CT Image Segmentation

Supervisor - Prof Yukun Lai

Zhongwang Li | Aug 2021

Abstract

With the widespread use of computers and the rapid development of artificial intelligence, more and more things are being handed over to computers and artificial intelligence to reduce staff workload. The segmentation of CT images is one of the tasks people want to handle with artificial intelligence models. As a hot topic in computer vision, researchers have been working to improve the accuracy and reliability of CT image segmentation. Today there are several established solutions and models to solve this problem. This project will focus on how one of the famous models, U-NET, performs on this task and can be adjusted and improved to improve performance further. Several approaches were tried, including increasing training data sets, using different feature extraction methods and adjusting the Skip-connection structure of the U-Net network. We measured the accuracy of these models, the Dice Score and the size of the models. We also visualize the prediction results of each version of the models to compare the differences between them better. Finally, the limitations of the project and the models used are discussed and possible future improvements.

Contents

CT Image Segmentation 1
Abstract 2
Contents 3
1. Introduction 4
2. Background 5
2.1 Current research 5
2.2 CT images and its segmentation 6
2.3 Convolutional Neural Network 7
3. Methodology 8
3.1 Loss 8
3.2 Optimization 9
3.3 U-Net 9
3.4 Feature Extraction module 10
3.5 Skip-connection 11
3.6 Data augmentation 12
4. Implementation 13
4.1 Framework 13
4.2 Code Deatails and Explanation 13
4.3 Model Version 18
5. Result & Discussion 19
6. Limitation and Future work 23
7. Conclusion 24
8. Reflective Learning 25
Acknowledgments 28
References 29

1. Introduction

In the medical area, there is a huge variety of images can have a major influence on doctors' diagnosis. Examining these images manually necessitates a significant amount of time and effort, such as detecting tissue borders, estimating tumor sizes and understanding the anatomical structure of organs [1]. Fortunately, a number of computer applications can assist doctors and researchers in analyzing the image, reducing their burden significantly. In these analysis programs, one basic process is the prerequisite for most of the further analysis: image segmentation. Once the target region has been isolated from the rest of the picture, doctors can better study it and acquire more quantitative measures from the segmentation result. Therefore, computer researchers and algorithm engineers have constantly been working on how to accurately separate target regions.

In today's rapidly developing artificial intelligence (AI) environment, as one of AI's key branches, computer vision plays a vital role in medical image segmentation. Until now, AI researchers have used a variety of models and algorithms to distinguish target regions in medical pictures. Several of these algorithms and models have shown excellent outcomes. This project also aims to use computer vision models that have done well in previous studies to challenge CT image segmentation.

CT images are usually large in number, with relatively simple texture and greyscale color. These CT image properties are beneficial to the computer vision model. The model does not need to learn and retain a large number of characteristics because of the greyscale color and basic texture. The CT image data size ensures that it has enough samples to obtain and learn the information and features. All of this demonstrates that utilizing computer vision to complete the task of segmentation is sensible and feasible.

This project aims to build a machine learning model that can distinguish the target area/organ and the background in a CT image. The model will be trained with CT images and its masks. The mask is a binary color and pixel match image to the input CT image, with white representing the target area and black representing the undesired background. The final model will take CT pictures as input and output the CT image mask.

The heart of the implementation in this project is the U-NET architecture with convolutional feature extraction module. Also, some attempts inspired by articles and to challenge oneself have been tried implementing into the project. Although not necessarily improving the final model in terms of score, these experimental attempts provide an additional way of thinking and evidence of the author's thought process.

2. Background

2.1 Current research

Researchers have designed numerous frameworks that exploit the features of Convolutional Neural Networks (CNNs) to deal with medical image segmentation. Long et al. [2] presented the famous Fully Convolutional Network (FCN) to address the constraint that typical CNN networks can only classify but not segment images. This makes it possible to use CNN to output objects' mask.

However, the FCN has its limitations; it is unable to produce a clear and detailed result. To improve the accuracy of the segmentation boundaries, Chen et al. [3] proposed DeepLab v1. DeepLab v1 is modified based on the VGG-16 network, which is another well-known CNN structure. Compared to FCN, DeepLab v1 adjusted the use of the pooling layer and added atrous convolution. Therefore, DeepLab v1 has a wider receptive field, and the density of the feature maps increases, leading to a better segmentation result. After that, DeepLab v2[4], DeepLab v3[5], and DeepLab v3+[6] were proposed one by one. They used ResNet-101 as the backbone and raised the idea of atrous spatial pyramid pooling (ASPP), and designed the cascaded or parallel atrous convolution module to improve the result. In DeepLab v3+, a decoder module is used to refine the boundary of the segmentation target, which shows an effective way to improve the performance.

The SegNet[7] goes one step farther than the DeepLab v3+; it is an encoder-decoder symmetric structure based on the FCN end-to-end approach for pixel-level image segmentation. The encoder is used to extract and analyze object information, while the decoder is used to construct the segmentation result using the information obtained during the encoder phase.

In 2015, one of the most famous and widely used structure, U-Net was proposed by Ronneberger et al [8]. It has a SegNet-like encoder-decoder design, and a unique skip-connection design which can merge the low-level and high-level features. The low-level features provide details to help improving accuracy, whereas the high-level features allow model to extract features with more complicated expression. The combination of these two special designs is why U-Net is outperformance in the medical image segmentation task. The U-Net also has various of derivative such as V-Net[9], W-Net[10] and Res-UNet[11].

2.2 CT images and its segmentation

Computed tomography scan is usually known as CT scan, which uses the property that x-rays attenuate differently in substances of different densities, using x-rays and its receivers to scan, calculate and image the cross-sections of the human body. Due to the imaging principle, the images will be grey, with varying shades of color indicating different densities of bodily tissue. Image segmentation process can divides an image into several groups and by classifying each pixel [12]. Through segmentation algorithms, different organs or objects can be separated and used for further analysis.

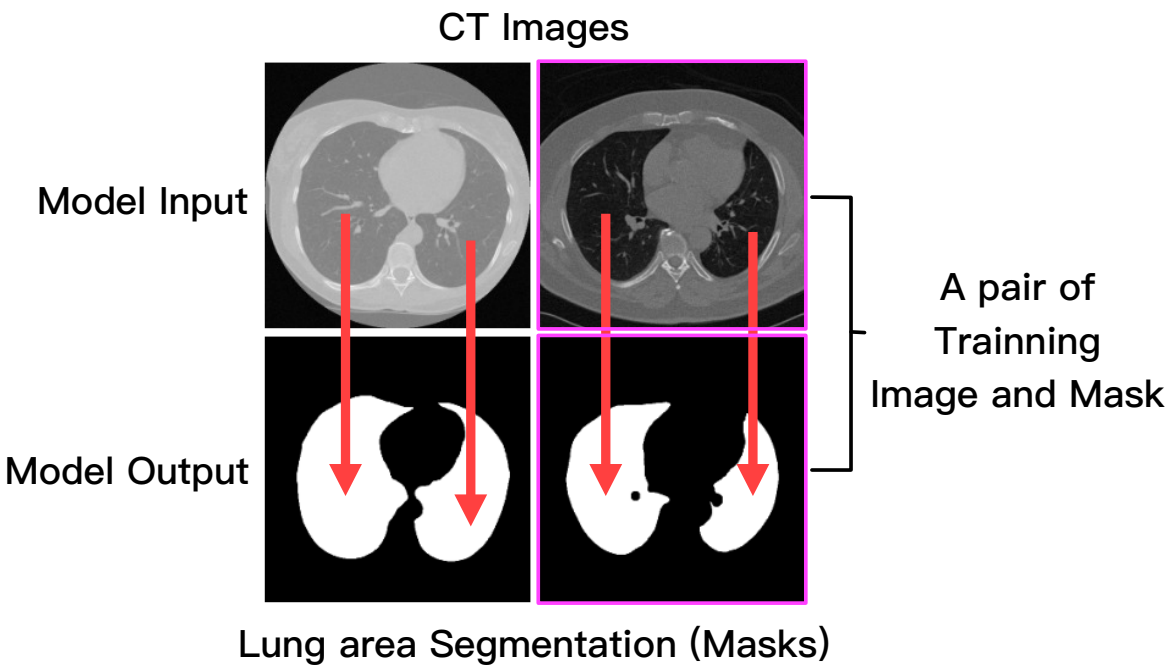

Figure 1. An example of CT image and it's mask

2.3 Convolutional Neural Network

Convolutional neural network (CNN) is a very common artificial neural network applied in the computer vision area [13].

