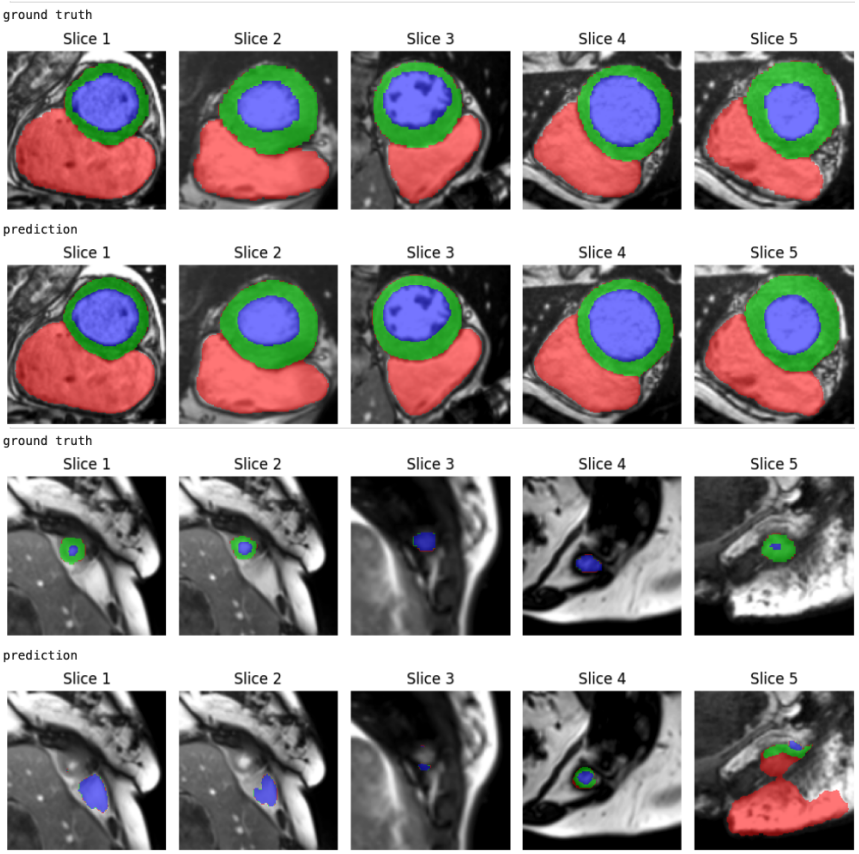


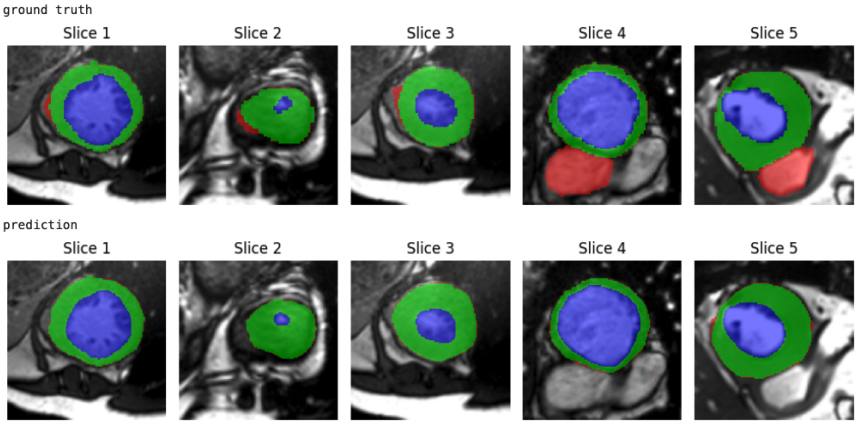
Figure 4: Example of segmentation result on test set. Top: best performance. Bottom: worst performance. In all figures, red, green and blue represent the RV, MYO and LV, respectively.

Furthermore, we analyze the score distributions for different slices. Since the total number of slices per patient is different, we center them by their middle slice (index=5) and take the average scores on the test set. The result is shown in Figure 6. As expected, on average, the scores decrease from center to extremity. Among the 3 structures, the performance of the left ventricle cavity is the most consistent for different positions, and the variation of the right ventricle cavity is the largest.

