University of Nottingham
UK | CHINA | MALAYSIA

Exploring The Techniques for Improving Right Ventricle Segmentation of Multi-Domain Dataset

Submitted **June 2023**, in partial al fulfillment
of the conditions of the award of the degree
MSc Computer Science (Artificial Intelligence).

Vincent Shao Sen, Chueh
Student ID: 20316799

Supervised by
Professor Michael Pound

School of Computer Science
University of Nottingham

I declare that this dissertation is all my own work, except as indicated in the text.

Abstract

This research examines the “Multi-Disease, Multi-View & Multi-Center Right Ventricular Segmentation in Cardiac MRI (M&Ms-2)” challenge. In this challenge, 2D and 3D images of hearts are provided, and the goal is to segment the right ventricles from other parts of the hearts. In this project, I reproduced the methods proposed by Li et al. [1], which combine both 2D and 3D deep learning segmentation models in order to allow each pair of 2D and 3D images to provide information for 2D and 3D segmentation. Furthermore, in order to further tackle the multi-domain nature of the M&Ms-2 dataset, I have explored more techniques that could pre-process the multi-domain dataset, or boost the generalizability of the deep learning models.

Acknowledgements

I would like to express my sincere gratitude to my supervisor Professor Michael Pound, who has provided me with great help and guidance with patience, which is truly inspiring and helpful for me throughout the whole period of this thesis project. Also, I would like to thank my dear parents who have given all the love and resources in the 2 years of studying my master's degree. Last but not least, I want to thank all the friends from the Nottingham Cathedral Choir, who have given me warm and unforgettable memories in Nottingham.

Contents

Introduction

This research examines the right ventricle (RV) segmentation task from the "Multi-Disease, Multi-View & Multi-Center Right Ventricular Segmentation in Cardiac MRI (M&Ms-2)" challenge by the University of Barcelona. The purpose of this challenge is that the RV segmentation is very difficult due to some reasons that will be mentioned later in this paragraph. The last RV segmentation challenge by MICCAI was in 2012, when deep learning has not been the mainstream method for image segmentation. Thus, in view of the outstanding performance of deep learning in recent years, it's worth trying to use it to solve the RV segmentation task.

There are few challenging points for the RV segmentation. First of all, the shapes of RVs are highly variable and complex, which could affect the prediction ability of a deep learning model. What's more, this heart image dataset contains images with various heart pathologies, 2 of which appear only in the testset. This will require the model to have strong generalizability in order to accurately segment the right ventricles. The second challenge is that this dataset contains both 2D and 3D heart images, which cannot be segmented with a single deep learning model. Thus, I have come up with a method to transform the 2D image coordinate system in order to combine the 2 types of images and transfer certain features of the 2D images to boost the accuracy of the 3D segmentation result. Due to the challenging points mentioned above, I think the RV segmentation is a topic that is worth doing research in. Thus, I chose the MnM2 Challenge as my thesis topic.

The method of this project is to first reproduce the results of the network architecture proposed in the paper by Li et al[1], which was a limited study and did not provide the necessary details of the network architecture. Then, I will use the result of this architecture as the baseline, and explore more techniques such as

fourier domain adaptation, and transfer learning to see if those methods could further improve the accuracy of the RV segmentation.

Literature Review

1. Image Segmentation

Image Segmentation is the task that partitions an image into several regions called image segments. The purpose of image segmentation is to simplify the structure of an image so that it's easier for further image processing and analysis. There are different types of image segmentation, such as semantic segmentation, instance segmentation, and panoptic segmentation, etc. The difference of these types of segmentation lies in the level of information provided by the result. Before deep learning becomes the mainstream method, there are several traditional image segmentation algorithms that have achieved acceptable results. Abdulateef and Salman [2] have generally introduced these image segmentation techniques. For instance, thresholding-based segmentation takes advantage of the gray-scale change to distinguish different regions. It can be used to extract the foreground and the background of an image. Another example is region-based segmentation. This method aims to arrange pixels into clusters based on similarity such as pixel intensity, color, or distance. However, there are a few disadvantages of the traditional techniques. Mahony et al. [3] points out that traditional computer vision algorithms rely heavily on domain knowledge to select which features are important for this task. It could be quite challenging and tends to take a lot of effort in the trial and error process.

2. Deep Learning

In recent years, with the outstanding performance of deep learning in many fields, including computer vision, it has gradually become the mainstream method for image segmentation. Compared to deep learning, conventional machine learning requires more domain knowledge to implement steps like pre-processing, feature extraction and selection, which have a massive impact on the performance of these models. On the other hand, deep learning is a type of representation learning that can take into very raw form of data. Through multiple non-linear layers networks, a very complex function can be learned [4]. Deep learning has outperformed other conventional algorithms in many fields. LeCun et al. [5] have indicated that deep learning is especially good at discovering complex structures in high-dimensional data, and has achieved better performance than other algorithms in fields like medicine, pharmacy, and natural language processing.

In the field of computer vision, deep learning has also beaten most of the conventional algorithms in most applications. With Convolutional Neural Networks becoming the mainstream backbone of deep learning in computer vision, many models have been proposed, such as AlexNet, VGG, and ResNet, etc [6][7][8]. With the structures becoming deeper, the model capacity becomes greater and thus more complex functions can be learned, which push the accuracy further. As for image segmentation, with enough data, CNN-based deep learning models have also beaten conventional methods.

Long et al. [9] proposed a fully convolutional network (FCN) that relies only on CNN structures to achieve semantic segmentation. Compared to other types of neural networks, FCN does not have any fully connected layers. The

purpose of only using CNN structures is that it can be trained end-to-end, pixel-to-pixel due to the unique nature of semantic segmentation. Ronneberger et al. [10] proposed the Unet based on the FCN, but it improves the upsampling method and adds the skip connection technique to further increase the model's prediction accuracy of pixel-wise details. There are many models with outstanding performances based on the Unet structures such as [11] [12] due to its excellent structure.

3. Domain Adaptation

In order to handle the multi-domain nature of real-world images, many domain adaptation techniques have been proposed. Hoffman et al. [13] proposed a domain adaptation model based on CycleGAN technique. However, this model only contains 2 domains, which is very unlikely in the real-world situations. Zhao et al. [14] proposed a domain adaptation model based on a similar CycleGAN mechanism but it can be used in multi-domain images. However, those GAN-based methods are very difficult to train, especially when applied to more than 2 domains, the cost of training multiple models could be very high. Thus, Yang and Soatto [15] proposed a fourier domain adaptation using just a Fourier Transform and its inverse. This method does not require training an extra DL model, and its performance is just as good as those GAN-based models.

4. M&Ms-2 Challenge

The Multi-Disease, Multi-View & Multi-Center Right Ventricular Segmentation in Cardiac MRI (M&Ms-2) Challenge is a challenge that aims to

create generalisable ml/dl models that can be applied across clinical centers. The first MnMs Challenge was in 2020 and the paper was published in 2021 [16]. In the MnM2 Challenge, the task is to segment the right ventricle (RV). Li et al. [1] proposed a 2-step model that consists of a 2D and a 3D model with a 2D transition mechanism in the middle. However, as the author admits, in this paper the 2D and 3D model are trained separately and then used the 2D result to inform the 3D model. In this way, the 2D model is not fully trained since the 3D information is wasted in the 2D training. Thus, in this paper, I want to explore a 1-step, end-to-end training of the 2D and 3D models. Also, I want to explore some novel ways of combining the 2D model output with the 3D model input in the research.

5. Transfer Learning

The training of deep neural networks requires a larger amount of data compared to training traditional machine learning algorithms because the model capacity is larger. However, data in the real world can be hard to acquire, let alone the labeled data. With limited training data, it is very common to suffer from overfitting. Therefore, it is useful to take advantage of transfer learning to increase the generalizability of the model. As indicated by Weiss et al [17], transfer learning can take advantage of the information that is learned from other domains to help improve the learning in the target domain. Hosna et al [18] defines transfer learning as training the target dataset with a model that has been trained on other similar datasets or similar taks. In this way, the model does not need to be trained from scratch. Also, because the model has learned various features from other datasets, it could allow the

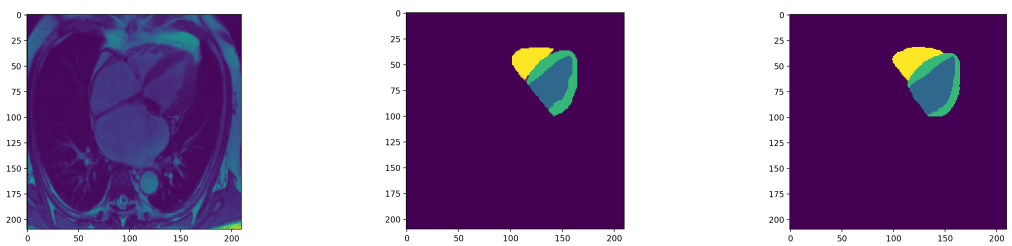
model to generalize better on the data that has not been seen by the model before.

Dataset

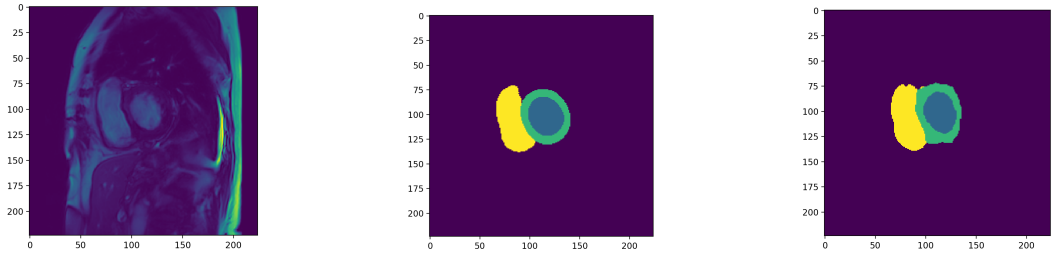
The Multi-Disease, Multi-View & Multi-Center Right Ventricular Segmentation in Cardiac MRI (M&Ms-2) dataset is the dataset that contains both 2D (long-axis view, LA) and 3D (short-axis view, SA) heart images. In total, there are 360 patients' images in the dataset. Each patient will have 2 LA-view images, and 2 SA-view images. The reason that there are 2 images of each view is that there are 2 types of phase of heart, the End-diastolic (ED) phase and the End-systolic (ES) phase.

The 2D and 3D images can be transformed onto the same coordinate system using the transformation matrix lies in the file header. I will explain this transformation process later. There are 4 classes on the mask of both 2D and 3D images: the background, the right ventricle(RV), the left ventricle(LV), and the left ventricular myocardium (MYO). The purpose of this task is to segment the right ventricle(RV) from other parts of the image. Below are the visualization of some images from the dataset:

LA image samples (left : image, mid : mask, right : prediction)



SA image samples (left : image, mid : mask, right : prediction)



The file format of this dataset is called Neuroimaging Informatics Technology Initiative (NIfTI). This file format is proposed by the NIfTI Data Format Working Group. The main purpose of this data format is to provide coordinated service of neuro-imaging in order to speed up the process of software development. This file format is widely used in medical imaging, and The coordinate information lies in the file header. With this format, software developers can have more control over the coordinate systems of the images and make the process more convenient. In order to deal with NIfTI data, I will use a package named Nibabel. This package is created in order to deal with common neuroimaging file formats. In this project, I mainly use this package to input/output the images and the coordinate transformation.

Lastly, in order to make the dataset suitable for training, I need to rearrange the file directory so that I can use the Dataset Class in Pytorch to access the training data as a dataset object that is ready to put into the training pipeline. In the original dataset, the file directory is arranged according to each patient. There are 360 patients in this dataset, and thus there are 360 folders in the dataset folder. In the folder of each patient, there are 5 long-axis(LA, 2D) images and 5 short-axis(SA, 3D) images. The 5 images from each axis include 2 ED-, ES-phase images, 2 ED-, ES-

phase target masks, and 1 CINE image(won't be used in this project). In order to make it more suitable for training, I firstly need to separate the SA and LA images since the training for 2D and 3D images are separate. What is more, I also need to separate the images and the masks so that they can be accessed in 2 different folders. Finally, I need to split the images and masks into training, validation, and testing sets.

