PRACTICAL FEASIBILITY OF DEEP-LEARNING BASED MEDICAL IMAGE SEGMENTATION FOR THE AORTA

PRACTICAL FEASIBILITY OF DEEP-LEARNING BASED MEDICAL IMAGE SEGMENTATION FOR AORTA

By
XUANMING YAN

A Report
Submitted to the School of Graduate Studies
in Partial Fulfillment of the Requirements
for the degree
Master of Engineering in Computing and Software

McMaster University

Abstract

Medical imaging segmentation can help health workers analyze and examine abnormal changes in an organ or in tissue due to injury or disease, without invading the patient's body. The application of deep learning makes this process more efficient. There are many studies have been presented in this field, especially in the segmentation of liver, heart, kidney and lung with very successful results. However, there are still many challenges of using deep learning on other organs' segmentation, such as the aorta. This report presents the feasibility of using deep learning on aortic segmentation. Firstly, we explored different methods for generating accurate ground-truth segmentation as training data, concluded the average time cost for each scan is around 3 hours. Secondly, we compared to the segmentation result of kidney using a large training set, analyzed the practical constraints which prevent us for getting an equivalently good result, including the training time (on the order of hundreds of hours), and the cost for accessing the computing hardware is also on the order of thousands of US dollars. These practical constraints are usually not disclosed by those successful studies, but are crucial for those researchers who would like to perform the deep learning approach to the medical image segmentation. This report summarizes these challenges and provides lessons learned for future practitioners.

Acknowledgments

Thanks to my supervisor, **Smith Spencer**, for all his help during my two years of post-graduate study. The two years of study was also an unprecedented experience for me, because this time it coincided with the outbreak of the COVID-19, and I completed my studies entirely online. Starting from the fall of 2020, Dr. Smith had hosted meetings with me every week to discuss the progress of my various studies. He not only cared about the progress of this project, but also very minded about the progress of my other courses. He gave me useful advice in every meeting, encouraged me to challenge difficulties, and helped me seek outside help when resources were tight. This project would not have been possible without his assistance to me and his dedicated involvement in every step of the process.

I would also like to thank **Zahra Motamed** and her Ph.D student **Seyedvahid Khodaei** who introduced me to the basic concepts and clinical applications of medical image segmentation, allowing me, a student who has never been exposed to the field of medical imaging, been able to understand the concepts and clinical applications of medical imaging and its its cutting-edge research.

I would also like to show gratitude to **Derek Lipiec**, the Computing Facilities Manager at University of McMaster, who helped me allocate the university's AI training computing resources so that I could complete this project more efficiently.

My sincere thank you to **Kailin Chu**, summer research student of Dr. Smith, her work was adapted for use in this project for data annotation.

Finally, thank all the faculty and staff for their help and support. A big thank you to my parents for providing me financial support along the way, and also to my girlfriend **Sijia Ding**, for her endless support and companionship when I encountered illness during my graduate study.

Contents

Chapter 1

Introduction

Medical Imaging (MI) plays a crucial role in medical diagnosis and case analysis. As a noninvasive procedure, it does not involve the puncturing of the skin or an incision, or the introduction into the body of foreign objects or materials. This non-invasive diagnosis method can effectively check the pathological formations in the patient body without wounding the patient [3].

Segmentation is one of the most important and popular tasks in MI analysis. It is a process of extracting the Regions Of Interest (ROIs) from 3D image data, such as from Magnetic Resonance Imaging (MRI) or Computed Tomography (CT) scans [4]. As a hot field of research, it not only attracts researchers and scientists working in the medical and health field, but also those in the field of computer vision, and computer graphics because of their interest in image processing and modeling. The countless efforts of these frontline researchers have made great contributions to the implementation of new medical image-processing algorithms. The segmentation of organs from medical images gives health workers the possibility of using a quantitative diagnostic result to analyze the clinical parameters, such as volume and shape, providing great improvement in diagnostic efficiency and accuracy.

X-ray, Computed Tomography (CT), Magnetic Resonance Imaging (MRI) and ultrasound are four major classes of MI equipments that help clinicians diagnose diseases. With the rapid development of deep learning, medical image processing based on deep neural networks has become one of the most focused research topics. Prior arts have already studied the deep learning neural network based implementations of liver and liver-tumor segmentation [5] [6], brain and brain-tumor segmentation [7] [8], lung segmentation [9], heart segmentation [10], etc. However, segmentation of many other organs using deep learning are still unaddressed. This is due to the difficulty of feature representation, in addition, the raw data obtained from these MI equipments generally has the issue of blur, low contrast, inconsistency, etc.

1.1 Objective

In this study, we will explore the feasibility of using deep learning algorithms for aortic medical image segmentation and reproduce some existing deep learning network based segmentation implementations on other open access datasets. At the same time, the report will serve as a software design of a deep learning neural network architecture to generate a trained model using pre-labeled ground truth aortic segmentation dataset. Ideally, we would like the model to perform a segmentation task on new chest CT data, generating segmentation masks for each image slice and ultimately building its 3D geometry. We will analyze the challenges and difficulties encountered in this process, to summarize the technical and practical difficulties in the clinical application of deep learning algorithms in the field of MI, and what needs to be done in the future.

1.2 Background

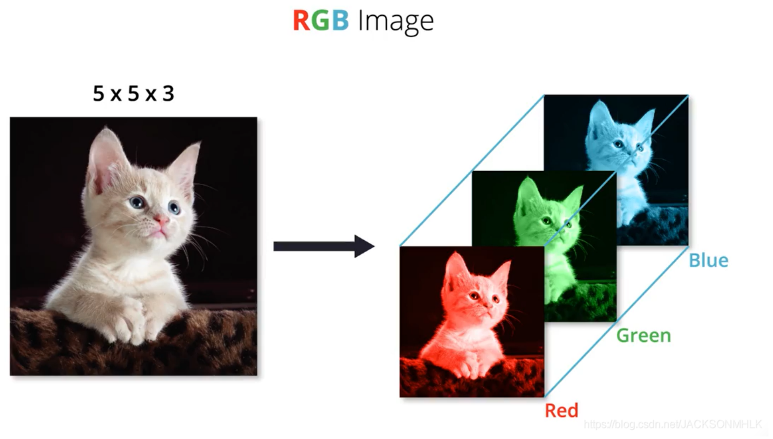


Figure 1.1: Example of 2D RGB image

Typically, when we talk about the image data, we are referring to the RGB data obtained from visible light, like photos captured by cameras. These are two-dimensional images are represented by a 3D array: **[height, width, channel]**, in which **channel** represents the dimension the color information as an example show in Figure 1.1. We can also add depth information to these images, we call them RGB-D images (2.5-dimensional images), RGB-D images are not true 3D, because the way they encode is that each pixel has a distance information between the image plane and the corresponding object in the RGB image. 3D

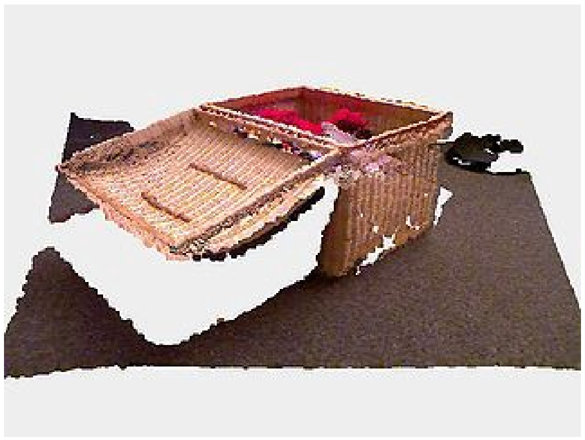


Figure 1.2: Example of 3D image stored as point cloud data

images are obtained from Lidar or structured light, and they are stored as point cloud data or voxel (Volume Pixel) data as shown in Figure 1.2. In image processing, they are all referred to as natural images because they are all generated from visible light. Medical images also have 2D or 3D data, normally generated by the computation of the reflections of X-ray, ultrasound or magnetic resonance. The objects of these images are basically derived from human or animal tissues. In this document, we focus on 3D medical images. Most MI data are in the form of a sequence of image files, for example, computed tomography (CT) scan, are sequences of files storing gray-level data values called CT numbers, or the Hounsfield scale, which represents the relative radio-density the detectors received.

$$HU = 1000 \times \frac{\mu - \mu_{water}}{\mu_{water} - \mu_{air}}$$

where μ_{water} and μ_{air} are respectively the linear attenuation coefficients of water and air.

In general, medical images belong to the sub-category of images, so there is no problem in applying the method of image segmentation to them. But before that, it is necessary to understand the difference between image segmentation and image semantic segmentation. The more commonly heard term is image semantic segmentation, that is, assigning a specified label to each pixel in the image (pixel-level category prediction problem). MI segmentation belong to the category of image semantic segmentation. In summary, MI segmentation is the process of classifying pixels into groups that correspond to the same tissue type with special semantic information (such as tumors (Figure 1.3), organs (Figure 1.4), blood vessels (Figure 1.5, etc.)), but the number of categories of medical image segmentation is generally not as large as natural image semantic segmentation. For example, VOC2012 [11] contains 20 categories and a background category, but many medical image segmentation problems are binary classification problems.

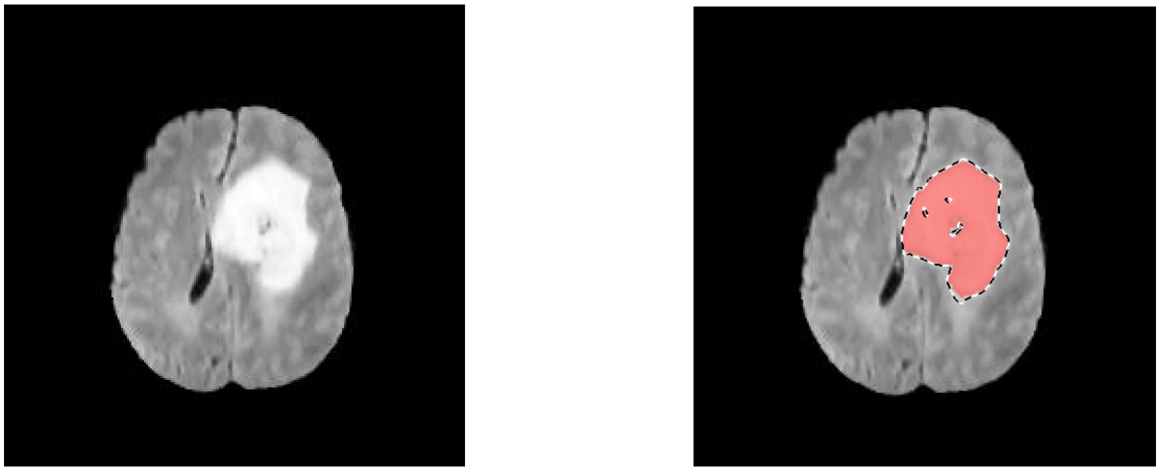


Figure 1.3: Brain tumor segmentation (Generated using method in Chapter 3)

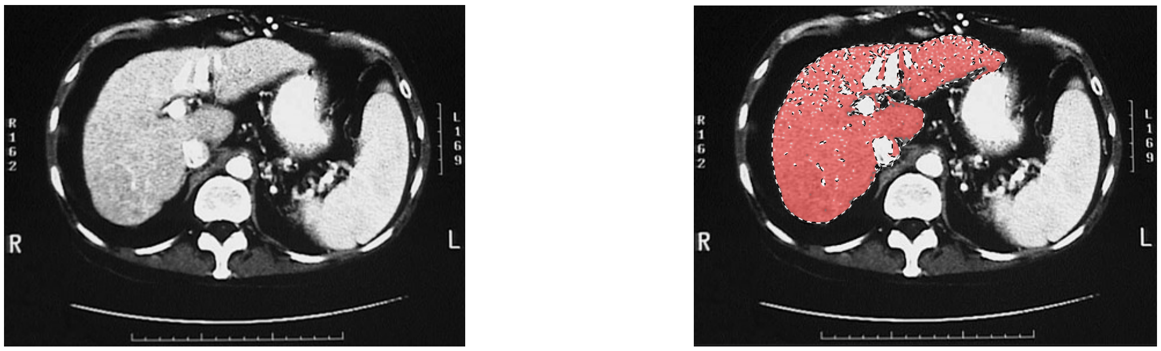


Figure 1.4: Liver segmentation (Generated using method in Chapter 3)

Before implementing this software, there are some challenges for getting and preprocessing the training data:

- We have to pre-process the aorta training data. The ground truth labelling MI data for the aorta is hard to obtain. We could not find an open source library.
- In addition, medical images (2D and 3D) generally have a high resolution, and current GPUs cannot directly and efficiently process the entire image. Therefore, it is generally necessary to crop the image into small images for processing, which in turn restricts the model from correctly capturing spatial information and spatial relations.
- Different types of medical equipment will generate unique and difficult-to-detect noise patterns (deviations), which will reduce the accuracy of model inference and

make it difficult to apply the model to different types of equipment.

