Xi'an Jiaotong-Liverpool University
西交利物浦大学

# Domain Adaptation for Medical Image Segmentation on Lung Tumour Datasets


**Author** Zirui Zhou

**Student ID** 1927924

**Supervisor** Erick Purwanto

**Accessor** Jianjun Chen

# Contents

# List of Figures

# List of Tables

# Acronyms

**CNN** Convolutional Neural Network

**CT** Computed Tomography

**DCMQI** DICOM for Quantitative Imaging

**DICOM** Digital Imaging and Communications in Medicine

**GAN** Generative Adversarial Network

**KL** Kullback–Leibler

**MRI** Magnetic Resonance Imaging

**MSA** Multihead Self-attention

**MSD** Medical Segmentation Decathlon

**NIfTI** Neuroimaging Informatics Technology Initiative

**NLP** Natural Language Processing

**NSCLC** Non-Small Cell Lung Cancer

**RIDER** The Reference Image Database to Evaluate Therapy Response

**SCLC** Small Cell Lung Cancer

**TCIA** The Cancer Imaging Archive

**UDA** Unsupervised Domain Adaptation

**U-Net** Convolutional Networks for Biomedical Image Segmentation

**UNETR** UNet Transformer

**VAE** Variational Autoencoder

**ViT** Vision Transformer

# Abstract

Over the past few decades, advances have been made in the classification, diagnosis and treatment of lung cancer in many ways. Machine learning-based lung cancer prediction models such as semantic segmentation have been proposed to assist clinicians in the management of incidentally detected or screened-out indeterminate lung nodules. Semantic segmentation transforms raw medical images into clinically relevant, spatially structured information, such as outlining tumour boundaries and is an essential prerequisite for abundant clinical applications. However, the normal segmentation models may not have qualified performance, due to the lack of labelled medical imaging datasets. Domain adaptation can be introduced to finetune the target datasets on one pre-trained model for better accuracy. This project focuses on applying a segmentation and shaping model based domain adaptation framework with 3D U-Net and UNETR as its backbone. Experimental results demonstrate that the Dice gap between a certain method and direct tests is around 2% in most cases, where three datasets are imported including MSD, RIDER, and NSCLC. Domain adaptation significantly improves the performance on the small dataset NSCLC of models pre-trained on the large dataset RIDER. In addition, the 3D U-Net with simple network architecture trained on the relatively abundant datasets RIDER reaches the highest scores among the current results.

**Key Words:** Deep Learning, Convolutional Neural Network, Medical Segmentation, Unsupervised Domain Adaptation, Lung Tumour Dataset

# Acknowledge

I am grateful to all of those who have helped me during this project, especially my supervisor Prof. Erick Purwanto.

I would like to thank my parents, my girlfriend and my roommates, although we get together less and leave more.

I also thank PyCharm and my NVIDIA RTX 2060 for spending many lonely nights with me.

# Chapter 1

# Introduction

## 1.1 Motivation, Aims and Objective

Semantic segmentation is a computer vision task that aims to assign a class to each pixel in the image using that image as input [8]. If multiple objects of the same class are accessible, they can be simply labelled with their class [8]. Semantic segmentation has many applications for medical image analysis, such as segmenting pancreas tumour regions in portal venous phase Computed Tomography (CT) scans [8]. This project focuses on some lung cancer datasets which require the participants to annotate the tumour in the lungs, such as Non-Small Cell Lung Cancer (NSCLC), considering large-ranging foreground size as a challenge. NSCLC is any type of epithelial lung cancer other than Small Cell Lung Cancer (SCLC), where the most common types of NSCLC are squamous cell carcinoma, large cell carcinoma, and adenocarcinoma [9, 10]. However, machine learning techniques for computer-aided medical image analysis are often plagued by domain transfer problems caused by different distributions between source and target data [11]. For example, the normal segmentation models may not have qualified performance, due to the lack of labelled medical imaging datasets. This project aims to build a machine-learning model based on principles of domain adaptation to solve the task of segmenting objects of interest in medical images by training on a general dataset. The model can be easily fine-tuned to fit other datasets with a similar task without extra training.

## 1.2 Literature Review

For medical image segmentation, Convolutional Networks for Biomedical Image Segmentation (U-Net) is a widely-used fully convolutional network under the encoder-decoder backbone, which works with very few training images and yields precise segmentations [12]. 3D U-Net is a derivative network developed for sparsely annotated volumetric images, such as CT volume [5]. Furthermore, Zhang et al. [13] introduce Deep Residual U-Net (ResUNet) to accomplish feature accumulation in recursive residual convolution layers based on U-Net, considering the time dependency of image sequences. An adversarial-based method is a promising approach for domain adaptation to training robust deep networks by complex samples across diverse domains [14]. The image-to-image translation for converting images from source to target domain can be realised by a Generative Adversarial Network (GAN) [15]. In addition, Liu et al. [16] proved that Variational Autoencoder (VAE) could learn the shape distribution for a specific organ in unsupervised domain adaptation. Synergistic Image and Feature Adaptation (SIFA) is one of unsupervised domain adaptation framework which presents synergistic fusion of adaptations from both image and feature perspectives, guided by adversarial losses [17].

## 1.3 Industrial Relevance

Semantic segmentation transforms raw medical images into clinically relevant, spatially structured information, such as outlining tumour boundaries and is an essential prerequisite for abundant clinical applications, such as radiotherapy planning and treatment response monitoring, and provides new insights into the early diagnosis of the corresponding disease [18]. For example, early detection of abnormal signs of diabetic retinopathy can lead to effective treatment before its initial onset and prevent blindness in more than 50% of cases [19]. However, manual segmentation of medical images for tissues such as retinal blood vessels is a lengthy and tedious task, requiring extra training and professional skill [20]. Furthermore, deep learning networks adaptable for a particular clinical problem may not necessarily generalise well to different, unexplored tasks [18]. Unsupervised domain adaptation can be seen as an approach for image segmentation by some labelled data without human intervention. This method would enhance the technical scalability to allow many new applications in computer-aided diagnosis, biomarker extraction, surgical intervention planning, disease prognosis, etc. [18].

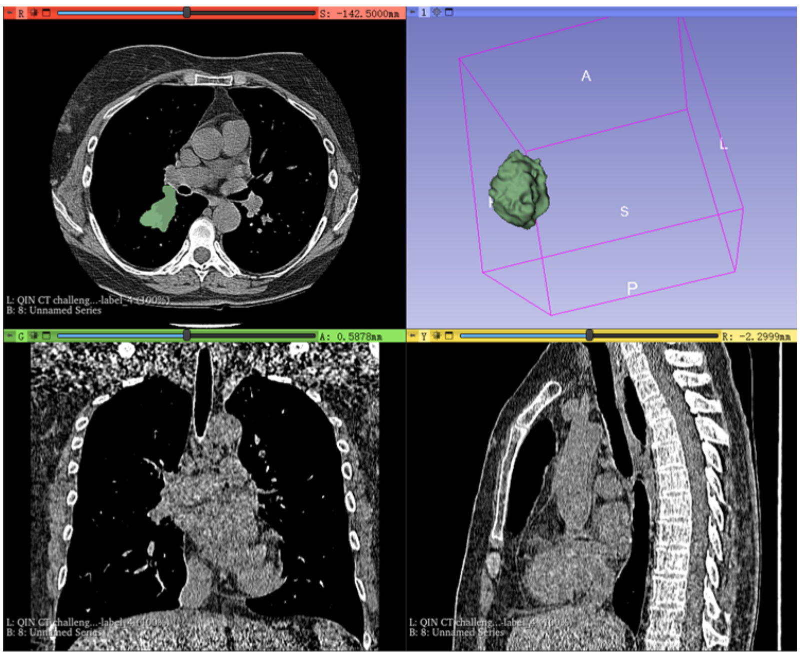


Fig. 1.1: The model of one CT set in RIDER Lung CT displayed in 3D Slicer.

# Chapter 2

# Datasets

## 2.1 Dataset Introduction

For domain adaptation, the variation of preferred modality and scanning protocol should be considered to improve the network's versatility. The Cancer Imaging Archive (TCIA) [21] is identified as the primary dataset source for retrieving data for this project to avoid data conflicts. This project selects a subset of the Non-Small Cell Lung Cancer (NSCLC) dataset [1], The Reference Image Database to Evaluate Therapy Response (RIDER) lung CT dataset [22], as the source dataset to train the source network, considering its data size and data diversity, Another subdataset of NSCLC, NSCLC Radiomics Interobserver1 [23], and Lung Tumours dataset from Medical Segmentation Decathlon (MSD) [18] are selected as the transfer data to finetune the target network.

### 2.1.1 Medical Segmentation Decathlon

This Medical Segmentation Decathlon (MSD) challenge and dataset aims to provide such resource through the open sourcing of large medical imaging datasets on several highly different tasks, and by standardising the analysis and validation process [18].

#### Lung Tumours

The dataset consists of preoperative thin-section CT scans from 96 patients with non-small cell lung cancer. This data set was selected due to the challenge of segmenting small tumour regions in an image with a large field-of-view [18]. The dataset consists of training set, testing set, and label set.

1. For training set, it includes 64 CT NIfTI files.
2. For testing set, it includes 32 CT NIfTI files.
3. For label set, it includes 64 segmentation NIfTI files.

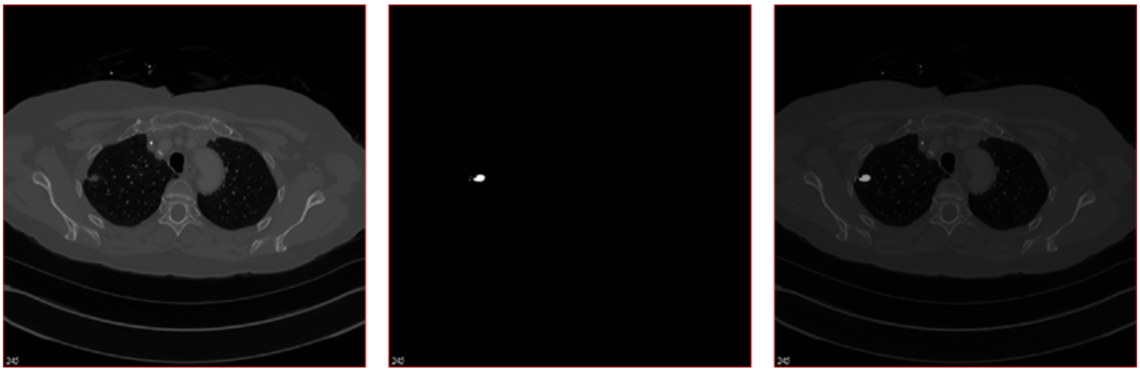

Fig. 2.1: The training, label, and composite CT image of ”lung_001.nii.gz” in MSD lung tumour dataset at frame 245.

### 2.1.2 Non-Small Cell Lung Cancer

The Non-Small Cell Lung Cancer (NSCLC) dataset is published in Nature Communications which applies a radiomic approach to computed tomography data of 1,019 patients with lung or head-and-neck cancer [1]. Radiomics refers to the comprehensive quantification of tumour phenotypes by applying a large number of quantitative image features [1]. This project introduces two lung tumour relevant datasets through manual delineation of the 3D volume of the tumor.

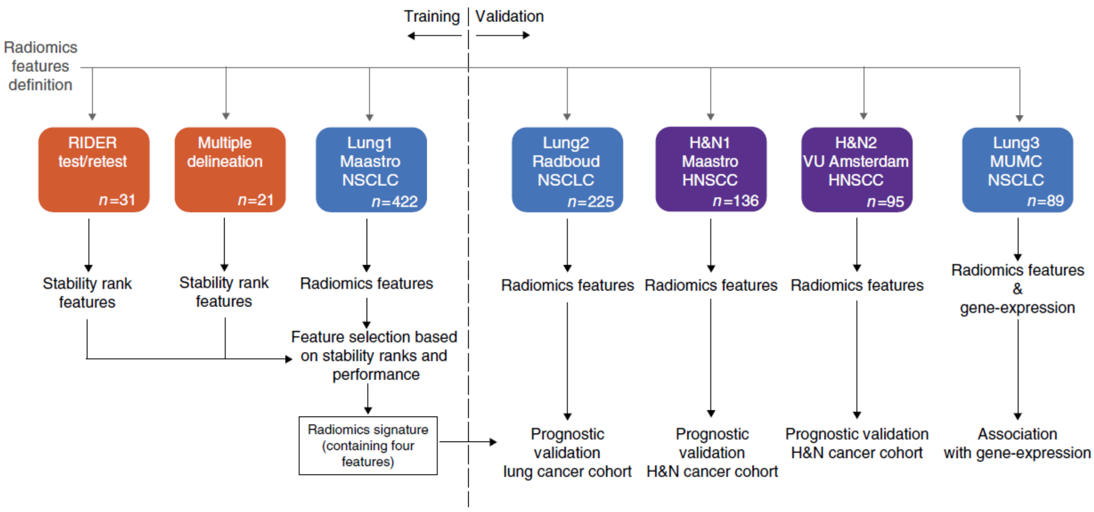


Fig. 2.2: Analysis workflow of this datasets collection which includes RIDER datasets and NSCLC Radiomics datasets [1].

#### RIDER Lung CT

The Reference Image Database to Evaluate Therapy Response (RIDER) Lung CT collection was constructed as part of a study to evaluate the variability of tumor unidimensional, bidimensional, and volumetric measurements on same-day repeat CT scans in patients with non–small cell lung cancer [22]. The dataset consists of raw resources (RIDER Lung CT) [22] and third party analyses

(RIDER-LungCT-Seg) [24].

1. For RIDER Lung CT, it includes 63 CT DICOM series from 63 patients.
2. For RIDER-LungCT-Seg, it includes 59 relevant SEG and RESTRUCT DICOM series.

