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

ShaoSen, Chueh
School of Computer Science,
University of Nottingham
Wollaton Rd, Lenton, Nottingham
NG8 1BB, United Kingdom
psxsc15@nottingham.ac.uk

Supervised by
Professor Michael Pound
School of Computer Science,
University of Nottingham
Wollaton Rd, Lenton, Nottingham
NG8 1BB, United Kingdom

***Abstract*—This research aims to solve the "Multi-Disease, Multi-View, and Multi-Center Right Ventricular Segmentation in Cardiac MRI (M&M-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. [16], 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.**

Keywords—MnM2 Challenge, Deep Learning, Unet, 3D Unet, Domain Adaptation, Transfer Learning.

I. Introduction

In this research, I aim to solve 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, 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 test set. 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 methods of this project will be based on the network architecture proposed by Li et al. [1]. 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.

II. Literature Review

A. Image Segmentation and Deep Learning

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 only acceptable results. 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 [2]. 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 [3]. Deep learning has outperformed other conventional algorithms in many fields. LeCun et al. [4] 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 [5][6][7]. 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. [8] 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. [9] proposed the Unet based on the FCN, but it improves the up-sampling 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 [10] [11] due to its excellent structure.

B. Domain Adaptation

In order to handle the multi-domain nature of real-world images, many domain adaptation techniques have been proposed. Hoffman et al. [12] 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. [13] 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 [14] 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.

C. M&M-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 generalizable ml/dl models that can be applied across clinical centers. The first M&Ms Challenge was in 2020 and the paper was published in 2021 [15]. 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.

D. 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 task. 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.

III. 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 phases 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. However, since I'm not really participating in this challenge, I want to explore the method that can best segment all parts at a time. The main challenge of this dataset is that the images of the dataset come from multiple resources. This could affect the prediction capability of a deep learning model.

IV. 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).

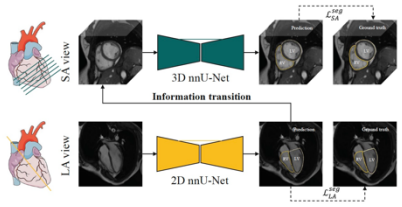

Fig. 1. The proposed RV segmentation framework for both SA and LA images. The framework includes three steps: the LA segmentation, ROI extraction from SA with assistant of LA information, and the SA segmentation. Here, the 3D cardiac image adopted from Kevil et al. [6].

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 part I want to put more emphasis on is the down sampling in the 3D Unet. The 3D Unet has four down sampling 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 must solve the problem that there are four down sampling layers in the 3D Unet. The solution that I've used in this project is that in the first down sampling layer, I only down sample (max pooling) the W and H dimension, and the Z dimension remains the same. This means that the Z dimension is only down sampled 3 times. What's more, since the input and the output of the 3D Unet should be identical, the up-sampling procedure should be the inverse of down sampling. Thus, in the last layer of the up sampling, I only up sample the W and H dimension. Therefore, the dimension of the input and the output are the same.

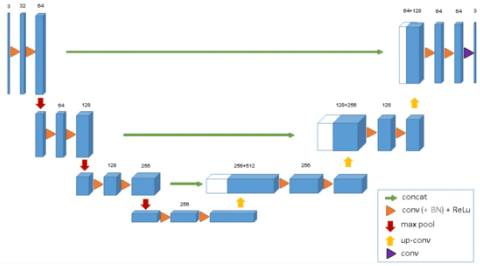

Fig.2. This figure in the paper [18] by Ö. Çiçek et al. indicate the process of the 3D Unet down sampling (the red arrow) and the dimension change.

V. Model Training

A. 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 are 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 Data Loader, it is essential to crop and pad these images. However, there are many factors when changing the size of the images. First, 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 is 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 must take how it will be processed by the 3D Unet into consideration. The 3D Unet will down sample the image every 2 convolutions, and the dimension size will be half of what it originally was. There are 4 down sample 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 must 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 must 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 3 has shown the effect of padding the LA image from four sides and only from right and bottom (the line in the middle is the transformed LA image that is concatenated with the SA image. The detail of the coordinate transformation will be explained later). 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.