- Another potential difficulty in the field of medical image segmentation is the lack of a large amount of accurately labeled data, which encourages researchers to explore semi-supervised and unsupervised models.

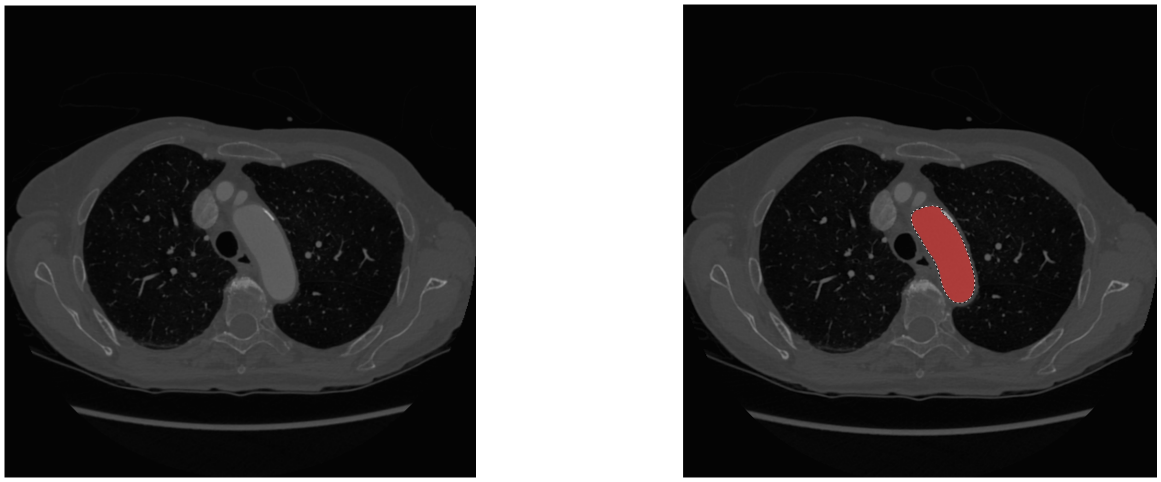


Figure 1.5: Segmentation of the arch of aorta from one slice of CT scan (Generated using method in Chapter 3)

1.3 Problem Statement

According to the World Health Organization, cardiovascular disease is a leading cause of mortality worldwide, killing an estimated 17.9 million people each year [12]. Cardiovascular disease refers to a group of heart and vascular disorders that includes coronary heart disease, cerebrovascular disease, rheumatic heart disease, and other conditions. Among various cardiovascular diseases, aortic disease are a group of conditions affecting the aorta.

The aorta is the main blood vessel that supplies blood to the body as shown in Figure 1.6. It runs from the heart down through the chest and abdomen. As people grow older, the wall of the aorta can become weak. if the walls expand they can develop into Aortic aneurysm (AA) and dissection (AD). This type of disease doesn't usually have any symptoms, but both are life-threatening [13]. Large aneurysms are rare, but they can be very serious. Left untreated, the wall of the aorta can become very weak and could burst. Ac-

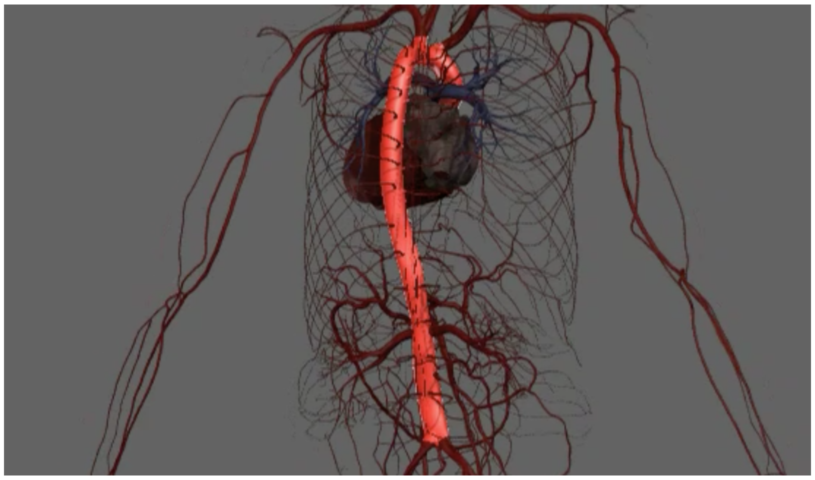


Figure 1.6: Aorta position in human body

cording to the CDC's Aortic Aneurysm Fact Sheet [14] [15], an estimated 200,000 persons in the United States are diagnosed with abdominal aortic aneurysms each year. Because many people have no symptoms, it is believed that more than one million people have an undetected abdominal aortic aneurysm. If this symptom can be detected during screening or testing for other reasons, the patient can receive treatment before the aneurysm becomes larger, This brings the needs of a quick and accurate way to do MI diagnosis for AA.

However, segmentation of an aortic aneurysm still remains difficult. Manual segmentation is a time-consuming method that is unsuitable for regular and reoccurring use. To solve this restriction, numerous automated aortic aneurysm segmentation strategies, such as edge detection-based methods, partial differential equation methods, and graph partitioning methods, have been developed. However, due to significant pixel resemblance to nearby tissue and a lack of color information in the medical picture, automated segmentation of aortic aneurysm is problematic [17], limiting the prior work from being applied to tough instances.

There are some studies using deep learning to detect AA, for instance DeepAAA from JT Lu, R Brooks et al [18], which uses a modified 3D U-Net combined with ellipse fitting that performs aorta segmentation and AAA detection. However, a full 3D reconstruction of the aorta using deep learning still remains difficult. This is probably due to the lack of labeled data sets and the complexity of organs' structure inside the abdomen. Compare to other autonomous segmentation approaches, deep learning may seem simpler and more flexible, but it requires more labeled data during training, and once the trained model generates, how it actually works is opaque (so-called "black boxes"). This limits the widespread adoption of deep learning in clinical practice.

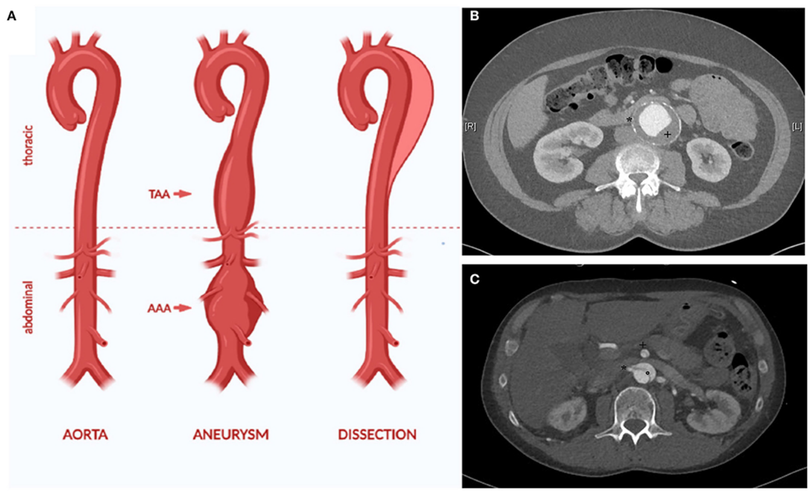


Figure 1.7: AA is a chronic dilatation of the whole vessel wall[16]

In the following chapters of the report, we will explore several deep learning algorithms, as well as the required preliminary work, and compare the final results, analyze the research resources required for similar projects, and its clinical feasibility. Chapter 2 provides some technical concepts that are used in this project. Chapter 3 introduces the preparation work for the training dataset. Chapter 4 shows the deep learning architecture and the training algorithms, as well as the prediction results generated from the trained model. Chapter 5 is going to analyze the effort and the clinical feasibility for this project, Chapter 6 contains conclusions.

Chapter 2

Background

This chapter will briefly introduce some technical concepts involved in this project, mainly image processing, deep learning algorithms, 3D geometric reconstruction, and models for determining the accuracy of results. Current progress on deep learning algorithms for image segmentation will also be summarize.

2.1 Image Annotation

In general, image segmentation groups similar areas or segments of an image under their respective class labels. It is a subfield of computer vision and digital image processing. However, for MI segmentation, it can also be treated of as pixel-wise classification, since for each pixel in the input image we label whether it is part of a segmentation mask. If the pixel is part of the segmentation mask, we label it 1, otherwise we label it 0. In other words, we want to input an image and then output a decision of a category for every pixel in that image.

Supervised learning uses labeled training data and test data, which has strict requirements on the accuracy of data labeling. This means that the quality of the training data determines the quality of the model generated by the machine learning algorithm. Usually, the training data used for supervised learning has very high labeling accuracy; however, to obtain the high accuracy of the training data, these labels are often done by purely manual work.

Most medical equipment is able to generate the DICOM format images. DICOM is standard image format for medical-imaging information and related data. It consists of the image slices and associated metadata of the patient info. However, for image annotation purpose, we are only interested in the image slices content. For research and commercial purpose, it is vital to remove sensitive patient information stored in the DICOM files to protect the patient's privacy.

2.2 Training Algorithm

The most commonly used artificial neural network for imagery processing is Convolutional Neural Network (CNN). UNet, which evolved from the traditional CNN, is used for biomedical image segmentation. This section will briefly introduce some concepts of these two networks.

2.2.1 CNN [1]

A CNN consists of one or more convolutional layers and a top fully connected layer (corresponding to a classical neural network), as well as associated weights and a pooling layer. This structure enables convolutional neural networks to exploit the two-dimensional structure of the input data. Compared to other deep learning structures, convolutional neural networks can give better results in image and speech recognition.

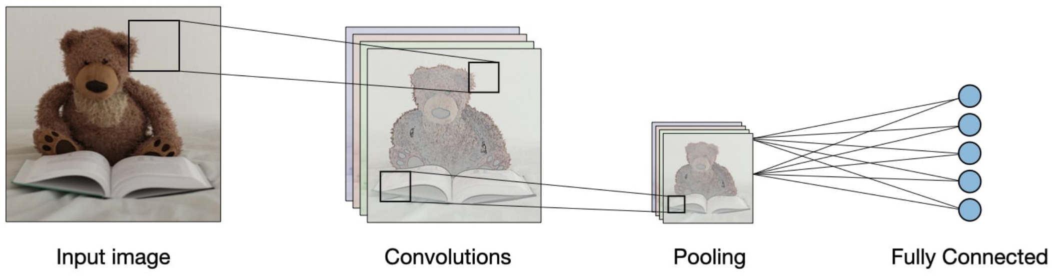


Figure 2.1: Visualization of CNN structure [19]

As shown in the Figure 2.1, there are three main components for a CNN, Convolutional Layer (CONV), Pooling Layer (POOL), and Fully-Connected Layer (FC). The first two layers are the main differences between CNN and a regular neural network.

- **CONV**: contains a set of filters (or kernels), parameters of which are to be learned. In this layer, an image with dimension $X \times Y \times Z$ will be processed by a filter with size F and stride S (Stride is a parameter of the neural network's filter that modifies the amount of movement over the image), then the image abstracted to a size O, called a feature map or an activation map.

- **POOL**: A pooling layer (POOL) is a down-sampling operation, usually applied after a CONV, with some spatial invariance. For example, max pooling and average pooling are a special kind of pooling that take maximum and average values, respectively.
- **FC**: Regular multi-layer perceptron neural network.