Thus, after the update, there are 2 folders under the dataset, 1 for LA images and 1 for SA images. Under each folder, there are 3 folders for training, validation and testing. Furthermore, each of the folders contains image and mask folders.

Model Architecture

In this project, I choose to begin with the similar architecture proposed in [1] by Li et al. The reason that I choose this architecture is that it has reached high accuracy in the test dataset. Also, its structure is concise but has some flexible parts that I will be able to experiment with. The architecture consists of 3 main parts: the 2D nnU-Net, 3D nnU-Net, and the image transformation from LA (2D) to SA (3D).

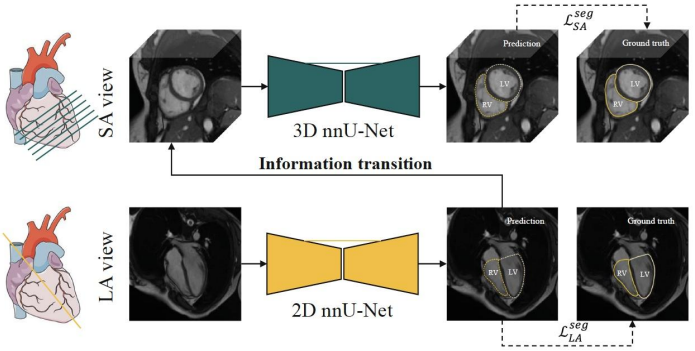


Fig. 1. The proposed RV segmentation framework for both SA and LA images. The framework includes three steps: the LA segmentation, ROI extraction from SA with assistant of LA information, and the SA segmentation. Here, the 3D cardiac image adopted from Kevil et al. [6].

Fig.1. Network Architecture of [1] by Li et al.

The authors of the paper[1] choose to use the 2D and 3D nnU-Net as its backbone of the training architecture. They first train the 2D nnU-Net, transform the output of the 2D model into the 3D coordinate, combine it with the 3D (SA) images, and put them into the 3D nnU-Net. However, the details of the implementation of the 2D to 3D transformation is not described clearly in this paper. Thus, I'm interested in experimenting with some tricks to see which methods could improve the accuracy.

The structure of the nnU-Net is almost the same as the original basic Unet without those recent modifications such as residual connections and attention mechanisms. The reason is that the authors have observed that during training, those state-of-the-art modifications of the Unet usually equip the model with too much capacity and result in overfitting. The difference is that it's a framework that automatically configures the training steps and parameters, such as preprocessing, loss selection, optimizer setting, and potential post-processing.

However, in this project, I want to have more control over these factors in order to observe which factors would affect the accuracy of the model. Thus, I choose to use the basic Unet (both 2D and 3D) in this project. Although this would mean that the final accuracy of this project might not be as good as the experiment by Li et al[1], the purpose of this project is to explore useful techniques that can improve the accuracy and can be further used in other segmentation tasks.

The structure of the Unet (figure 2) is mainly made of 3 parts, the downsampling, the upsampling, and the skip connection. The downsampling, or the contracting path, is the same as Fully Convolutional Networks (FCN) [9]. It's a network that only contains convolutional layers. Convolutional Neural Network (CNN) has been proved to have excellent performance on many image processing tasks such classification, recognition, and segmentation. In the first few convolutional layers, the network

extracts some detailed and basic features, such as a line or a simple shape. The deeper layers then extract some larger, more concrete features such as an object or the outline of an image. Thus, it is clear that the first few layers are responsible for learning the details of an image, while the deeper layers focus on learning the localization information of an object. Thus, the Unet utilizes this information to improve the structure of FCN.

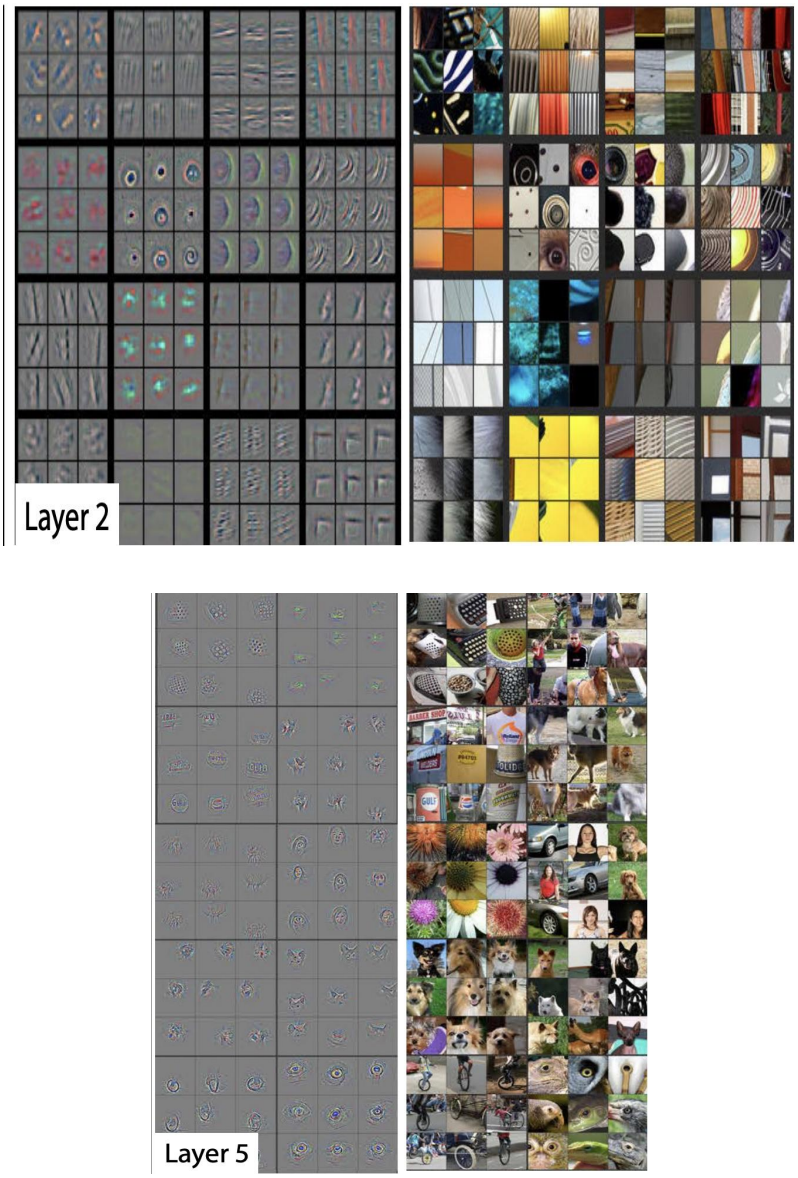


Fig.2. The features learned in layer 2 and layer 5 in CNN [19].

The difference between the Unet and the FCN is that the FCN uses only 1 deconvolution layer to upsample the image to its original scale. On the other hand, the Unet has more upsampling layers, which can provide more localization information for the model. What is more, when upsampling the image, it will inevitably lose some image details. Thus, in order to allow the output to have higher resolutions, the Unet adds the skip connection between the contracting path and the expansive path. Because the Unet's contracting and expansive paths are almost symmetric, it takes advantage of this feature to connect the 2 paths. In this way, while the deeper layers of the model learn larger features and localization information, the skip connections can add the detailed information to the output. In this way, the output can reach a higher resolution compared to the FCN.

As for the 3D Unet, the structure is very similar to the basic Unet, except that the input adds one more z dimension. Thus, one has to adjust CNN to adopt 3D operations. For instance, it contains 3D convolutions, 3D max pooling, and 3D up-convolutional layers in its structure. When dealing with 3D images, there's also another option. I can slice those 3D images into 2D images, and use 2D Unet to segment the RV. However, this means that when training the Unet model to segment the RV, the model only learns based on the single image. On the other hand, the 3D Unet can learn to segment the RV based on not only the single 2D image information, but also in the context of other slices of 2D images. Thus, as stated in the paper [20] by Ö. Çiçek et al. that the 3D Unet requires a small number of annotated images to train the model.

The last part I want to put more emphasis on is the downsampling in the 3D Unet. The 3D Unet has four downsampling layers (as shown in figure 4) that uses max pooling technique to reduce the size of an image. This would not be a problem for

the W and H dimension, since the values of the 2 dimensions are large enough that can be adjusted more flexibly. However, the Z dimension is not large enough to be divided by 2 four times. (there will be more detailed explanations about the image size in the Adjusting Image Size part). Thus, I choose to set the Z dimension to 8, which is 2 to the power of 3. However, I still have to solve the problem that there are four downsampling layers in the 3D Unet. The solution that I've used in this project is that in the first downsampling layer, I only downsample (max pooling) the W and H dimension, and the Z dimension remains the same. This means that the Z dimension is only downsampled 3 times. What's more, since the input and the output of the 3D Unet should be identical, the upsampling procedure should be the inverse of downsampling. Thus, in the last layer of the upsampling, I only upsample the W and H dimension. Therefore, the dimension of the input and the output are the same.

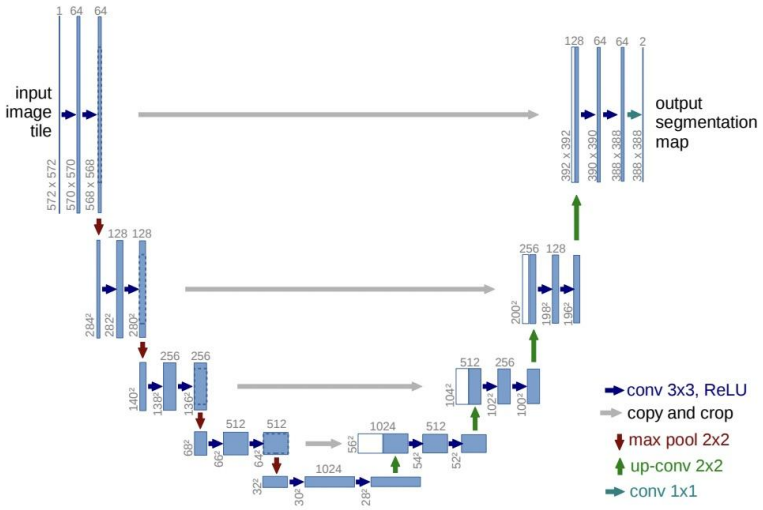


Fig.3. The Unet architecture. [10].

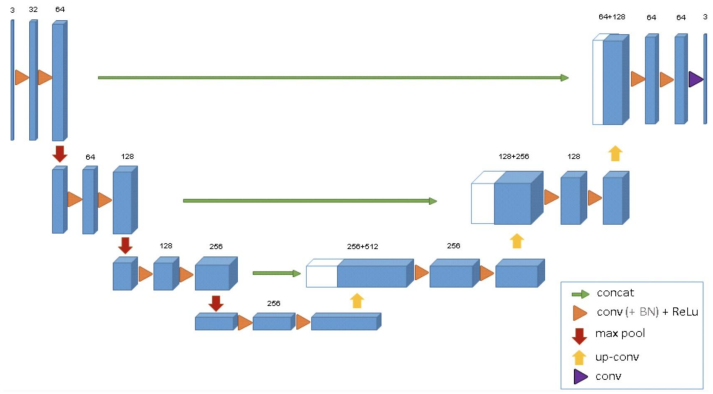


Fig.4. This figure in the paper [20] by Ö. Çiçek et al. indicate the process of the 3D Unet downsampling (the red arrow) and the dimension change.