The convolutional layer is the key design of CNN. A filter, also known as 'kernel', will sweep over the input image in the convolutional layer. The kernel functions as a detector for a certain item, and there may be hundreds or even thousands of them searching for and capturing different objects or characteristics in the input. By integrating the search results of these kernels, CNN will be taught to do tasks such as object identification, image classification, and object segmentation. The CNN training process determines what type of object detector (kernel) it will require to do its mission. The item that a kernel need to detect is represented by the parameters in the kernel. In conclusion, during the training process, the algorithms attempt to adjust all the kernels' parameters to recognize the valuable features for the model to finish its task.

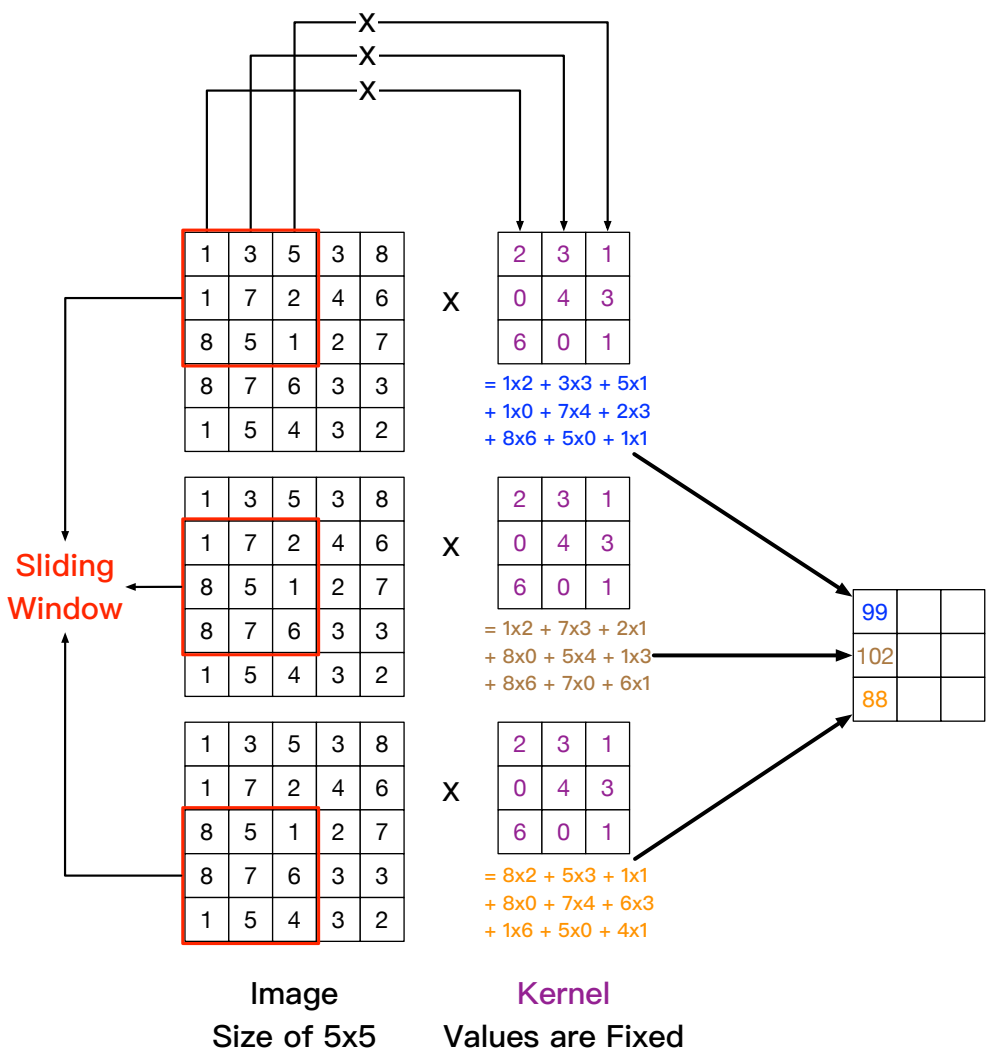

Figure 2. Kernel working mechanism example

3. Methodology

3.1 Loss

Loss function is used to obtain how difference between the predict result and the ground truth. A good loss function can avoid gradient explosion and gradient disappear problems while also shortening the model's convergence time. A classic loss function for the binary classification task called binary cross-entropy loss[14] (BCE) is a good start for this project's segmentation challenge.

BCE equation:

$$Loss = -\frac{1}{output\ size} \sum_{i=1}^{output\ size} y_i * \log \hat{y}_i + (1 - y_i) * \log (1 - \hat{y}_i)$$

The $\hat{y}_i$ is the ith scalar value of the model output, y_i is the corresponding target value, and output size is the number of scalar values in the model output. However, the model's prediction value, $\hat{y}_i$, might be some unreasonable large or small, which will cause the BCE loss to become unstable. An advanced loss function called 'BCE WITH LOGITS LOSS' is introduced and applied in this project.

BCEWITHLOGITSLOSS:

$$Loss = -\frac{1}{output\ size} \sum_{i=1}^{output\ size} y_i * \log \sigma(\hat{y}_i) + (1 - y_i) * \log (1 - \sigma(\hat{y}_i))$$

BCEWITHLOGITSLOSS[15] will perform a sigmoid function on the $\hat{y}_i$ before calculating the BCE loss. The sigmoid function[16] will normalize the $\hat{y}_i$ and converts it to a number between 0 and 1. The advantage of doing so is that it avoid the instability produced by the ludicrous prediction and speeds up the convergence time of the models.

Sigmoid function:

$$\sigma(\mathrm{x}) = \frac{1}{1 + e^{-x}}$$

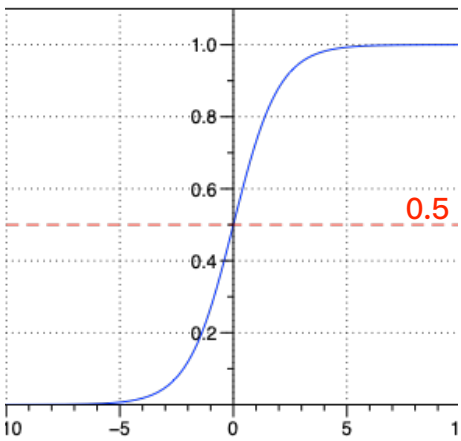

Figure 3. Sigmoid function

3.2 Optimization

The loss function tells the model how the difference between its prediction and the true result. An optimizer is then required to correct its internal parameters and improve the prediction result. The gradient descent approach is the fundamental theory of optimizer. It tries to discover the potential global minimum point of the loss function. Optimizer used in this project called Adaptive Moment Estimation[17] (Adam).

Adam optimizer can against high variance during the derivative calculation and correct the gradient vanishing due to an unsuitable learning rate. Also, Adam can converge the model rapidly. The tradeoff of these benefits is that it requires more computational power because the first and second moments is required to be obtained.

3.3 U-Net

The U-Net framework is the backbone of this project. Various changes and adjustments will be implemented on the U-Net.

The encoder, bottom convolution, and decoder are the three major components. The number of layers in the encoder and decoder are the same. Both the encoder and decoder portions employ double convolution.

Convolution is applied to the input in each encoder layer to double the picture channels, and then max pooling is used to reduce the image size. The convolution part at the bottom is identical to the encoder section, except that the image size is not decreased any more. The decoder, on the other hand, employs convolution to cut the image channels in half and then uses up-convolution to expand the image size.

One of the most important features of U-NET, the skip-connection, is how the decoder concatenates the result of the preceding encoder at the same level to the current input of the decoder, which doubling the picture channels.

In the final part of U-Net, the image is convolved to produce a masked image; the number of channels is the number of types of segmentation required.

U-Net's encoder-decoder design allows each layer's convolution kernel view to be gradually expanded, resulting in features of various sizes and kinds. Some of the data lost during encoding stage can be restored by skip-connection. These two designs enable U-Net to conduct image segmentation with greater accuracy while decreasing the number of parameters that need to be trained.

In the U-Net, inspired by papers, feature extraction modules in the encoder-decoder and skip-connection can be modified to improve outcomes.