CNN transforms the original pixel valued image to a final classification model. The parameters in the CONV/FC layer will be trained with some optimization algorithms, for example:

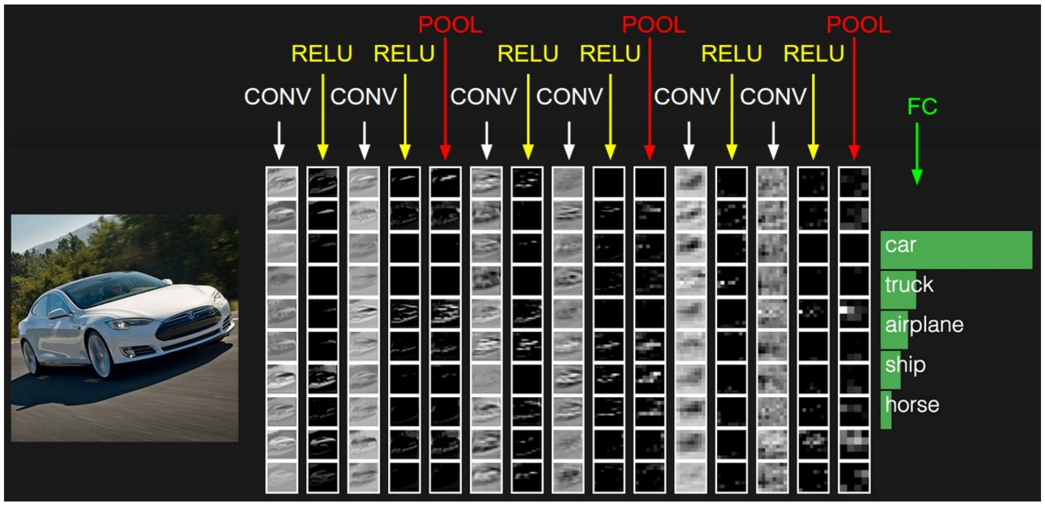


Figure 2.2: Example of CNN architecture. [19]

2.2.2 UNet [2]

As we saw from the previous CNN example, traditional CNN is mostly used for image classification, where the input is an image and output is one label. However, for medical images, it is required to do the pixel level classification in order to produce a image mask or a segmentation. Therefore, the input and output should have the same dimensions. Luckily, medical images only have one channel of color information, meaning that the third dimension of the input image size is 1 in most cases.

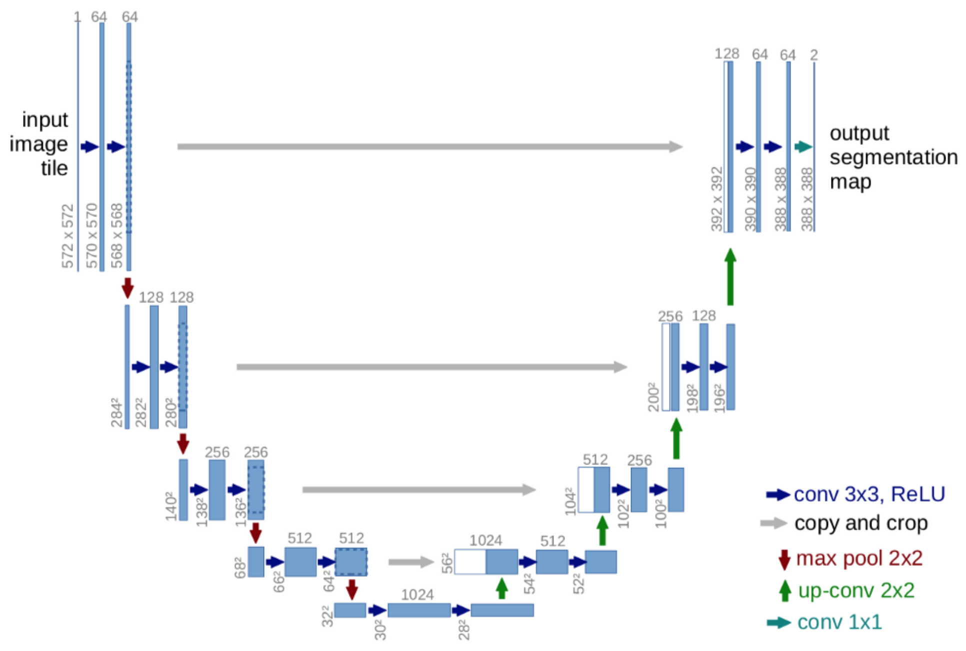


Figure 2.3: UNet architecture

```
[[255, 10], [234, 12]]  # each number is a pixel
```

Listing 2.1: input image size 2×2

If we set the threshold of a pixel value of above 100 to label 1, below 100 to 0, the output will have the same size, in this value:

```
[[1, 0], [1, 0]]  # Consider we are using binary classification
```

Listing 2.2: input image size 2×2

From Figure 2.3, we may have a clue why this CNN variant network is named as UNet, it has a ”U” shape. This symmetric shape consists of a contracting path (involves a general convolution process) and an expansive path (a series of up-convolutions and connections with high-resolution features from contracting paths). The network has a total of 23 convolutional layers. A python implementation can be found in Appendix A.

2.3 Metrics for Semantic Segmentation Model Accuracy

For semantic segmentation, the goal is to predict the mask within the ROI. But how do we know that our segmentation model is performing well? Obviously, we cannot rely on human visual inspection to judge the accuracy of the prediction results. Such metrics lack objectivity and are time-consuming and labor-intensive; therefore, automation of these metrics is needed. This project mainly uses two method for the model evaluation: pixel accuracy and Dice coefficient (F1 score).

2.3.1 Pixel Accuracy

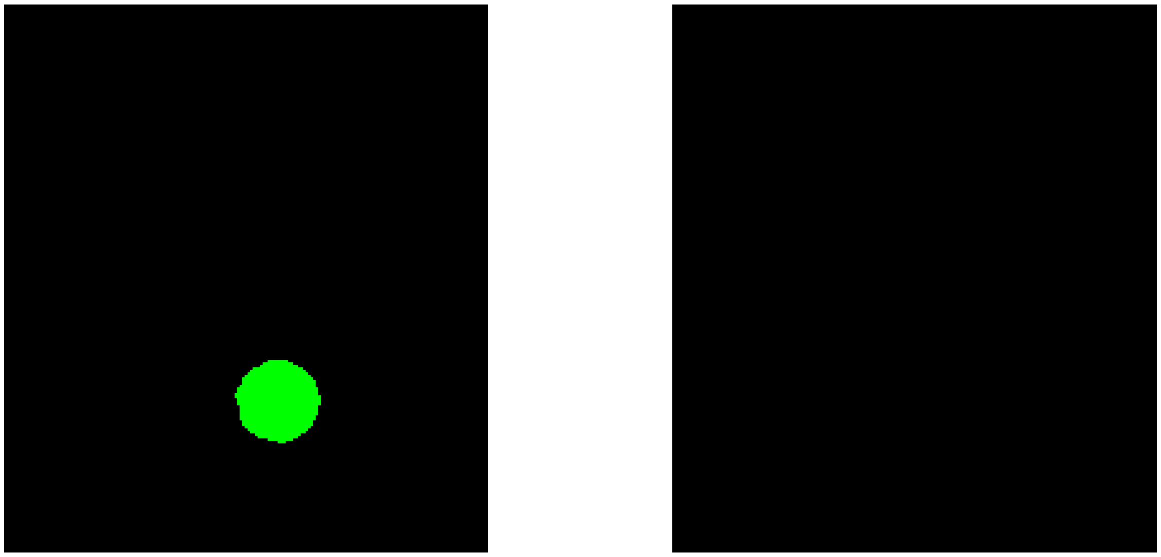


Figure 2.4: Pixel Accuracy of 90%

Pixel Accuracy is the percentage of pixels in an image that are correctly classified, as shown in this equation:

$$PA = \frac{\sum_{i=1}^{k} n_{ii}}{\sum_{i=1}^{k} t_i}$$

where n_{ii} is the total number of pixels both classified and labeled as class i. t_i is the total number of pixels labeled as class i.

This might be the most intuitive metrics for evaluating the model, however, there are some problems. As we mentioned earlier, for many medical images, the area ratio of the object of interest in these images is not high, in other words, most of the pixels are background, (unlabeled area). This is called class imbalance. For most MI segmentation, class imbalance is inevitable. For example in Figure 2.4, where the left figure is the ground truth.

The pixel accuracy for a completely black image is 90%. Which means that high pixel accuracy does not imply an excellent segmentation capabilities.

2.3.2 Dice Score

Consider an MI segmentation task, with class i as the object of interest, and class j as the background. The dice score is a measure of how similar the ground truth object and predicted object are. In other words, it is the size of overlap of the two segmentations divided by the total size of the two objects:

$$\text{Dice Score} = \frac{2 \times (\sum n_{ii} + \sum n_{jj})}{2 \times (\sum n_{ii} + \sum n_{jj}) + \sum n_{ij} + \sum n_{ji}}$$

In which n_{ii} and n_{jj} are the total number of pixels both classified and labeled as class i or j (True positive), n_{ij} is the total number of pixels should be class j but is classified as class i (False positive), n_{ji} is the total number of pixels should be class i but is classified as class j (False negative). Simplify above expression:

$$\text{Dice Score} = \frac{2 \times |\text{Mask i} \cap \text{Mask j}|}{|\text{Mask i}| + |\text{Mask j}|}$$

We can quickly calculate the dice score for two images shown in Figure 2.4 using below code:

```python
import cv2
import numpy as np

#load images
y_pred = cv2.imread('segment.png')
y_true = cv2.imread('black_030.png')

# Dice similarity function
def dice(pred, true, k = 1):
    intersection = np.sum(pred[true==k]) * 2.0
    dice = intersection / (np.sum(pred) + np.sum(true))
    return dice

dice_score = dice(y_pred, y_true, k = 1)
```

```python
print ("Dice Similarity: {}".format(dice_score))
```

Listing 2.3: Calculate dice score

and the output with *Dice Similarity: 0.0*, which means those two images are completely different.

2.4 Literature Review

With the rapid development and tremendous progress in deep learning algorithms, researchers and medical workers have created many clinical applications based on these algorithms. Medical workers use these applications on the MI diagnostic analysis for various different diseases, including diseases related to the brain, heart, liver and lung. The following literatures are some profound achievements made by researchers in recent years. It proves the feasibility of deep learning in the field of medical image segmentation.

- **Brain Disease**
  - Deep learning algorithm has great potential in stroke lesion segmentation. Chen et al.[20] use CNN for a fully automatic acute ischemic lesions segmentation with an average Dice score of 0.67. Liu et al [21] uses U-shaped network (Res-CNN) and achieve a Dice score of 0.742.
  - Deep learning also shown great promise in accurate detection of intracranial aneurysms. In a study from Stember et al. [22], they use CNN based U-net to train 250 MRA maximum intensity projection (MIP) images, the model identifies aneurysms in 85/86 (98.8% of) testing set cases, and also correctly predicts the area trend for the set of aneurysms. Ueda et al. [23] utilize ResNet for this task and achieve a sensitivity of 91%–93% improving aneurysm detection by 4.8%–13% for internal and external test datasets, respectively.
  - Recent studies also make good progress on the fully automatic brain tumor segmentation using CNN as shown in Havaeia et al. [24]. In addition, the work from Ranjbarzadeh et al. [25] present a promising segmentation result on the BRATS 2018 dataset, achieves a mean whole tumor, enhancing tumor, and tumor core dice scores of 0.9203, 0.9113 and 0.8726, respectively.
- **Heart Disease**
  - A number of CNN variant deep networks are presented for cardiac structures segmentation, in which AtriaNet developed by Xiong et al [26] achieves a Dice

score of 0.940 and 0.942 for the left atrial epicardium and endocardium, respectively..

Chapter 3

Datasets Preparation

Using deep learning method to get 3D models from 2D images already has promising implementations on camera based images [27]. Normally analysts uses convolutional neural networks (CNN) to generate features from these images, and then uses other deep neural network models like generative adversarial network (GAN) for 3D reconstruction. In this paper [28], they use an encoder and decoder method to map features from 2D images to their ground-truth 3D models. The encoder computes a set of features for the decoder to recover the 3D shape of the object. The decoder is responsible for transforming information of 2D feature maps into 3D volumes. Both are built by many layers of CNN.

Medical imaging doesn't have important information of camera parameters, light and shadows, which are used in traditional computer vision method. Moreover, there is a lack of ground-truth 3D models as training sets. Therefore, having high-accuracy training datasets is crucial for the deep learning projects working with medical images.

However, obtaining the ground truth training data from the raw data that is generated from the medical equipment is not an easy task. Converting DICOM to PNG format while preserving the image slices sequence order is a common approach for MI Annotation. The annotation should generate a one-to-one mapping between the PNG file and its segmentation mask file. For MI annotation, it is essentially a binary classification problem, the segmentation mask file only has two regions, the labeled area (annotated in white color, high brightness), and the unlabeled area (annotated in black color, zero brightness).

An unsupervised learning approach is an alternative. Since it doesn't require labeled data as training sets. In this case, there is no need for ground-truth 3D models. In fact, ground-truth 3D data is difficult and expensive to collect even for real-world objects as needed to train camera based images with supervised learning. There is an attempt to use unsupervised learning to get 3D structure from 2D camera based images [29], which uses conditional generative models with latent variables. This approach will not be explained in detail in this report. Due to the limitation of research resources, we have not had the opportunity to do research in this direction. This project mainly uses the method of supervised learning and leaves the unsupervised learning as future work.