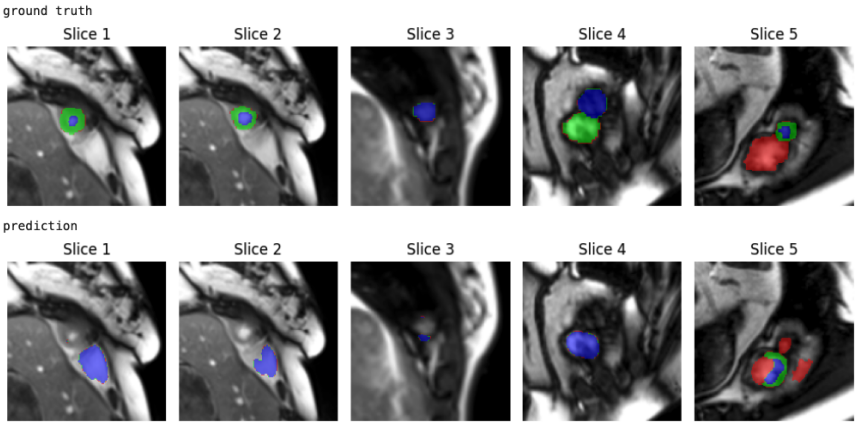
4.4 Future work

These failure cases shed light on the challenges posed by the smaller size and complex spatial relationships of cardiac structures, especially in the extremity slices. The segmentation model needs to be refined to accurately capture these nuances and ensure consistent adherence to the anatomical rules governing the relationships between different cardiac structures. Thus, we might need to adopt a multi-stage model to better segment cardiac structures regardless of their spatial sizes and positions. We can keep the current model as the first stage, and in later stages, we make the model learn the geometry relationships of these structures and take care of the extremity slices by giving the model well-segmented center slices as references.

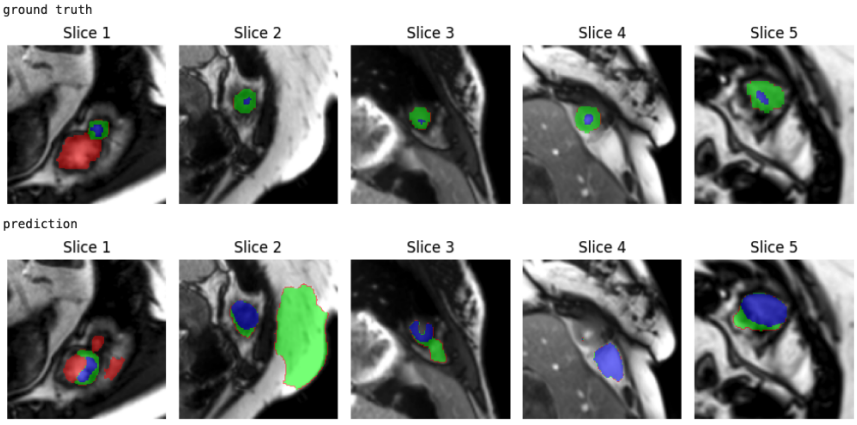
Also, we can try other segmentation models and introduce more carefully designed post-processing methods to remove the visible errors from the prediction.



(a) Right ventricle cavity



(b) Myocardium



(c) Left ventricle cavity

Figure 5: Fail cases (worst performance) for all three cardiac structures. In all figures, red, green and blue represent the RV, MYO and LV, respectively.

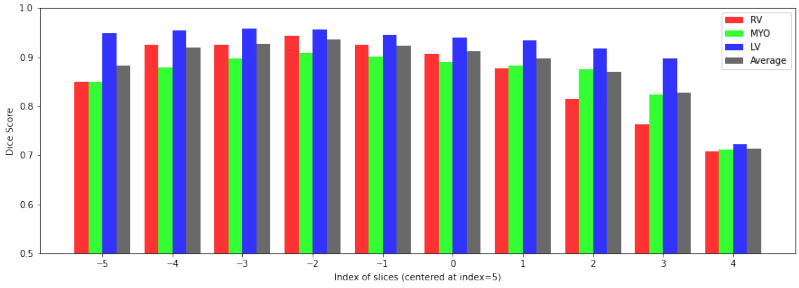


Figure 6: The distribution of DICE scores per slice

5 Conclusion

To conclude, in this project, we have proposed a well-designed pipeline of cardiac structure segmentation. We have used a number of pre-processing to unify the input images, several data augmentation methods to enlarge our relatively small dataset, adopted U-Net as the main segmentation model, and some post-processing to try our best to correct the predicted mask. Finally, we have obtained a quite good result, with the average DICE score near 0.9, comparable to the SOTA work.

However, due to the limited time, there are still a lot of things we could do to improve the results. After this project, we will continue to work on it and try our best to make the model better in the future!

The code of our project is on Github: https://github.com/MunchkinChen/IMA206_project