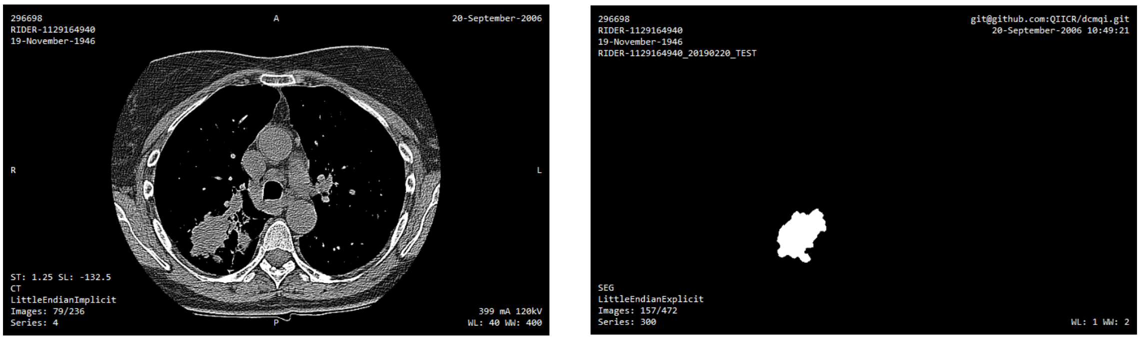


Fig. 2.3: The raw and segmentation CT image of ”RIDER-1129164940” in RIDER lung tumour dataset at frame 157.

### NSCLC-Radiomics-Interobserver1

This collection contains clinical data and CT from 22 non-small cell lung cancer radiotherapy patients [23]. The dataset consists of CT and segmentation as well.

1. For CT, it includes 22 CT DICOM series from 22 patients.
2. For segmentation, it includes 22 relevant SEG and RESTRUCT DICOM series.

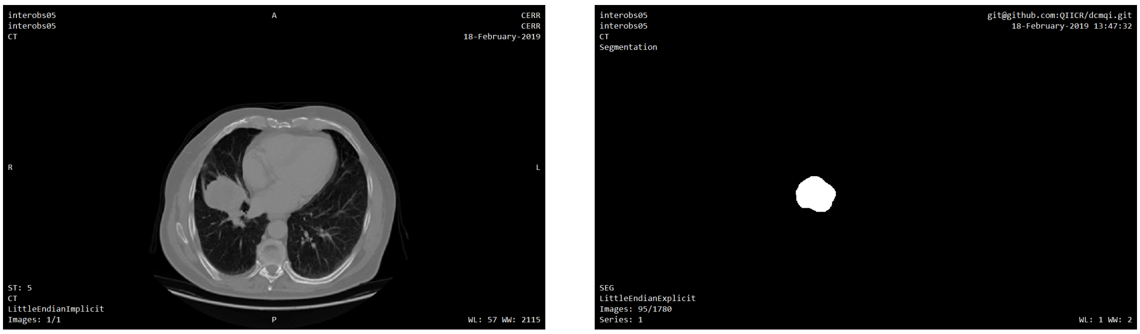


Fig. 2.4: The raw and segmentation CT image of ”interobs05” in NSCLC Radiomics Interobserver1 lung tumour dataset at frame 95.

## 2.2 Dataset Format

### 2.2.1 Neuroimaging Informatics Technology Initiative

Neuroimaging Informatics Technology Initiative (NIfTI) is a new Analyze-style data format as a short-term measure to facilitate inter-operation of functional MRI data analysis software packages [25]. The original format of MSD datasets is NIfTI.

```
>>> import nibabel as nib
>>> path = r"dataset/nih_data/Task06_Lung/imagesTr/lung_001.nii.gz"
>>> image = nib.load(path)
>>> type(image.get_fdata())
<class 'numpy.ndarray'>
>>> image.get_fdata().shape
(512, 512, 304)
```

Fig. 2.5: The NIfTI file loading process of ”lung_001.nii.gz” in MSD lung tumour dataset.

### 2.2.2 Digital Imaging and Communications in Medicine

Digital Imaging and Communications in Medicine (DICOM) is the international standard for medical images and related information [26]. The raw CT radiomic images in the NSCLC datasets are all in the DICOM format. DICOM for Quantitative Imaging (DCMQI) is a protocol with minimum dependencies to support standardized communication of quantitative image analysis research data using DICOM standard [27]. The segmentation slices in the NSCLC datasets are all in the DCMQI format.

### 2.2.3 Medical Format Normalization

Considering the less metadata and better performance in 3D volumn representation, this project prefers to use NIfTI as an intermediate format towards NumPy array. NumPy arrays as a standard matrix format provide more transformation functions and better compatibility with PyTorch. The specific reformatting path is depicted in the Fig 2.6, where some third-party tools are introduced, such as dcm2niix and dcmqi.

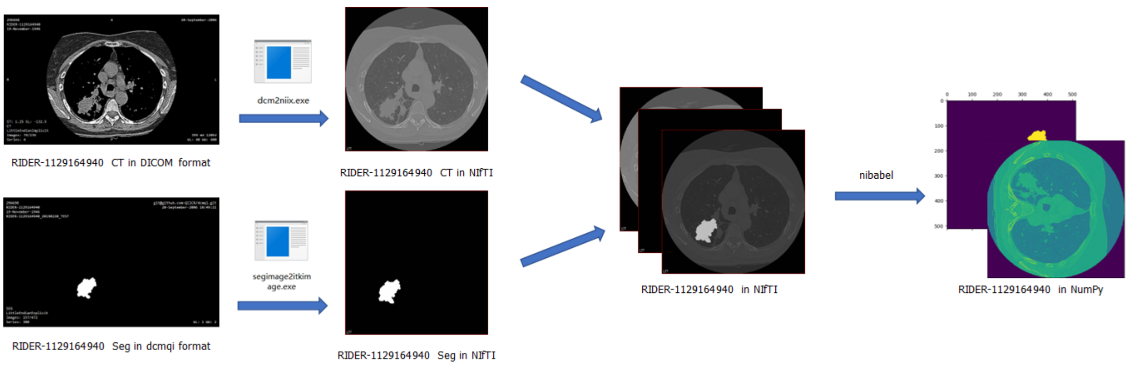


Fig. 2.6: The files reformating process from DICOM and DCMQI to NIfTI and finally NumPy arrays.

## 2.3 Dataset Preprocess

### 2.3.1 Resampling

Due to the different voxel coordinates of CT and segmentation series, the CT and segmentation volumes are resampled to the unit voxel size of $1mm \times 1mm \times 1mm$. The resampling transformation is applied by matrix multiplication with the affine matrix, which is defined as

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} m_{1,1} & m_{1,2} & m_{1,3} & a \\ m_{2,1} & m_{2,2} & m_{2,3} & b \\ m_{3,1} & m_{3,2} & m_{3,3} & c \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \\ k \\ 1 \end{bmatrix} \tag{2.1}$$

### 2.3.2 Cube Bounding

Cube bounding is utilized to crop the CT and segmentation volumes to emphasize the tumour area and reduce unwanted background. The cube box is centered on the center of the tumour volumes, with enough side length and fixed pads to cover the whole target volumes and necessary backgrounds.

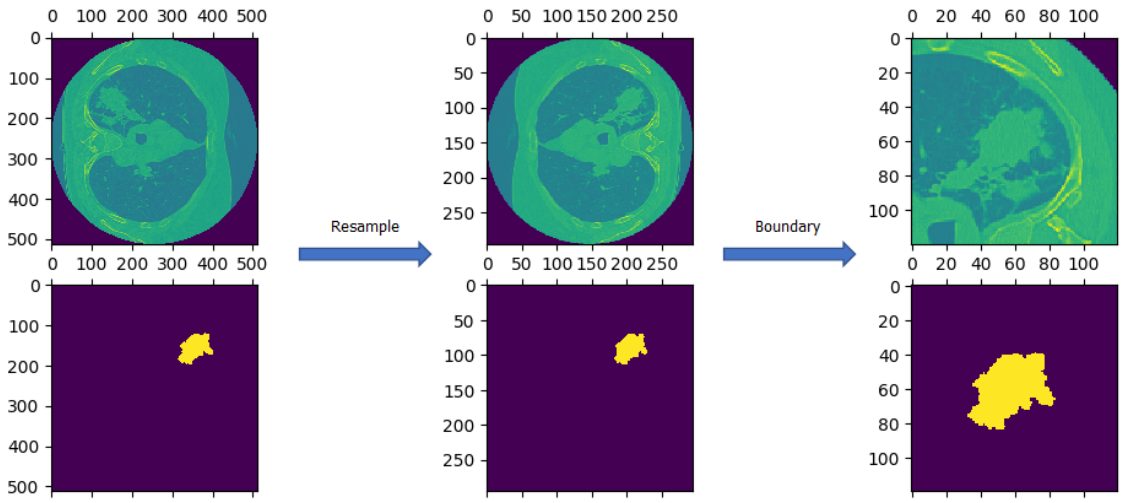


Fig. 2.7: The data preprocessing process through resampling and cube bounding.

## 2.4 Data Augmentation

Data augmentation is often used to handle the problem of the data shortage and insufficient training samples, especially for 3D segmented medical images. In model fitting, it can alleviate the overfitting problem for models with strong generalization capabilities and robustness [28]. This project applies geometric methods for this process, including scaling, rotation and cropping in a random range. The volumes are through normalization mapped to $[0, 1]$ at beginning and centered in the patch volumes in the end to standardize the volume size and range.

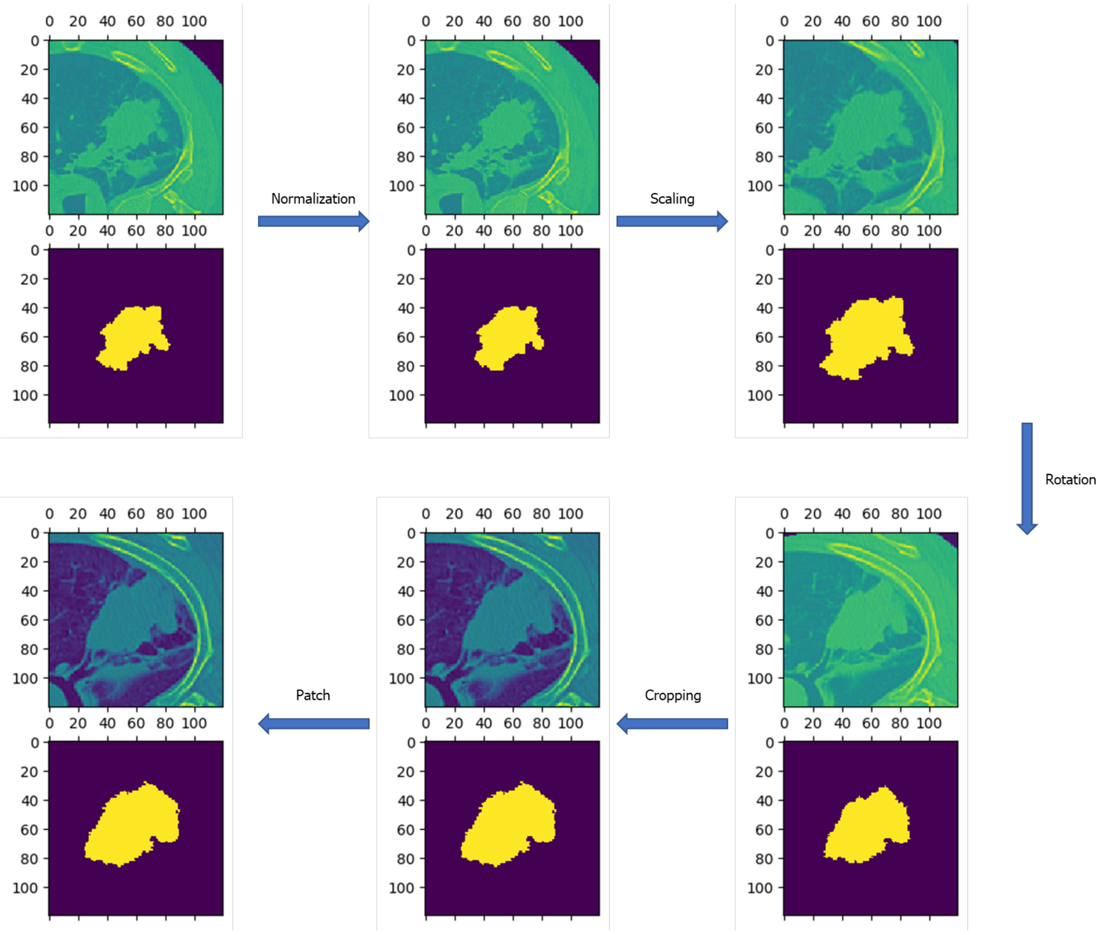


Fig. 2.8: The data augmentation process through normalization, scaling, rotation, cropping and patch.

# Chapter 3

# Methodology

## 3.1 Method Background

### 3.1.1 3D Convolutional Neural Network

Convolutional Neural Network (CNN) is a neural network primarily applied on images. 3D CNN is constructed by convolutional layers convolving 3D kernels to the cube formed by stacking multiple-dimensional spatial features, compared with 2D CNN, where convolutions are applied on the 2D feature maps to compute features from the single layers only [2]. According to Ji et al. [2], the formula of the value at position $(x, y, z)$ on the $j$th feature map in the $i$th layer is given by

$$v_{ij}^{xyz} = \tanh\left(b_{ij} + \sum_{m}\sum_{p=0}^{P_i-1}\sum_{q=0}^{Q_i-1}\sum_{r=0}^{R_i-1} w_{ijm}^{pqr} v_{(i-1)m}^{(x+p)(y+q)(z+r)}\right), \tag{3.1}$$

where $(P_i, Q_i, R_i)$ is the size of the 3D kernel, and $w_{ijm}^{pqr}$ is the ($p$, $q$, $r$)th value of the kernel connected to the $m$th feature map in the previous layer.