Model Training

1. 2-Step Training vs 1-Step Training

In the paper [1], the author trains the network separately. This means that the researchers train the 2D network first, transform the output to 3D space, combine with the SA image, and then train the 3D network. However, this 2-step method requires more processing steps. What is worse, the final loss (loss of 3D network) is not propagated backward to the 2D network, which, in theory, means that the accuracy of the 2D network could have reached an even higher level. In this project, I originally intended to explore a 1-step training pipeline that includes the 2D network, 3D network, and transformation process. However, although in theory this is feasible, in reality this would

require too much memory space and time in training. The reason is that the time complexity of long-axis to short-axis coordinate transformation (which will be explained in the “Long-Axis to Short-Axis Image Coordinate Transformation” section) is $O(n^3)$, which is considered very expensive. If the model is trained as 1-step, the coordinate transition process has to be done in each batch, which means that the total times of coordinate transition will be (number of batches)*(number of epochs). Thus, although the 1-step training could potentially have a good impact on the accuracy, it is too inefficient and costly to implement in reality. However, although in the training phase it is unlikely to be done as 1-step, in the evaluation phase (test set) it is possible to do it as 1-step, which makes the model intuitively easier to use.

2. Adjusting Image Size

There are various sizes of images in this dataset, both LA and SA. In the LA image set, the width and height dimension range is from (208*208) to (512*512), while the channel number is 1 since the images in this dataset are grayscale images. As for the SA image set, the (W,H) dimension range is from (240, 196) to (352,352), while the Z dimension is from 5 to 14. Therefore, in order to put it into the DataLoader, it is essential to crop and pad these images. However, there are many factors when changing the size of the images. First of all, the range of the image size is very wide. Originally, I consider cropping all the LA images to the smallest size (208*208). However, when I visualize the cropped images, in some of the images with larger original size (ex. 512*512), part of the hearts are cropped out. Thus, I adjust the LA image size to (448*448) to ensure that the heart image is not cropped

out. As for the SA image, since the range of size is smaller, so I set the W*H dimension of the SA image to be (320*320).

As for the selection of the Z dimension of the SA image, it has to take how it will be processed by the 3D Unet into consideration. The 3D Unet will downsample the image every 2 convolutions, and the dimension size will be half of what it originally was. There are 4 downsample layers in the 3D Unet, which means that each dimension will be divided by 2 four times. The W and H dimension is large enough so it is not a big problem. However, since the Z dimension is much smaller than other dimensions, it has to be dealt with very carefully. Originally, I wanted to pad the Z dimension to 16, which is two to the power of four. However, since most of the Z dimension of the SA images is less than 8, it will almost double the image size and thus require much more memory space. The other option is to reduce it to only 8, and then make the 3D Unet skip one down sample layer. In this way, although this method sacrifices some slices of the Z dimension (some images with the Z dimension that's larger than 8 are cropped out), it helps strike a balance between cropping and padding.

The other factor to consider is that the padding may affect the transformation. In order to transform the LA image (2D) into the same space (3D) as the SA image, I have to transform each coordinate point (W, H, 0) on the LA image onto the 3D space (the details will be explained later). If I pad the image on four sides of the image, the points on the actual LA image will be transformed to the wrong points in the 3D space. The solution in this project is to pad the image only from right and bottom. In this way, the coordinate points on the original LA image will remain the same. The figure 4

has shown the effect of padding the LA image from four sides and only from right and bottom. As for the padding of the SA image, in order to keep the SA coordinate points at the same place, it is only padded from the right and the bottom as well.

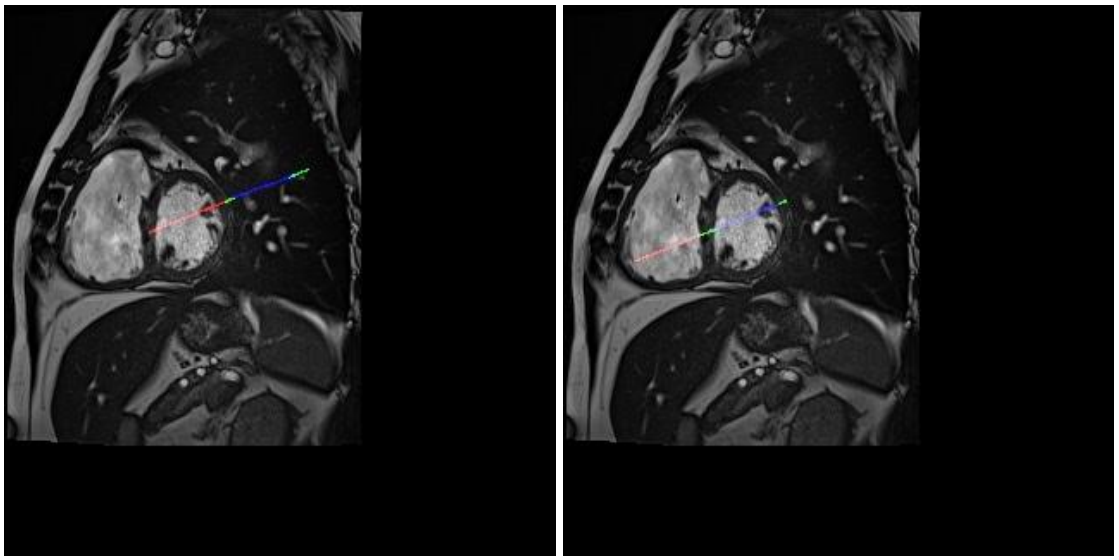


Fig.5. The two images are sliced from the Z dimension of the SA (3D) image. The lines drawn on the images are the transformed LA image. The left image shows the transformed line of a LA image padded from four sides, which is off the right position. The right image shows the transformed line of a LA image padded from only the right and the bottom, which is on the right position.

3. Image Augmentation for 2D and 3D Unet

The purpose of the image augmentation is that it can effectively reduce the chance of over-fitting. Usually, the number of training images that are available is quite limited, just as the MnM2 dataset. It only contains the heart images from 360 patients. What's more, labeling these images takes a lot of time and labor. Therefore, in order to increase the diversity of the dataset, image augmentation is quite helpful. It creates new images based on the original images that are already in the dataset. There are many ways of doing

this, such as geometric augmentations, or some pixel-wise augmentations. The most important thing is that when geometric transformations are done to the image, such as the position, angle, or padding, the mask should be changed the same way. This is because the researchers must make sure that the high-level features of the image and the mask match with each other so that the network can predict the segmentation based on the high-level features. On the other hand, when doing the low-level feature augmentation, such as blurring and normalization, only the image should be augmented. The reason is that the model should predict the result based on high-level features, instead of the pixel-level features. Thus, the image augmentations can help the model learn which features should matter in predicting.

In order to achieve the task of augmenting the image and the mask at the same time, I choose to use the transform function provided in the Albumentations package. This package allows researchers to change the image and mask in pairs. For instance, when I want to change the angle of the image1, I can make sure the mask1 is changed the same way. The reason that I don't transform the image and mask separately using the same transform composition (Pytorch torch.nn.Sequential) is that there are usually some random operations in augmentation, such as random cropping, random flipping, etc. Even if using the same transform composition to transform the image and the mask, the researchers still can't ensure that the matched image and mask are transformed the same way. Thus, with the Albumentations package, I can pair the image and the mask, and then transform the pair together.

4. Selection of The Optimization Algorithms

When training the model, in order to make the training converge or converge faster, the selection of the optimization algorithms is important. In this project, I choose to use Adam as the optimization function. Adam is proposed by Diederik P. Kingma, Jimmy Ba [21], and is actually the combination of a few optimization techniques, including gradient descent, momentum, RMSProp.

I. Momentum

Momentum is a way of optimizing the gradient descent process. In the traditional gradient descent, the current movement only considers the current gradient. However, training with traditional gradient descent could be very slow when the gradient is very small (such as in plateau, saddle point, or local minima). Thus, in order to solve this problem, the momentum technique considers not only the current gradient, but also the previous movement. In this way, the momentum of the previous movement can help get out of the saddle point or where the gradient is small. The mathematical formula can be briefly generalized as the figure below:

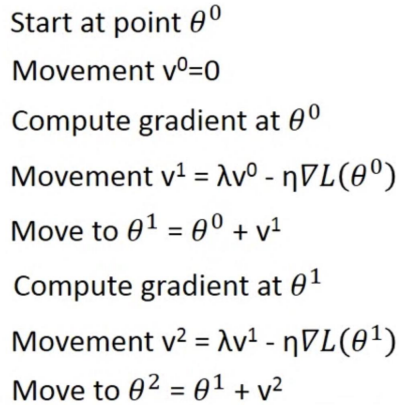


Fig.6. figure by H.-Y. Lee [22].

II. RMSProp

In deep network training, the loss surface tends to be very complex, which means that the learning rate should be adapted very quickly. Therefore, in RMSProp, when updating the parameters with learning rate*gradient, the learning rate is divided by a new variable α that allows researchers to control how to adapt the learning rate according to the network.

RMSProp

$$w^1 \leftarrow w^0 - \frac{\eta}{\sigma^0} g^0 \quad \sigma^0 = g^0$$

$$w^2 \leftarrow w^1 - \frac{\eta}{\sigma^1} g^1 \quad \sigma^1 = \sqrt{\alpha(\sigma^0)^2 + (1-\alpha)(g^1)^2}$$

$$w^3 \leftarrow w^2 - \frac{\eta}{\sigma^2} g^2 \quad \sigma^2 = \sqrt{\alpha(\sigma^1)^2 + (1-\alpha)(g^2)^2}$$

$$\vdots$$

$$w^{t+1} \leftarrow w^t - \frac{\eta}{\sigma^t} g^t \quad \sigma^t = \sqrt{\alpha(\sigma^{t-1})^2 + (1-\alpha)(g^t)^2}$$

Fig.7. figure by H.-Y. Lee [22].

5. Long-Axis to Short-Axis Image Coordinate Transformation

The idea of the transformation from LA (2D) to SA (3D) space is quite straightforward. The LA image is a 2D image, but I can describe a LA(2D) point in a SA(3D) coordinate space as a vector (x, y, 0), as the value in the z dimension is zero. In order to transform a LA (2D) vector into the same coordinate system as the SA (3D) image, I will need to have the transformation matrix that can allow the 2 coordinate system to transform from one to the other. In this dataset, the transformation matrix is provided in the file header of each image. Each LA and SA image are matched in pairs, and each image has its own transformation matrix. The LA matrix and the SA matrix will transform the 2 types of images onto the common coordinate

system. However, the goal in this project is to transform the LA image to the SA space. Thus, I will have to multiply the LA point vector with the LA matrix first, and apply the inverse of the SA matrix in order to transform the point to the SA space. The mathematics of the transformation process can be written as below:

Step 1: transform LA image into the common coordinate (and SA the opposite)

$LA_c = f_{la}(LA)$

$SA_c = f_{sa}(SA)$

Step 2: transform the LA image into the SA coordinate (and SA the opposite)

$LA' = f_{la}^{-1}(LAc)$

$SA' = f_{sa}^{-1}(SAc)$

LA, SA=LA, SA image in its original coordinate

LA_c, SA_c=LA, SA image in the common coordinate

LA', SA'=LA image in the SA space, and vice versa

$f_{la}()$, $f_{sa}()$*=transformation matrix of LA, SA*

After the LA image is transformed into the SA coordinate, it intersects with each slice (according to the axis z) of the SA image by a line. Therefore, if I draw the LA image mask onto the SA image, and slice the SA image according to the axis z, the image will be a heart image with a line drawn on it as below:

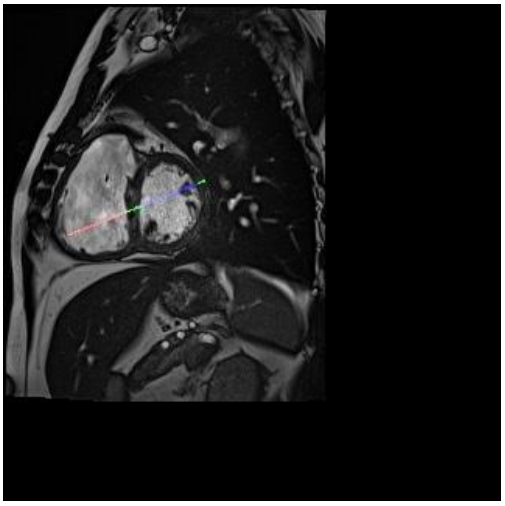


Fig.8. The SA image with the transformed LA line.