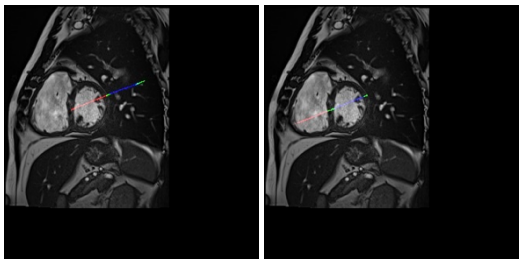

Fig.3. The two images are sliced from the Z dimension of the SA (3D) image. The lines drawn on the images are the transformed LA image. The left image shows the transformed line of a LA image padded from four sides, which is off the right position. The right image shows the transformed line of a LA image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

B. *Long-Axis to Short-Axis 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 systems 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:

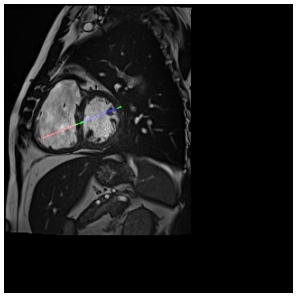

Fig.4. The SA image with the transformed LA line.

This line in figure 4 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).

C. *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 (because 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%).

D. Evaluation

In the MnM2 challenge, there are 2 evaluation metrics used for evaluating the results of each model, which are dice coefficient score and Hausdorff distance.

The concept of dice coefficient score is to calculate the intersection between 2 subjects, let's say A and B. The equation can be written as

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

for dH denotes the Hausdorff distance.

However, during the training, the Hausdorffff distance code I 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 like the results reported in the paper [16], I am not able to explain this phenomenon even after looking 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.

E. Domain Adaptation

Due to the multi-domain nature of this dataset, I believe it is worth trying to apply domain adaptation techniques on the preprocessing. 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 model I had considered is CyCADA by Isola et al [12]. CyCADA is a domain adaptation technique that's based on the idea of a CyCleGAN [19].

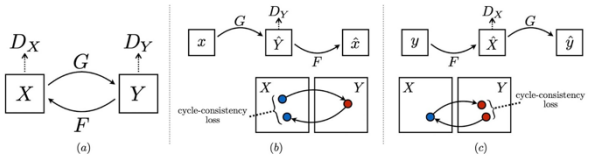

Fig. 5. CyCleGAN [19].

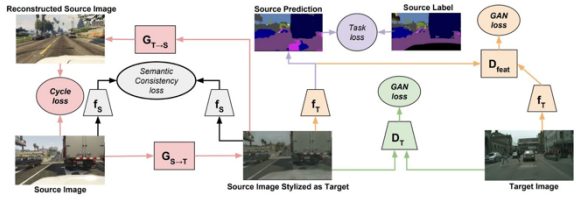

Fig. 6. CyCADA [12]

However, 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 must 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 must 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. [14]. 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.

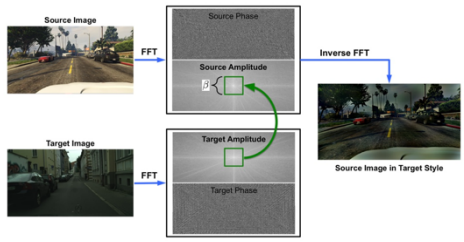

Fig. 7. Fourier Domain Transform [14].

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 is 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.

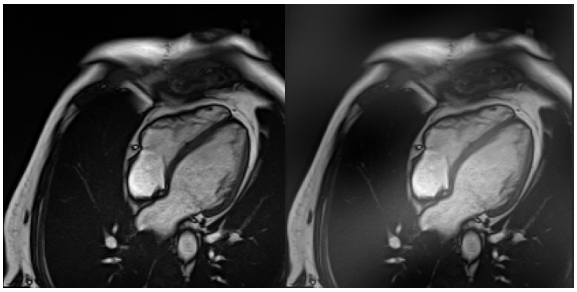

Fig. 8. The source image (left), and the source image after the FDA (right).

F. Transfer Learning

In this MnM2 dataset, the number of images and target masks is quite limited in terms of training a deep learning model. Therefore, in order to increase the generalization ability of the model, I tried to use transfer learning on the Medical Segmentation Decathlon dataset.

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, 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.

VI. Results

In the result section, I will report the results according to different stages. First, 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 can 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.

A. 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
Train Loss	0.01	0.007	0.009
Validation Loss	0.02	0.013	0.008
Test Loss	0.02	0.015	0.008

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 reach 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 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.