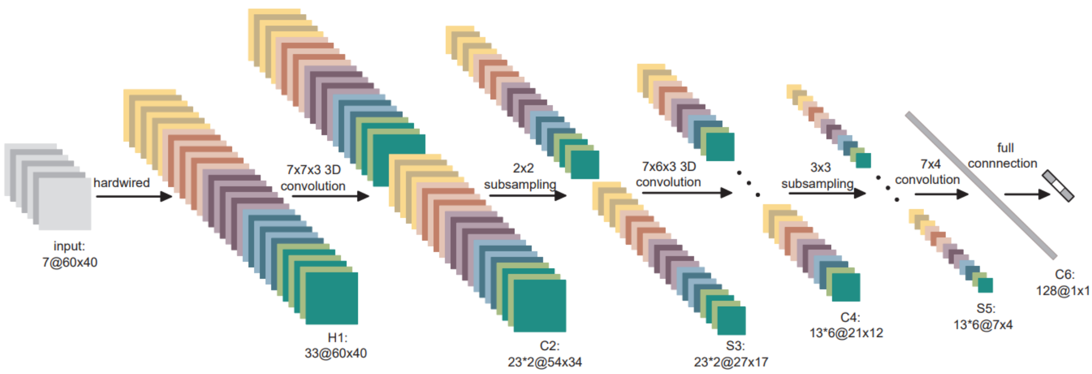


Fig. 3.1: An example 3D CNN architecture for human action recognition Ji et al. [2].

In this project, 3D CNN serves as the basic network block for feature extraction. With relevant deconvolutional networks, segmentation mask can be constructed in image segmentation task.

### 3.1.2 Vision Transformer

Self-attention-based architectures, such as Transformers, have been widely used in the field of Natural Language Processing (NLP). In image recogniztion, Dosovitskiy et al. [3] introduce Vision Transformer (ViT) architecture which abandons the traditional CNN structure and applies a standard Transformer directly to images. The images are divided into patches linked with the linear sequence, which are seen as tokens in term of NLP [3].

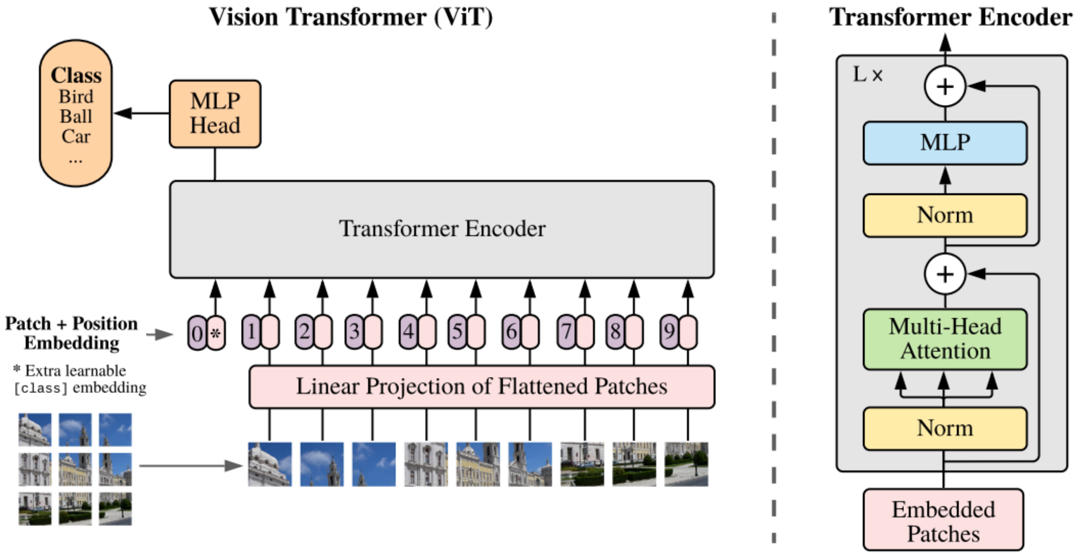


Fig. 3.2: The Vision Transformer architecture [3].

One transformer encoder process can be defined as

$$
\begin{aligned}
\mathbf{z}_0 &= \left[\mathbf{x}_{\text{class}}\,; \mathbf{x}_p^1\mathbf{E}; \mathbf{x}_p^2\mathbf{E}; \cdots ; \mathbf{x}_p^N\mathbf{E}\right] + \mathbf{E}_{pos}, && \mathbf{E} \in \mathbb{R}^{\left(P^2\cdot C\right)\times D}, \mathbf{E}_{pos} \in \mathbb{R}^{(N+1)\times D} \\
\mathbf{z}'_\ell &= \mathrm{MSA}\left(\mathrm{LN}\left(\mathbf{z}_{\ell-1}\right)\right) + \mathbf{z}_{\ell-1}, && \ell = 1 \ldots L \\
\mathbf{z}_\ell &= \mathrm{MLP}\left(\mathrm{LN}\left(\mathbf{z}'_\ell\right)\right) + \mathbf{z}'_\ell, && \ell = 1 \ldots L \\
\mathbf{y} &= \mathrm{LN}\left(\mathbf{z}_L^0\right)
\end{aligned}
\tag{3.2}
$$

where MLP is the multilayer perceptron block, MSA is the Multihead Self-attention block, and Layernorm (LN) and residual connection are applied between each network blocks [3].

Multihead Self-attention (MSA) is an extension of standard $QKV$ self-attention with $k$ paralleled self-attention operations [3]. The MSA block process can be defined as

$$
\begin{aligned}
[\mathbf{q}, \mathbf{k}, \mathbf{v}] &= \mathbf{z}\mathbf{U}_{qkv} && \mathbf{U}_{qkv} \in \mathbb{R}^{D \times 3D_h} \\
A &= \mathrm{softmax}\left(\mathbf{q}\mathbf{k}^\top / \sqrt{D_h}\right) && A \in \mathbb{R}^{N \times N} \\
\mathrm{SA}(\mathbf{z}) &= A\mathbf{v} && \\
\mathrm{MSA}(\mathbf{z}) &= [\mathrm{SA}_1(z); \mathrm{SA}_2(z); \cdots; \mathrm{SA}_k(z)]\, \mathbf{U}_{msa} && \mathbf{U}_{msa} \in \mathbb{R}^{k \cdot D_h \times D}
\end{aligned} \tag{3.3}
$$

where $q$, $k$, $v$ are query, key and value matrix [3].

Althrough transformers lack some of the inductive biases inherent to CNNs leading to its poor generalization performance on insufficient amounts of data, some researches, such as UNet Transformer (UNETR) [6] and TransUNet [29], show reasonable segmentation scores by combination of CNNs and transformers [3].

## 3.2 Main Framework

### 3.2.1 VAE Pipeline

Yao et al. [4] developed an Unsupervised Domain Adaptation (UDA) framework to import inherent shape statistics into a standard medical image segmentation model, which is based on a teacher-student learning paradigm with a dual-loss function.

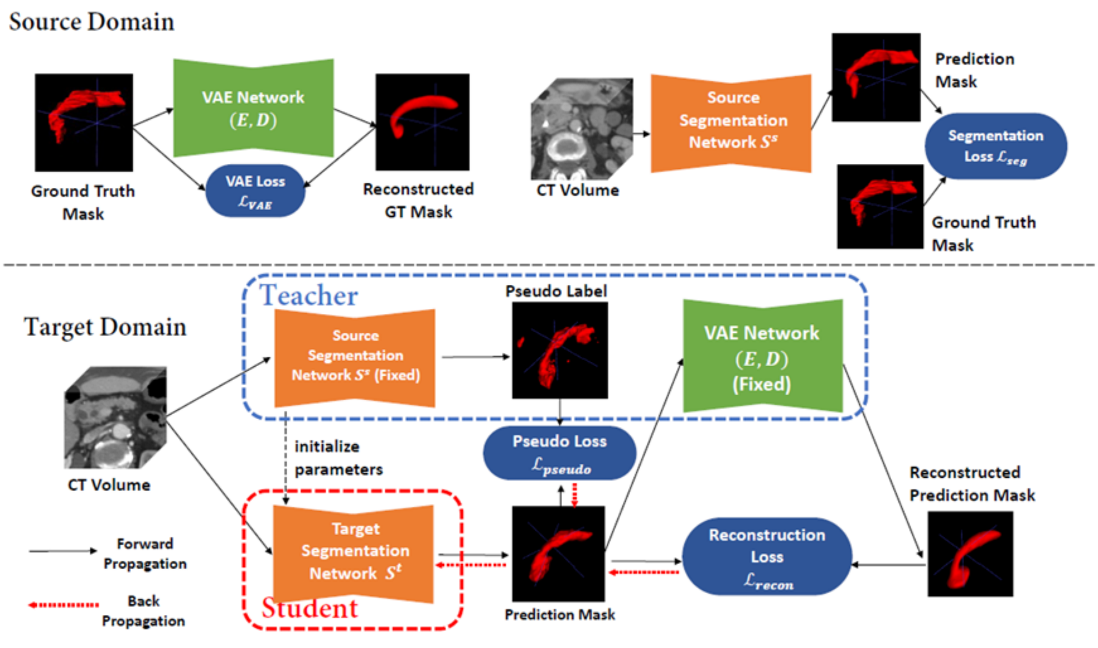


Fig. 3.3: Proposed VAE-based pipeline of Unsupervised Domain Adaptation [4].

On the source domain, the organ shape information is extracted through a pre-trained VAE network in ground truth masks [4]. One dataset is under the training of source segmentation network (3D

U-Net in this project) to initialise it as a teacher network. On the target domain, the two trained networks, i.e., the VAE network and source segmentation network, are fixed as teacher networks [4]. The target segmentation network is initialised by the source one and loads another unlabeled dataset in a different style from the first dataset for domain adaptation [4]. The pseudo label is generated by the source segmentation network in the teacher group for a pseudo-loss calculation to fine-tune the segmentation labels [4]. The distribution of shape for a specific organ is from the VAE network in the teacher group to predict reconstructed masks and acquire reconstruction loss [4]. The dual loss is combined by a loss function with a hyperparameter offset $\lambda$, defined in Equation 3.4, considering their adverse effects [4].

$$L_\theta\left(S^t, x^t\right) = \lambda_{\text{recon}} \cdot \mathcal{L}_{\text{recon}} + \mathcal{L}_{\text{pseudo}} \tag{3.4}$$

## 3.3 Segmentation Network

### 3.3.1 3D U-Net

3D U-Net is a CNN-based network for image segmentation which learns from sparsely annotated volumetric images [5]. The network extends the previous U-Net architecture from Ronneberger et al. [12] by replacing all 2D operations in dimensions with 3D counterparts [5]. Similar to the standard U-Net, this network contains two paths, including an analysis one and a synthesis one, each with four resolution layers, depicted in Fig 3.4 [5].

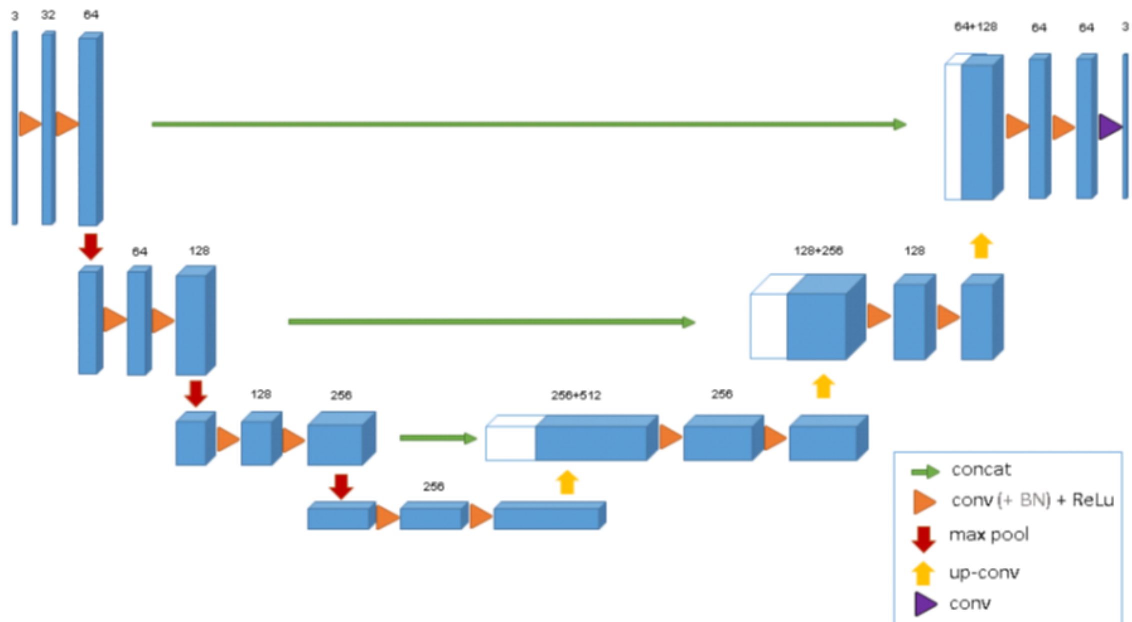


Fig. 3.4: The 3D U-Net network architecture [5].

This means one medical image can be labelled in the target domains to realise semantic segmentation. In this project, the framework is modified to fit the dataset magnitude.

1. In the analysis path, all four "Down" layers contain one $2 \times 2 \times 2$ convolution layer alone with a stride of two in all dimensions, and three same convolution layers each followed by an activation function (ReLu or Softplus) and a normalisation layer (Instance Normalisation or Batch Normalisation). The max pooling layer in the original 3D U-Net model is not applied, which is replaced by an optional dropout layer.
2. In the synthesis path, all four "Up" layers consist of the up-convolution layers whose structure and parameters are similar to the "Down" ones but in the opposite direction.
3. Shortcut connections from layers of equal resolution between analysis and synthesis paths remain to import high-resolution features realised by concatenation.
4. In the last layer, a convolution layer reduces the number of output channels to the number of labels. Then, a Softmax function is attached to get the final labels for each point.