This line clearly defines the border of each class of a heart. In this way, it might help boost the accuracy when training the model since the right ventricle (RV)'s shape is highly variable and complex. With the help of the line, it might be easier for the model to tell the border of the right ventricle (RV).

6. Image Concatenation

There are many options for combining 2D and 3D images. Since the LA image transformation is done, I now have the SA image and the predicted LA mask on the SA coordinate system. At first, I tried to directly draw the transformed LA mask onto the SA image. However, I have found out that the value range of the SA image is too large (sometimes more than 4000), while the predicted LA mask ranges from 0 to 3 (cause there are four classes). If I draw those LA values onto the SA image, then the effect could be extremely small or even none since those values are so small that the model could not even detect them. In order to solve this problem, I will have to normalize the transformed LA mask and the SA image separately. However, since the transformed LA image is drawn on the SA image, it is a bit complicated to normalize them separately when doing transformation. Therefore, I choose not to directly draw the transformed LA image onto the SA image. Instead, I first initialize an empty space (a zero tensor) with the same size as the SA image. I transform the LA mask onto this empty space, and then concatenate the space with the SA image by the channel dimension. Since the original

images are all grayscale with 1 channel, the new concatenated images are now 2 channel 3D images. In order to input the 2-channel images into the 3D Unet, I also need to set the 3D Unet input size as 2 channels while the output remains 4 (since there are 4 classes). In this way, I can now normalize the transformed space and the original SA image separately.

Another point that has affected the training a lot is now to normalize the concatenated image. At first, I did not think about normalizing the concatenated image because when I input the data I already normalize the images already. Thus, the values of the original SA image range from 0 to 1, while the transformed LA image ranges from 0 to 3. This has slightly affected the training accuracy (slightly lower than the original 3D Unet training). Then, as mentioned before, I at first normalize the concatenated image together, and the accuracy does not improve at all since the transformed LA values are too small compared to the values on the SA image. Thus, I tried to normalize the transformed LA space and the SA image separately, and the result has improved significantly (increase from about 77% to 85%).

7. Loss Function Selection

There are a few commonly used loss functions in image segmentation tasks. In this project, I mainly consider using cross entropy and dice coefficient since they are the most common losses used in image segmentation. First of all, cross entropy (CE) is widely used in various machine learning tasks. It is very intuitive and is better for training optimization

compared to other ways of loss calculation such as mean square error (MSE). It can be written mathematically as

$$e = -\sum_{i} \widehat{\boldsymbol{y}}_i ln \boldsymbol{y}'_i$$

The e=the cross entropy value, y head=label, y'=the predicted value. The goal of training is to minimize the cross entropy value, which is proportional to the difference in values of y head and y'. However, it might be a bit confusing when it comes to calculating the cross entropy in an image segmentation task, since the label and the output are images. I would like to explain calculation of cross entropy in image segmentation with the figure below:

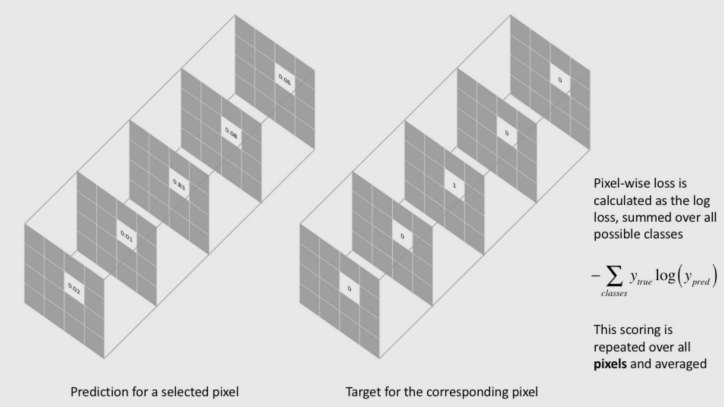


Fig. 9. Image by J. Jordan [23].

In figure 9, there are 5 channels in the image and the mask (Target), which means that there are 5 classes to be segmented in this task. The calculation of cross entropy is calculated for each pixel in an image, and each pixel has 5 channels. Therefore, we can view the 5 values into a vector, and the target is a 1-hot vector, which classifies each pixel to 1 class. At last, we can calculate the cross entropy between the 2 vectors. After calculating the cross entropy of all pixels, we can average the values and get the final cross entropy value.

As for the dice loss, it is also very easy to understand. The basic idea is to calculate the intersection between 2 subjects, let's say A and B. The equation can be written as

$$\text{Dice score} = \frac{2 \cdot |A \cap B|}{2 \cdot |A \cap B| + |B \backslash A| + |A \backslash B|} = \frac{2 \cdot |A \cap B|}{|A| + |B|}$$

The idea is that when the overlap between A and B is very large, the Dice score will approach 1, until they overlap completely with each other, while if they do not overlap with each other, the dice score is 0.

The implementation of dice loss in image segmentation is quite different from cross entropy.

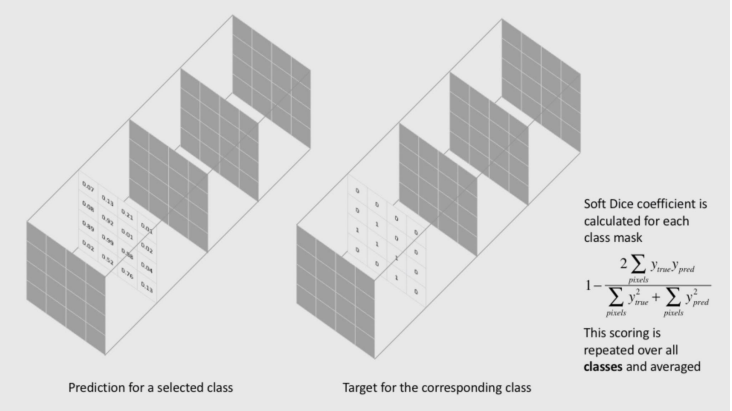


Fig. 10. Image by J. Jordan [23].

When calculating cross entropy, we first calculate the ce values of each pixel (with all 5 channels), and then average over all pixels. However, in dice score, we first calculate the dice score according to each channel, and then average the scores of all 5 channels (as shown in figure 10.).

Overall, it is hard to say which loss function is better than the other. However, because I have used dice score as the evaluation standard, I am afraid that using the same standard as loss function would keep me from

evaluating the result comprehensively and objectively. Therefore, I decided to use cross entropy as the loss function in this project.

8. Evaluation Metrics

In the MnM2 challenge, there are 2 evaluation metrics used for evaluating the results of each model. The first metric is the dice score, which has been introduced in the “Loss Selection” section earlier. Although the implementation of dice scores might be different in loss function and evaluation (whether to require gradient or not), the concept should be identical. The other metric that has been adopted in the MnM2 challenge is hausdorff distance.

Hausdorff Distance is used to measure the distance between 2 different sets. For example, given 2 different sets X and Y, we can use hausdorff distance to calculate how much distance, or how much do we need to expand one set to completely cover the other set. There are 2 ways to calculate the distance. The first one is by calculating the distance the set X needs to expand, while we can also calculate the opposite way. There are a few options to decide the final hausdorff distance between the 2 values. For example, the average Hausdorff distance chooses the mean of the 2 values as its final result. In this project, however, I choose the maximum of the 2 values as the final distance because in the original paper by Li et al [16], the authors did not clearly indicate which methods they use, so I would presume they use the maximum hausdorff distance to maintain the high standard of this project. This idea can be mathematically expressed as

$$d_{\mathrm{H}}(X, Y) = \max \left\{ \sup_{x \in X} d(x, Y), \sup_{y \in Y} d(X, y) \right\}$$

or d_H denotes the hausdorff distance.

However, during the training, the Hausdorff distance codeI have used behaved quite strangely. In the first few epochs of the 2D model training, it keeps indicating that the distance is infinite, but after about 50 to 100 epochs, it would suddenly behave normally. On the other hand, there's no such error when training the 3D model. Although I can still get the final values of the hausdorff distance after training, which is quite similar to the results reported in the paper [16], I am not able to explain this phenomenon even after taking a look into the code. Thus, in order to ensure the accuracy of the results in this project, I decided to use only the dice score as the evaluation standard.

9. Image Visualization

It is of great importance that I visualize the images during each stage of the whole process. The reason is that only when I visualize the images will I be able to know whether I am on the right track. For example, when inputting the dataset, I need to first take a look at the image to understand what I am dealing with. I would like to extract some sample images at each stage to help understand what has happened at each stage. First of all, the LA image and the one slice of the SA image look like the figure 11.

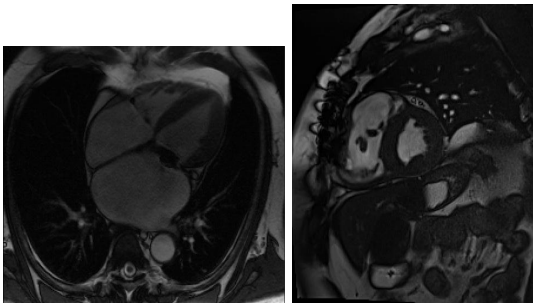


Fig. 11. LA image(left) and sliced SA image(right).

They are actually the same heart but are sliced in different angles by the magnetic resonance scanner. Also, since the SA-view image is in 3D, it is impossible to visualize it as a whole. Therefore, I choose to slice a piece of image from the SA image so that it can be shown as a 2D image. After putting the dataset into the data loader, I will first do some image augmentations on the images. Below are some augmentations on the LA images:

Fig.12. GaussNoise.

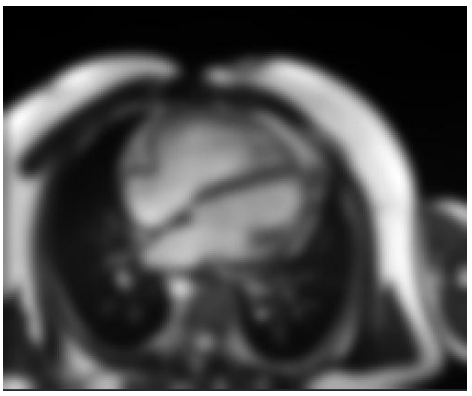


Fig. 13. Gaussian Blur

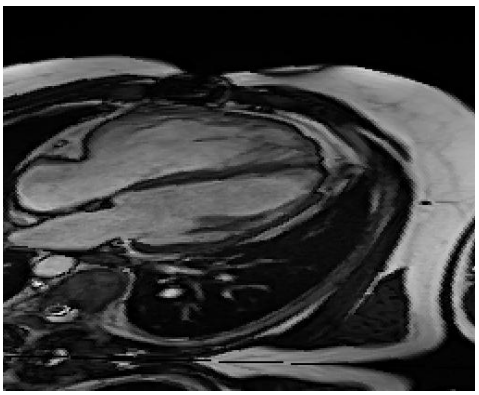


Fig. 14. Elastic Transform

These are just some examples of some augmentations that I have applied on the training set images. With the help of visualizing those images, I would be able to truly understand what and how much effect is applied on the images so that I could control those factors when training the model.

When I was experimenting how to transform the LA mask onto the SA coordinate, visualizing the image is also extremely important, since it's the only way that I would know whether I've transformed the LA image onto the right place. For example, if transformed correctly, the 3 classes (RV, LV, MYO), shown as 3 colors, of the transformed LA mask (see the right image of Fig.3) should match the right place in the SA image. However, when experimenting how to pad the images, the 3 classes are obviously in the wrong place. Thus, with the help of visualizing it, I would know whether I am on the right track.