We obtained 6 upper chest CT images for this project, all of them are stored in DICOM format. Since the pixel information in the DICOM file is stored as binary data, we need to convert the file formate to visualize the image slices and convert them to a 3D numpy array (2D image) or 4D numpy array (3D image) for image processing, ans described in the next section.

3.1 Convert DICOM slices to PNG

There are many ways to visualize DICOM files. One package that we chose is called *pydicom*. This is a pure Python package that is designed to manipulate data elements in DICOM files with python code.

First, load the DICOM files:

```
import pydicom
import numpy as np
import matplotlib.pyplot as plt
import sys
import glob
from PIL import Image as im

# load the DICOM files
files = []
path = '/Users/yanxuanm/Downloads/CT patient data sample/43681283/*.dcm'
print('glob: {}'.format(path))
for fname in glob.glob(path, recursive=False):
    files.append(pydicom.dcmread(fname))

print("file count: {}".format(len(files)))
```

Listing 3.1: Load the DICOM files

We are using a DICOM series with 1128 slices for our example.

Some CT scans may contain scout views slices, which are a survey of the region of interest used by the technician to select the area of dedicated image acquisition. These slices don't have location info and generally are not used for diagnostic purposes, so we can safely skip them.

```
# skip files with no SliceLocation (eg scout views)
```

```python
slices = []
skipcount = 0

for f in files:
    if hasattr(f, 'SliceLocation'):
        slices.append(f)
    else:
        skipcount = skipcount + 1

print("skipped, no SliceLocation: {}".format(skipcount))
```

Listing 3.2: Skip files with no slice Location

In our sample dataset, there are no scout view images. All the 1128 slices contains spatial information. The spatial information of each slice is stored in the metadata of the DICOM series, including the physical distances between each slice, and even the distance between two pixels. It is also essential that we keep the right order of these slices.

```python
# ensure they are in the correct order
slices = sorted(slices, key=lambda s: s.SliceLocation)

# pixel aspects, assuming all slices are the same
ps = slices[0].PixelSpacing
ss = slices[0].SliceThickness
ax_aspect = ps[1]/ps[0]
sag_aspect = ps[1]/ss
cor_aspect = ss/ps[0]
```

Listing 3.3: Sort the slices and record spatial info

Now we have the spatial information stored, we may proceed with converting the DICOM to a Numpy array. Once the data is stored in Numpy array, we can visualize the data using PNG format images.

```python
# create 3D array
img_shape = list(slices[0].pixel_array.shape)
img_shape.append(len(slices))
img3d = np.zeros(img_shape)

# fill 3D array with the images from the files
for i, s in enumerate(slices):
    img2d = s.pixel_array
    img3d[:, :, i] = img2d
    threshold = 1290 # Adjust as needed
    image_2d_scaled = (np.maximum(img2d, 64) / (np.amax(img2d) +
    threshold)) * 255.0

    data = im.fromarray(image_2d_scaled)
    if data.mode == 'F':
        data = data.convert('RGB')
    data.save('gfg_dummy_pic{0}.png'.format(i))
```

Listing 3.4: Store DICOM info to Numpy array

This will convert all 1128 slices of DICOM images to PNG Figure 3.1, and the indices are sorted with the corresponded location.

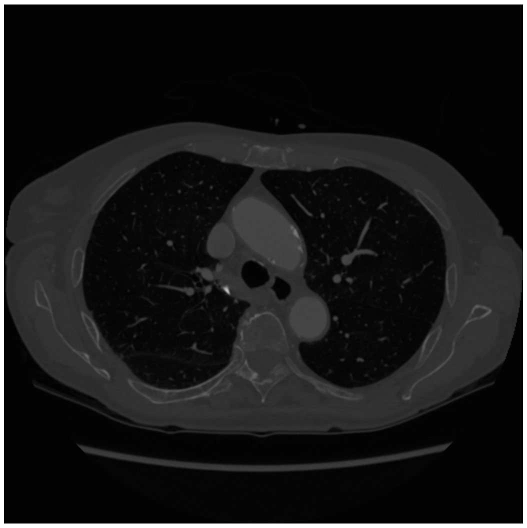


Figure 3.1: Sample PNG file

We can also visualize the Numpy arrays Figure 3.2:

```
# plot 3 orthogonal slices
a1 = plt.subplot(2, 2, 1)
plt.imshow(img3d[:, :, img_shape[2]//2])
a1.set_aspect(ax_aspect)

a2 = plt.subplot(2, 2, 2)
plt.imshow(img3d[:, img_shape[1]//2, :])
a2.set_aspect(sag_aspect)

a3 = plt.subplot(2, 2, 3)
plt.imshow(img3d[img_shape[0]//2, :, :].T)
a3.set_aspect(cor_aspect)

plt.show()
```

Listing 3.5: plot 3 orthogonal slices

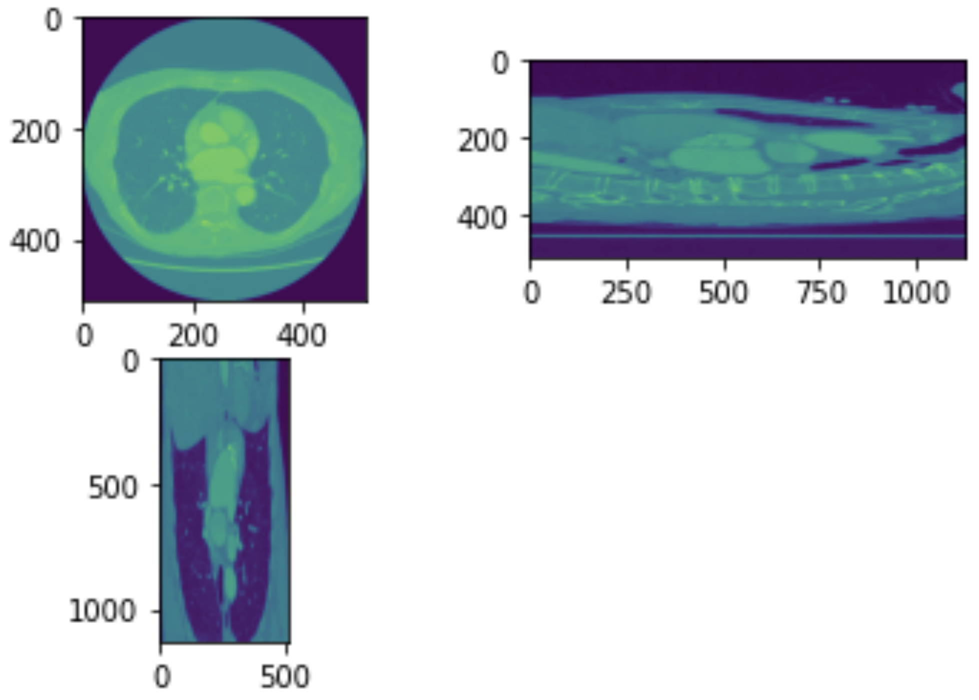


Figure 3.2: Plot 3 orthogonal slices

3.2 Training and Test Data Set

To use supervised learning algorithm for MI segmentation, we need to have access to sufficiently large, well-curated training data sets. However, preparation of such data is a tedious and time-consuming process. In fact, the lack of annotated data has been a major obstacle for this project for a very long time. So far, this project has only had access to only 6 upper chest CT MI data sets. This is not likely to be enough data for supervised learning. Ideally, we would like to have more than one hundred labeled datasets. **KiTS19** has a training set of 210 cross sectional CT images and a test set of 90 CT images [30] [31]. **LiTS17** has a training data set of 130 CT scans and the test data set 70 CT scans [32]. There are many objective reasons for the paucity of date. The reasons are all the common pain points in the general preparation of medical imaging data for development of deep learning models.

- **Data sharing and storage systems** are complex and expensive, and many hospitals do not have the spare funds to support such facilities. Therefore, it is very difficult for medical AI researchers to query a good amount of representative data.

- For the purposes of research and **reproducible results**, MI data used in the research projects are often publicly accessible. Having patient data publicly available through such repositories may constitute a breach of personal privacy if the data is not properly di-identified. There are existing software for automating the process of removing sensitive information from the MI data; however, this research [33] indicates that it is extremely difficult to eliminate all protected health information from DICOM images using automated software while retaining all useful information.

- **Data transfer**. To download data from the host server, and then upload to a shared drive can be time-consuming. This step heavily depend on the speed of ISP network service provide.

- **Data labelling is a challenge**. The quality of annotated data is an essential part to the supervised deep learning frameworks. It can determine if the training process is able to generate a good image classification model. This is the hardest and most time-consuming step for the data preparation. It often needs to be done manually to ensure the accuracy of the labeling. Annotators often follows certain guideline and operates in a best-guess manner [34]. However, considering that there are usually hundreds of DICOM slices in one CT scan, manual annotation can become unrealistic and unpractical.

3.3 Approaches for Data Labelling

This project explored several approaches to obtain high-quality annotated data. Including the use of existing MI segmentation software, for example ITK-snap, ParaView, 3D slicer, etc. And the use of script programs based on Simple-ITK library. Our goal is to generate PNG files that are annotation masks for each DICOM slice. Specifically, given a chest radiographic image (Figure 3.3 left), the idea is to predict a mask that shows the position of the aorta in an image (Figure 3.3 right).

3.3.1 Using ITK-snap

ITK-snap is an interactive software that provides the functionality for automatic and manual segmentation. This software can read the entire MI scan DICOM series, and show the images in an interactive 3D view, as show in Figure 3.4. Using ITK-snap to do manual segmentation requires assigning labels to each voxel in the structure. It has a tool help user to draw polygons on top of the axial, sagittal and coronal slice windows and then paint the closed polygons slice by slice. We can see that manual segmentation is time-consuming

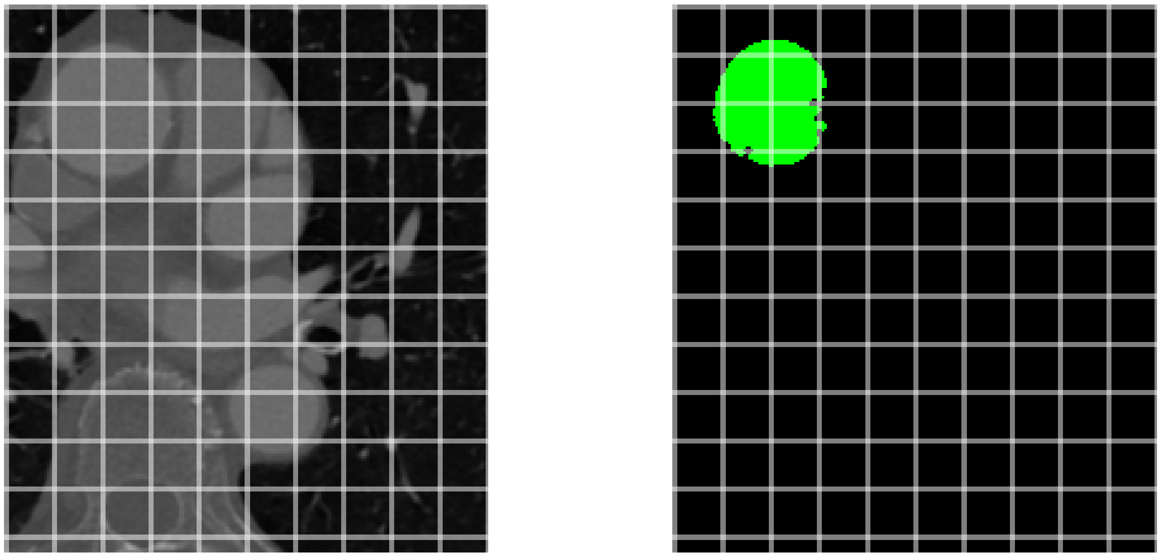


Figure 3.3: Sample Image and Upper Aorta Segmentation Mask

and not efficient, the user needs to draw polygons on every slice, which means large amount of repetitive work.

Luckily, ITK-snap also has the ability to do automatic segmentation, with very little human interaction. By manually adjust the image upper and lower threshold values, we can isolate the region we are interested in. We can then manually place bubbles that are then expanded to fill up the entire structure.