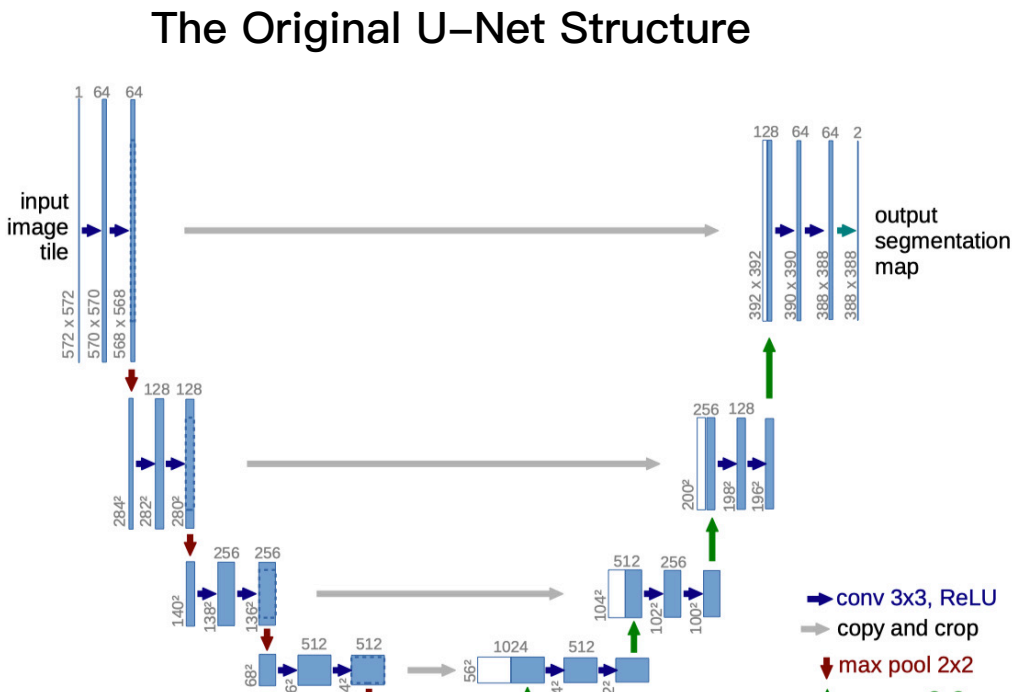

Figure 4. U-Net structure from the original paper

3.4 Feature Extraction module

Feature Extraction module is used in bottom convolution and every encoder and decoder layer. The main job of a feature extraction module is to learn one feature from the current input. During the inference time, the feature extraction module will sweep through the input and find out if a part of the image has a similar feature as the one it holds. A well-designed feature extraction module is responsive to different image features while requiring little computational resources.

3.4.1 DoubleConv

Double convolution is the one used in the original paper. It is an application of two unpadded convolutions, each followed by a ReLU. A batch normalization unit is added after ReLU to balance the data between layers and reduce training time.

3.4.2 SPConv

A split based convolutional operation module, called SPConv is proposed in [18]. In this paper, it was found that some of the features obtained by the feature extraction module are

duplicated or very identical; if a mechanism can be developed to minimise the redundant features, training and inference may be sped up even further. SPConv was one approach of achieving this aim. The SPConv split features into two parts, one for representative feature learning and the other to supplement minor useful information as a redundant portion. The result demonstrates that this plug-and-play module has a better performance than Vanilla convolution (the ground truth used in this paper).

In this project, due to the nature of CT images, the issue of redundant features will undoubtedly occur. SPConv was implemented and expected to mitigate this problem.

3.4.3 MFEConv

Multi-scale Feature Extraction (MFE) module

Lesion regions in CT scans vary in size and shape for different people. The ordinary convolution module can only extract feature information at one fixed scale, which cannot capture multi-scale representation and high-dimensional information. Although, the encoder-decoder can help with this issue in the model structure, the model's performance will improve further if the multi-scale feature can be extracted in each convolution. Inspired by [19], a MFEConv module is designed. The MFEConv split the channels or features of the input into several parts. To generate different sizes of features, different convolution and pooling methods will be used to each component. The features of various scale will then be combined into a single feature map, which will be added to the original input to improve the learnt multi-scale feature.

3.4.4 CIEConv

Context Information Extraction (CIE) module

One thing that is inevitable during the encoding phase is that information from the input will be lost due to the Max Pooling compression procedure. U-Net attempted to solve this problem with the skip-connection design. However, if more context content characteristics can be collected during feature extraction, greater information loss may be compensated. The CIE module [19] was created with this in mind. On the input feature map, CIEConv performs a sequence of various rate of dilation convolutions, then fuses the dilation result to allow the feature map to incorporate additional context information.

3.5 Skip-connection

3.5.1 Normal

In the U-Net's original architecture, there exist connecting pathways between the corresponding layer level of encoding and decoding phase, as described in the U-Net section. These routes allow the decoder to acquire information prior to Max Pooling loss, resulting in improved segmentation accuracy and performance.

3.5.2 Residue Attention (RA) Module

On the decoder, skip-connection performs well with supplement detail information; however, it brings the information before feature extraction, which may affect the segmentation result.

In the [19], a residue attention (RA) module is proposed to regulate the weights on the connections, with the goal of suppressing irrelevant information and emphasize key features. RA module requires current level encoder result and previous level decoder result as the input. The below diagram illustrates the basic workflow of RA module.

3.6 Data augmentation

Multiple transformations were applied to the dataset to increase the dataset size and force the model to learn against noise and the background. Rotation, horizontal and vertical flipping and elastic transform were the major transformations applied.

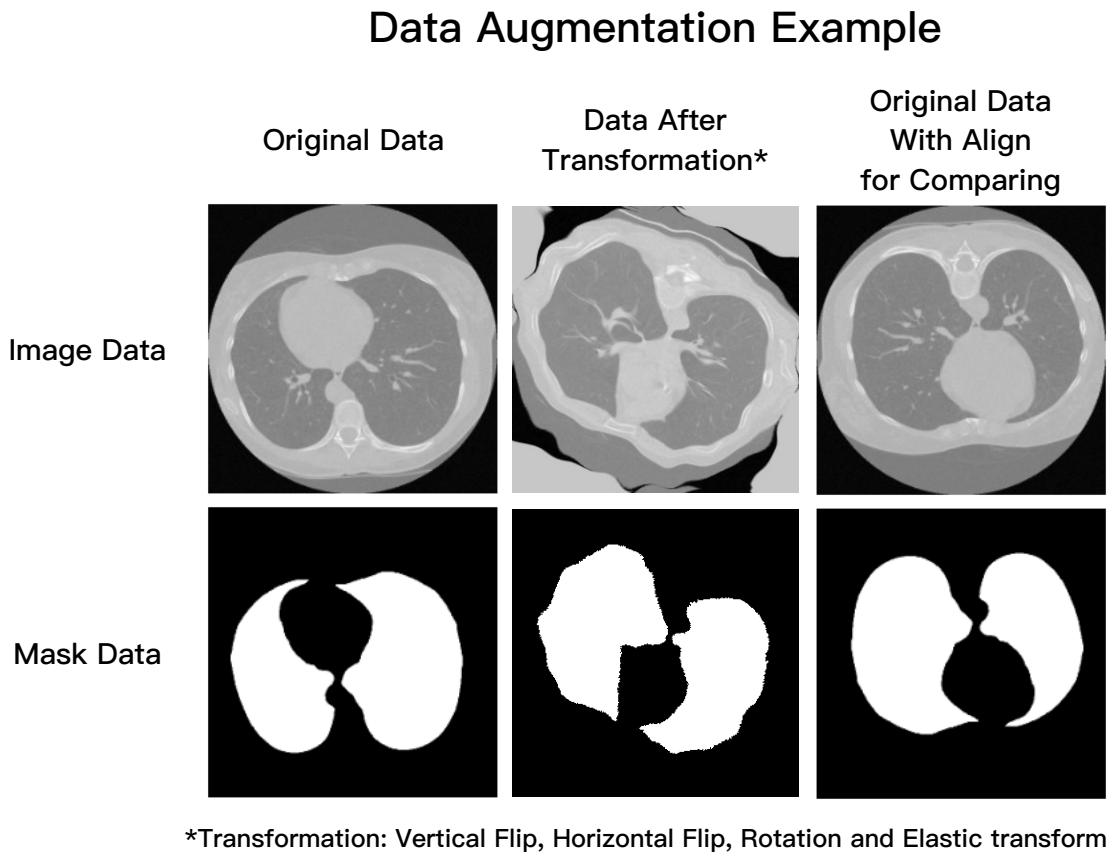

Figure 5. An example of Data transformation in Data Augmentation process

4. Implementation

4.1 Framework

4.1.1 Python

The programming language used in this project was Python. Python is the most popular programming language in the fields of data science and artificial intelligence at the moment. It is simple to learn, and there are many mature and reliable libraries and packages, which can greatly reduce the workload.

4.1.2 PyTorch[20]

PyTorch is one of the most famous machine learning libraries in python. It contains countless classes and methods related. The superclass of the model used in this project inherits from the nn.Module of the PyTorch and optimization method used to train the model and reduce model loss is from the Optim module of PyTorch.