### 3.3.2 UNet Transformer

UNet Transformer (UNETR) follows 3D U-Net “U-shaped” network design for the encoder and decoder, but replaces the CNN layers with a ViT transformer in the encoder to consider volume sequence representations and capture the multi-dimension information [6].

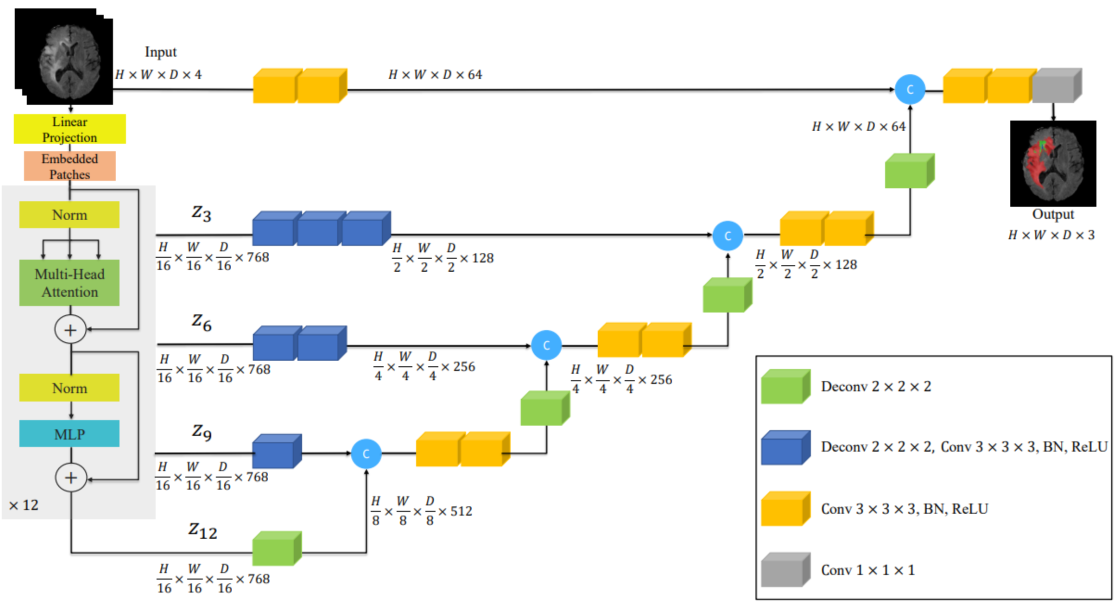


Fig. 3.5: The UNETR network architecture [6].

According to ViT, a 1D sequence of a 3D input volume is initialized by splitting it into flattened patches and embeded with its positional sequence at beginning [6]. In encoder path, the twelve transformer blocks are divided into four stack of transformers, which can be seen as encoder layers [6]. Similar to 3D U-Net, features from multiple resolutions of the encoder are merged with the decoder by concatenating volumes in the same level [6]. However, the hidden layer outputs need to reshaped into the target volume size and encoded through some deconvolution blocks [6]. Compared with standard 3D U-Net, UNETR discards max-pooling layers between levels, which increases the parameter size.

## 3.4 Model Shaping Network

### 3.4.1 Variational Autoencoder

The Variational Autoencoder (VAE) is applied for the recognition, denoising, representation and visualisation by learning the approximate posterior inference model [7]. The VAE network consists of Encoder and Decoder at the architecture level [7]. To allow back propagation, the reparameterisation trick is introduced in the sampling function, which combines the distribution parameters (mean and standard deviation) and a random sampled number in a scale [7]. Through KL divergence, Gaussian distribution, and Bayes formula, the final result formula of the VAE can be presented as the one in Equation 3.5, under the reparameterisation trick [7]. This means one image can be sampled into features as a latent variable and re-sampled to restore the original image.

$$
\begin{aligned}
\mathcal{L}\left(\theta, \phi; x^{(i)}\right) \simeq & \frac{1}{2} \sum_{j=1}^{J}\left(1+\log \left(\left(\sigma_j^{(i)}\right)^2\right)-\left(\mu_j^{(i)}\right)^2-\left(\sigma_j^{(i)}\right)^2\right) \\
& +\frac{1}{L} \sum_{l=1}^{L} \log p_\theta\left(x^{(i)} \mid \mathbf{z}^{(i, l)}\right) \\
\text{where } \mathbf{z}^{(i, l)} = & \mu^{(i)}+\sigma^{(i)} \odot \epsilon^{(l)} \text{ and } \epsilon^{(l)} \sim \mathcal{N}(0, \mathbf{I})
\end{aligned}
\tag{3.5}
$$

In this project, the reconstructed mask, i.e., the shape information of labels, must be extracted by the VAE network trained with ground truth masks. The structure follows the original design of the VAE network but imports the Dice coefficient as the loss term.

1. In the encoder block, all five encoding layers contain one $2 \times 2 \times 2$ convolution layer alone and three same convolution layers, followed by an activation function and a normalisation layer. The encoding layers and their parameters are analogous to the ”Down” layer in the 3D U-Net, which reuse the layer-building function.

2. In the decode block, all five decoding layers consist of the up-convolution layers whose structure and parameters are similar to the encoding ones but in the opposite direction, which is analogous to the ”Up” layer.

3. The mean and standard deviation are calculated in the middle layers for back propagation. For the reparameterisation trick, the random sample between zero and a given scale is optional.

4. In the last layer, a convolution layer reduces the number of output channels to the number of labels. Then, a Softmax function is attached to get the final labels for each point.

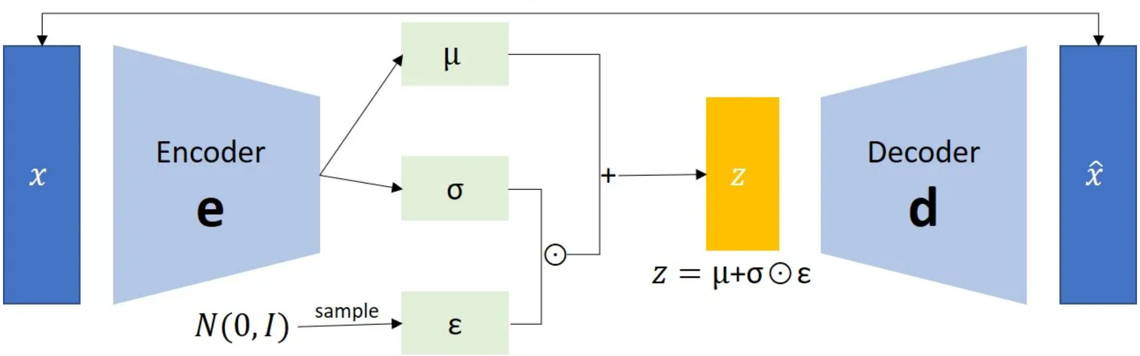


Fig. 3.6: The VAE architecture [7].

## 3.5 Loss Function

Tumour areas in this lung tumour segmentation task represent a very small fraction of the full image in some slices. Such unbalanced problems can cause improper fitting such as a tendency to mark all to non-tumour leading to false negative results, applying inappropriate loss functions [30]. This project prefers to apply the Dice loss for evaluation and validation.

### 3.5.1 Dice Loss

The Dice score coefficient is a measure of overlap widely used to assess segmentation performance when ground truth is available. The Dice loss can be expressed as

$$DL_s = 1 - \frac{2|X \bigcap Y| + \epsilon}{|X| + |Y| + \epsilon} \tag{3.6}$$

where $|X \bigcap Y|$ is the intersection between X and Y, $|X|$ and $|Y|$ are the numbers of elements of X and Y, and$\epsilon$ is the smoothing parameter for Laplace smoothing [30].

Multiple labels Dice loss for Tensor is implemented, as it is not realized in PyTorch.

```
def dice_coef_multiclass(inputs, targets, num_class, smooth=1):
   dice = 0
   for index in range(num_class):
      dice += dice_coef(inputs[:, index, ...], targets[:, index, ...],
          smooth)
   return 1 - dice / num_class

def dice_coef(inputs, targets, smooth=1):
   inputs = inputs.flatten()
   targets = targets.flatten()
   intersection = (inputs * targets).sum()
   dice = (2. * intersection + smooth) / (inputs.sum() + targets.sum()
      + smooth)
   return dice
```

# Chapter 4

# Experiments

## 4.1 Experiment Process

### 4.1.1 Train on source domain

Three datasets (MSD, RIDER, and NSCLC) have been trained on the two segmentation models (3D U-Net and UNETR) and shaping model (VAE) respectively with relevant labelled segmentaion masks to access the upper bound scores. The pre-trained models are saved for best model selection and further domain adaptation in target domain.

### 4.1.2 Train on target domain

The pre-trained segmentation and shaping models of one dataset are selected as the source dataset to initialize parameters in the two fixed teacher networks. The other two datasets left are target datasets to fine-tune the student segmentation network, where their ground truth masks are neglected as they are in unsupervised domain adaptation. The three datasets are determined to be the source dataset to observe their performance.

## 4.2 Results

### 4.2.1 Model Performance

Table 4.1 and Table 4.2 present the segmentation results on 3D U-Net and UNETR as their segmentation models, which are evaluated with mean Dice score. In detail, the data in columns have same target datasets. The upper bound score means the result in the segmentation model trained by relevant labelled datasets, which can be seen as ceiling score. The direct score means validate the dataset on the model trained by another dataset without any finetune. The UDA score is the final score through the complete model. Domain adaptation do polish up the score from direct validation. Moreover, due to more fitting on a larger dataset, the UDA even have better score on the small NSCLC dataset.

| | | MSD | RIDER | NSCLC |
|---|---|---|---|---|
| Upper Bound | | 0.6676 | 0.7879 | 0.6963 |
| Direct | MSD | / | 0.7247 | 0.7248 |
| | RIDER | 0.6537 | / | 0.7511 |
| | NSCLC | 0.6065 | 0.6382 | / |
| UDA | MSD | / | **0.7415** | 0.7145 |
| | RIDER | **0.6625** | / | **0.7637** |
| | NSCLC | 0.6065 | 0.6669 | / |

Table 4.1: Performance of this UDA segmentation method on 3D U-Net.

| | | MSD | RIDER | NSCLC |
|---|---|---|---|---|
| Upper Bound | | 0.6708 | 0.7568 | 0.7020 |
| Direct | MSD | / | 0.6811 | 0.7265 |
| | RIDER | 0.6547 | / | 0.7424 |
| | NSCLC | 0.5994 | 0.6192 | / |
| UDA | MSD | / | **0.7326** | 0.7386 |
| | RIDER | **0.6624** | / | **0.7432** |
| | NSCLC | 0.6041 | 0.6357 | / |

Table 4.2: Performance of this UDA segmentation method on UNETR.

### 4.2.2 Training Process

Data visualization on training process of validation score is shown in the Fig 4.1, Fig 4.2, and Fig 4.3, which are based on UNETR model. As the figures show, the models in source domain can converge quickly in 20 epochs and be stabilised to a upper limit with some fluctuations. The target domain do not provide reasonable finetune towards target datasets. All charts including training losses and validation scores on 3D U-Net and UNETR are listed in the Appendix A.

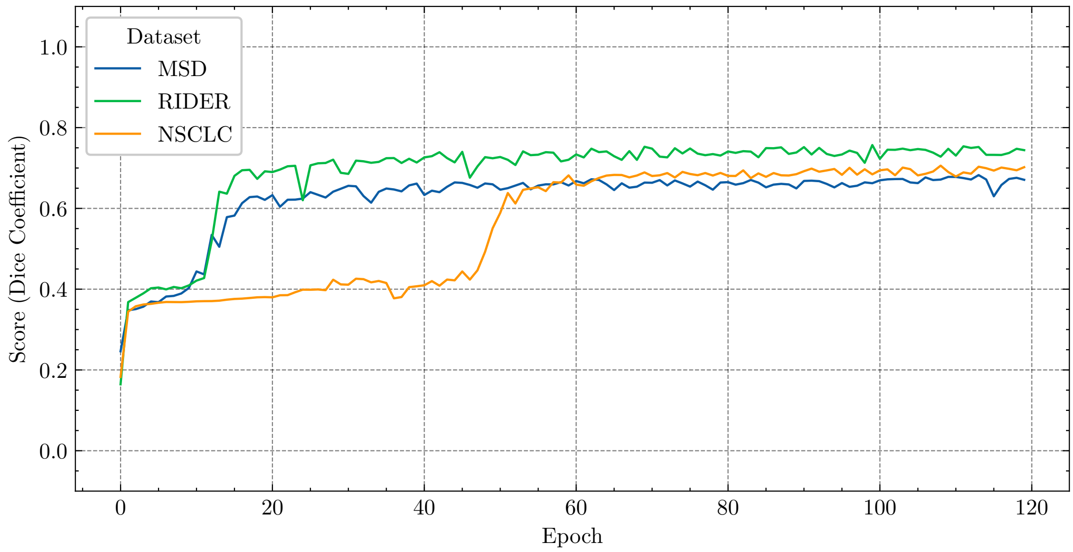


Fig. 4.1: The validation score chart for three datasets in UNETR segmentation model.