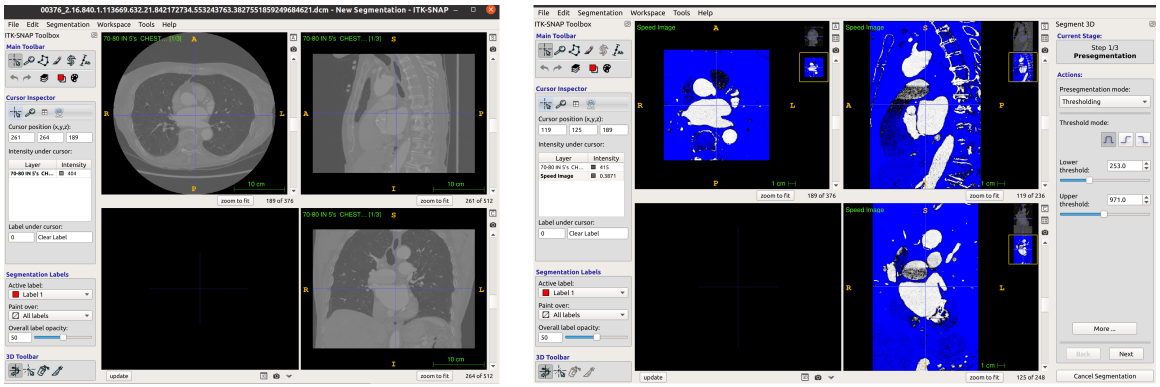
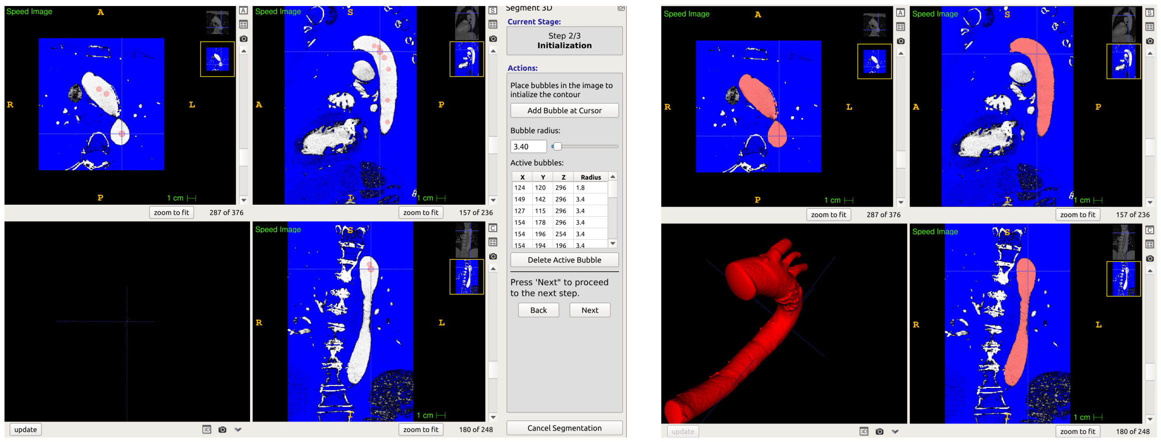


Figure 3.4: ITK-snap 3D view main window and threshold

However, using ITK cannot generate the PNG masks for each slice. It generates the



Segment 3D

Current Stage:

Step 2/3
Initialization

Actions:

Place bubbles in the image to intialize the contour

Add Bubble at Cursor

Bubble radius:

3.40

Active bubbles:

X	Y	Z	Radius
124	120	296	1.8
149	142	296	3.4
127	115	296	3.4
154	178	296	3.4
154	196	254	3.4
154	194	196	3.4

Delete Active Bubble

Press 'Next" to proceed to the next step.

Back Next

Figure 3.5: ITK-snap bubbles and 3D reconstruction

3D geometric structure of the segmentation. However, we can use scripts to map this 3D structure to each 2D slice. We have 6 CT DICOM series image for this project, and we use ITK-snap to generate 3D structures for all of them as shown in Figure 3.6. These 3D structures can be stored in NIFTI format with shape [**num-slices, height, width**], slice thicknesses range from 1mm to 5mm.

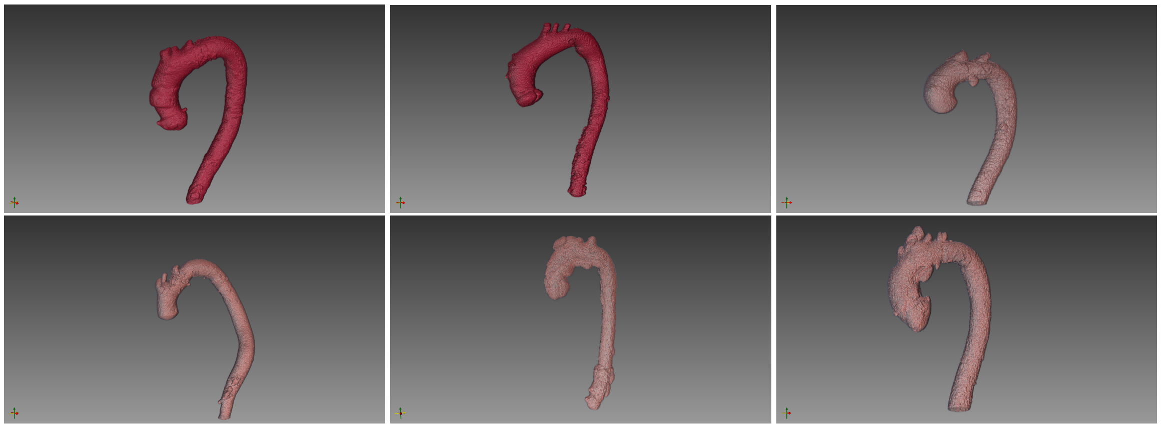


Figure 3.6: 3D Structure of Aorta for 6 CTs

For example, we process the case_00000 dataset, and map the 3D ground truth structure to 2D masks. First, we load the files:

```
BASE_IMG_PATH=os.path.join('','/content/drive/MyDrive/data_set/
    case_00000')
glob(os.path.join(BASE_IMG_PATH, '*'))
```

```
all_images=glob(os.path.join(BASE_IMG_PATH,'imaging*'))
all_masks = [x.replace('imaging', 'segmentation') for x in all_images]
print(len(all_images),' matching files found:',all_images[0], all_masks
    [0])

# Should print message : 1  matching files found: /content/drive/MyDrive
    /data_set/case_00000/imaging_00000.nii.gz /content/drive/MyDrive/
    data_set/case_00000/segmentation_00000.nii.gz
```

Listing 3.6: Load NIFTI file

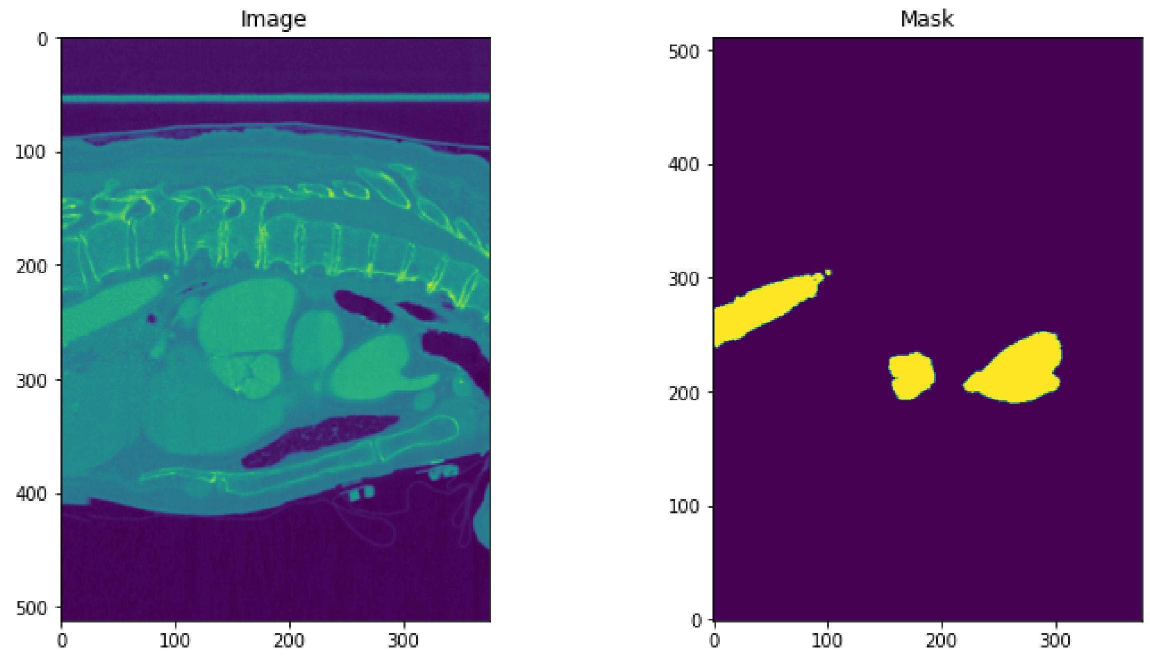


Figure 3.7: Mask mapping

Then we can visualize the corresponding raw image and mask image using this script:

```
%matplotlib inline
import nibabel as nib

test_image=nib.load(all_images[0]).get_data()
test_mask=nib.load(all_masks[0]).get_data()
fig, (ax1, ax2) = plt.subplots(1,2, figsize = (12, 6))
ax1.imshow(test_image[test_image.shape[1]//2].squeeze(axis=2))
ax1.set_title('Image')
ax2.imshow(test_mask[test_image.shape[1]//2], origin='lower')
```

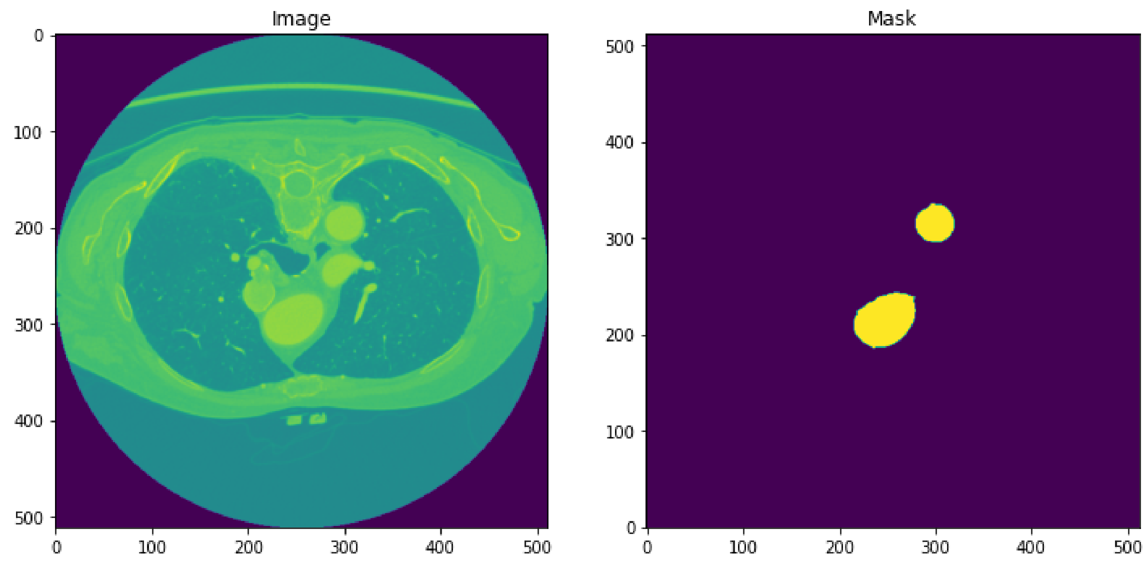


Figure 3.8: Another projection view

```
ax2.set_title('Mask')
```

Listing 3.7: 2D mask mapping

We can obtain the one to one mapping of the raw data and mask image as shown in Figure 3.7.