At last, after finishing training the model, although the researchers can evaluate the accuracy by some standards such as dice coefficient or IoU (Intersection Over Union), it is still very important to visualize the results so that the researchers could intuitively understand how good the results are. Below are some samples of the model output.

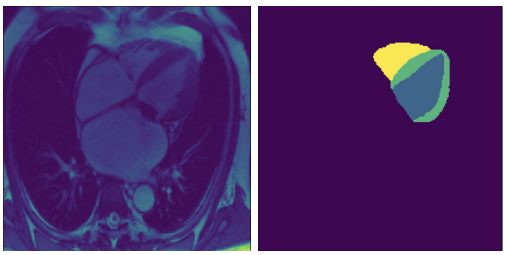


Fig. 15. The results of the LA mask predicted by the 2D Unet.

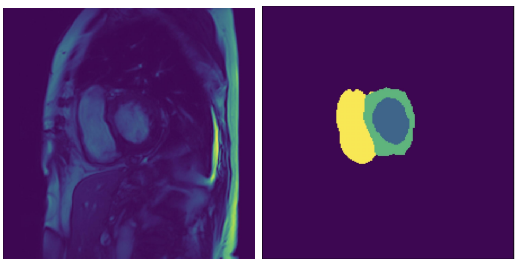


Fig. 16. The results of the SA mask predicted by the 3D Unet.

10. Memory Space

The “Cuda out of memory” is the problem that I have encountered most often when working on model training. When I was training the 2D Unet, even if I already set the batch size to be 1, it still kept showing that Cuda was out of memory error. Thus, I was sure that the problem does not lie in the batch size of the dataset since the image size is very common in other computer vision tasks. I checked line by line of the training process, the thing that could take a lot of memory during training is the losses record and the evaluation record in each epoch. Therefore, I checked the types of the losses and the evaluation results. It turns out that the types of the 2 records are torch.cuda.tensor, instead of an integer. Saving these records as tensors will take a very large amount of memory, which results in the cuda out of memory problem. Therefore, when saving these records, I first use the .item() function provided

by Pytorch to set these records as integers before appending them into the array.

As for training the 3D Unet, because the file size of a 3D image is much larger than a 2D image, so it's very natural that I may encounter the out of memory problem. I first require 32g memory space and 2 GPUs, and then set the batch size to only 10 to see whether it would work. However, 10 is still too large for 1 batch. I try to reduce the size 1 by 1, until it works. It turns out that 6 is the largest size that I can put into 1 batch. Thus, although a smaller batch size means that it might take more time to train, overall the training pipeline works fine and the mechanism (such as batch normalization) inside the model also works with this batch size.

At last, I intended to train the whole pipeline (2D Unet, transformation, concatenation, and 3D Unet) together so that this could be an end-to-end network. However, there are 2 main problems when working with this pipeline. First of all, this training process takes a very long time. It can only train 1 epoch in about an hour, since it needs to do the training, prediction, and transformation in each batch operation. The other problem is that when it comes to training the validation set, the cuda out of memory occurs. Therefore, since I have not figured out how to speed up the whole training process and have no access to more cuda memory, I decide to train each part separately instead of an end-to-end way.

11. Domain Adaptations

Due to the multi-domain nature of this dataset, I believe it is worth trying to apply domain adaptation techniques on the preprocessing, in addition to the

basic augmentation techniques. However, the most challenging point is that there are more than 2 domains in this dataset, while most papers of domain adaptations only discuss the transformation between 2 domains. Therefore, I mainly consider 2 multi-domain adaptation techniques in this project.

The first paper I had considered is CyCADA by Isola et al [23]. CyCADA is a domain adaptation technique that's based on the idea of a CyCleGAN [24]. The idea of CyCleGAN is to train 2 generative adversarial networks G and F (as shown in figure 17). The network G is to transform the image in X domain to Y domain, while the network F does the opposite (from Y domain to X domain). The Discriminators X and Y are used to encourage the transformation between domains (distinguish whether the transformed images are similar to the new domain images enough). The reason that CyCleGAN first transforms from X to Y then Y to X is that the author wants to make sure that after transforming between domains, they can still ensure that the images being transformed are consistent (or the networks might simply transform an image into a random image in another domain).

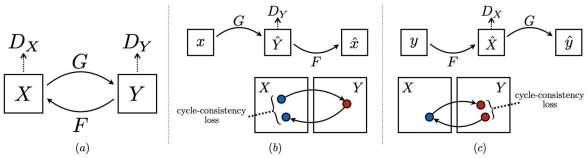


Fig. 17. CyCleGAN [23].

Based on this idea, CyCADA further sets a semantic consistency loss between the source image and the image that has been transformed twice (from source->target->source domain) back to the source domain. This makes sure that during the transformations between domains, the semantics of an image remains intact. What's more, this method can be extended to be used

in multi-domain dataset, in theory, which makes it suitable for the MnM2 dataset. The whole process of the CyCADA is shown in figure 18.

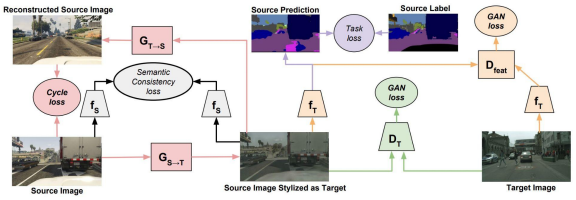


Fig. 18. CyCADA [23]

However, there is a significant disadvantage of using GAN-based methods, which is that these methods are very computationally expensive. A basic GAN itself consists of 2 networks: the generator and the discriminator. The CyCleGAN is based on 2 opposite GANs, so there are 4 networks to train in a basic CyCleGAN. As mentioned above, in addition to the basic CyCleGAN, there's one more semantic consistency loss in CyCADA that has to be calculated. The number of networks is only for the transformation between 2 domains. This MnM2 dataset contains 3 domains in total, which means that the number of networks have to be doubled. Therefore, with the limited time, I choose not to implement this method in this project.

Due to the reasons mentioned above, I chose to implement the Fourier Domain Adaptation (FDA) proposed by Yang et al [15]. This is a non-adversarial, non-neural networks technique that is much more time and effort efficient, and more practical in the real world where data is collected from multiple domains. In order to understand how this method works, first I need to understand Fourier Transform and its application on image processing.

Fourier Transform is a technique that can transform signals sampled in time or space into its temporal or spatial frequency. The image data can be

viewed as 2-dimension signals in the spatial domain. The fourier transform on an image can be mathematically expressed as

$$\mathcal{F}(x)(m,n) = \sum_{h,w} x(h,w) e^{-j2\pi\left(\frac{h}{H}m + \frac{w}{W}n\right)}, j^2 = -1$$

for x = image, and (m,n) represents the value of each point. Thus, I can use Fourier Transform to transform an image into its frequency domain. The purpose of getting the frequency information about a signal (image in this case) is that we can take advantage of the characteristics of the frequency domain to achieve domain adaptations. Fig. 19 has shown the process of Fourier Transform and Inverse Fourier Transform, which transform it from frequency domain back to spatial domain.

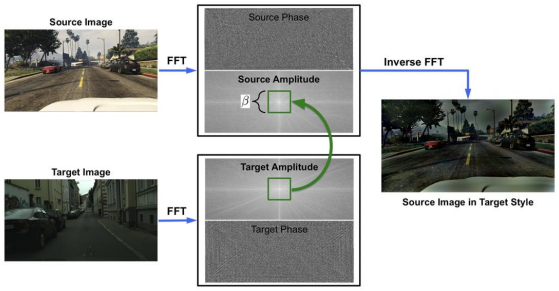


Fig. 19. Fourier Domain Transform [15].

Then, the key technique in FDA is the swapping of the low frequencies of the source image amplitude and the target image amplitude. The frequency of an image represents how drastically the pixel values change in a specific span of space. For example, the high frequency means that the values change drastically, and thus the high-frequency part is responsible for the semantic of an image, which looks obviously sharper compared to the background. On the other hand, the low frequency part will look smoother and have not much impact on the semantic of an image. Thus, when changing the low frequency of an image, it won't affect the semantic of an image. Instead, it will change

the pixel distribution of the smooth parts. Therefore, the idea of Fourier Domain Adaptation is to copy the low frequency of an image from the target domain, and use it to replace the low frequency of the image in the source domain. In this way, it can reduce the difference in the pixel distribution of images in different domains, and allow the deep neural networks to focus on learning from the semantics of the images. Figure 16 is a sample of the source image and the image after the Fourier Domain Adaptation. From the image on the right it can be observed that the semantics of the image are identical from the original image. However, it is also obvious that the pixel distribution between the 2 images are different due to the swapping of the low frequencies with the image in the target domain.

In this project, however, because I use padding to make all images the same size, the padding space would affect the low frequency of an image. Therefore, I choose an LA image (141_LA_ED.nii.gz) with the size that's larger than 448*448 as the target image. In this way, I can make sure that I will not need to pad this image when making all images to the same size in training. In addition, for the 3D training, I first use Fourier Domain Adaptation to transform all the SA images, and then concatenate them with the transformed 2D output information.

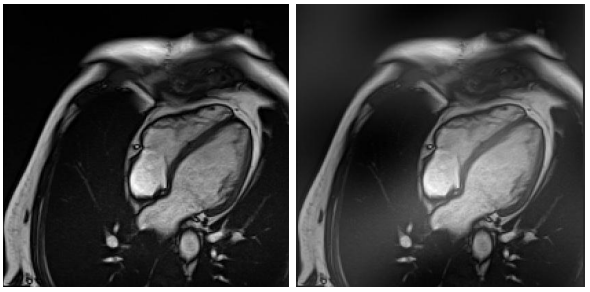


Fig. 20. The source image (left), and the source image after the FDA (right).

12. Transfer Learning

In this MnM2 dataset, the number of images and target masks is quite limited in terms of training a deep learning model. Although simply training a basic Unit has achieved a pretty good result in the dice score, the result of hausdorff distance is still not satisfactory compared to the original paper by Li et al [1]. Therefore, in order to increase the generalization ability of the model, I tried transfer learning, pre-training the model on the Medical Segmentation Decathlon dataset.

In human learning, people usually borrow similar previous experiences when acquiring new skills or knowledge, which can also be the case when training the model. The general implementation of transfer learning is that we pre-train the model on datasets that have similar domains or similar tasks. Then, we train it with the target dataset to fine tune the parameters of the model. The idea of transfer learning is that when pre-training the model on similar tasks or domains, the model can learn some general features and patterns of other similar data, which could improve the generalization ability of the model. In this way, compared to training a model with randomly-initialized parameters, it could reduce the risk of overfitting and the time for training.

In this project, I choose to use the Medical Segmentation Decathlon dataset to pre-train the Unet and 3D Unet. This is a 3D medical image dataset that includes 10 organs. There are a few reasons why I choose to pre-train the model on this dataset. First of all, this dataset that includes 10 datasets of different organs are all medical images in the domains that are very similar to

the MnM2 dataset, which is very suitable for transfer learning. The second reason is that the size of this dataset is very large. The sources of medical images are very limited, let alone labeling these images pixel-by-pixel for segmentation tasks. Therefore, it is very rare to have access to such a large medical image dataset. Lastly, the versatility of this dataset is significant. Originally, I hoped to find an already pre-trained Unet and 3D Unet from other sources and directly train on the MnM2 dataset. However, most of the pre-trained models on github have only trained on certain tasks or organs. What is worse is that I have adjusted the 3D Unet since the Z dimension of the MnM2 SA(3D) images is only set to 8, so that it's impossible to find a pre-trained adjusted version of the 3D Unet model from github. Therefore, I chose the Medical Segmentation Decathlon for pre-training the model.

However, since this is a 3D medical image dataset, it requires some tricks to use it to train the 2D Unet model. During training, I use a loop to calculate the loss slice by slice of a 3D image. In this way, I can make each slice a 2D image and train the 2D Unet. Also, when slicing the 3D images into 2D slices, there tends to be many empty masks in the first few and last few slices. In order to reduce the training time (since I am using a loop, which is very time-consuming) and memory space, I skip those slices when preprocessing the data. In this way, I can also avoid the model predicting all the results as empty tensors because in some cases that the number of empty slices is much larger than the number of slices with target class.