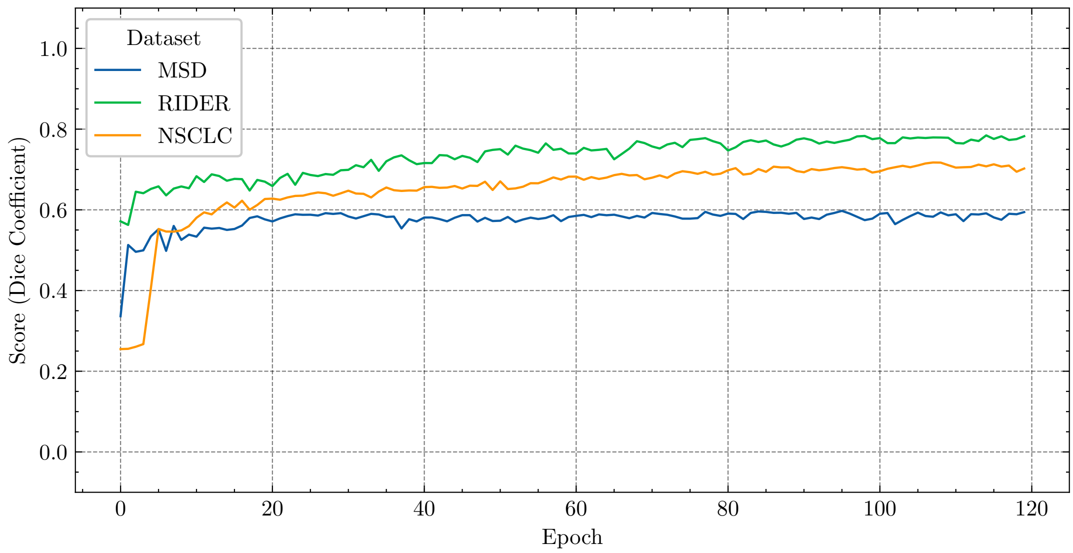


Fig. 4.2: The validation score chart for three datasets in VAE shaping model.

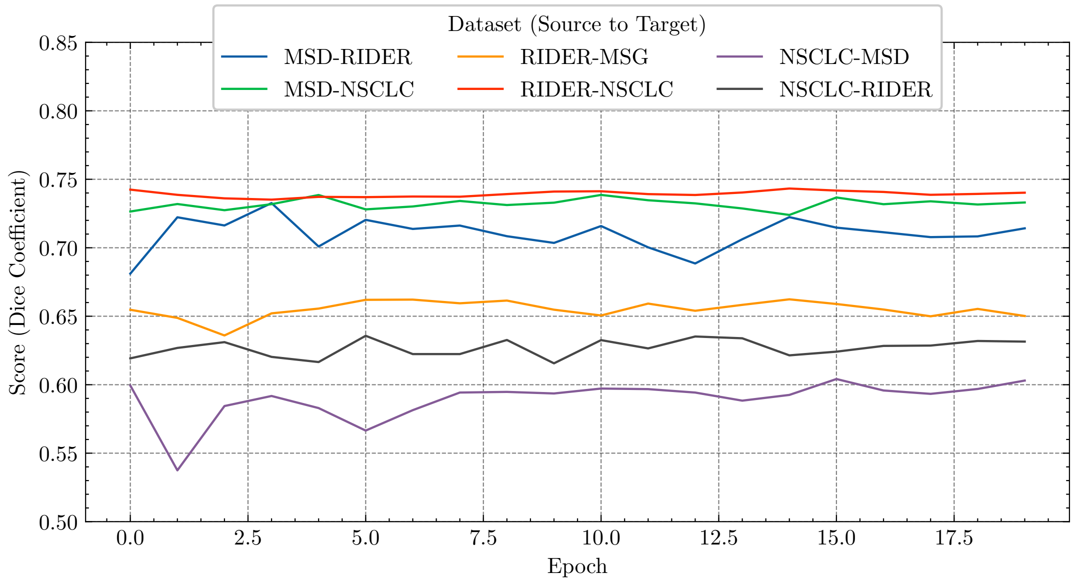


Fig. 4.3: The validation score chart for six dataset pairs in UDA framework based on UNETR.

### 4.2.3 Visualization Result

As the original medical CT volumes are through data preprocessing and augmentation, it is hard for the outputs of the model to be converted to a standard medical segmentation format (i.e. NIfTI or DCMQI) for further 3D model reconstruction. Fig 4.4 and Fig 4.5 show two slices with different states for model performance display in both segmentation model and UDA framework.

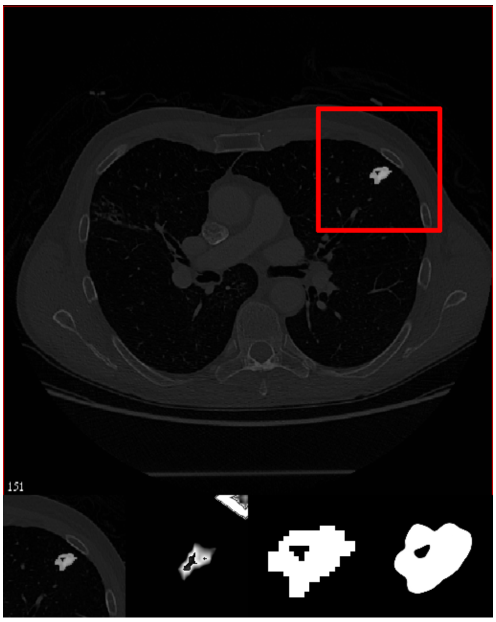


Fig. 4.4: The original segmentation result of ”lung_070.nii.gz” in the MSD dataset at frame 151 (from left to right, they are origin, preprocessed, label, and prediction).

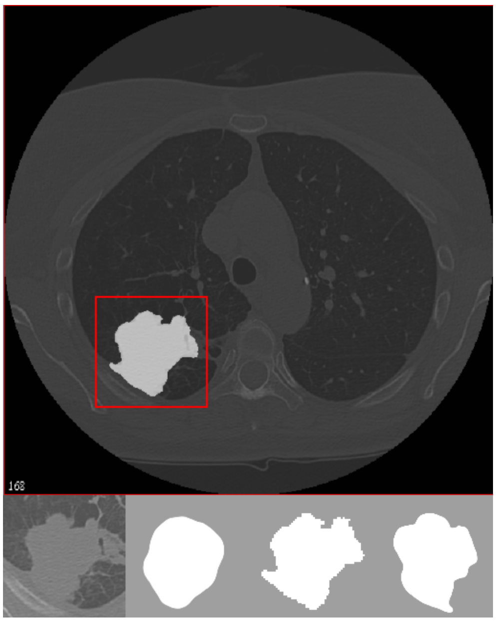


Fig. 4.5: The domain adaptation segmentation result of “RIDER-1129164940” in the RIDER dataset at frame 168 (from left to right, they are origin, reconstruction, label, and prediction).

# Chapter 5

# Conclusion and Future Work

## 5.1 Conclusion

In conclusion, the framework based on segmentation and shaping models can improve the model performance on unlabelled datasets by unsupervised domain adaptation. The Dice gap between a certain method and the upper bound is around 2% in most cases. The 3D U-Net with simple network architecture trained on the relatively abundant datasets RIDER reaches the highest scores among the current results, which satisfies the weakness of ViT in labelled dataset shortage. In the small dataset NSCLC, the model pre-trained on other datasets provide more accurate segmentation than the upper bound result trained by the same dataset.

## 5.2 Future Work

This project needs further modification and maintenance in the future. More datasets with different scanning styles, such as combination of CT and MRI, will be collected to analyse the bottlenecks of the current network. Some state-of-the-art components and mechanisms will be introduced into the network to handle the possible drawbacks, including accuracy, training time, and model size. Statistics about comparison with other segmentation methods will be evaluated in the future work.

# Reference


[1] H. J. Aerts, E. R. Velazquez, R. T. Leijenaar, C. Parmar, P. Grossmann, S. Carvalho, J. Bussink, R. Monshouwer, B. Haibe-Kains, D. Rietveld *et al.*, "Decoding tumour phenotype by noninvasive imaging using a quantitative radiomics approach," *Nature communications*, vol. 5, no. 1, p. 4006, 2014.

[2] S. Ji, W. Xu, M. Yang, and K. Yu, "3d convolutional neural networks for human action recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 1, pp. 221–231, 2012.

[3] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[4] Y. Yao, F. Liu, Z. Zhou, Y. Wang, W. Shen, A. Yuille, and Y. Lu, "Unsupervised domain adaptation through shape modeling for medical image segmentation," in *International Conference on Medical Imaging with Deep Learning*. PMLR, 2022, pp. 1444–1458.

[5] Ö. Çiçek, A. Abdulkadir, S. S. Lienkamp, T. Brox, and O. Ronneberger, "3d u-net: learning dense volumetric segmentation from sparse annotation," in *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2016: 19th International Conference, Athens, Greece, October 17-21, 2016, Proceedings, Part II 19*. Springer, 2016, pp. 424–432.

[6] A. Hatamizadeh, Y. Tang, V. Nath, D. Yang, A. Myronenko, B. Landman, H. R. Roth, and D. Xu, "Unetr: Transformers for 3d medical image segmentation," in *Proceedings of the IEEE/CVF winter conference on applications of computer vision*, 2022, pp. 574–584.

[7] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[8] S. Asgari Taghanaki, K. Abhishek, J. P. Cohen, J. Cohen-Adad, and G. Hamarneh, "Deep semantic segmentation of natural and medical images: a review," *Artificial Intelligence Review*, vol. 54, pp. 137–178, 2021.

[9] N. Duma, R. Santana-Davila, and J. R. Molina, "Non–small cell lung cancer: epidemiology, screening, diagnosis, and treatment," in *Mayo Clinic Proceedings*, vol. 94, no. 8. Elsevier, 2019, pp. 1623–1640.

[10] P. Goldstraw, D. Ball, J. R. Jett, T. Le Chevalier, E. Lim, A. G. Nicholson, and F. A. Shepherd, "Non-small-cell lung cancer," *The Lancet*, vol. 378, no. 9804, pp. 1727–1740, 2011.

[11] H. Guan and M. Liu, "Domain adaptation for medical image analysis: a survey," *IEEE Transactions on Biomedical Engineering*, vol. 69, no. 3, pp. 1173–1185, 2021.

[12] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*. Springer, 2015, pp. 234–241.

[13] Z. Zhang, Q. Liu, and Y. Wang, "Road extraction by deep residual u-net," *IEEE Geoscience and Remote Sensing Letters*, vol. 15, no. 5, pp. 749–753, 2018.

[14] E. Tzeng, J. Hoffman, K. Saenko, and T. Darrell, "Adversarial discriminative domain adaptation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 7167–7176.

[15] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *Communications of the ACM*, vol. 63, no. 11, pp. 139–144, 2020.

[16] F. Liu, Y. Xia, D. Yang, A. L. Yuille, and D. Xu, "An alarm system for segmentation algorithm based on shape model," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 10 652–10 661.

[17] C. Chen, Q. Dou, H. Chen, J. Qin, and P.-A. Heng, "Synergistic image and feature adaptation: Towards cross-modality domain adaptation for medical image segmentation," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, no. 01, 2019, pp. 865–872.

[18] M. Antonelli, A. Reinke, S. Bakas, K. Farahani, A. Kopp-Schneider, B. A. Landman, G. Litjens, B. Menze, O. Ronneberger, R. M. Summers, B. van Ginneken, M. Bilello, P. Bilic, P. F. Christ, R. K. G. Do, M. J. Gollub, S. H. Heckers, H. Huisman, W. R. Jarnagin, M. K. McHugo, S. Napel, J. S. G. Pernicka, K. Rhode, C. Tobon-Gomez, E. Vorontsov, J. A. Meakin, S. Ourselin, M. Wiesenfarth, P. Arbeláez, B. Bae, S. Chen, L. Daza, J. Feng, B. He, F. Isensee, Y. Ji, F. Jia, I. Kim, K. Maier-Hein, D. Merhof, A. Pai, B. Park, M. Perslev, R. Rezaiifar, O. Rippel, I. Sarasua, W. Shen, J. Son, C. Wachinger, L. Wang, Y. Wang, Y. Xia, D. Xu, Z. Xu, Y. Zheng, A. L. Simpson, L. Maier-Hein, and M. J. Cardoso, "The Medical Segmentation Decathlon," *Nature Communications*, vol. 13, no. 1, p. 4128, jul 2022. [Online]. Available: https://www.nature.com/articles/s41467-022-30695-9

[19] X. Jin, H. Guangshu, H. Tianna, H. Houbin, and C. Bin, "The multifocal erg in early detection of diabetic retinopathy," in *2005 IEEE Engineering in Medicine and Biology 27th Annual Conference*. IEEE, 2006, pp. 7762–7765.

[20] A. Asad, A. T. Azar, N. El-Bendary, A. E. Hassaanien *et al.*, “Ant colony based feature selection heuristics for retinal vessel segmentation,” *arXiv preprint arXiv:1403.1735*, 2014.

[21] K. Clark, B. Vendt, K. Smith, J. Freymann, J. Kirby, P. Koppel, S. Moore, S. Phillips, D. Maffitt, M. Pringle, L. Tarbox, and F. Prior, “The Cancer Imaging Archive (TCIA): Maintaining and Operating a Public Information Repository,” *Journal of Digital Imaging*, vol. 26, no. 6, pp. 1045–1057, 2013. [Online]. Available: https://doi.org/10.1007/s10278-013-9622-7

[22] B. Zhao, L. H. Schwartz, and M. G. Kris, “Data from rider lung ct. the cancer imaging archive,” *The Cancer Imaging Archive*, 2015.

[23] L. Wee, H. Aerts, P. Kalendralis, and A. Dekker, “Data from nsclc-radiomics-interobserver1 [data set],” *The Cancer Imaging Archive*, vol. 10, 2019.

[24] ——, “Rider lung ct segmentation labels from: Decoding tumour phenotype by noninvasive imaging using a quantitative radiomics approach [data set],” *The Cancer Imaging Archive*, 2020.

[25] S. Jonnalagadda and R. Srinivasan, “Nifti: An evolutionary approach for finding number of clusters in microarray data,” *BMC bioinformatics*, vol. 10, pp. 1–13, 2009.

[26] M. Mustra, K. Delac, and M. Grgic, “Overview of the dicom standard,” in *2008 50th International Symposium ELMAR*, vol. 1. IEEE, 2008, pp. 39–44.

[27] C. Herz, J.-C. Fillion-Robin, M. Onken, J. Riesmeier, A. Lasso, C. Pinter, G. Fichtinger, S. Pieper, D. Clunie, R. Kikinis *et al.*, “Dcmqi: an open source library for standardized communication of quantitative image analysis results using dicom,” *Cancer research*, vol. 77, no. 21, pp. e87–e90, 2017.