It's also very easy to change the projection view by swapping the axes as shown in Figure 3.8.

```
test_image=nib.load(all_images[0]).get_data()
test_image=np.swapaxes(test_image,0,2)
test_mask=nib.load(all_masks[0]).get_data()
test_mask=np.swapaxes(test_mask,0,2)
fig, (ax1, ax2) = plt.subplots(1,2, figsize = (12, 6))
ax1.imshow(test_image[test_image.shape[1]//2+10].squeeze(axis=2))
ax1.set_title('Image')
ax2.imshow(test_mask[test_image.shape[1]//2+10], origin='lower')
ax2.set_title('Mask')
```

Listing 3.8: Change a view angle

We can easily generate more one to one mappings using this method:

```
test_image=nib.load(all_images[0]).get_data()
test_mask=nib.load(all_masks[0]).get_data()
fig, axs = plt.subplots(10,10, figsize = (50, 50))
```

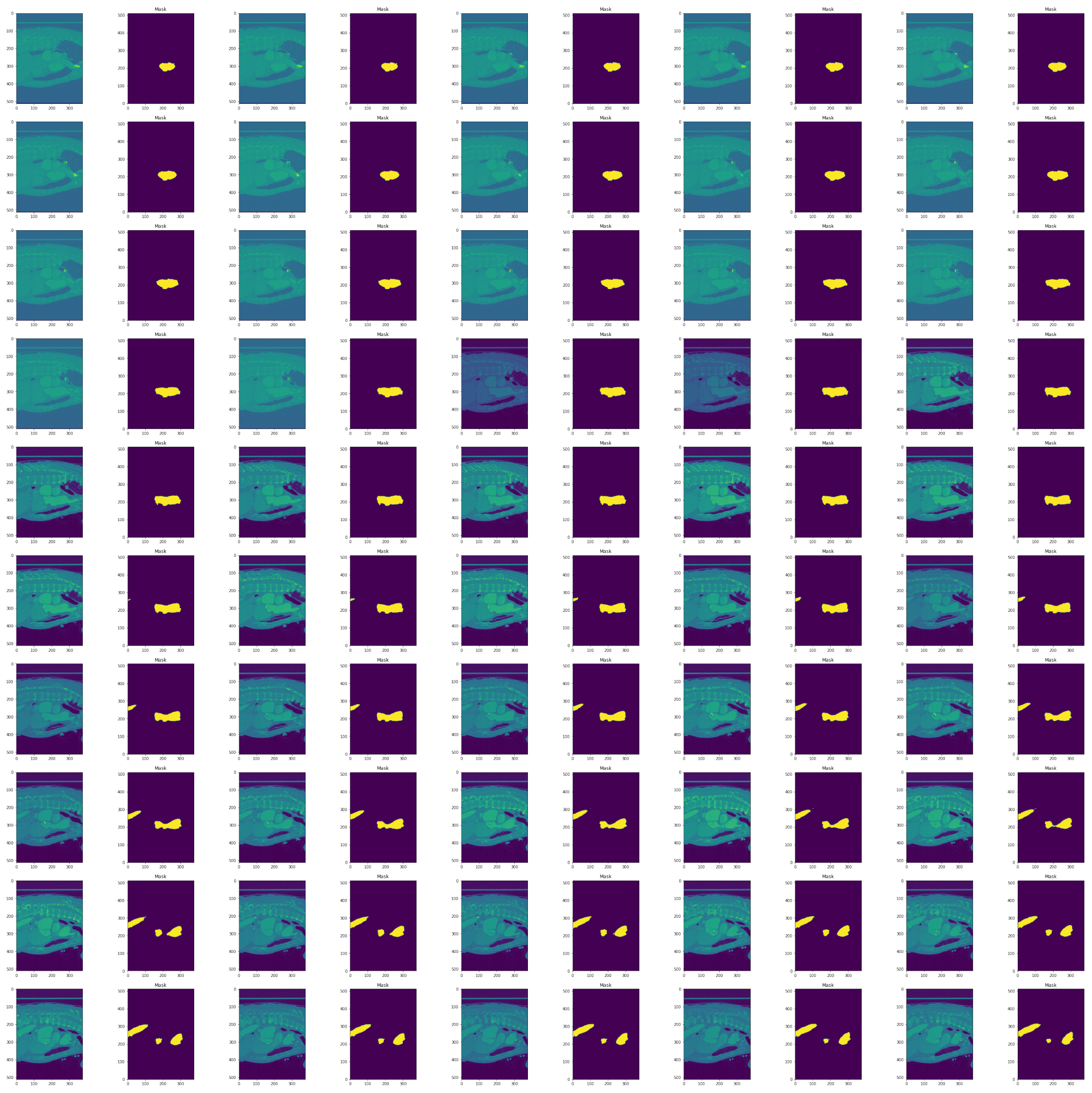


Figure 3.9: Mask mapping

```

for i, ax in zip(range(201), axs.ravel()):
    if i % 2 == 0:
        ax.imshow(test_image[test_image.shape[1]//2+(i//2)-40].squeeze(axis=2))
        ax1.set_title('Image')
    else:
        ax.imshow(test_mask[test_image.shape[1]//2+(i//2)-40], origin='lower')
        ax.set_title('Mask')
```

Listing 3.9: Generate more masks mapping

The results are shown in Figure 3.9

From visual inspection of the ground truth data generated by this method, we conclude it to be relatively reliable.

3.3.2 Using SimpleITK

The official introduction to SimpleITK states: SimpleITK is an image analysis toolkit with many components supporting general filtering operations, image segmentation and registration [35]. The implementations for this method can be found in Appendix C. Figure 3.10 are the mask generated by this method.

Using SimpleITK requires us to carefully place some seeds within the segmented area. It will use this seed as a growth point for the highlighted threshold region. Code implementations can be found in Appendix B. After the segmentation, we can map each slices of the results to a black background PNG file.

```
for i in trange(curve_seed[2],cropped_images[ct_number].GetDepth()):

    # perform segmentation on slice i
    seg, total_coord, centre, l, u, seeds = circle_filter_arch(i,centre_previous,seeds_previous,"cropped")

    # determine whether the size of this slice "qualifies" it to be accurate
    # if((total_coord > 1/factor_size * original_size) and (total_coord < factor_size * original_size) and (total_coord < 2*previous_size)):
    counter = 0
    new_image_cropped[:,:,i] = (seg>0) | new_image_cropped[:,:,i]
    if (i%5==0):
        print(cropped_images[ct_number].GetDepth()-i)
        myshow(sitk.LabelOverlay(black_images[:,:,i], seg>0))

```

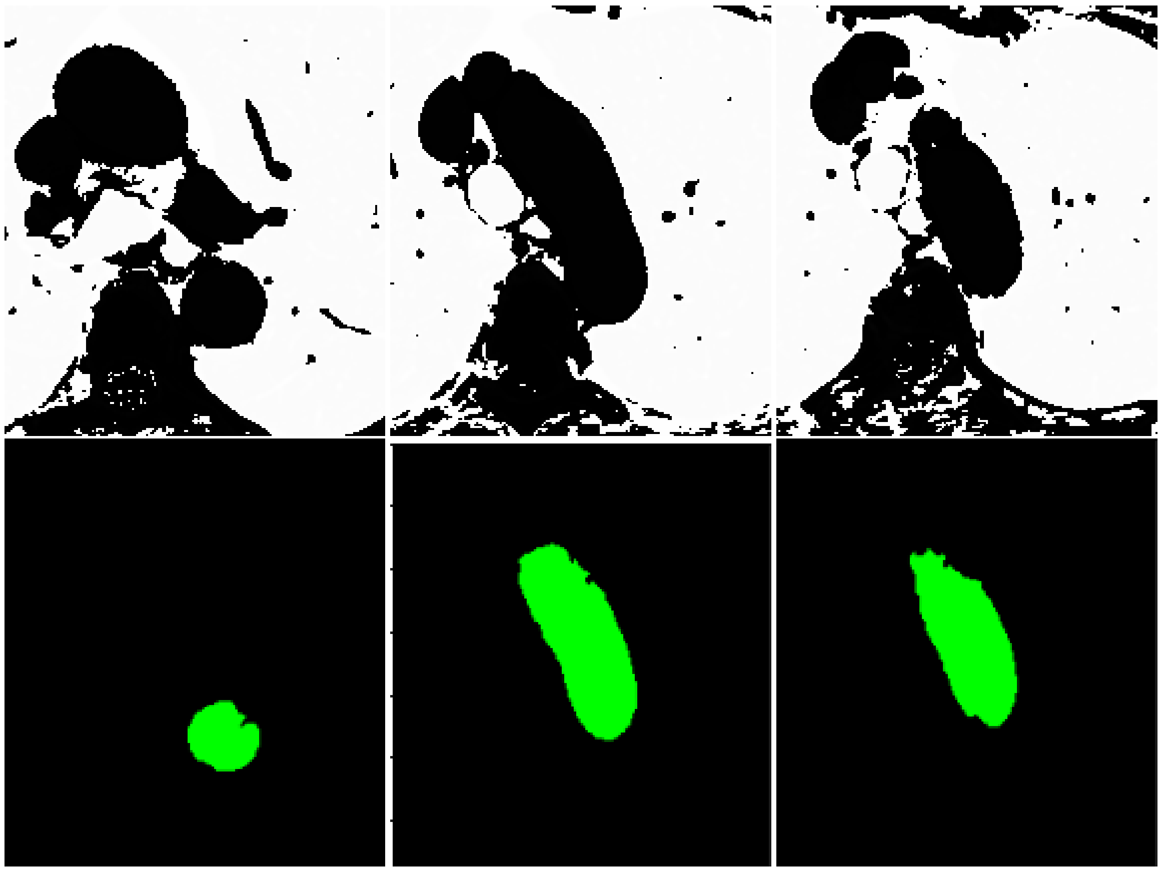


Figure 3.10: Mask generated from SimpleITK

```
        sitk.WriteImage(sitk.LabelOverlay(black_images[:,:,i], seg>0), os.path.join("output",'{0:04d}.tiff'.format(cropped_images[ct_number].GetDepth()-i)))
        for j in range(3):
            with Image(filename = os.path.join("output",'{0:04d}.tiff'.format(cropped_images[ct_number].GetDepth()-i))) as image_png:
                image_png.format = 'png'
                image_png.save(filename=os.path.join("output/png",'{0:04d}.png'.format((cropped_images[ct_number].GetDepth()-i)*3-j)))

        # Image.fromarray(image_png).save(os.path.join("output/png",'{}.png'.format(cropped_images[ct_number].GetDepth()-i)))
```
Listing 3.10: Mapping each slices segmentation results to a black background PNG file.

However, there is a problem with this approach. To get more accurate segmentation results, the raw data of the CT scans will be converted to float 32 images and stored in tiff format.

```
sitk.Show(cropped_images[ct_number])

for i in range(376):

    sitk.WriteImage(sitk.Cast(cropped_images[ct_number][:,:,i], \
                        sitk.sitkFloat32), \
                        os.path.join("input",'{}.tiff'.format(i)))
```

Listing 3.11: Convert raw data to tiff file

This means the training data sets will have float images. This will significantly increasing the computational requirements for this project. Although we can cast those float format images to integer format, some information will be lost and eventually cause distortion. Below Figure 3.11 are the comparisons of a same slice displayed in float format and cast down in integer format.

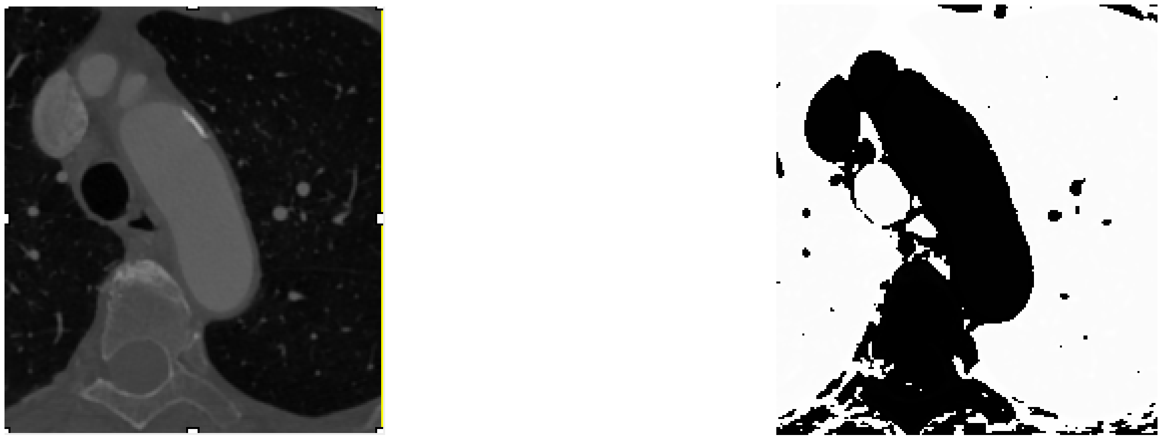


Figure 3.11: Left side is slice in tiff format, right side is cast to integer format

Base on the above discussion, this project uses ITK-snap for generating ground truth training datasets.

Chapter 4

Architecture and Training Algorithms

The training uses the MIScnn pipeline [36], MIScnn features an open model interface to load and switch between the provided state-of-the-art CNN models like the U-Net model we mentioned previously. This training pipeline also requires that training datasets be arranged in a specific structure as shown below.

In the previous section we discussed how to obtain the ground truth data set. After ensuring that each slice of each CT scan has a good mask, we reconstruct the 3D structure of the aorta from the mask information and store it in a NIfTI file.

Training datasets are arranged in this structure, each case folder contains one CT scan raw data (imageing.nii.gz) and it's ground truth labeled data (segmentation.nii.gz).

```
data
├── case_00000
|   ├── imaging.nii.gz
|   └── segmentation.nii.gz
├── case_00001
|   ├── imaging.nii.gz
|   └── segmentation.nii.gz
...
├── case_00005
|   ├── imaging.nii.gz
└── └── segmentation.nii.gz
```

This chapter introduces the training process and the steps of the setup training pipeline. First, we need to establish the data I/O interface, let the pipeline consume the training datasets that we provided. Then it is necessary to configure the data augmentation to prevent overfitting. After that, we can select a neural network model for training. Finally, we perform the validation process for the training model.