4.1.3 Colab[21]

Colab is a cloud computing platform provide by Google. It allows user to implement and test their model and code with Google's powerful hardware and GPU. Models in this project are all trained and test on the Colab.

4.2 Code Detail and Explanation

4.2.1 Dataset

A class called 'ImageDataset' inherited from the 'Dataset' class from 'torch.utils.data' is created. The path to the images' and masks' folders, as well as the transformation the user wants to apply to the images and masks, are the class's inputs. According to the specified index, the class will return the image and its mask pair. The image will be opened and converted to three-channel RGB data, after which it will be stored as a np.array type. Unlike the image, the mask will be transformed to single channel 'L' data, which indicates greyscale, and then stored as float32 np.array type data. The mask data will then be adjusted to [0,1] from [0,255].

If any transform is fed in, it will be applied to both the image and mask. Mask should be still able to cover all the target areas of the image after the transformation. Finally, the image and mask will be returned.

4.2.2 Dataset augmentation

A dataset class script and a data augmentation script are used to increase the input data size. The dataset script is nearly identical to the 'dataset.py' script, which is used to construct dataset objects to store the image and mask. The dataset class used here is called 'LungCTImagesDataset'.

Package 'albumentations' is commonly used in data augmentation. The transformations

'Resize', 'Rotate', 'HorizontalFlip', 'VerticalFlip', and 'ElasticTransform' are specified in a pipeline workflow called 'my transforms' in the 'data augmentation.py' file. Then, as two 'LungCTImagesDataset' objects, the training and validation data are read in and loaded.

The training and validation datasets are then read ten times, with the produced image and mask files being saved independently each time. The image-mask pair's transformation will be randomized for each time of image generation.

4.2.3 Model

4.2.3.1 U-Net

The core structure of U-Net is defined in this class 'UNET', which is inherited from 'nn.Module'. The default values for 'in channels', 'out channels', and 'features' in the 'UNET' class are 3, 1 and [64, 128, 256, 512] accordingly. Five subparts of 'UNET' are defined in the initialization section.

'self.pool' is the Max Pooling function used to reduce the image size during the encoding phase.

'self.up' is a list of DoubleConv functions with gradually increased in_channels and features (out channels). All these DoubleConv are used in the encoding phase. The number of DoubleConv equals the number of encoding layers.

'self.down' is a list of ConvTranpose2d and DoubleConv. These modules are used in the decoding phase. ConvTranpose2d is used to double the image size. DoubleConv functions have gradually decreased in channels and out channels.

'self.bottleneck' is a DoubleConv used to double the feature number (image channels). It is the bottom part of the U-Net.

'self.final_conv' is a Conv2d module with a kernel size equal to one. This convolution 'combines' all of the features to determine the class of a pixel point. As a result, the input image channel has the fewest amount of features in the 'features' list, while the output channel has only one.

The data flow of the input is defined in the forward function of the 'UNET' class. The components in the 'self.down' processed input during the encoding phase. The result after each processing was saved in the 'skip_connections', then the result is shrinked by 'self.pool'.

The 'self.bottleneck' is then applied, and the 'skip_connection' was reversed.

In the decoding phase, input was first processed by the 'ConvTranspose2d' in the 'self.up' to recover the image size, then resized to match the saved data of the corresponding level in the 'skip_connection'. The result of the 'skip_connection' is then concatenated with the data from the 'skip connection'. To fuse the channels and learn the features, 'DoubleConv' is applied to the data.

Finally, 'self.final_conv' is applied to obtain a single channel black and white mask image.

4.2.3.2 Feature Extraction

Feature Extraction is another part in the 'model.py'. It normally contains one or more class used in the 'UNET' class.

- DoubleConv

  A sequential module list is defined. It has two Conv2d with kernel size 3x3. Each of the Conv2d followed by a BatchNorm2d and a ReLU. It requires user to define the number of 'in_channels' and 'out_channels'.

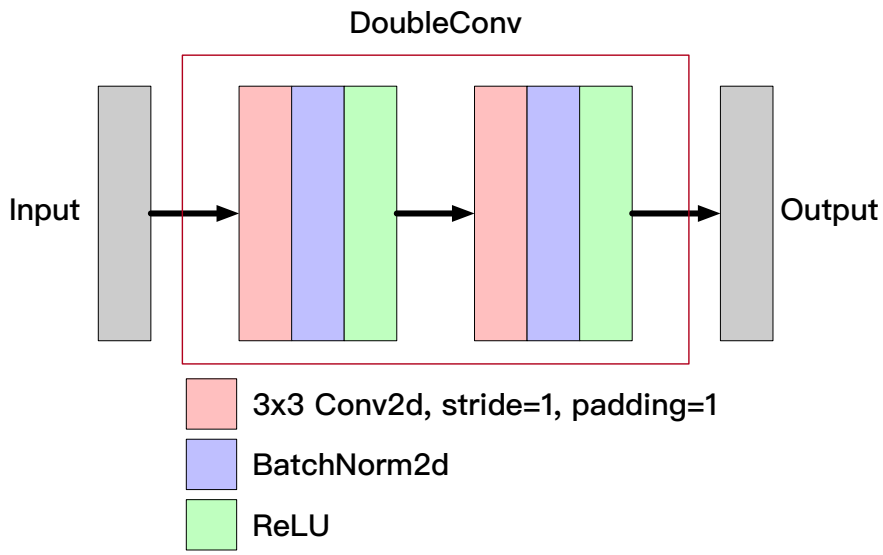

Figure 6. DoubleConv Structure

- SPConv

  This module's code comes from the code example provide by the original paper. In this project, one version of model uses SPConv instead of the Conv2d in the DoubleConv module.

- MFEConv

  Because MFEConv only works when the input channel number exceeds 8, the input channels must first be verified during the initialization phase. If the channel number is less than 8, the input will be sent via a Conv2d to raise it to 8. The number of channels needed for each branch is then calculated. This calculated result will then be utilized as the Conv2d module's setting in each branch. MaxPool2d is defined since it will be used in one of the branches. The 1x1 Conv2d module 'self.smoothConv1x1' is used to fuse the four branches results.

  The input channel will be checked and raised if it is less than 8 in the forward section. The 'torch.split' method will then split the input into four parts in the view of the channel. The splitting ratio is 1/4, 1/8, 1/2 and 1/8. Branch one will use a 1x1 convolution, whereas branch two will use MaxPool first and then a 1x1 convolution. Branch three does a 1x1 and a 3x3 convolution in turn, while branch four performs a 1x1 and two 3x3 convolutions in turn. 'torch.cat' will concatenate the results of the four branches in the channel's view. Then 'self.smoothConv1x1' will fuse this result and add it to the original input (the one before the channel was enlarged) as the final result.

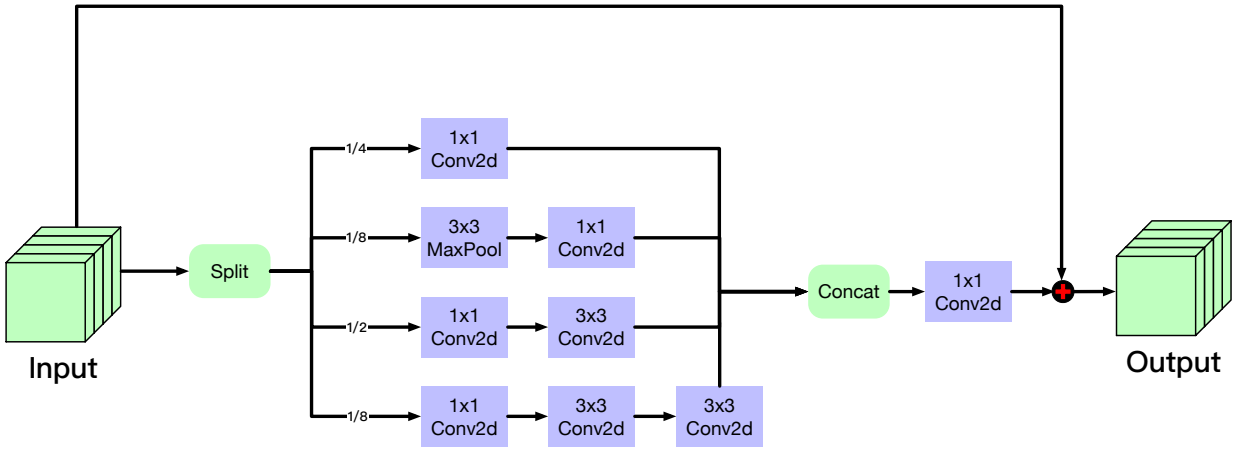

Figure 7. MFEConv Structure

- CIEConv

  The initialization of CIEConv requires three settings: 'in channels', 'kernel size', and 'dilation'. By default, 'kernel size' is set to 3 and 'dilation' is set to [2,6,10]. Then three Conv2d are constructed, each with a different dilation rate set in the 'dilation' parameter. The 'self.Conv1x1' is a 1x1 Conv2d with three times of 'in_channel' as input channel number and 'in_channels' as the output channel number. This is because the input must restore to its original shape after passing through the three dilation convolutions and being concatenated together.