[28] L. Taylor and G. Nitschke, “Improving deep learning with generic data augmentation,” in *2018 IEEE symposium series on computational intelligence (SSCI)*. IEEE, 2018, pp. 1542–1547.

[29] J. Chen, Y. Lu, Q. Yu, X. Luo, E. Adeli, Y. Wang, L. Lu, A. L. Yuille, and Y. Zhou, “Transunet: Transformers make strong encoders for medical image segmentation,” *arXiv preprint arXiv:2102.04306*, 2021.

[30] C. H. Sudre, W. Li, T. Vercauteren, S. Ourselin, and M. Jorge Cardoso, “Generalised dice overlap as a deep learning loss function for highly unbalanced segmentations,” in *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support: Third International Workshop, DLMIA 2017, and 7th International Workshop, ML-CDS 2017, Held in Conjunction with MICCAI 2017, Québec City, QC, Canada, September 14, Proceedings 3*. Springer, 2017, pp. 240–248.

# Appendix A

# Charts

## A.1 3D U-Net Segmentation

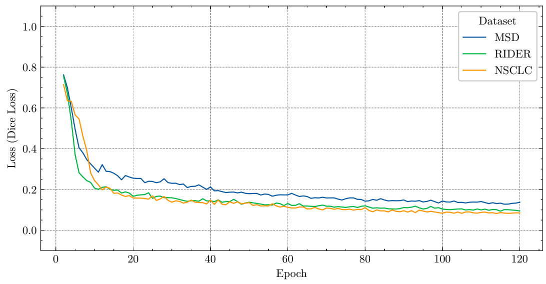


Fig. A.1: The training loss chart for three datasets in 3D U-Net segmentation model.

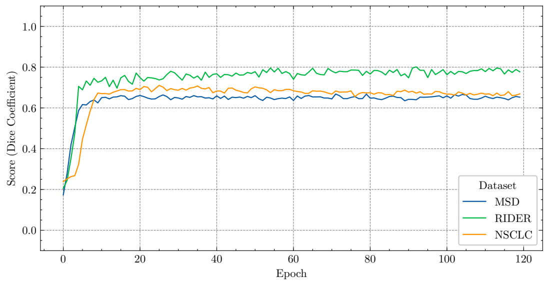


Fig. A.2: The validation score chart for three datasets in 3D U-Net segmentation model.

## A.2 UNETR Segmentation

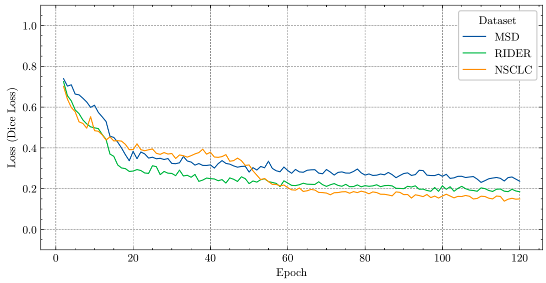


Fig. A.3: The training loss chart for three datasets in UNETR segmentation model.

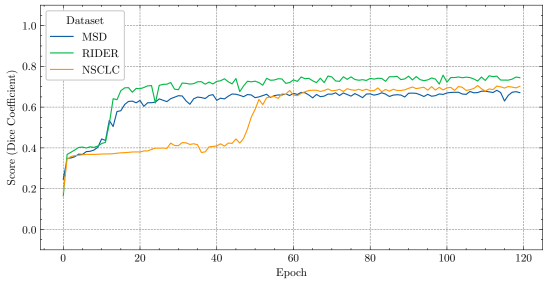


Fig. A.4: The validation score chart for three datasets in UNETR segmentation model.

## A.3 VAE Shaping

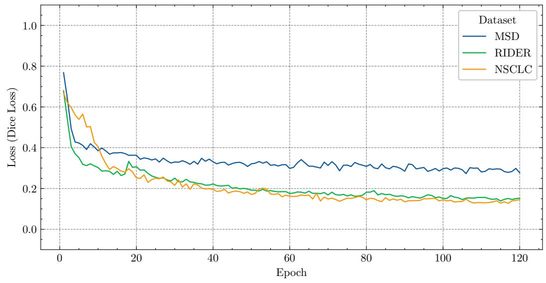


Fig. A.5: The training loss chart for three datasets in VAE shaping model.

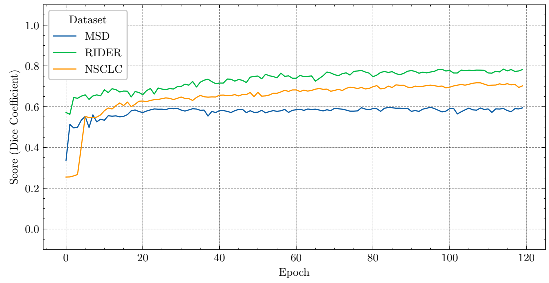


Fig. A.6: The validation score chart for three datasets in VAE shaping model.

## A.4 3D U-Net UDA framework

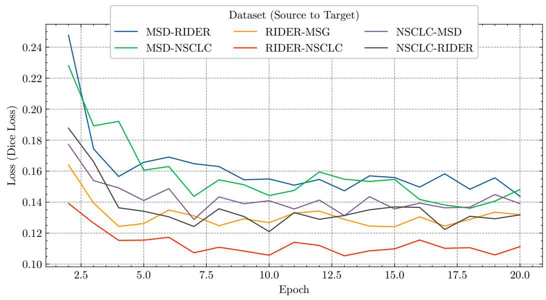


Fig. A.7: The training loss chart for six dataset pairs in 3D U-Net UDA framework.

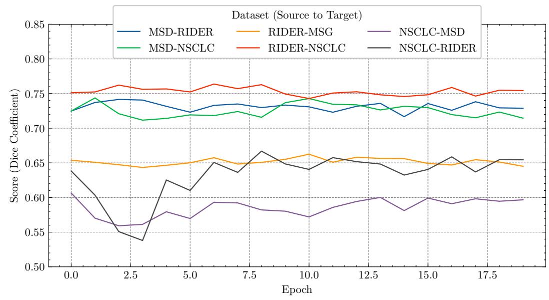


Fig. A.8: The validation score chart for six dataset pairs in 3D U-Net UDA framework.

## A.5 UNETR UDA framework

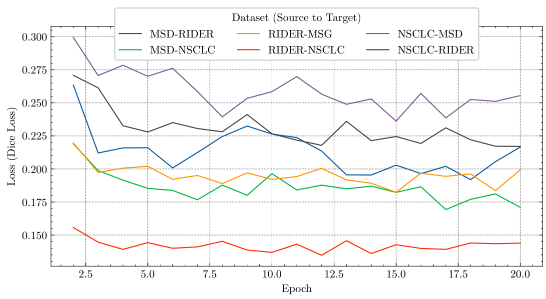


Fig. A.9: The training loss chart for six dataset pairs in UNETR UDA framework.

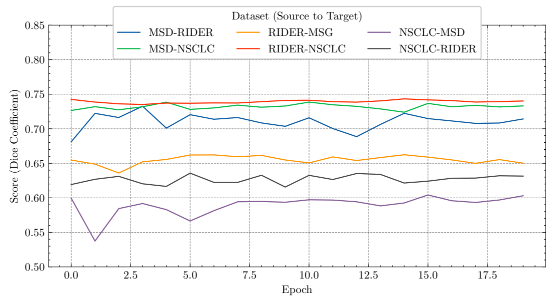


Fig. A.10: The validation score chart for six dataset pairs in UNETR UDA framework.

# Appendix B

# Project Code

## B.1 dcm_to_nii.py

```python
import os

dcm2niix_path = ".\\tools\\MRIcron\\Resources\\dcm2niix.exe"
dcm2niix_command = "{} -o \"{}\" -b n -z y -f \"{}\" \"{}\""

dcmqi_path = ".\\tools\\dcmqi\\bin\\segimage2itkimage.exe"
dcmqi_command = "{} --inputDICOM \"{}\" --outputDirectory \"{}\" --prefix \"{}\" -t nifti"


ct_target_path = "./dataset/dcm_data/NSCLC1/CT"
ct_output_path = "./dataset/nih_data/NSCLC1/CT"
seg_target_path = "./dataset/dcm_data/NSCLC1/Seg"
seg_output_path = "./dataset/nih_data/NSCLC1/Seg"


def convert_ct():
    for root, dirs, _ in os.walk(ct_target_path, topdown=False):
        for name in dirs:
            os.makedirs(ct_output_path, exist_ok=True)
            os.system(dcm2niix_command.format(
                dcm2niix_path,
                ct_output_path,
                name,
                os.path.join(root, name),
            ))


def convert_seg():
    for root, dirs, files in os.walk(seg_target_path, topdown=False):
        for name in files:
            os.makedirs(seg_output_path, exist_ok=True)
            os.system(dcmqi_command.format(
                dcmqi_path,
                os.path.join(root, name),
                seg_output_path,
```

```python
            os.path.basename(root)
        ))


def main():
    convert_ct()
    convert_seg()


if __name__ == "__main__":
    main()
```

## B.2 data_process.py

```
import numpy as np
import nibabel as nib
import skimage
import os
import glob
import tqdm
import re

pad = [32, 32, 32]


def resample(nih, **kwargs):
    scale = np.diagonal(nih.affine)[: 3]
    image = np.flip(nih.get_fdata(), np.flatnonzero(scale > 0))
    new_shape = (image.shape * np.abs(scale)).astype(int)
    image = skimage.transform.resize(image.astype(np.float64),
        new_shape, **kwargs)
    return image


def get_box_index(label):
    mask_index = np.array((label > 0).nonzero())
    max_index = (mask_index.max(axis=1) + pad).clip(max=label.shape)
    min_index = (mask_index.min(axis=1) - pad).clip(min=0)
    center = np.ceil((max_index + min_index) / 2).astype(int)
    length = np.ceil((max_index - min_index) / 2).max().astype(int)
    max_index = (center + length).clip(max=label.shape)
    min_index = (center - length).clip(min=0)
    return max_index, min_index


def get_image_box(image, box):
    max_index, min_index = box
    return image[
            min_index[0]:max_index[0], \
            min_index[1]:max_index[1], \
            min_index[2]:max_index[2], \
        ]


def main():
    image_path = '../dataset/nih_data/NSCLC1/CT'
    label_path = '../dataset/nih_data/NSCLC1/Seg'
    to_path = './nih/nsclc1_test'
    os.makedirs(to_path, exist_ok=True)

    names = glob.glob(os.path.join(image_path, '*.gz'))
    names = [os.path.split(f)[1] for f in names]

    for img_name in tqdm.tqdm(names):
        label_name = re.sub(r"(?=.nii.gz)", "-1", img_name)

        image = nib.load(os.path.join(image_path, img_name))
```

```
        label = nib.load(os.path.join(label_path, label_name))

        image = resample(image)
        label = resample(label, anti_aliasing=False, order=0)

        label_box = get_box_index(label)

        image = get_image_box(image, label_box)
        label = get_image_box(label, label_box)

        path_prefix = os.path.join(to_path, img_name.split('.')[0])
        os.makedirs(path_prefix, exist_ok=True)

        np.save(os.path.join(path_prefix, 'img.npy'),
            image.astype(np.int16))
        np.save(os.path.join(path_prefix, 'label.npy'),
            label.astype(np.int8))
        np.save(os.path.join(path_prefix, 'merge.npy'),
                np.stack((image, label), axis=-1).astype(np.int16))


if __name__ == '__main__':
    main()
```

### B.3 main_source.py

```
import argparse
import os

from batchgenerators.transforms.sample_normalization_transforms import
    RangeTransform
from batchgenerators.transforms.spatial_transforms import
    SpatialTransform
from batchgenerators.transforms.abstract_transforms import Compose
from monai.networks.nets import UNETR
import torch
from torch.utils.data import DataLoader
import tqdm

from models import Segmentation, VAE
from utils import CropResize, TensorBoardWriter, filedict_from_json,
    DiceLoss, NiiDataset, KL_loss


parser = argparse.ArgumentParser()
parser.add_argument("prefix")
parser.add_argument("--method", default='unet_train')
parser.add_argument("--batch_size", type=int, default=2)
parser.add_argument("--max_epoch", type=int, default=120)
parser.add_argument("--save_epoch", type=int, default=20)
parser.add_argument("--data_index", default='./data/data_index.json')
parser.add_argument("--train_list", default='MSD_train')
parser.add_argument("--val_list", default='MSD_val')
parser.add_argument("--train_data_root", default='./data/nih/msd')
parser.add_argument("--val_data_root", default='./data/nih/msd')
parser.add_argument("--save_root", default='./model')
parser.add_argument("--display_root", default='./tensorboard')
parser.add_argument("--checkpoint_name", default="best_model.ckpt")
parser.add_argument("--lr", type=float, default=1e-2)
parser.add_argument("--weight_decay", type=float, default=0)
parser.add_argument("--test_only", action='store_true')
args = parser.parse_args()

prefix = args.prefix
method = args.method
train_batch = args.batch_size
val_batch = 1
max_epoch = args.max_epoch
save_epoch = args.save_epoch
data_index = args.data_index
train_list = args.train_list
val_list = args.val_list
train_data_root = args.train_data_root
val_data_root = args.val_data_root
save_path = os.path.join(args.save_root, prefix)
display_path = os.path.join(args.display_root, prefix)
checkpoint_name = args.checkpoint_name
lr = args.lr
weight_decay = args.weight_decay
```

```python
test_only = args.test_only

num_workers = 4
torch.backends.cudnn.benchmark = True

patch_size = [128, 128, 128]
num_class = 2

for path in [save_path, display_path]:
    os.makedirs(path, exist_ok=True)


def main():
    print("Loading data")
    train_data_list = filedict_from_json(data_index, train_list)
    val_data_list = filedict_from_json(data_index, val_list)

    transforms = {
        "train": Compose([
            CropResize(
                data_key="data",
                label_key="label",
                output_size=patch_size
            ),
            RangeTransform(
                data_key="data",
                label_key="label"
            ),
            SpatialTransform(
                patch_size,
                [dis // 2 - 5 for dis in patch_size],
                random_crop=True,
                scale=(0.85, 1.15),
                do_rotation=True,
                angle_x=(-0.2, 0.2),
                angle_y=(-0.2, 0.2),
                angle_z=(-0.2, 0.2),
                data_key="data",
                label_key="label",
            ),
        ]),
        "val": Compose([
            CropResize(
                data_key="data",
                label_key="label",
                output_size=patch_size
            ),
            RangeTransform(
                data_key="data",
                label_key="label"
            ),
            SpatialTransform(
                patch_size,
                do_rotation=False,
                do_scale=False,
```

```
            do_elastic_deform=False,
            random_crop=False,
            data_key="data",
            label_key="label"
        ),
    ])
}

train_dataset = NiiDataset(train_data_root, train_data_list, transforms["train"])
val_dataset = NiiDataset(val_data_root, val_data_list, transforms["val"])

train_loader = DataLoader(train_dataset, batch_size=train_batch, shuffle=True,
                    num_workers=num_workers, drop_last=True, pin_memory=True)
val_loader = DataLoader(val_dataset, batch_size=val_batch, shuffle=False,
                    num_workers=num_workers, pin_memory=True)

print("Building model")

if method == 'vae_train':
    model = VAE(n_channels=1, n_class=num_class, norm_type=1, dim=128).cuda()
elif method == 'unet_train':
    model = Segmentation(n_channels=1, n_class=num_class, norm_type=1).cuda()
elif method == 'unetr_train':
    model = UNETR(in_channels=1, out_channels=num_class, img_size=(128, 128, 128)).cuda()
else:
    raise ValueError("Try a valid method")

criterion = DiceLoss(num_class=num_class)
optimizer = torch.optim.SGD(
    model.parameters(),
    lr=lr,
    weight_decay=weight_decay,
    momentum=0.9
)

best_result = 0
saver = TensorBoardWriter(display_path)

print("Start training")

for epoch in tqdm.tqdm(range(max_epoch)):
    if not test_only:
        model.train()
        total_loss = 0.0
        for idx, batch in enumerate(train_loader):
            optimizer.zero_grad()