After pre-training the 2D Unet model on the Medical Segmentation Decathlon dataset, I directly use the whole pre-trained model on the MnM2 dataset. However, the training on the target dataset behaved weirdly. The loss

and dice score had completely stuck from the beginning, even after about 150 epochs of training. I have not been able to figure out the reason behind this phenomenon. In order to overcome the stagnation, I experimented with only saving the pre-trained parameters of certain layers of the Unet encoder (down-sampling layers), and training the decoder layers from scratch. When only keeping some layers of the encoder parameters, the loss and dice score were still stuck at the beginning. However, after some epochs, it would break the stagnation and start training normally. I also observed that the more layers of pre-trained parameters I keep, the more epochs it would take in order to break the stagnation. Although I am still not able to explain this phenomenon, I would presume that even if I want to keep the whole pre-trained model, it would still work but would have to take hundreds of epochs. In this project, due to limited time, I only experimented saving layers of the encoder.

13. Transformer

Transformer was proposed in 2017, and has achieved state of the art performance in many Natural Language Processing tasks. Therefore, many researchers have been trying to apply the similar mechanism on computer vision tasks. The core part of the transformer is the attention mechanism. It aims to make each unit of the input to attend to every other unit. In this way, the model can consider the whole context of the input before predicting the output.

In the application of transformers on computer vision tasks, a naive way is to make each pixel attend to every other pixel. However, as Alexey

Dosovitskiy et al [25] points out in their paper, this method is computationally expensive and requires a huge memory space due to the amount of parameters required, which means that this method won't be able to scale well. Thus, in the paper [25] by Alexey Dosovitskiy et al, they propose the Vision Transformer model that fixed the problem. Compared to language model BERT, whose input is the 1D sequence of tokens, the Vision Transformer reshape the 2D image into a number flattened small patches as its tokens. In this way, it would allow the model to scale since it has drastically reduced the number of tokens.

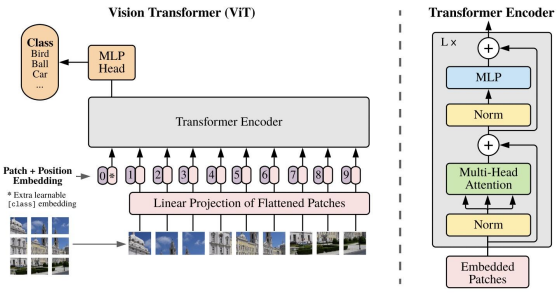


Fig. 21. Vision Transformer [26].

However, the vision transformer model is designed for classification tasks, instead of segmentation tasks, which requires pixel-level prediction. In the paper by Chen et al [27], they proposed a model that combines the Unet structure and the Vision Transformer mechanism to achieve the image segmentation tasks.

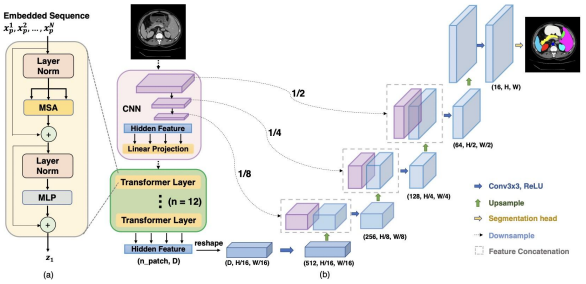


Fig. 22. TransUnet [27].

Instead of simply using the basic Unet, the researchers introduce the vision transformer after the downsampling layers. As the authors pointed out in the paper, it's actually possible to directly upsample the vision transformer layers. However, because the upsampling is so aggressive that the output will lose much detail information of the original image. Therefore, with the skip connections of the Unet, it can add in detailed information for the output and thus boost the accuracy of the result.

However, there are also some disadvantages of training a model with a transformer. The transformer mechanism has increased the model capacity, which means that it allows the model to overcome much more complex tasks. But on the other hand, it also means that the model with great capacity will require more data in order to train well. If there is only limited data available, the model will very likely encounter overfitting during training. Unfortunately, the data of medical images tends to be limited, let alone the labeled target masks. In this project, the MnM2 dataset contains only 360 patients, and about 720 2D (LA) images to train the model. It is very small in size compared to the dataset in the TransUnet paper (3779 CT scans). Therefore, when I train the TransUnet on the MnM2 dataset, it only achieves about 88% on dice score (compared to original training at 91%). What is worse, there's a very number of versions of 3D TransUnet implementation on github, and since the 3D Unet in this project is adjusted so that it can adapt to the limited z dimension, it could require quite some effort to adjust the TransUnet structure to fit for this dataset. Thus, I gave up on training the 3D TransUnet due to

limited research time and the size of the dataset. However, I think it is still a topic that's worth exploring in the future.

Results

In the result section, I will report the results according to different stages. First of all, I trained both 2D and 3D networks using the basic Unet model with some augmentations. After training the basic model as the baseline in this project, I explored different methods that are meant for boosting the accuracy of the model. I experimented with fourier domain adaptation (FDA), transfer learning, and transformer mechanism. At the end, I compared these results with the baseline in order to tell which techniques are able to improve the original result. Also, in this project I mainly use Dice Coefficient as the evaluation metric, although I also implemented the Hausdorff Coefficient in this project (since these are the 2 metrics that are used in the original MnM2 challenge) for reference. I will explain the reason in the last Hausdorff Distance section.

1. Basic 2D Unet

	2D Unet (4-classes)	2D Unet (1-class, RV)	2D Unet (1-class, RV) with augmentations
Train Dice	0.91	0.91	0.88
Validation Dice	0.85	0.85	0.88
Test Dice	0.83	0.83	0.88

The first result is the basic Unet, and the goal is to segment all 4 classes (RV, LV, MYO, background) provided in the dataset target mask. The dice score of the train set reaches 91%, while the validation and test set reaches only about 85% and 83%. As for the loss, the train loss has been lowered to 0.01, while the validation loss is still 0.2 (and so is the test loss). I think this indicates that the training has encountered the problem of overfitting. Thus, later I explored some more augmentations on the dataset to see whether I could further boost the accuracy of the validation set.

After the training of the 4-classes segmentation, I am curious about whether reducing the number of classes could raise the accuracy to a higher level. Thus, I adjusted the original target mask. I set the values of 1 and 2 on the tensor to be 0, which is the value of the background. In this way, there are only 2 values on the mask tensor, 0, the background, and 3, the target RV. Also, I had to set the channel number of the model output to be 2, which means that the output will be a 2-class tensor, and the number of classes of the evaluation function had to be set to 2, instead of 4. The result of the single class training, however, is barely better than the multi-class training. The loss is slightly lower than the multi-class model, but the dice coefficient score is almost identical to the multi-class version. Thus, the number of classes of tasks does not have a significant impact on the prediction accuracy.

Since both versions of the training all suffered from the problem of over-fitting, I explored some more augmentations in order to enhance the generalizability of the model. I add gaussian blur, elastic transform, and horizontal flip to the augmentation list. I set each of the application probability to 0.6. I have experimented with some other techniques such as random

noise, gaussian noise, random blur, etc, but the results are not as good as the combination I have introduced above. The training dice score drops a little bit from 91% to 88%. However, the validation dice score rises from 85% to 88%. The loss data also shows a similar trend. This result indicates that augmentations can actually solve the problem of overfitting during training. However, it takes quite some time and effort to figure out what is the better combination of augmentation techniques. On one hand, I hope the augmentation is strong enough to solve the overfitting problem; on the other hand, the augmentation should be limited so that it still allows the model to learn the key features.

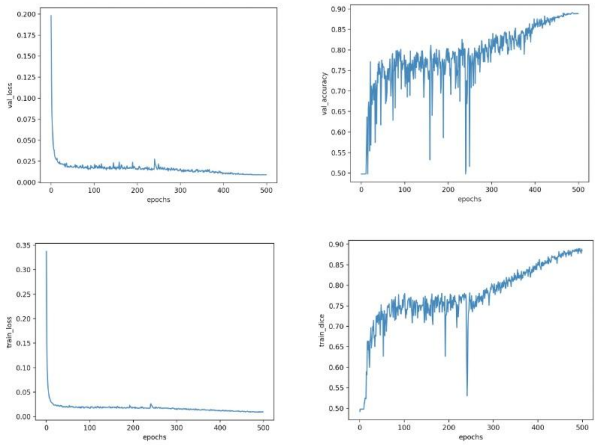


Fig. 22. The Loss data and the dice score of the final result of 2D training

2. Basic 3D Unet

	3D Unet	3D Unet with 2D Info (3-class)	3D Unet with 2D Info (3-class) and aug	3D Unet with 2D Info (1-class)
Train Dice	0.85	0.88	0.65	0.91
Validation Dice	0.84	0.85	0.75	0.91
Test Dice	0.84	0.84	0.74	0.91

As for the 3D model, I first tried to train a 3D Unet without any 2D information, the result is pretty good already even without any augmentations. Both train and validation loss have reached a pretty low level, and the accuracy of both sets are similar (85% for train set and 84% for validation set), and so is the test set. This indicates that the training set has converged successfully and the overfitting problem is not obvious. In this situation, I prefer not to add more augmentations to the dataset in order not to reduce the accuracy of the training set.

Secondly, I add the 2D information (after transformation) to the 3D data, and train the 3D model again, to see whether the information could help boost the accuracy. The result seems slightly better. The dice score on the validation set reached 1% higher than the previous result, while the loss is almost identical. However, the train set is much better than the previous one. The dice score reached 88% (compared to the previous 85%), and the loss reached about 0.01 (compared to the previous 0.02). This indicates that this training result has encountered overfitting. Hence, I train another 3D model with slightly more augmentation (add random noise to the images). However, the result has obviously failed. Although the overfitting problem is resolved (the validation result is much better than the training result). However, the result of the train set is much lower than the previous version, so the validation result is also worse than before. So far, I have not found a balance between overfitting and training convergence, but I believe, in theory, when solving the overfitting problem, the result could still be better than the score I have got.

At last, I reduced the number of classes on the target mask from 4 to 2 (only the RV and the background) to see whether this factor could further improve the result. Different from the 2D Unet, when I reduce the number of classes, the result has improved significantly on both the train set and the validation set. The dice score on all 3 sets reached 91% compared to 85%, while the train loss reached 0.005 compared to 0.02. This result is a bit surprising for me, and there's no obvious overfitting problem in this training.

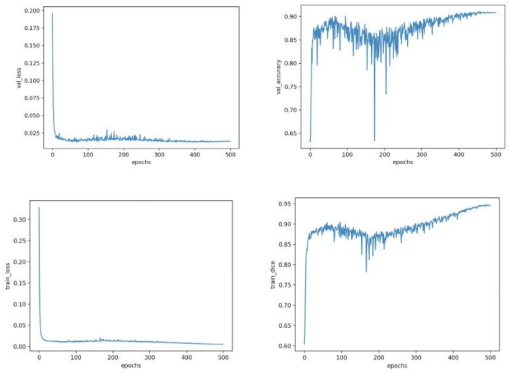


Fig. 23. The loss and dice score of the 1-class 3D Unet training.