  In the forward part, the input will go through the three dilation convolutions individually, stack together, and fuse by 'self.Conv1x1'. This result will then be added to the input.

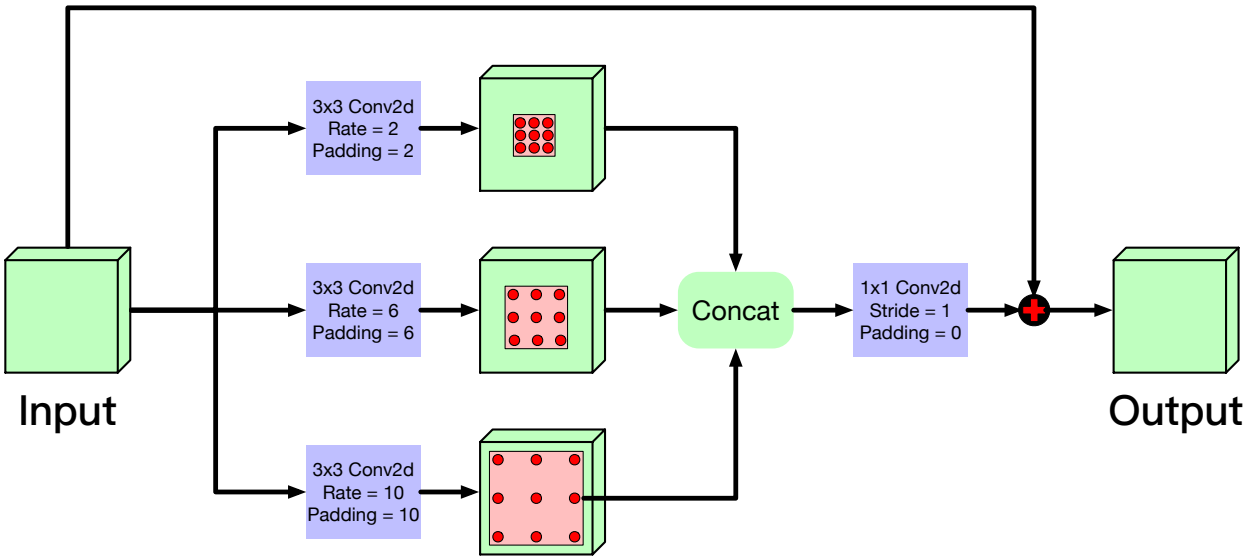

Figure 8. CIEConv Structure (red spots represent different dilation rate of Conv2d)

- ConvModule

  A module combines the CIEConv, DoubleConv and MFEConv.

4.2.3.3 Residue Attention (RA) module

The RA module is in the process of being decoded. Conv2D modules are designed to match the current result and the next level's skip-connection data to the same channels and sizes. Then, using 'nn.Unsample' with a scale factor of 2 and bilinear mode, they are added together and up-sampled. After that, ReLU, a 1x1 Conv2d, and Sigmoid modules rectify and normalize this result. The result or weight map is referred to as alpha, and it is multiplied by skip-connection data before being added to the same skip-connection data. This result will be taken as the new skip-connection data.

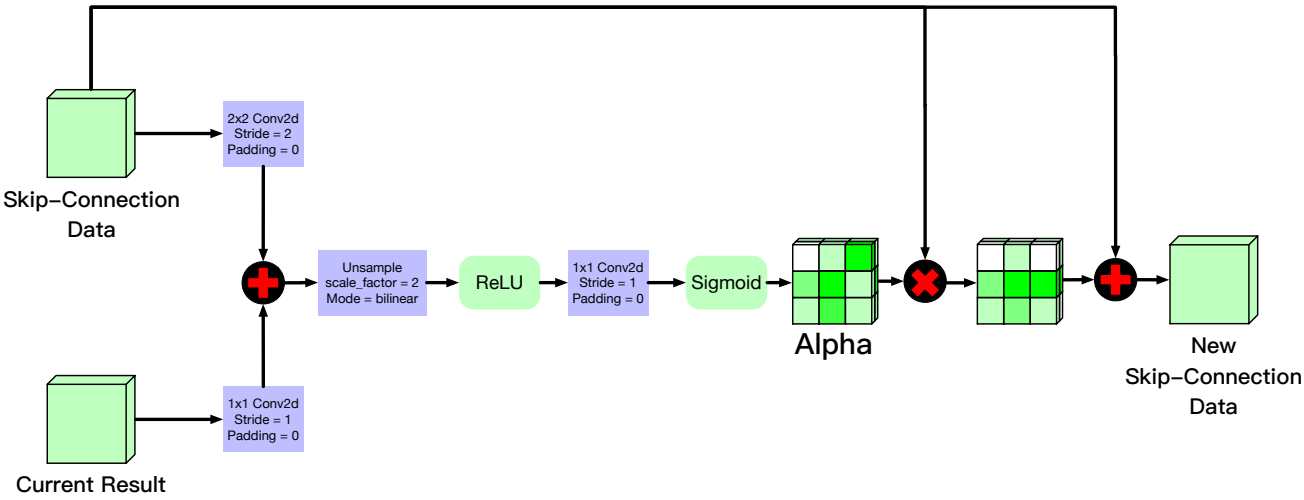

Figure 9. RA Structure

4.2.4 Utils

This file contains all of the tools needed for the training process. The checkpoint function saves and loads model states, allowing the user to extract and restore the trained model.

The 'get loaders' function is used to transform the input images and masks into batch size dataloader objects, which make training the model easier.

The 'check_accuracy' function will give the accuracy and dice score[22] feedback on the given (validation) dataset under the current model.

The file's last function, 'save_predictions_as_imgs', does exactly what its name implies: it uses the current model state to predict the mask of the supplied images and saves the result as a file. This feature allows the user to more immediately and intuitively assess the model's performance.

4.2.5 Train

The core training procedure is contained in this file. At the top of the file, the hyperparameters are defined. The training and optimization in each epoch takes place in the function 'train_fn'. It will read the data in batches, with the model attempting to predict the outcome for each batch. The loss function will then compare the predicted and true outcomes. After that, the optimization function will update the model parameters in each layer based on the loss function result. After the 'train_fn' go through the whole data once, one epoch of training is finished.

The transformation used in the train and validation dataset is first defined in the 'main' function. The model, loss function, and optimizer are then established. Following these initial parameters, the function will use the 'get loaders' function to acquire the data loader and begin training the model. Several epochs will be trained by the model. In each epoch, the 'train_fn' will run, then the checkpoint, which is the model's current status, will be saved. Following that, the forecast image as well as the current model's accuracy will be created and stored.

4.3 Model Version

- Ver1 UNET with DoubleConv, No Data Augmentation
- Ver2 UNET with DoubleConv, Data Augmentation
- Ver3 UNET with SPConv used in DoubleConv, Data Augmentation
- Ver4 UNET with DoubleConv, RA design implemented, Data Augmentation
- Ver5 UNET with ConvModule, RA design implemented, Data Augmentation

5. Result & Discussion

The following two figures show the prediction results of five versions of the model on the validation set after training for 100 epochs. These images provide a visual representation of the model's performance.