4.1 Establish Data I/O

The first step in the MIScnn pipeline is to establish a Data I/O. MIScnn offers the utilization of custom Data I/O interfaces for fast integration of a specific data structure into the pipeline. In the preparation stage, we already encoded the Aorta scans in 3D NIfTI format, therefore we deploy a Data I/O class with the NIfTI interface for handling the NIfTI format. Since this segmentation is binary, we initialize the NIfTI I/O interface and configure the images as one channel (gray-scale) and 2 segmentation classes (background, Aorta), as shown in the following code.

```
# Library import
from miscnn.data_loading.interfaces.nifti_io \
    import NIFTI_interface
from miscnn.data_loading.data_io import Data_IO

# Initialize the NIfTI I/O interface and configure the images as one
   channel (grayscale) and three segmentation classes (background,
   Arota)
interface = NIFTI_interface(pattern="case_00[0-9]*",
                            channels=1, classes=2)

# Specify the Aorta data directory
data_path = "/u40/yanx24/Documents/data_set"
# Create the Data I/O object
data_io = Data_IO(interface, data_path)
```

Listing 4.1: MIScnn Data I/O

4.2 Create and Configure a Data Augmentation class

After the Data I/O Interface initialization, the Data Augmentation class can be configured. Data augmentation is a way to reduce overfitting on models, where we increase the amount of training data using information only in our training data. A Preprocessor object with default parameters automatically initialize a Data Augmentation class with default values, but here we initialize it by hand to illustrate the exact workflow of the MIScnn pipeline.

The parameters for the Data Augmentation configure which augmentation techniques should be applied to the data set. Since the training sets is very small, we have to increase the amount of data by adding slightly modified copies of already existing data using these augmentation techniques. In this case, we are using all possible augmentation techniques to run extensive data augmentation and avoid overfitting [37].

```
# Library import
from miscnn.processing.data_augmentation import Data_Augmentation

# Create and configure the Data Augmentation class
data_aug = Data_Augmentation(cycles=2, scaling=True, rotations=True,
    elastic_deform=True, mirror=True,
                                    brightness=True, contrast=True, gamma=True,
     gaussian_noise=True)
```

Listing 4.2: Data Augmentation

4.3 Select Sub-functions for Preprocessing

For the Preprocessor class, we define which Sub-functions we want to execute on our data set. Sub-functions are pre- and post-processing functions that will be applied on the data to boost performance. It is possible to add already provided Sub-functions from MIScnn or implement custom Sub-functions and pass these to the MIScnn pipeline.

Here, we initialize three state-of-the-art preprocessing methods.

- **Pixel Value Normalization**: Pixel values are normalized through the Z-Score formula.
- **Clipping**: Pixel value ranges are clipped according to a provided min/max value. In this project, for each CT image, each pixel is assigned a value of grayscale level between 0 and 255.
- **Resampling**: MRT and CT images can have different voxel spacings (slice thickness). Therefore, a normalization of these voxel spacings to a common scale is required. The resampling process also lead to a smaller image size. MIScnn provides resampling subfunctions. Note, this resampling is depended on the input datasets. We need to first read the metadata of the CT scan DICOM to determine the optimal voxel spacing. These step is recommended to perform on each of the CT scans.

We are adding all Sub-function objects to a list, which we then be passed to the Preprocessor class.

```python
# Library imports
from miscnn.processing.subfunctions.normalization import Normalization
from miscnn.processing.subfunctions.clipping import Clipping
from miscnn.processing.subfunctions.resampling import Resampling

# Create a pixel value normalization Subfunction through Z-Score
sf_normalize = Normalization(mode='z-score')
# Create a clipping Subfunction between 10 and 255
sf_clipping = Clipping(min=10, max=255)
# Create a resampling Subfunction to voxel spacing 1.62 x 1.62 x 1.62
sf_resample = Resampling((1.62, 1.62, 1.62))

subfunctions = [sf_clipping, sf_normalize]
```

Listing 4.3: Sub-function for preprocessing

4.4 Create a Neural Network Model

With the preparation work done, we can config the Neural network.

To show the simplicity and performance of MIScnn, we stick with the simple 3D U-Net Architecture for our Neural Network, without any special tricks or optimizations. For training, we are using the Dice-Crossentropy (sum of soft Dice and categorical crossentropy) as loss, but also passing the soft Dice and the Tversky Loss as additional metrics for performance evaluation. The choice of these metrics is based on this article: *Generalizing Dice and cross entropy-based losses to handle class imbalanced medical image segmentation* [38].

With the batch_queue_size and number of workers, we are able to specify the parallel prepared batches and the number of threads for multiprocess batch generation, as follows:

```python
# Library import
from miscnn.neural_network.model import Neural_Network
from miscnn.neural_network.metrics import dice_soft, dice_crossentropy,
    tversky_loss

# Create the Neural Network model
model = Neural_Network(preprocessor=pp, loss=tversky_loss, metrics=[
    dice_soft, dice_crossentropy],
                       batch_queue_size=3, workers=3)
```

Listing 4.4: Create a Neural Network model

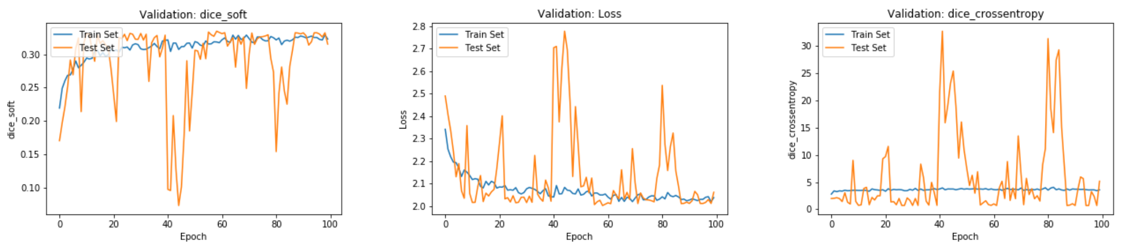


Figure 4.1: Fold 0

4.5 Perform a 3-fold Cross-Validation

We now run the cross_validation function and define the number of k-folds as 3. Due to computational resources constraints, the number of epochs and iterations as 100 and 100, respectively. The training process with these parameters will take more than 300 hours using a RTX 2070 GPU (Table 5.2).

For summary, a cross-validation runs the training and prediction process several times with different combinations of data set parts. Therefore, this step takes long time. The cb_lr and cb_es are Callback parameters, which indexes to automatically reduce the learning rate, if no loss improvement happens in the last 20 epochs.

```
# Library import
from miscnn.evaluation.cross_validation import cross_validation
# Run cross-validation function
cross_validation(validation_samples, model, k_fold=3, epochs=100,
    iterations=100,
                  evaluation_path="evaluation", draw_figures=True,
    callbacks=[cb_lr, cb_es])
```

Listing 4.5: 3-fold Cross-Validation

4.6 Results

Ideally, we would like to see the validation for loss reach below 0.5, and the dice cross entropy converges below 1. However, from Figure 4.1, 4.2 and 4.3, the dice score for all three folds are around 0.3, validation loss still above 2. These metrics are all above the desired values. Moreover, Figure 4.4 shows the prediction of a test data set using the trained model. The result is not accurate in details, but it still roughly reflects the general shape and the location of the aorta. Such a result is expected as this training set has only 6 images. We believe that with enough training data, this training network can generate a

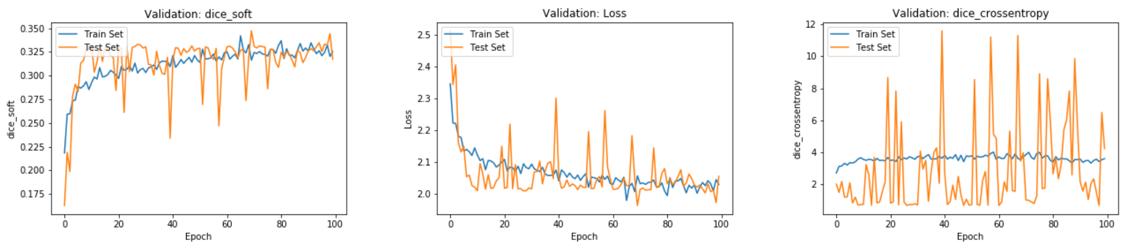


Figure 4.2: Fold 1

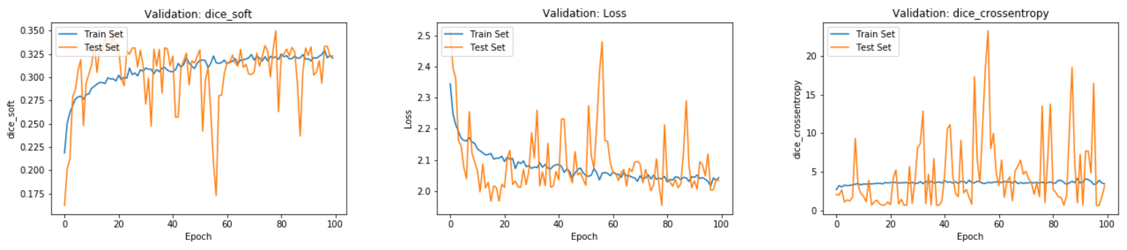


Figure 4.3: Fold 2

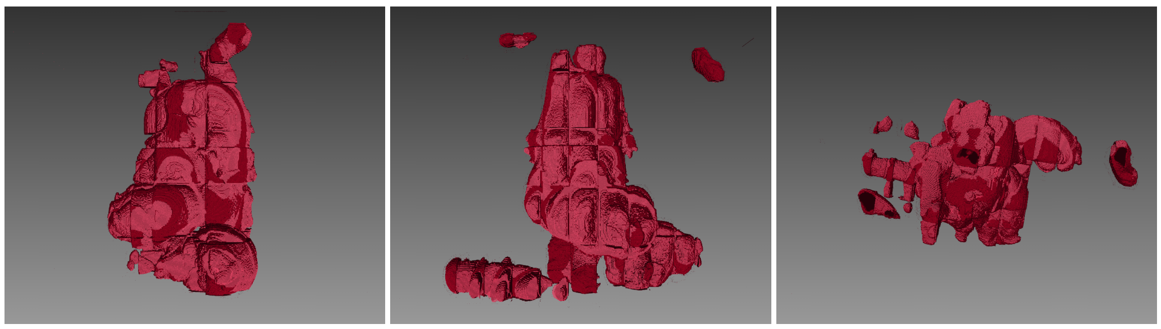


Figure 4.4: Prediction for 3D geometry from the training model

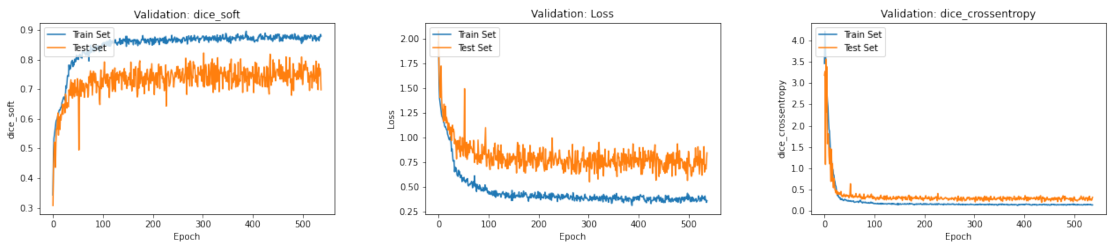


Figure 4.5: Fold 0 for KiTS19

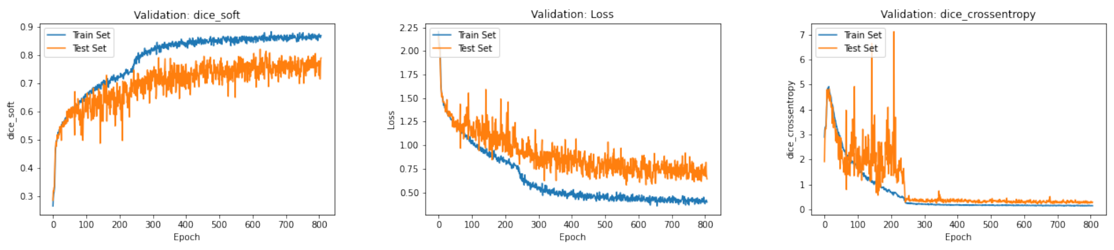


Figure 4.6: Fold 1 for KiTS19

good prediction model. However, for various reasons explained earlier, it is not feasible to obtain more images of the aorta, so to test this conjecture, we can only use the existing data of other organs as the training set. Specifically, we test our approach applied to kidney segmentation.