```

```python
            batch["data"] = batch["data"].cuda()
            batch["label"] = batch["label"].cuda()
            batch["label"] = batch["label"].type(torch.cuda.LongTensor)
            one_hot = torch.cuda.FloatTensor(
                batch["label"].size(0),
                num_class,
                batch["label"].size(2),
                batch["label"].size(3),
                batch["label"].size(4)
            ).zero_()
            batch["label"] = one_hot.scatter_(1, batch["label"].data, 1)

            if method in ['unet_train', 'unetr_train']:
                predict_mask = model(batch["data"])

                dice_loss = criterion(batch["label"], predict_mask)
                final_loss = dice_loss

                print('[%3d, %3d] loss: %.4f' %
                      (epoch + 1, idx + 1, final_loss.item()))
            elif method in ['vae_train']:
                reconstruct_mask, mean, std = model(batch["label"])

                dice_loss = criterion(batch["label"], reconstruct_mask)
                kl_loss = KL_loss(mean, std)
                final_loss = dice_loss + 0.00002 * kl_loss

                print('[%3d, %3d] loss: %.4f, %.4f' %
                      (epoch + 1, idx + 1, dice_loss.item(), kl_loss.item()))
            else:
                raise ValueError("Try a valid method")

            total_loss += final_loss.item()
            final_loss.backward()
            optimizer.step()
        saver.add_scale("train_loss", total_loss, epoch)

    print("Start validation")

    model.eval()
    current_result = 0.0
    with torch.no_grad():
        for idx, batch in enumerate(val_loader):
            batch["data"] = batch["data"].cuda()
            batch['label'] = batch['label'].type(torch.cuda.LongTensor)
            one_hot = torch.cuda.FloatTensor(
                batch['label'].size(0),
                num_class,
                batch['label'].size(2),
                batch['label'].size(3),
                batch['label'].size(4)
            ).zero_()
```

```
                batch['label'] = one_hot.scatter_(1, batch['label'].data, 1)

                if method in ['unet_train', 'unetr_train']:
                    prediction = model(batch["data"])
                    current_result += criterion(batch["label"], prediction).item()
                elif method in ['vae_train']:
                    prediction, mean, std = model(batch["label"])

                    dice_loss = criterion(batch["label"], prediction)
                    kl_loss = KL_loss(mean, std)
                    current_result += (dice_loss + 0.00002 * kl_loss).item()
                else:
                    raise ValueError("Try a valid method")

                h = prediction.shape[4] // 2
                saver.add_image(
                    "val_display",
                    torch.cat(
                        (
                            batch["data"][0:1, 0:1, :, :, h],
                            batch["label"][0:1, 1:2, :, :, h],
                            prediction[0:1, 1:2, :, :, h]
                        ),
                        dim=0
                    ),
                    idx + epoch * (len(batch))
                )

            current_result = 1 - current_result / (len(val_loader))

            saver.add_scale("val_score", current_result, epoch)
            print('epoch %d validation result: %f, best result %f.' %
                  (epoch + 1, current_result, best_result))

        if test_only:
            break

        if (epoch + 1) % save_epoch == 0:
            print('Saving model')
            torch.save(
                {
                    'epoch': epoch + 1,
                    'model_state_dict': model.state_dict(),
                    'optimizer_state_dict': optimizer.state_dict()
                },
                os.path.join(save_path, f'model_epoch{epoch + 1}.ckpt')
            )
            if current_result > best_result:
                best_result = current_result
                torch.save(
                    {
                        'epoch': epoch + 1,
                        'model_state_dict': model.state_dict(),
```

```
                  'optimizer_state_dict': optimizer.state_dict()
               },
               os.path.join(save_path, 'best_model.ckpt')
            )

   print('Finished Training')


if __name__ == "__main__":
   main()
```

## B.4 main_target.py

```
import argparse
import os

from batchgenerators.transforms.sample_normalization_transforms import
    RangeTransform
from batchgenerators.transforms.spatial_transforms import
    SpatialTransform
from batchgenerators.transforms.abstract_transforms import Compose
from monai.networks.nets import UNETR
import torch
from torch.utils.data import DataLoader
import tqdm

from models import Segmentation, VAE
from utils import NiiDataset, filedict_from_json, CropResize,
    TensorBoardWriter, DiceLoss


parser = argparse.ArgumentParser()
parser.add_argument("prefix")
parser.add_argument("--method", default='unet_train')
parser.add_argument("--batch_size", type=int, default=2)
parser.add_argument("--max_epoch", type=int, default=120)
parser.add_argument("--save_epoch", type=int, default=20)
parser.add_argument("--data_index", default='./data/data_index.json')
parser.add_argument("--train_list", default='MSD_train')
parser.add_argument("--val_list", default='MSD_val')
parser.add_argument("--train_data_root", default='./data/nih/msd')
parser.add_argument("--val_data_root", default='./data/nih/msd')
parser.add_argument("--save_root", default='./model')
parser.add_argument("--display_root", default='./tensorboard')
parser.add_argument("--checkpoint_name", default="best_model.ckpt")
parser.add_argument("--lr", type=float, default=1e-2)
parser.add_argument("--weight_decay", type=float, default=0)
parser.add_argument("--lambda_vae", type=float, default=0.1)
parser.add_argument("--test_only", action='store_true')
parser.add_argument("--load_prefix_seg", default=None)
parser.add_argument("--load_prefix_vae", default=None)
args = parser.parse_args()

prefix = args.prefix
method = args.method
train_batch = args.batch_size
val_batch = 1
max_epoch = args.max_epoch
save_epoch = args.save_epoch
data_index = args.data_index
train_list = args.train_list
val_list = args.val_list
train_data_root = args.train_data_root
val_data_root = args.val_data_root
save_root = args.save_root
save_path = os.path.join(save_root, prefix)
```

```python
display_path = os.path.join(args.display_root, prefix)
checkpoint_name = args.checkpoint_name
lr = args.lr
weight_decay = args.weight_decay
lambda_vae = args.lambda_vae
test_only = args.test_only
load_prefix_seg = args.load_prefix_seg
load_prefix_vae = args.load_prefix_vae

num_workers = 4
torch.backends.cudnn.benchmark = True

patch_size = [128, 128, 128]
num_class = 2

for path in [save_path, display_path]:
    os.makedirs(path, exist_ok=True)


def main():
    train_data_list = filedict_from_json(data_index, train_list)
    val_data_list = filedict_from_json(data_index, val_list)

    transforms = {
        "train": Compose([
            CropResize(
                data_key="data",
                label_key="label",
                output_size=patch_size
            ),
            RangeTransform(
                data_key="data",
                label_key="label"
            ),
            SpatialTransform(
                patch_size,
                [dis // 2 - 5 for dis in patch_size],
                random_crop=True,
                scale=(0.85, 1.15),
                do_rotation=True,
                angle_x=(-0.2, 0.2),
                angle_y=(-0.2, 0.2),
                angle_z=(-0.2, 0.2),
                data_key="data",
                label_key="label",
            ),
        ]),
        "val": Compose([
            CropResize(
                data_key="data",
                label_key="label",
                output_size=patch_size
            ),
            RangeTransform(
                data_key="data",
```

```
                label_key="label"
            ),
            SpatialTransform(
                patch_size,
                do_rotation=False,
                do_scale=False,
                do_elastic_deform=False,
                random_crop=False,
                data_key="data",
                label_key="label"
            ),
        ])
    }

    print("Loading data")

    train_dataset = NiiDataset(train_data_root, train_data_list, transforms["train"])
    val_dataset = NiiDataset(val_data_root, val_data_list, transforms["val"])

    train_loader = DataLoader(train_dataset, batch_size=train_batch, shuffle=True,
                              num_workers=num_workers, drop_last=True, pin_memory=True)
    val_loader = DataLoader(val_dataset, batch_size=val_batch, shuffle=False,
                            num_workers=num_workers, pin_memory=True)

    print("Building model")

    if method in ["unet_train"]:
        model = {
            "teacher": {
                "seg": Segmentation(n_channels=1, n_class=num_class, norm_type=1).cuda(),
                "vae": VAE(n_channels=1, n_class=num_class, norm_type=1, dim=128).cuda(),
            },
            "student": {
                "seg": Segmentation(n_channels=1, n_class=num_class, norm_type=1).cuda()
            }
        }
    elif method in ["unetr_train"]:
        model = {
            "teacher": {
                "seg": UNETR(in_channels=1, out_channels=num_class, img_size=(128, 128, 128)).cuda(),
                "vae": VAE(n_channels=1, n_class=num_class, norm_type=1, dim=128).cuda(),
            },
            "student": {
                "seg": UNETR(in_channels=1, out_channels=num_class, img_size=(128, 128, 128)).cuda()
```

```
         }
      }
   else:
      raise ValueError("Try a valid method")

   criterion = DiceLoss(num_class=num_class)
   optimizer = torch.optim.SGD(model["student"]["seg"].parameters(),
                        lr=lr, weight_decay=weight_decay, momentum=0.9)

   print("Loading prefix")

   if load_prefix_seg:
      model_path = os.path.join(save_root, load_prefix_seg, checkpoint_name)
      model_state_dict = torch.load(model_path)['model_state_dict']
      model["teacher"]["seg"].load_state_dict(model_state_dict)
      model["student"]["seg"].load_state_dict(model_state_dict)

   if load_prefix_vae:
      model_path = os.path.join(save_root, load_prefix_vae, checkpoint_name)
      model_state_dict = torch.load(model_path)['model_state_dict']
      model["teacher"]["vae"].load_state_dict(model_state_dict)

   for item in model["teacher"].values():
      for param in item.parameters():
         param.requires_grad = False
      item.eval()

   best_result = 0
   saver = TensorBoardWriter(display_path)

   print("Start training")

   for epoch in tqdm.tqdm(range(max_epoch)):
      if not test_only:
         model["student"]["seg"].train()
         total_final_loss = 0.0
         for idx, batch in enumerate(train_loader):
            optimizer.zero_grad()
            model["student"]["seg"].train()
            batch["data"] = batch["data"].cuda()
            batch["label"] = batch["label"].cuda()
            batch["label"] = batch["label"].type(torch.cuda.LongTensor)
            one_hot = torch.cuda.FloatTensor(
               batch["label"].size(0),
               num_class,
               batch["label"].size(2),
               batch["label"].size(3),
               batch["label"].size(4)
            ).zero_()
            batch["label"] = one_hot.scatter_(1, batch["label"].data, 1)

            pseudo_label = model["teacher"]["seg"](batch["data"])
```

```
            predict_mask = model["student"]["seg"](batch["data"])
            restruct_predict_mask, _, _ = model["teacher"]["vae"](predict_mask)

            recon_loss = criterion(restruct_predict_mask, predict_mask)
            pseudo_loss = criterion(pseudo_label, predict_mask)
            final_loss = lambda_vae * recon_loss + pseudo_loss
            total_final_loss += final_loss

            print('[%3d, %3d] loss: %.4f, %.4f, %.4f' %
                  (epoch + 1, idx + 1, final_loss, recon_loss, pseudo_loss))

            final_loss.backward()
            optimizer.step()

        saver.add_scale("train_loss", total_final_loss, epoch)

    print("Start validation")

    model["student"]["seg"].eval()
    current_result = 0.0
    with torch.no_grad():
        for idx, batch in enumerate(val_loader):
            batch["data"] = batch["data"].cuda()
            batch["label"] = batch["label"].cuda()
            batch["label"] = batch["label"].type(torch.cuda.LongTensor)
            one_hot = torch.cuda.FloatTensor(
                batch["label"].size(0),
                num_class,
                batch["label"].size(2),
                batch["label"].size(3),
                batch["label"].size(4)
            ).zero_()
            batch["label"] = one_hot.scatter_(1, batch["label"].data, 1)

            pseudo_label = model["teacher"]["seg"](batch["data"])
            predict_mask = model["student"]["seg"](batch["data"])
            restruct_predict_mask, _, _ = model["teacher"]["vae"](predict_mask, if_random=False, scale=0)

            dice_loss = criterion(predict_mask, batch["label"])
            current_result += dice_loss

            h = predict_mask.shape[4] // 2
            saver.add_image(
                "val_display",
                torch.cat(
                    (
                        batch["data"][0:1, 0:1, :, :, h],
                        batch["label"][0:1, 1:2, :, :, h],
                        predict_mask[0:1, 1:2, :, :, h],
                        pseudo_label[0:1, 1:2, :, :, h],
```

```
                     restruct_predict_mask[0:1, 1:2, :, :, h],
                  ),
                  dim=0
               ),
               idx + epoch * (len(batch))
            )

         current_result = 1 - current_result / (len(val_loader))

         saver.add_scale("val_score", current_result, epoch)
         print('epoch %d validation result: %f, best result %f.' %
              (epoch + 1, current_result, best_result))

      if test_only:
         break

      if (epoch + 1) % save_epoch == 0:
         print('Saving model')
         torch.save(
            {
               'epoch': epoch + 1,
               'model_state_dict':
                  model["student"]["seg"].state_dict(),
               'optimizer_state_dict': optimizer.state_dict()
            },
            os.path.join(save_path, f'model_epoch{epoch + 1}.ckpt')
         )
         if current_result > best_result:
            best_result = current_result
            torch.save(
               {
                  'epoch': epoch + 1,
                  'model_state_dict':
                     model["student"]["seg"].state_dict(),
                  'optimizer_state_dict': optimizer.state_dict()
               },
               os.path.join(save_path, 'best_model.ckpt')
            )

   print('Finished Training')


if __name__ == "__main__":
   main()
```