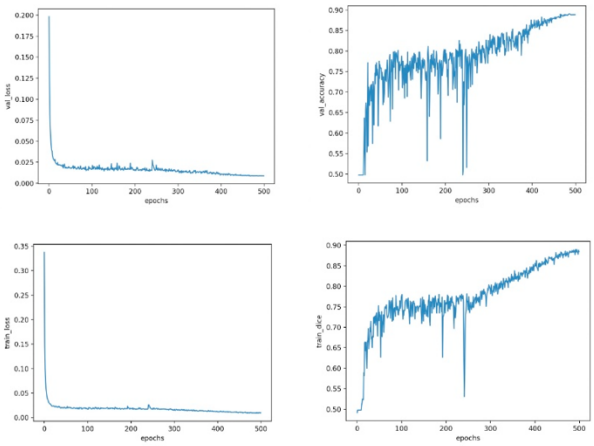

Fig. 9. The Loss data and the dice score of the result of 2D training

B. Basic 3D Unet

	3D Unet	3D Unet with 2D Info (3-class)	3D Unet with 2D Info (3-class) and augmentation	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
Train Loss	0.02	0.01	0.05	0.005
Validation Loss	0.02	0.02	0.04	0.012
Test Loss	0.02	0.02	0.04	0.014

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 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.

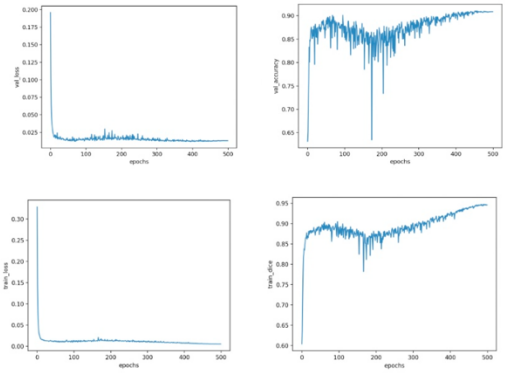

Fig. 10. The loss and dice score of the 1-class 3D Unet training.

C. *Fourier Domain Adaptation*

The training with FDA is shown as below:

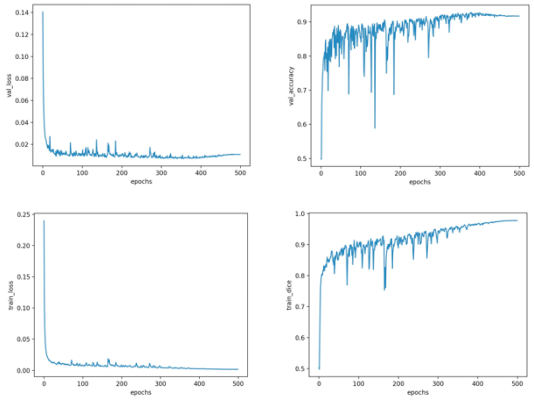

Fig. 11. The loss and dice score of the 2D Unet training with FDA

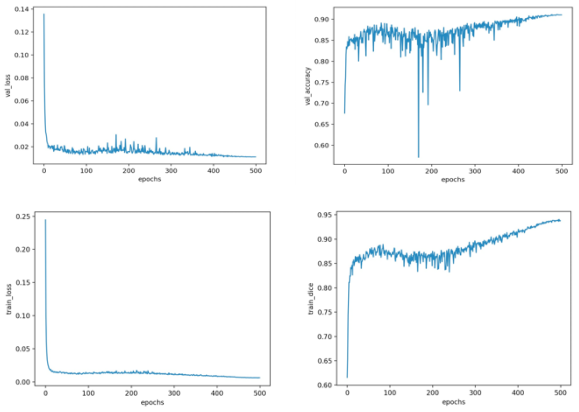

Fig. 12. The loss and dice score of the 3D Unet training with FDA

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 [1]. 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.

D. *Transfer Learning*

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

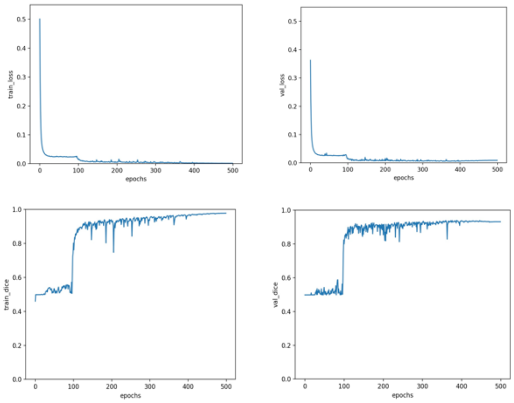

Fig. 13. The loss and dice score of the 2D Unet training with pre-trained parameters

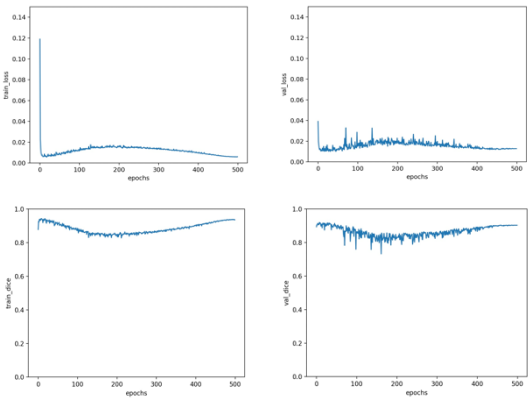

Fig. 14. The loss and dice score of the 3D Unet training with pre-trained parameters

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 (down sampling 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 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. 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 criterion. 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 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 result is about 89%, which is lower than other choices of scheduler hyper-parameters.