3. Fourier Domain Adaptation (FDA)

The results of the training with FDA are shown as below:

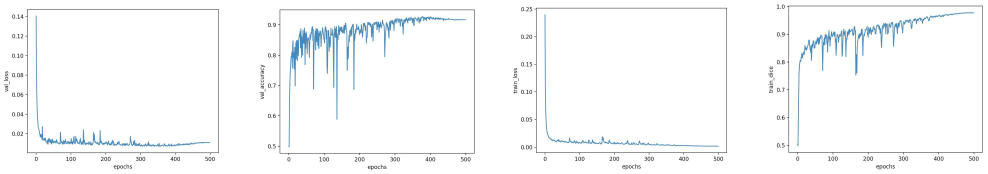


Fig. 24. The loss and dice score of the 2D Unet training with FDA

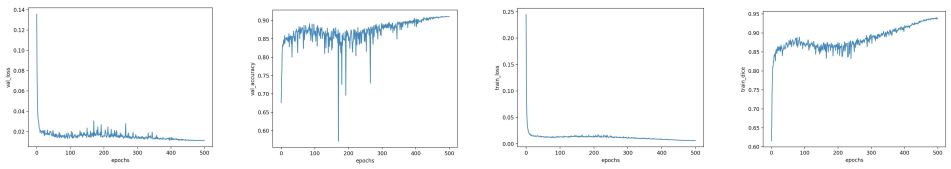


Fig. 25. The loss and dice score of the 3D Unet training with FDA

After applying fourier domain adaptation to preprocess the LA (2D) images, the training result has improved significantly. The original dice score for basic 2D Unet on validation set was only about 0.88. However, with FDA, the dice score has reached about 0.92 stably during training, which is slightly higher than the dice score reported in the paper by Li et al [16]. The dice score on the training set is about 0.94, so I tried to apply more harsh augmentations on the images. However, the result is even worse on both training and validation sets.

On the other hand, after applying the FDA on the SA (3D) images, the result is almost identical to the original basic 3D Unet. The dice scores on validation set for both are 0.91. Although I am not certain about the reason for this result, I infer that the reason might be because the 3D images contain more context information, which allows the model to learn the features more easily. Thus, even without the FDA, it has reached a very good level already (about the same level as the results of the paper [16] at 0.91), and the FDA might not be able to further improve the performance.

4. Transfer Learning

The results of the training with pre-trained model are shown as below:

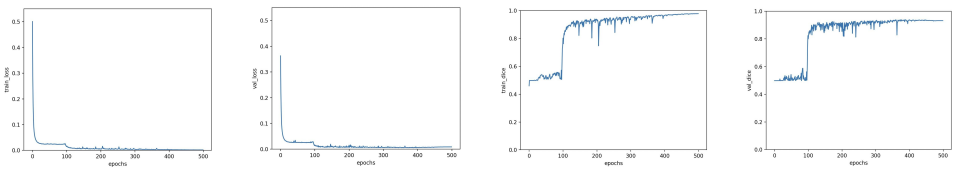


Fig. 26. The loss and dice score of the 2D Unet training with pre-trained parameters

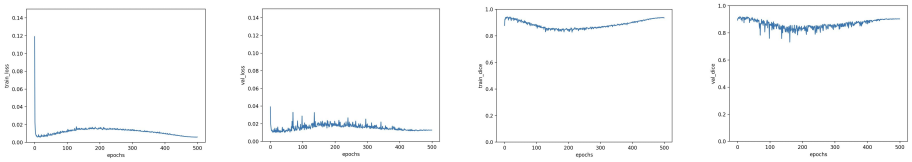


Fig. 27. The loss and dice score of the 3D Unet training with pre-trained parameters

When applying transfer learning to the 2D and 3D model, it has a significant impact on the accuracy and training efficiency. However, there are some unexpected phenomena when training the MnM2 dataset with the pre-trained models.

After I have finished pre-training the 2D model, I applied the whole pre-trained 2D model on the MnM2 dataset. However, it is very strange and surprising that the training is completely stuck and not training at all. I was not able to understand or explain this phenomenon. I tried to make the maximum learning rate larger and smaller (the original maximum learning rate is 0.01), but it did not make any difference. I was not sure about the reason behind this, but I was sure that the training procedure is correct, and the key factor should be in the pre-trained model. Therefore, I tried only keeping only certain layers of the pre-trained parameters (adding one layer at a time). When keeping the layers of the Unet encoder (downsampling layers), the training still behaves weirdly. At first, the training was still stuck, and the dice score did not change for about 90-100 epochs. However, at about the 100th epoch, the dice score would suddenly skyrocket to about 90% within 20-30 epochs. This happens no matter how many pre-trained layers of the encoders I have kept. Since I was unable to comprehend why this happens, I turned to look at the loss graph during training. The loss behaves pretty differently from the dice score. It has indeed drastically lowered at the first few epochs. However, it would be stuck after the drastic downward trend for a while, until at about the 100th epoch, and the loss begins going down again. The second downward

trend of the loss matches the upward trend of the dice score. Although I am still unable to explain this, I presume that there are some factors of the pretrained model that have caused this, but after a certain point, when the criteria is reached, the training would begin quickly and smoothly. Actually, when I kept the first 4 layers of the pre-trained encoder, the dice score reached 93.9%, which is significantly better than the original training at 91% (when applying FDA).

However, after applying the pre-trained decoder, the training was still stuck with the original one-cycle scheduler and maximum learning rate at 0.01. After discussing with my supervisor, I was suggested to use the cosine annealing scheduler to replace the one-cycle schedule and setting a much larger learning rate. The reason for these changes is that the training was stuck, probably because the learning rate is not large enough to reach the criteria where the pre-trained model would begin training. Therefore, setting a much larger learning rate maximum would help reach that criteria. Also, the reason for replacing the one-cycle scheduler is that the initial learning rate would begin from 0 and reach the peak between 100-200 epochs (as shown in figure 19). The learning rate at the first 100 epochs might be too small so that the dice score was unable to be moved. Thus, the cosine annealing scheduler (as shown in figure 20), which begins with the maximum learning rate, might allow the upward trend of the dice score to begin earlier. In fact, after setting the maximum learning rate at 0.5 with cosine annealing, I was finally able to train the MnM2 dataset with the whole pre-trained model, even though the dice score was still stuck for a while. However, the final result of using the whole pre-trained model is not as good as I had expected. The final dice score is

only about 87%, which is obviously worse than using only the pre-trained encoder at about 93%. Despite this unsatisfactory result, I would assume that, in theory, using the whole pre-trained model could have the best performance. It is just that in this project, I was unable to find the appropriate combination of hyper-parameters that would allow it to reach its best performance.

Except for the experiments mentioned above, I have also tried to adjust the timing of the peak of the one-cycle scheduled learning rate. I tried to make the learning rate peak earlier to see if that would allow the dice score to work earlier instead of stuck for about 100 epochs. As shown in figure 21, I shorten the time for the warmup of the learning rate, and make it peak at the 50th epoch. This change has indeed allowed the dice score to begin rising earlier, but I have also observed that the training became unstable at first probably because the initial learning rates rose too fast. Also, the final result is about 89%, which is lower than other choices of scheduler hyper-parameters.

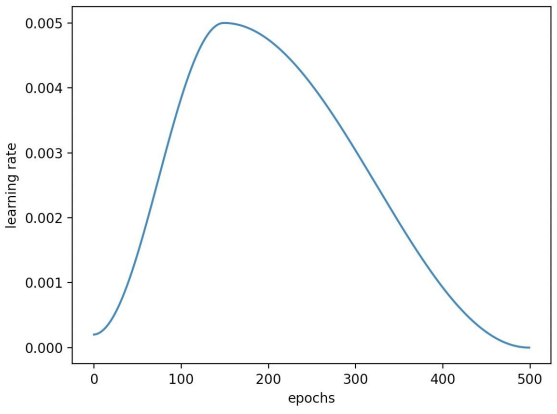


Fig. 28. One-cycle scheduled learning rate.

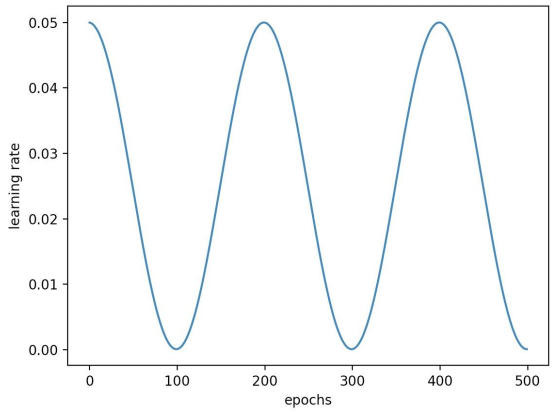


Fig. 29. Cosine-annealing scheduled learning rate.

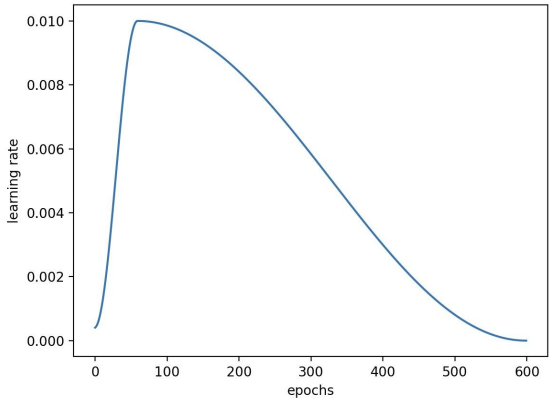


Fig. 30. One-cycle scheduled learning rate with the peak at the 50th epoch.

On the other hand, transfer learning behaves very differently on the 3D model. Training the MnM2 dataset has worked well from the beginning, and the dice score is able to reach 90% very easily within 10 epochs. However, it is likely that after about 30 epochs, it will likely peak before the 30th epoch with dice score at about 91%. As shown in figure 20, after the dice score reaches its peak, it slowly goes down to about 83% between 100th and 200th epochs, then it goes up again. At first, this seems to be quite an unexpected trend, but I think taking a look at the loss values would probably be able to explain this trend. In the 3D training, I use the one-cycle scheduler with the loss peaking between 100-200 epochs. This matches when the dice score

reaches its bottom and begins going up again. I reckon that it might be because when the learning rate is about to reach its peak, the value is a bit too high that makes the training noisy, which has shown in the downward trend of the dice score. Thus, although this phenomenon might seem to be a bit weird, I think the main reason might be that when using the pre-trained model, it has allowed the dice score to reach its peak so fast even before the loss reaches its peak, and the rest of the training seems superfluous. In conclusion, although using transfer learning on training the 3D model does not have much impact on the final result, it has allowed the model to converge much faster.

5. Transformer Mechanism for 2D Unet

Understanding that training a neural networks model with a transformer mechanism would require a large amount of data due to its larger model capacity, I was not expecting a much better result before training with the TransIent model. However, the results performed poorly, which was even worse than I hoped for. The training results are shown in figure 31:

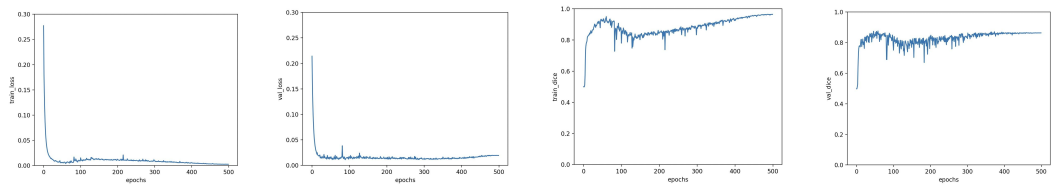


Fig. 31. Training with TransUnet.