## B.5 models.py

```python
import torch


def Normalization(norm_type, out_channels, num_group=1):
    if norm_type == 1:
        return torch.nn.InstanceNorm3d(out_channels)
    elif norm_type == 2:
        return torch.nn.BatchNorm3d(out_channels, momentum=0.1)


class DoubleConv(torch.nn.Module):
    def __init__(self, in_ch, out_ch, norm_type=2, soft=False):
        super().__init__()
        activation = torch.nn.Softplus() if soft else
            torch.nn.ReLU(inplace=False)
        self.conv = torch.nn.Sequential(
            torch.nn.Conv3d(in_ch, out_ch, 3, padding=1),
            Normalization(norm_type, out_ch),
            activation,
            torch.nn.Conv3d(out_ch, out_ch, 3, padding=1),
            Normalization(norm_type, out_ch),
            activation,
            torch.nn.Conv3d(out_ch, out_ch, 3, padding=1),
            Normalization(norm_type, out_ch),
            activation
        )

    def forward(self, x):
        x = self.conv(x)
        return x


class Conv(torch.nn.Module):
    def __init__(self, in_ch, out_ch, norm_type=2, num_group=1,
        activation=True, norm=True,
                soft=False):
        super().__init__()
        activation = torch.nn.Softplus() if soft else
            torch.nn.ReLU(inplace=True)
        self.conv = torch.nn.Sequential(
            torch.nn.Conv3d(in_ch, out_ch, 3, padding=1),
            Normalization(norm_type, out_ch),
            activation,
        )

    def forward(self, x):
        x = self.conv(x)
        return x


class Up(torch.nn.Module):
    def __init__(self, in_ch, out_ch, norm_type=2, kernal_size=(2, 2,
        2), stride=(2, 2, 2),
```

```
             soft=False):
      super().__init__()
      self.conv = torch.nn.Sequential(
         torch.nn.ConvTranspose3d(in_ch, in_ch, kernal_size,
            stride=stride, padding=0),
         DoubleConv(in_ch, out_ch, norm_type, soft=False)
      )

   def forward(self, x):
      x = self.conv(x)
      return x


class Down(torch.nn.Module):
   def __init__(self, in_ch, out_ch, norm_type=2, kernal_size=(2, 2,
      2), stride=(2, 2, 2),
             soft=False):
      super().__init__()
      self.conv = torch.nn.Sequential(
         torch.nn.Conv3d(in_ch, in_ch, kernal_size, stride=stride,
            padding=0),
         DoubleConv(in_ch, out_ch, norm_type, soft=False)
      )

   def forward(self, x):
      x = self.conv(x)
      return x


class VAE(torch.nn.Module):
   def __init__(self, n_channels, n_class, norm_type=2, n_fmaps=None,
             dim=1024, soft=False):
      super().__init__()
      if n_fmaps is None:
         n_fmaps = [8, 16, 32, 64, 128, 256]
      self.in_block = Conv(n_class, n_fmaps[0], norm_type=norm_type,
         soft=False)
      self.down1 = Down(n_fmaps[0], n_fmaps[1], norm_type=norm_type,
         soft=False)
      self.down2 = Down(n_fmaps[1], n_fmaps[2], norm_type=norm_type,
         soft=False)
      self.down3 = Down(n_fmaps[2], n_fmaps[3], norm_type=norm_type,
         soft=False)
      self.down4 = Down(n_fmaps[3], n_fmaps[4], norm_type=norm_type,
         soft=False)
      self.down5 = Down(n_fmaps[4], n_fmaps[5], norm_type=norm_type,
         soft=False)
      self.fc_mean = torch.nn.Linear(16384, dim)
      self.fc_std = torch.nn.Linear(16384, dim)
      self.fc2 = torch.nn.Linear(dim, 16384)
      self.up1 = Up(n_fmaps[5], n_fmaps[4], norm_type=norm_type,
         soft=False)
      self.up2 = Up(n_fmaps[4], n_fmaps[3], norm_type=norm_type,
         soft=False)
```

```
        self.up3 = Up(n_fmaps[3], n_fmaps[2], norm_type=norm_type, soft=False)
        self.up4 = Up(n_fmaps[2], n_fmaps[1], norm_type=norm_type, soft=False)
        self.up5 = Up(n_fmaps[1], n_fmaps[0], norm_type=norm_type, soft=False)
        self.out_block = torch.nn.Conv3d(n_fmaps[0], n_class, 3, padding=1)
        self.final = torch.nn.Softmax(dim=1)
        self.n_class = n_class

    def forward(self, x, if_random=False, scale=1, mid_input=False, dropout=0.0):
        if not mid_input:
            x = self.in_block(x)
            x = self.down1(x)
            x = self.down2(x)
            x = self.down3(x)
            x = self.down4(x)
            x = self.down5(x)
            x = x.view(x.size(0), 16384)
            x_mean = self.fc_mean(x)
            x_std = torch.nn.ReLU()(self.fc_std(x))
            z = torch.randn(x_mean.size(0), x_mean.size(1)).type(torch.cuda.FloatTensor)
            if if_random:
                x = self.fc2(x_mean + z * x_std * scale)
            else:
                x = self.fc2(x_mean)
        else:
            x = self.fc2(x)
        x = x.view(x.size(0), 256, 4, 4, 4)

        x = self.up1(x)
        if dropout: x = torch.nn.functional.dropout(x, p=dropout, training=True)
        x = self.up2(x)
        if dropout: x = torch.nn.functional.dropout(x, p=dropout, training=True)
        x = self.up3(x)
        if dropout: x = torch.nn.functional.dropout(x, p=dropout, training=True)
        x = self.up4(x)
        if dropout: x = torch.nn.functional.dropout(x, p=dropout, training=True)
        x = self.up5(x)
        if dropout: x = torch.nn.functional.dropout(x, p=dropout, training=True)
        x = self.out_block(x)
        x = self.final(x)

        if not mid_input:
            return x, x_mean, x_std
        else:
            return x
```

```python


class Segmentation(torch.nn.Module):
    def __init__(self, n_channels, n_class, norm_type=2, n_fmaps=None):
        super().__init__()
        if n_fmaps is None:
            n_fmaps = [8, 16, 32, 64, 128, 256]
        self.in_block = Conv(n_channels, n_fmaps[0],
            norm_type=norm_type, soft=False)
        self.down1 = Down(n_fmaps[0], n_fmaps[1], norm_type=norm_type,
            soft=False)
        self.down2 = Down(n_fmaps[1], n_fmaps[2], norm_type=norm_type,
            soft=False)
        self.down3 = Down(n_fmaps[2], n_fmaps[3], norm_type=norm_type,
            soft=False)
        self.down4 = Down(n_fmaps[3], n_fmaps[4], norm_type=norm_type,
            soft=False)

        self.up2 = Up(n_fmaps[4], n_fmaps[3], norm_type=norm_type,
            soft=False)
        self.up3 = Up(n_fmaps[3], n_fmaps[2], norm_type=norm_type,
            soft=False)
        self.up4 = Up(n_fmaps[2], n_fmaps[1], norm_type=norm_type,
            soft=False)
        self.up5 = Up(n_fmaps[1], n_fmaps[0], norm_type=norm_type,
            soft=False)
        self.out_block = torch.nn.Conv3d(n_fmaps[0], n_class, 3,
            padding=1)
        self.final = torch.nn.Softmax(dim=1)
        self.n_class = n_class

    def forward(self, x, dropout=0.0):
        x1 = self.in_block(x)
        x2 = self.down1(x1)
        x3 = self.down2(x2)
        x4 = self.down3(x3)
        x5 = self.down4(x4)
        x = self.up2(x5)
        if dropout: x = torch.nn.functional.dropout(x, p=dropout,
            training=True)
        x = self.up3(x) + x3
        if dropout: x = torch.nn.functional.dropout(x, p=dropout,
            training=True)
        x = self.up4(x) + x2
        if dropout: x = torch.nn.functional.dropout(x, p=dropout,
            training=True)
        x = self.up5(x)
        if dropout: x = torch.nn.functional.dropout(x, p=dropout,
            training=True)
        x = self.out_block(x)
        if dropout: x = torch.nn.functional.dropout(x, p=dropout,
            training=True)
        x = self.final(x)
        return x
```

## B.6 utils.py

```
import json
import os

from batchgenerators.transforms.abstract_transforms import
    AbstractTransform
import numpy as np
from skimage.transform import resize
from tensorboardX import SummaryWriter
import torch
from torch import nn
from torch.utils.data import Dataset
import torchvision


def filedict_from_json(json_path, key):
   with open(json_path, 'r') as f:
      json_dict = json.load(f)
   listdict = json_dict.get(key, [])
   return listdict


class NiiDataset(Dataset):
   def __init__(self, root_dir, img_dirs, transform=None):
      self.root_dir = root_dir
      self.img_dirs = img_dirs
      self.transform = transform

   def __len__(self):
      return len(self.img_dirs)

   def __getitem__(self, idx):
      merge_data = np.load(os.path.join(self.root_dir,
          self.img_dirs[idx]))
      image = merge_data[np.newaxis, np.newaxis, ...,
          0].astype(np.float32)
      label = merge_data[np.newaxis, np.newaxis, ...,
          1].astype(np.float32)
      data = {"data": image, "label": label}
      if self.transform:
         data = self.transform(**data)
      data = {"data": np.squeeze(data["data"], 0), "label":
          np.squeeze(data["label"], 0)}
      return data


class DiceLoss(nn.Module):
   def __init__(self, num_class, smooth=1):
      super().__init__()
      self.num_class = num_class
      self.smooth = smooth

   def forward(self, inputs, targets):
      dice = 0
```

```
        for index in range(self.num_class):
            dice += self.dice_coef(inputs[:, index, ...], targets[:,
                index, ...])
        return 1 - dice / self.num_class

    def dice_coef(self, inputs, targets):
        # flatten label and prediction tensors
        inputs = inputs.flatten()
        targets = targets.flatten()

        intersection = (inputs * targets).sum()
        dice = (2. * intersection + self.smooth) / (inputs.sum() +
            targets.sum() + self.smooth)

        return dice


class CropResize(AbstractTransform):
    def __init__(self, output_size, data_key="data", label_key="label"):
        self.output_size = output_size
        self.data_key = data_key
        self.label_key = label_key

    def __call__(self, **data_dict):
        image = np.squeeze(data_dict[self.data_key], axis=(0, 1))
        label = np.squeeze(data_dict[self.label_key], axis=(0, 1))

        label_box = self.get_box_index(label)

        image = self.get_image_box(image, label_box)
        label = self.get_image_box(label, label_box)

        image = resize(image, self.output_size)
        label = resize(label, self.output_size, order=0,
                       anti_aliasing=False)

        data_dict[self.data_key] = np.expand_dims(image, (0, 1))
        data_dict[self.label_key] = np.expand_dims(label, (0, 1))

        return data_dict

    @staticmethod
    def get_box_index(label):
        mask_index = np.array((label > 0).nonzero())
        max_index = (mask_index.max(axis=1)).clip(max=label.shape)
        min_index = (mask_index.min(axis=1)).clip(min=0)
        center = np.ceil((max_index + min_index) / 2).astype(int)
        length = np.ceil((max_index - min_index) / 2).max().astype(int)
        pad = int(length * 0.1)
        max_index = (center + length + pad).clip(max=label.shape)
        min_index = (center - length - pad).clip(min=0)
        return max_index, min_index

    @staticmethod
    def get_image_box(image, box):
```

```
        max_index, min_index = box
        return image[
             min_index[0]: max_index[0],
             min_index[1]: max_index[1],
             min_index[2]: max_index[2],
             ]


def KL_loss(mean, std):
    return torch.mean(0.5 * (
         torch.sum(torch.pow(std, 2), 1)
         + torch.sum(torch.pow(mean, 2), 1)
         - 2 * torch.sum(torch.log(std + 0.00001), 1)
    ))


class TensorBoardWriter():
    def __init__(self, logdir):
        self.writer = SummaryWriter(logdir=logdir)

    def add_image(self, tag, image, step):
        self.writer.add_image(
            tag,
            torchvision.utils.make_grid(image.detach()),
            step
        )

    def add_scale(self, tag, num, step):
        self.writer.add_scalar(
            tag,
            num,
            step
        )
```