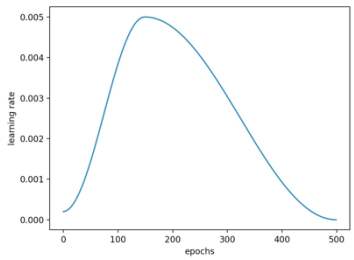

Fig. 15. One-cycle scheduled learning rate.

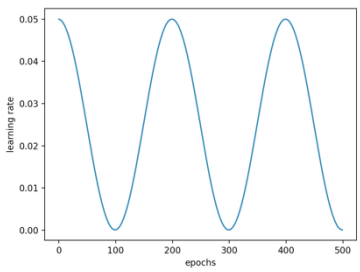

Fig. 16. Cosine-annealing scheduled learning rate.

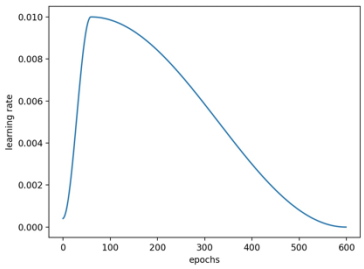

Fig. 17. One-cycle scheduled learning rate with the peak at the 50th epoch.

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 looking 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 result, it has allowed the model to converge much faster.

VII. Final Results Comparison

In conclusion, I have summed up the results of the basic Unet, Unet with Fourier Domain Adaptation, and Unet with transfer learning as below:

A. 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
Train Loss	0.009	0.002	0.002
Validation Loss	0.008	0.007	0.005
Test Loss	0.008	0.008	0.008

B. 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
Train Loss	0.005	0.006	0.009
Validation Loss	0.012	0.011	0.01
Test Loss	0.014	0.014	0.013

VIII. Conclusion

In this project, I reproduced the training pipeline of the paper by Li et al [16] 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, which has shown great improvement on the 2D model. At last, I applied transfer learning in training the model. It has also improved the 2D accuracy by about 1% and has allowed the 3D model to converge much faster than it used to be (within 20 epochs). To sum up, the contribution of this project is that the experiments that I have explored in this project can indeed improve the model accuracy and training efficiency, as supported by the results.

IX. Acknowledgment

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. Finally, I want to thank all the friends from the Nottingham Cathedral Choir, who have given me warm and unforgettable memories in Nottingham.

X. 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] N. O. Mahony et al., "Deep Learning vs. Traditional computer vision," arXiv [cs.CV], 2019.K. Elissa, "Title of paper if known," unpublished.

[3] 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.Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interface," IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].

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

[5] 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.

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

[7] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” arXiv [cs.CV], 2015.

[8] J. Long, E. Shelhamer, and T. Darrell, “Fully convolutional networks for semantic segmentation,” arXiv [cs.CV], 2014.

[9] O. Ronneberger, P. Fischer, and T. Brox, “U-Net: Convolutional Networks for Biomedical Image Segmentation,” arXiv [cs.CV], 2015.

[10] 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.

[11] J. Chen et al., “TransUNet: Transformers make strong encoders for medical image segmentation,” arXiv [cs.CV], 2021.

[12] J. Hoffman et al., “CyCADA: Cycle-Consistent Adversarial Domain Adaptation,” arXiv [cs.CV], 2017.

[13] S. Zhao et al., “Multi-source Domain Adaptation for Semantic Segmentation,” arXiv [cs.CV], 2019.

[14] Y. Yang and S. Soatto, “FDA: Fourier domain adaptation for semantic segmentation,” arXiv [cs.CV], 2020.

[15] 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.

[16] K. Weiss, T. M. Khoshgoftaar, and D. Wang, “A survey of transfer learning,” J. Big Data, vol. 3, no. 1, 2016.

[17] A. Hosna, E. Merry, J. Gyalmo, Z. Alom, Z. Aung, and M. A. Azim, “Transfer learning: a friendly introduction,” J. Big Data, vol. 9, no. 1, p. 102, 2022.

[18] 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.

[19] Zhu, J.-Y., Park, T., Isola, P., & Efros, A. A. (2017). Unpaired image-to-image translation using cycle-consistent adversarial networks. In arXiv [cs.CV]. http://arxiv.org/abs/1703.10593

[20]