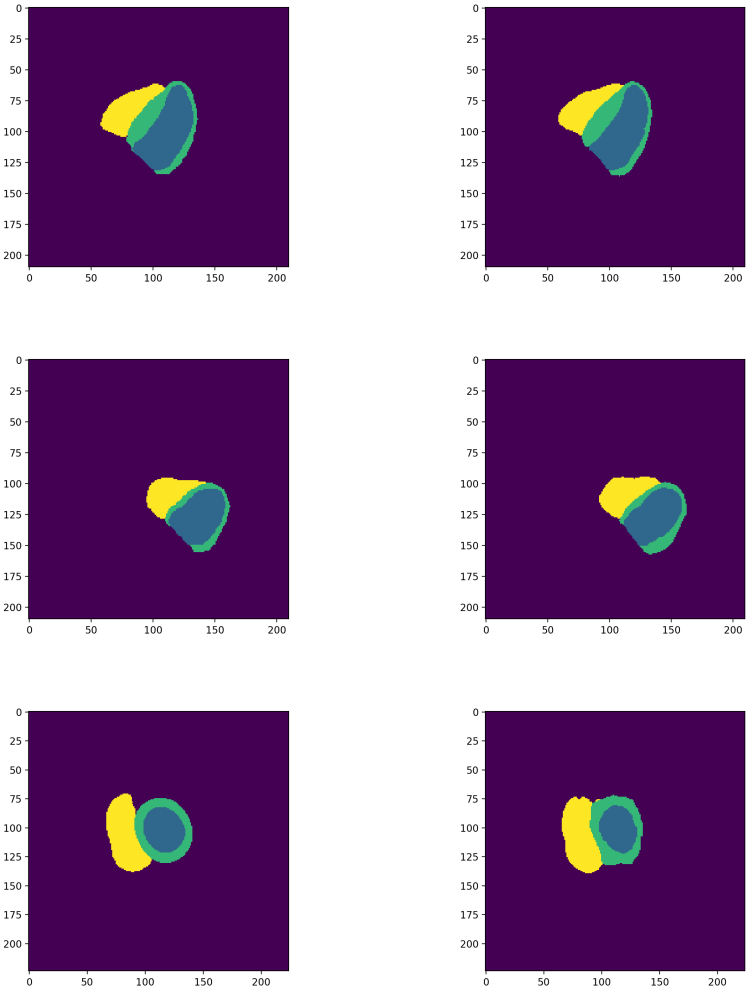
The result shown above is trained without any augmentations. Although it is clear that it is overfitting the training set (since the accuracy of the training set is obviously higher than the validation set), I was unable to find the appropriate augmentations that could effectively solve this problem. When I

add some augmentations to pre-process the dataset, it was likely to significantly impact the training set accuracy during training, which would lead to a even worse performance on the validation set. However, it is also possible that it was just that I have not tried many enough combinations of augmentations due to limited time. There might be an appropriate set of hyper-parameters that could make the TransUnet work well with the MnM2 dataset. However, in this project, the TransUnet performs much worse than the basic Unet.

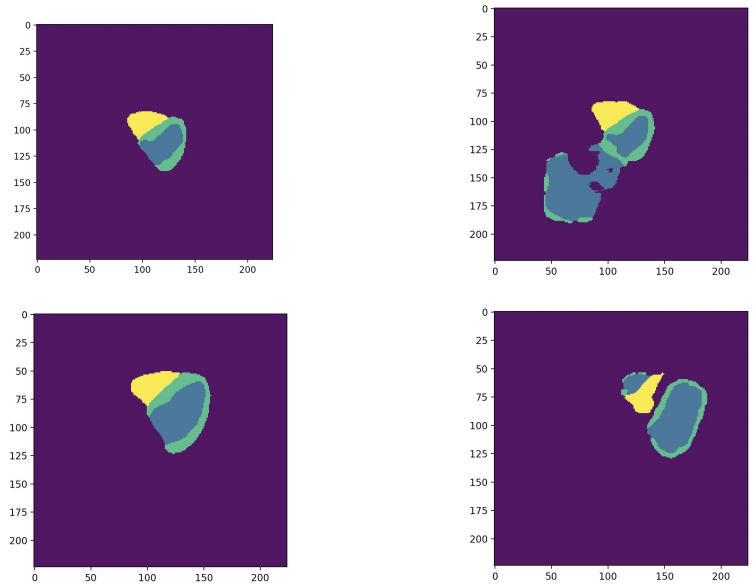
On the other hand, I did not apply TransUnet on the 3D model for a few reasons. First of all, due to limited amount of data, it performs poorly on the 2D model. Thus, I reckon that with the limited amount of data at hand, it is difficult to allow the TransUnet to fully reach its best performance, on both 2D and 3D model. The second reason is that modifying the TransUnet into a 3D version could be difficult. With the transformer, the structure of TransUnet is much more complex than the basic Unet. What is more, with only a few slices on the z dimension, I need to further modify certain layers of the networks. Considering the poor performance on the 2D dataset and limited amount of time, I think it might be a bit too costly to make these changes. However, I think it is still worth trying for further studies in the future.

6. Image Samples

After finishing all the training process, I have visualized the predicted results and the mask in order to have a more intuitive comparison. In most cases, the distance between the predicted output is actually very small to human eyes. The good samples are shown as below:
(left : mask, right : result)



However, there are still some cases that have gone very wrong. Below are some examples:



Although in this project I have not dived into the reasons behind those weird samples, I believe it is a very worth exploring topic in future studies because if able to recognize the reasons behind those failed predictions (although very few), researchers will be able to identify specific methods to tackle these problems.

7. Final Results Comparison

In conclusion, I have summed up the final results of the basic Unet, Unet with

Fourier Domain Adaptation, and Unet with transfer learning as below:

1. 2D Model

	2D Unet (1-class, RV) with augmentations	2D Unet (FDA)	2D Unet (FDA, Transfer Learning)
Train Dice	0.88	0.96	0.96
Validation Dice	0.88	0.92	0.939
Test Dice	0.88	0.91	0.92

2. 3D Model

	3D Unet with 2D Info (1-class)	3D Unet with 2D Info (FDA)	3D Unet with 2D Info (FDA, Transfer Learning)
Train Dice	0.91	0.93	0.94
Validation Dice	0.91	0.91	0.91
Test Dice	0.91	0.91	0.9

Conclusion

In this project, I reproduced the training pipeline of the paper by Li et al [1] and reached a similar accuracy reported in the paper. What is more, I have explored the fourier domain adaptation technique to tackle the multi-domain nature of this dataset. At last, I applied transfer learning in training the model.

In the basic Unet, the 3D network has already reached a similar accuracy level as the result in the paper by Li et al [1]. However, the result of the basic 2D Unet is much lower than the paper, even after I have tried various image augmentations such as gauss noise, elastic transform, etc. I would presume that with the appropriate combination of hyper-parameters, the basic 2D Unet could also reach a similar level as the nnUnet used in the paper, since the main difference between the 2 networks is not the network structure itself.

After applying the fourier domain adaptation to the dataset, the accuracy of the 2D network has been improved significantly (from 88% to 92% on the validation set, and 88% to 91% on the test set). Although this result has only allowed the model to reach a similar level as the original paper, it proves that fourier domain adaptation can boost the performance of the network. At last, after applying transfer learning, the accuracy of the 2D Unet has obviously outperformed the result in the paper, improving the dice score from 91% to 93.9% on the validation set. This also proves that pretraining the model on the datasets with similar tasks and domains is beneficial for image segmentation.

However, the 2 techniques do not seem to have much impact on the 3D Unet. I would presume the probable reason might be that the 3D image dataset has

provided more contextual information, which is enough to allow the basic 3D Unet model to reach high accuracy, and applying other techniques such as FDA and transfer learning would not have much impact. Nevertheless, the improvements of the accuracy on the 2D Unet have proved that these techniques are still very beneficial for training neural networks.

To sum up, the contributions of the research are that I have explored the impact of fourier domain adaptation and transfer learning on training the networks for image segmentation tasks, and the results have shown that they are helpful in training the 2D Unet, and although unable to further improve the accuracy of the 3D Unet, they do not have any negative impact in training. Therefore, I believe that these techniques are very worth applying when training deep neural networks. As for future research, I think it is worth further exploring the different performance between the 2D and 3D networks of the 2 techniques. Also, it is also worth exploring how to decide the combinations of hyper-parameters in training neural networks.

References

[1] L. Li, W. Ding, L. Huang, and X. Zhuang, “Right ventricular segmentation from short- and long-axis MRIs via information transition,” arXiv [eess.IV], 2021.

[2] S. Abdulateef and M. Salman, “A comprehensive review of image segmentation techniques,” Al-mağallaẗ al-‘irāqiyyaẗ al-handasaẗ al-kahrabā’iyyaẗ wa-al-ilikttrūniyyaẗ, vol. 17, no. 2, pp. 166–175, 2021.

[3] N. O. Mahony et al., “Deep Learning vs. Traditional computer vision,” arXiv [cs.CV], 2019

[4] L. Alzubaidi et al., “Review of deep learning: concepts, CNN architectures, challenges, applications, future directions,” J. Big Data, vol. 8, no. 1, p. 53, 2021.

[5] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” Nature, vol. 521, no. 7553, pp. 436–444, 2015.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” Commun. ACM, vol. 60, no. 6, pp. 84–90, 2017.

[7] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” arXiv [cs.CV], 2014.

[8] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” arXiv [cs.CV], 2015.
[9] J. Long, E. Shelhamer, and T. Darrell, “Fully convolutional networks for semantic segmentation,” arXiv [cs.CV], 2014.
[10] O. Ronneberger, P. Fischer, and T. Brox, “U-Net: Convolutional Networks for Biomedical Image Segmentation,” arXiv [cs.CV], 2015.
[11] F. Isensee, P. F. Jaeger, S. A. A. Kohl, J. Petersen, and K. H. Maier-Hein, “nnU-Net: a self-configuring method for deep learning-based biomedical image segmentation,” Nat. Methods, vol. 18, no. 2, pp. 203–211, 2021.
[12] J. Chen et al., “TransUNet: Transformers make strong encoders for medical image segmentation,” arXiv [cs.CV], 2021.
[13] J. Hoffman et al., “CyCADA: Cycle-Consistent Adversarial Domain Adaptation,” arXiv [cs.CV], 2017.
[14] S. Zhao et al., “Multi-source Domain Adaptation for Semantic Segmentation,” arXiv [cs.CV], 2019.
[15] Y. Yang and S. Soatto, “FDA: Fourier domain adaptation for semantic segmentation,” in 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020.
[16] V. M. Campello et al., "Multi-Centre, Multi-Vendor and Multi-Disease Cardiac Segmentation: The M&Ms Challenge," in IEEE Transactions on Medical Imaging, vol. 40, no. 12, pp. 3543-3554, Dec. 2021, doi: 10.1109/TMI.2021.3090082.
[17] Weiss, K., Khoshgoftaar, T. M., & Wang, D. (2016). A survey of transfer learning. Journal of BigData, 3(1). https://doi.org/10.1186/s40537-016-0043-6
[18] Hosna, A., Merry, E., Gyalmo, J., Alom, Z., Aung, Z., & Azim, M. A. (2022). Transfer learning: a friendly introduction. Journal of Big Data, 9(1), 102. https://doi.org/10.1186/s40537-022-00652-w.
[19] M. D. Zeiler and R. Fergus, “Visualizing and understanding convolutional networks,” arXiv [cs.CV], 2013.
[20] Ö. Çiçek, A. Abdulkadir, S. S. Lienkamp, T. Brox, and O. Ronneberger, “3D U-net: Learning dense volumetric segmentation from sparse annotation,” arXiv [cs.CV], 2016.
[21] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” arXiv [cs.LG], 2014.
[22] H.-Y. Lee, “ML Lecture 9-1: Tips for Training DNN,” 04-Nov-2016. [Online]. Available: https://www.youtube.com/watch?v=xki61j7z-30&list=PLJV_el3uVTsPy9oCRY30oBPNLCo89yu49&index=17. [Accessed: 24-Dec-2022].
[23] J. Jordan, “An overview of semantic image segmentation,” Jeremy Jordan, 22-May-2018. [Online]. Available: https://www.jeremyjordan.me/semantic-segmentation/. [Accessed: 07-Jan-2023].
[24] .-Y. Zhu, T. Park, P. Isola, and A. A. Efros, “Unpaired image-to-image translation using cycle-consistent adversarial networks,” in 2017 IEEE International Conference on Computer Vision (ICCV), 2017.

[25] A. Dosovitskiy et al., “An image is worth 16x16 words: Transformers for image recognition at scale,” arXiv [cs.CV], 2020.
[26] J. Chen et al., “TransUNet: Transformers make strong encoders for medical image segmentation,” arXiv [cs.CV], 2021.