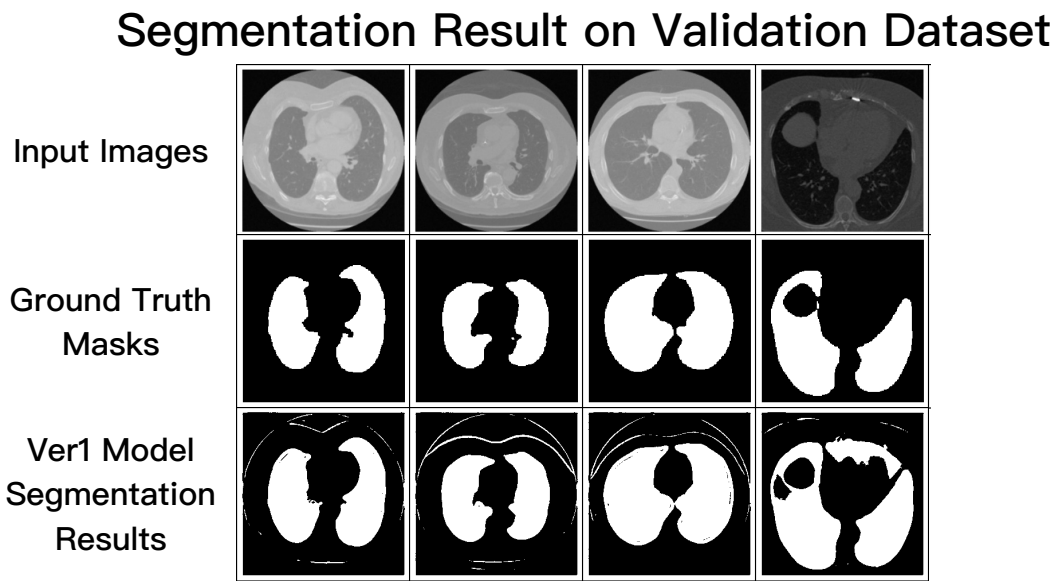

Figure 10. Predict result for the Model Version 1

The target region can be segmented basically using Model Version 1. There are, however, several apparent flaws. First, because the model has trouble distinguishing rounded edges, the edges in the results are left with white borders. Because the width of the white borders is so little, it has little effect on the accuracy, but it is plainly evident to the naked eye. The model also contains a significant amount of misclassification. These are most likely due to the limited training dataset. The model was unable to acquire enough features from the training data, particularly for the target region's edges.

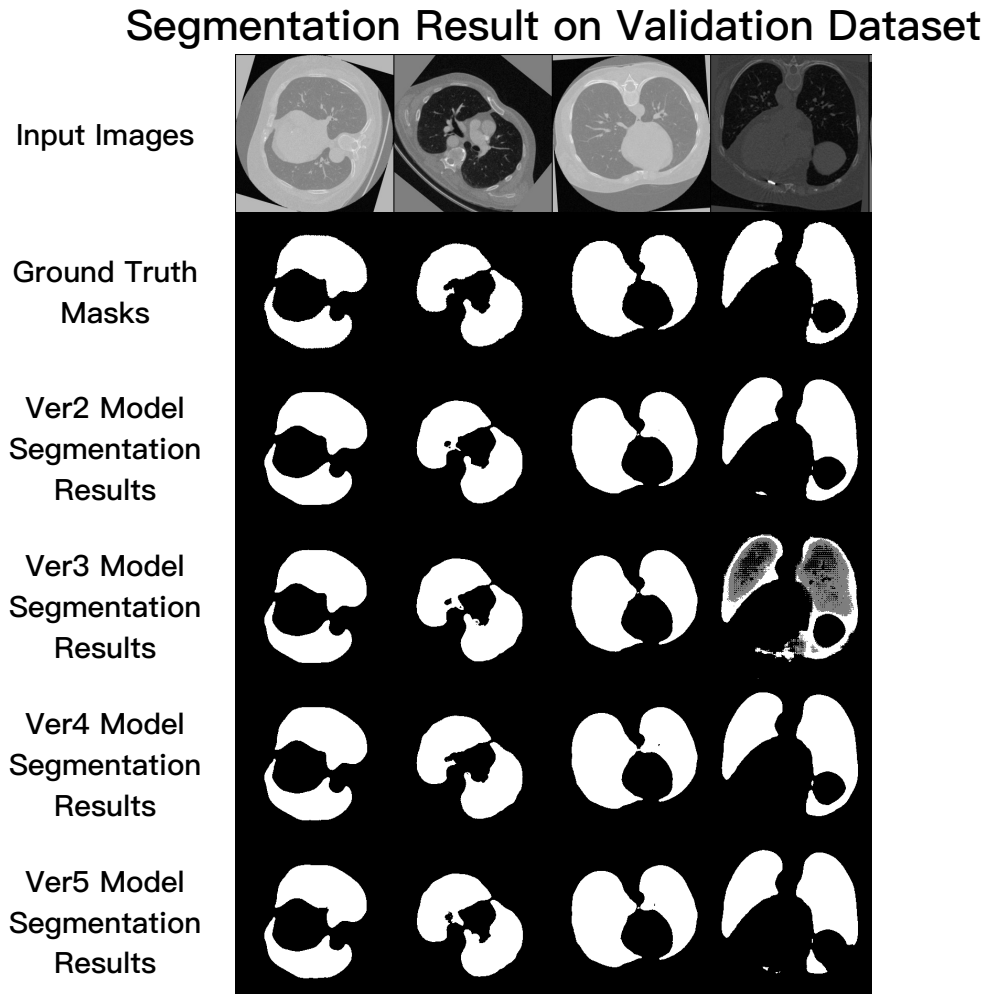

Figure 11. Predict results for the Model Version 2, Model Version 3, Model version 4 and Model Version 5

The next four model versions all solve the problem of white edges in the predictions and greatly enhance the segmentation results at the edges of the target region since the training dataset is 10 times larger. The fourth model had the highest visual performance, maybe because the RA module improved the classification results by improving the expressiveness of the features and weakening the irrelevant information while recovering the information lost during the encoding phase. Model Version 3 shows an apparent shortcoming, with unclear classification in the target region as a grid style. It's conceivable that this is due to SPConv's removal of too many redundant features, making it difficult to learn as many features as other models with the same number of training epochs and batch size hyperparameters. Model Version 5 contains a few more details and missing pieces than the solution, although it was only trained for 10 epochs.

The two graphs below demonstrate how the accuracy of the five model versions varies over time as they are trained.

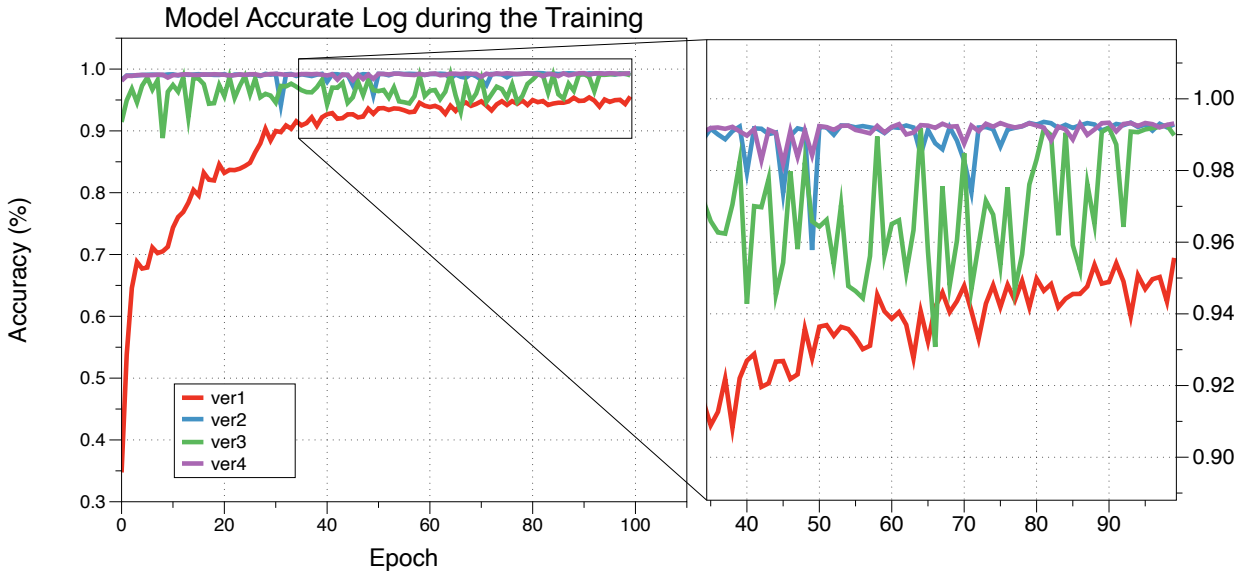

Figure 12. Accuracy Log of Models

At the 60th epoch, Model Version 1's accuracy begins to stabilize and eventually reaches 95%. The accuracy of the remaining four model variants all exceeds 90% at the start of training and finally settles around 99%. It's conceivable that this is due to the large size of the training dataset. Because the dataset is 10 times larger, one epoch of training for the previous four versions of the model is roughly equivalent to performing ten epochs for Model Version 1. As a result, these models can acquire better accuracy much more quickly. During training, the accuracy of the Model Version 2 and Model Version 4 models changes relatively slightly, with only a few major variations.

Model Version 3 has, on the other hand, seen unpredictable oscillations, dropping from 99 percent to 90 percent at times. This shaking was nevertheless taken into account since SPConv eliminated too many features, resulting in fewer stable features being learnt, which had a significant impact on the model's accuracy.

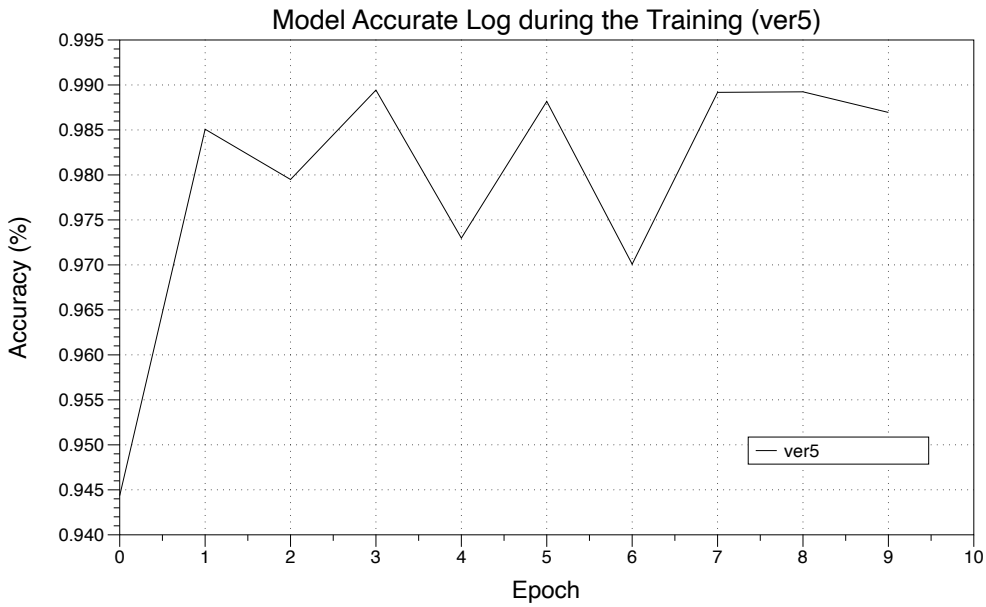

Figure 13. Accuracy Log of Model Version 5

Model Version 5 on the other hand, stabilized accuracy between 97% and 99% after 10 epochs of training, indicating that if the training is continued, it should at least attain similar performance to Model Version 2.

The accuracy, dice score, and average inference time of the final version of the model trained for 100 epochs are shown in the table below.

Model	Total Epoch Number	Final Accuracy (%)	Dice Score	Average Inference time (second per image)
ver1	100	95.56230307	0.941433191	0.029
ver2	100	99.28712249	0.985901713	0.095
ver3	100	98.98146987	0.980378211	0.141
ver4	100	99.31240082	0.986650705	0.129
ver5	10	98.69464636	0.969237983	0.538

Table 1. Accuracy, Dice score and Inference time result of the Models

Except for Model Version 1 and Model Version 5, the other three model versions had a Dice Score over 0.98 and an accuracy of about 99%. In terms of average inference time, Model Version 5 took the longest, taking nearly half a second to create the matching mask for the input. The structure of the Model Version 5, which has twice as many model parameters as the preceding versions, may be to blame for the lengthy inference time. Model Version 1 is the fastest, although it has the same parameters as Model Versions 2 and 4. The reason for this may be that Model Version 1 does not learn the parameters sufficiently. Therefore a large number of parameters within layers could not extract enough features to participate in the inference. The results might have been different if Model Version 1 could have performed a more extended training period.

The model's size, parameters, number of layers, and other information are listed in the table below.

Model	Total params	Trainable params	Forward/backward pass size (MB)	Params size (MB)	Estimated Total Size (MB)	Layers	ModelState Size (MB)
ver1	31,037,633	31,037,633	1272.03	118.4	1391.46	72	372.6
ver2	31,037,633	31,037,633	1272.03	118.4	1391.46	72	372.6
ver3	13,025,357	13,025,357	2945.22	49.69	2995.93	199	156.7
ver4	31,037,633	31,037,633	1266.84	118.4	1386.27	72	372.6
ver5	77,414,275	77,414,275	2826.71	295.31	3123.05	200	929.3

Table 2. Parameters Size of Models

Model Version 3 has the fewest trainable parameters, the smallest storage volume for parameters and the smallest archive of model states, but it requires the largest runtime volume, even more space than Model Version 5. However, both the Model Version 3 and the Model Version 5 contain about 200 layers in their models, thus this is a reasonable phenomena. Model Version 5's state collection is the largest, with 929.3 MB, and also contains the most parameters.

6. Limitation and Future work

There are several limits to this project. Although several computer vision models can be used to solve the medical image segmentation task, the focus on UNET in this project makes it impossible to compare the performance gap between different model frameworks. The author, on the other hand, lacks the time and resources to experiment with UNET variations such as VNET, UNET++, and Res-UNet. Furthermore, the research used just one dataset for training, making it hard to verify that the model will perform similarly in other medical images. It was unable to verify that all hyperparameters were constant due to the low performance of the machine utilized for training (e.g. batch size). This has an unquantifiable influence on the performance disparity between different models. Furthermore, owing to the authors' talents and knowledge, the majority of the module design and framework enhancements are based on solutions given in previous publications.

There are also several shortcomings in the way UNET is used. To begin with, the author lacks the capacity to architecturally optimize the neural network, thus these versions of the model will be computationally costly in certain areas. Second, while certain versions of the model perform well (acc>99), the author is unable to determine what variations in the versions have resulted in this performance. In subsequent research, the author is required to examine and justify these differences using the feature maps learnt by the model. Due to the model architecture and training dataset, which is typically insufficient in medical images segmentation, the models only accomplished two-class segmentation in the final output. Existing study findings have shown that multiple target locations of various classifications may be segmented. UNET will be able to detect and segment multiple target regions of various categories in the author's future research and model upgrades. Better feature extraction modules, more cutting-edge methods, and other factors will be required to attain this aim.

7. Conclusion

In conclusion, this project leaves behind results that meet the original objectives set. The implementation of UNET and various attempts to optimize its segmentation results were demonstrated in this project. Most of the versions were able to segment the target region successfully. The model can also be used for other image segmentations simply by replacing the data used for training. There are some regrets and incomplete parts of this project, which will hopefully be overcome in future research.

8. Reflective Learning

This project taught me a lot.

Initially, I attempted to run it from my computer, but it was not supported on my system, and there was no GPU support for training, so I went with Colab. I started by learning how to process image data and build a training set based on the images. I learnt how to use Colab to link data and code in the cloud, as well as how to install packages in the Colab environment. Although I had some theoretical knowledge of machine learning and computer vision, I had never studied how to create a model in a systematic way. Fortunately, there are a plenty of tutorials available on YouTube.

I didn't spend much time on the most fundamental model framework. On YouTube, I saw a pretty comprehensive tutorial on how to create a UNET model from the ground up. I went over the code in the video several times and utilized it to create a prototype. Despite the fact that the code was executable and the model was functional, I didn't completely get what was going on in the model at the time. So, line by line, I began to read and comprehend the logic in the code. From the code, I understood the basic construction of the PyTorch model and how to design code to measure the model's performance and to visualize and save the model predictions. However, sticking with the most basic version of the model was insufficient to address the project's requirements, also the initial version model did not perform as well as it should have after training. Although the accuracy display increased with time, the anticipated masks contained a lot of random noise. The theory was that the model had reached a local optimum, beyond which no further error reduction could be achieved. To improve performance further, the model used a method of giving values to specific pixels at random. This might be because the model was unable to extract and learn sufficient features from the data.

So I had two choices: increase the quantity of data used for training or use a better feature extraction module to enhance my model. I first chose to use data augmentation to try to enhance the quantity of data used for training. I reviewed the original UNET article and saw that stretching deformation was appropriate for the model's performance. I subsequently learnt how to use 'albumentation' to handle the data, which allowed me to tenfold the dataset size. The model performed better after being given a larger dataset, increasing its accuracy from 95% to 99 percent and resolving the noise issue. The findings of the segmentation were quite accurate.

The next phase of the research focused on using PyTorch and referring to relevant publications to better optimize the UNET. Firstly, I considered that the number of channels would increase during the down-sampling of the UNET and whether these features were necessary. According to one study, neighboring features in feature extraction are often quite similar. It would enhance the model's training and inference time if I could minimize feature redundancy properly. Then I looked for sample code and attempted to include the feature extraction module described in this article into an existing model. However, after installing it, I discovered that the module requires the specified number of input channels. I tried adding channels to the inputs in the dataset part's code at first, but the function still reported problems. After that, I tried directly adding a 1x1 convolution module to the model's initial input to increase the number of channels in the input to match the module's functioning criteria, which succeeded. When I tested the module in Colab, I discovered that the memory given by Colab was insufficient to run and train the model with the batch size set to the

original 16, thus for this version of the model, I reduced the batch size to 15.

After completing this version of the model, I decided to try to reproduce some of the modules myself that were worthy of reference and use. After reading the paper MC-Net, I felt I had a clear target. The functions of the MFE and CIE modules proposed in the article are potentially optimizing for the model. Their design is also not overly complicated. So, based on the architecture in the MC-Net article, I started building the MFE, CIE, and RA modules as I understood them. I started by replicating the RA module, because the skip-connection modification is about the model's structure and would be more interesting. The RA module was difficult since it was a variation on the skip-connection and would put my knowledge of the UNET model's architecture and code to the test. Because it does not need the addition of training parameters, the RA module was easily implemented after I grasped the UNET concept and the code.

Finally, I reproduced the MFE and CIE modules and embedded the model with DoubleConv, modelled on the MC-Net design. In the process, I was able to build relatively simple modules using PyTorch.

Throughout the process, I've come across a few minor problems that have been tough to fix and have taken a lot of time. The above-mentioned SPConv section is an example of when the channel input channel does not match. When validating the Dice indicator, I also encountered an issue; the value would initially surpass 1 when the software displayed the Dice index, but this was not allowed according to the Dice formula. As a result, I began checking the code in sections rather than in chunks. Finally, when I double-checked the Dice computation part, I discovered that the mask's pixel points were not in the 0 to 1 range as I had specified, but considerably exceeded it. As a result, I inserted the binarization code to the calculation area and was able to resolve the issue.

In working on this project, I discovered that many code-related errors are often hidden but end up looking like simple errors. However, it took a long time for me to realize what was wrong until I was able to pinpoint the source. So, as the project progressed, I gradually developed the practice of modularizing the functionality I wanted to accomplish and testing each module separately, only putting them together provided none of the needed components were defective. This design pattern has helped me decrease the number of errors I make when creating code and has given me a clear sense of what I'm attempting to accomplish.

This project has enabled me to make significant progress in both practical work and artificial intelligence modelling. I've improved my ability to handle training data, create reasonably simple models fast, and write the modules I need based on the articles as a result of this effort. I believe this is a milestone for my future study career. Furthermore, this project has aided me in improving my project management, time management, communication, and teamwork skills. I kept a daily journal of my work, articles read, and thoughts for the next stages throughout the project. As a result, I always have a clear picture of my project and a reference point for drafting my report. It also provided me a solid notion of the time range and room to alter my schedule because it was documented everyday. In the end, despite the fact that I did not have any technical issues that required assistance from my supervisor, my correspondence with him served as a fantastic source of inspiration when I hit a roadblock in my study. I couldn't have come up with these ideas on my own, and I realized once again that interacting with people to solve difficulties is essential.

Finally, I'd want to express my gratitude to this project for helping me to develop both technically and emotionally, make significant progress, and get a better understanding of my future research direction. I'll put what I've learned from this endeavor to good use in my future studies.

Acknowledgments

I'd want to express my gratitude to Professor Yukun Lai, my project supervisor, for his invaluable assistance.

Secondly, I would like to thank my parents for everything they have done for me.

Finally, I would like to thank my friends and classmates Chutong Zhong and Ce Guo for their support and friendship during my Master's degree.

References

[1] N. Sharma and L. M. Aggarwal, 'Automated medical image segmentation techniques', *J Med Phys*, vol. 35, no. 1, pp. 3–14, 2010, doi: 10.4103/0971-6203.58777.

[2] J. Long, E. Shelhamer, and T. Darrell, 'Fully Convolutional Networks for Semantic Segmentation', *arXiv:1411.4038 [cs]*, Mar. 2015, Accessed: Oct. 07, 2021. [Online]. Available: http://arxiv.org/abs/1411.4038

[3] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, 'Semantic Image Segmentation with Deep Convolutional Nets and Fully Connected CRFs', *arXiv:1412.7062 [cs]*, Jun. 2016, Accessed: Oct. 07, 2021. [Online]. Available: http://arxiv.org/abs/1412.7062

[4] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, 'DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs', *arXiv:1606.00915 [cs]*, May 2017, Accessed: Oct. 07, 2021. [Online]. Available: http://arxiv.org/abs/1606.00915

[5] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, 'Rethinking Atrous Convolution for Semantic Image Segmentation', *arXiv:1706.05587 [cs]*, Dec. 2017, Accessed: Oct. 07, 2021. [Online]. Available: http://arxiv.org/abs/1706.05587

[6] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, 'Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation', *arXiv:1802.02611 [cs]*, Aug. 2018, Accessed: Oct. 07, 2021. [Online]. Available: http://arxiv.org/abs/1802.02611

[7] V. Badrinarayanan, A. Kendall, and R. Cipolla, 'SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation', *arXiv:1511.00561 [cs]*, Oct. 2016, Accessed: Oct. 07, 2021. [Online]. Available: http://arxiv.org/abs/1511.00561

[8] O. Ronneberger, P. Fischer, and T. Brox, 'U-Net: Convolutional Networks for Biomedical Image Segmentation', *arXiv:1505.04597 [cs]*, May 2015, Accessed: Jul. 28, 2021. [Online]. Available: http://arxiv.org/abs/1505.04597

[9] F. Milletari, N. Navab, and S.-A. Ahmadi, 'V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation', *arXiv:1606.04797 [cs]*, Jun. 2016, Accessed: Oct. 07, 2021. [Online]. Available: http://arxiv.org/abs/1606.04797

[10] X. Xia and B. Kulis, 'W-Net: A Deep Model for Fully Unsupervised Image Segmentation', *arXiv:1711.08506 [cs]*, Nov. 2017, Accessed: Oct. 07, 2021. [Online]. Available: http://arxiv.org/abs/1711.08506

[11] F. I. Diakogiannis, F. Waldner, P. Caccetta, and C. Wu, 'ResUNet-a: a deep learning framework for semantic segmentation of remotely sensed data', *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 162, pp. 94–114, Apr. 2020, doi: 10.1016/j.isprsjprs.2020.01.013.

[12] G. Stockman and L. G. Shapiro, *Computer Vision*, 1st ed. USA: Prentice Hall PTR, 2001.

[13] M. V. Valueva, N. N. Nagornov, P. A. Lyakhov, G. V. Valuev, and N. I. Chervyakov, 'Application of the residue number system to reduce hardware costs of the convolutional neural network implementation', *Mathematics and Computers in Simulation*, vol. 177, pp. 232–243, Nov. 2020, doi: 10.1016/j.matcom.2020.04.031.

[14] K. P. Murphy and F. Bach, *Machine Learning: A Probabilistic Perspective*, Illustrated edition. Cambridge, MA: MIT Press, 2012.

[15] 'BCEWithLogitsLoss — PyTorch 1.9.1 documentation'. https://pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html (accessed Oct. 07, 2021).

[16] J. Han and C. Moraga, ‘The influence of the sigmoid function parameters on the speed of backpropagation learning’, in *From Natural to Artificial Neural Computation*, Berlin, Heidelberg, 1995, pp. 195–201. doi: 10.1007/3-540-59497-3_175.
[17] D. P. Kingma and J. Ba, ‘Adam: A Method for Stochastic Optimization’, *arXiv:1412.6980 [cs]*, Jan. 2017, Accessed: Oct. 07, 2021. [Online]. Available: http://arxiv.org/abs/1412.6980
[18] Q. Zhang *et al.*, ‘Split to Be Slim: An Overlooked Redundancy in Vanilla Convolution’, *arXiv:2006.12085 [cs]*, Jun. 2020, Accessed: Aug. 22, 2021. [Online]. Available: http://arxiv.org/abs/2006.12085
[19] H. Xia, M. Ma, H. Li, and S. Song, ‘MC-Net: multi-scale context-attention network for medical CT image segmentation’, *Appl Intell*, May 2021, doi: 10.1007/s10489-021-02506-z.
[20] ‘PyTorch’. https://www.pytorch.org (accessed Oct. 07, 2021).
[21] ‘Google Colaboratory’. https://colab.research.google.com/?utm_source=scs-index (accessed Oct. 07, 2021).
[22] A. Carass *et al.*, ‘Evaluating White Matter Lesion Segmentations with Refined Sørensen-Dice Analysis’, *Sci Rep*, vol. 10, p. 8242, May 2020, doi: 10.1038/s41598-020-64803-w.