Figure 4.5, 4.6 and 4.7 show the training loss and dice score for Kidney Tumor Segmentation Challenge 2019 (KITS19) data set [31], this data set has a training set of 210 cross-sectional CT images and a test set of 90 CT images. Compare to Figure 4.1, 4.2 and 4.3, with larger training sets and more training iterations, we are getting higher dice score and lower validation loss. Both validation loss and cross entropy are 10-times better than the aorta results, and dice score also increased about 3 times.

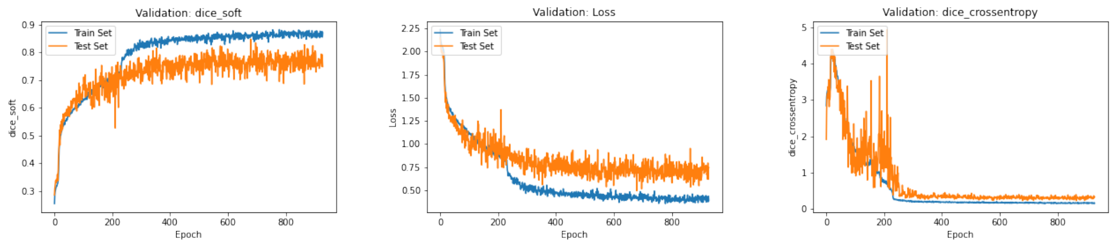


Figure 4.7: Fold 2 for KiTS19

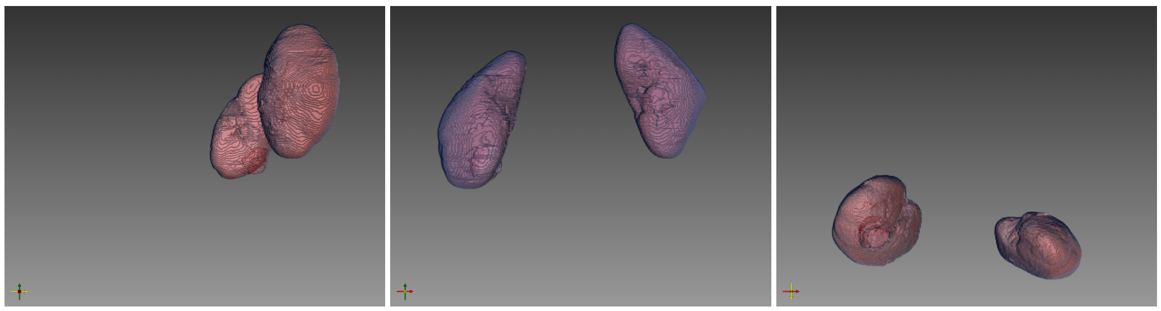


Figure 4.8: Prediction for 3D geometry from the KiTS19 training model

Chapter 5

Challenges and Constraints

This chapter is going to analyze the challenges and difficulties encountered in the previous chapters, Our goal is to summarize the technical and practical difficulties in the clinical application of deep learning algorithms in the field of MI. We especially highlight the feasibility problems that may be encountered by small research teams with limited resource and funding support.

5.1 Data Annotation

Generating ground truth data is the most tedious and time-consuming process for this project. Even with the automation features of ITK-snap, it still requires around 3 hours of single person manual work for one CT-scan images as shown in Table 5.1, since each CT scan has hundreds of image slices. Analyzing each slice's mask is necessary for ground truth accuracy. This means that getting the optimal number (assuming with 100 scans of training datasets and 20 scans of testing datasets) of CT scans for the datasets requires hundreds of hours of data labeling, equivalent to 9 weeks of work. Such workload is impractical for a small research group with a tight deadline.

Effort break down/scan	Average Time in Minutes
Select ROI	15
Manual Segmentation	40
Exam Results	120
Grouping Raw Data with Labeled Data	5
Total	180

Table 5.1: Effort break down for processing one CT scan

5.2 Computational Resources

To speed up the training process, we use two training machines to compute some tasks that can be processed separately. One of the machines was assembled by the researchers themselves, another one is a McMaster University server.

5.2.1 Cost

Personal device: Note: These hardware devices were purchased in mid-2021, at a time when cryptocurrencies were skyrocketing and chips were in short supply [39], This may have inflated the costs. Shown in Table 5.2. Detailed specs for the GPU shown in Figure 5.1.

Assembly	Components	Cost (in CAD)
CPU	AMD Ryzen R9 5950x 16 Cores	1130
GPU	GeForce RTX 2070 SUPER	800
Motherboard	X570 AORUS ELITE	330
Power	Corsair RMX Series, RM850x, 850 Watt	190
Memory	Corsair Vengeance LPX 32GB (2 X 16GB)	150
Total		2600

Table 5.2: Cost for assembled training machine

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 510.47.03    Driver Version: 510.47.03    CUDA Version: 11.6     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  GeForce RTX 207...   On  | 00000000:02:00.0 Off |                    0 |
| N/A   37C    P0    26W / 215W |      48MiB / 8191MiB |     12%      Default|
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
```

Figure 5.1: Specs for Personal device's GPU

McMaster University DL Server: Hardware configuration for the server is shown in Table 5.3. The GPU specs is shown in Figure 5.2. The research team did not pay to use the school's servers, however, to reflect the training costs of other potential research teams that do not have the privilege to utilize universities resources, we calculated the price of cloud computing (AWS) with the similar configuration.

Assembly	Components
CPU	Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz
GPU	Tesla P100 x 4
Memory	64GB

Table 5.3: Configurations of McMaster University DL Server

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 510.47.03    Driver Version: 510.47.03    CUDA Version: 11.6     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla P100-PCIE...  Off  | 00000000:02:00.0 Off |                    0 |
| N/A   37C    P0    26W / 250W |      2MiB / 16384MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   1  Tesla P100-PCIE...  Off  | 00000000:81:00.0 Off |                    0 |
| N/A   34C    P0    27W / 250W |      2MiB / 16384MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   2  Tesla P100-PCIE...  Off  | 00000000:84:00.0 Off |                    0 |
| N/A   54C    P0   132W / 250W |   4159MiB / 16384MiB |     80%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   3  Tesla P100-PCIE...  Off  | 00000000:85:00.0 Off |                    0 |
| N/A   55C    P0   126W / 250W |   3827MiB / 16384MiB |     75%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
```

Figure 5.2: Specs for University Server's GPU

AWS EC2 instance p3.8xlarge has the similar configurations (shown as 5.4)as the DL server. In next subsection, we provided the training time with 6 CT scans of training data on the DL server, which was around 170 hours (1.7 hours per epoch). This translates to a total cost of $2080 USD per run. Of course the training time will continue to increase as the training dataset gets larger. Estimation of this is based on the training time of 6 images of KiTTS19, which takes about half hour for each epoch, knowing that with 210 images of training data, it takes 14.17 hours, assume the training time increase proportionally, with the same amount of training data for the aorta, each epoch will take:

$$1.7 \times (14.17/0.5) = 48.178 \text{ hour} \tag{5.1}$$

Instance Size	GPUs - Tesla V100	Pricing in USD [40]
p3.2xlarge	1	$3.06 per hour
p3.8xlarge	4	$12.24 per hour
p3.16xlarge	8	$24.48 per hour

Table 5.4: Amazon EC2 P3 – Machine Learning and HPC - AWS

5.2.2 Training Time

On personal device, running 100 epochs, each epoch with 100 iterations:

```
Epoch 1/100
Iteration 3/100 [=>...........] - ETA: 2:54:15 - loss: 2.2282
                                - dice_soft: 0.2574
                                - dice_crossentropy: 3.2986
```

This shows that for each epoch, the training time is around 3 hours. The total training time is roughly 300 hours.

On the server, running 100 epochs, each epoch with 100 iterations::

```
Epoch 1/100
Iteration 1/100 [>............] - ETA: 1:42:32 - loss: 2.2982
                                - dice_soft: 0.2724
                                - dice_crossentropy: 3.1456
```

This shows that for each epoch, the training time is around 1.7 hours. The total training time is roughly 170 hours. However, this result is when the training dataset has only 6 CT scans. If we would like to train a dataset with hundreds of CT scans, it will increase the training time significantly as estimated in the equation 5.1.

By comparison, training with KiTS19, on personal device:

```
Epoch 1/100
100/100 [===========] - 50957s 510s/step - loss: 2.0408 - dice_soft: 0.3201
                      - dice_crossentropy: 3.4498 - val_loss: 2.0064
                      - val_dice_soft: 0.3310 - val_dice_crossentropy: 1.7724
```

Each epoch takes 51k seconds, that's about 14.17 hours. With similar results for the server.

5.3 Getting More Data

As mentioned in previous chapters, the research team did not have enough raw data to construct a good training set. Due to the lack of open source databases as raw data, we can only obtain the data directly from the university hospital's database. But this approach is not applicable to a large scale, because these data generally contain private patient information, which means not only a lot of procedural approvals, but also data desensitization.

Chapter 6

Conclusion

In this project, we explored the practical feasibility of deep learning methods for aortic segmentation from CT scans. We tried different approaches for generating high quality training datasets from raw DICOM images, then applied these data to the MIScnn pipeline for training. As shown in Figure 4.4, even though the prediction results are not very accurate in details, it still roughly reflects the general shape and the location of the aorta in the ROI. Such result was expected and acceptable based on the training datasets only having 6 CT scans. In addition, we also performed the training process on KiTS19, compared with the prediction from the trained model, and the result was very promising (as shown in Figure 4.8), this demonstrates the effectiveness of deep learning for medical image segmentation, and showed the importance of having enough data to train a deep learning model.

Through this project, we also summarized and quantified the challenges and constraints encountered, provided a lesson learned to the future research groups with limited resources, outlining some difficulties they would like to address first in similar projects. Specifically, the time and effort in data labeling suggests that researchers may need dedicated annotators to process the desired data. For this project, each CT scan cost around 3 hours to generate an accurate ground-truth segmentation. If the dataset has hundreds of scans (a reasonable amount for deep learning), the annotation process will be months of work. Also, the computational resources may also be very costly for a research team, a low-profile training device for this project still cost more than 2000 CAD. Whether you choose to build your own training machine or rent computing resources from a cloud computing company, the cost usually on the order of tens of thousands of dollars. Moreover, even we trained with relatively powerful AI server, the training time is still around several hundreds hours per run. Finally, getting sufficient medical imaging data from hospitals often introduces more complex approval mechanisms and processes such as data desensitization, which will lead to an increase in work to the research team.

Knowing that the biggest difficulty for this project was labeling the data and getting more data, in the future, we may want to have an infrastructure for hospitals to share their desensitized data to the research groups, or explore the possibility of using unsupervised deep learning for medical image analysis.

Bibliography

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[2] O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” *CoRR*, vol. abs/1505.04597, 2015. [Online]. Available: http://arxiv.org/abs/1505.04597

[3] “Minimally invasive procedure,” Feb 2022. [Online]. Available: https://en.wikipedia.org/wiki/Minimally_invasive_procedure

[4] “What is medical image segmentation and how does it work?” [Online]. Available: https://www.synopsys.com/glossary/what-is-medical-image-segmentation.html#:~:text=Medical%20image%20segmentation%20involves%20the,Computed%20Tomography%20(CT)%20scans.

[5] G. Chlebus, A. Schenk, J. H. Moltz, B. van Ginneken, H. K. Hahn, and H. Meine, “Automatic liver tumor segmentation in CT with fully convolutional neural networks and object-based postprocessing,” *Scientific Reports*, vol. 8, no. 1, p. 15497, Oct. 2018.

[6] W. Li, F. Jia, and Q. Hu, “Automatic segmentation of liver tumor in CT images with deep convolutional neural networks,” *Journal of Computer and Communications*, vol. 03No.11, p. 6, 2015.

[7] B. H. Menze, A. Jakab, S. Bauer, J. Kalpathy-Cramer, K. Farahani, J. Kirby, Y. Burren, N. Porz, J. Slotboom, R. Wiest, L. Lanczi, E. Gerstner, M. -A. Weber, T. Arbel, B. B. Avants, N. Ayache, P. Buendia, D. L. Collins, N. Cordier, J. J. Corso, A. Criminisi, T. Das, H. Delingette, Ç. Demiralp, C. R. Durst, M. Dojat, S. Doyle, J. Festa, F.

Forbes, E. Geremia, B. Glocker, P. Golland, X. Guo, A. Hamamci, K. M. Iftekharuddin, R. Jena, N. M. John, E. Konukoglu, D. Lashkari, J. A. Mariz, R. Meier, S. Pereira, D. Precup, S. J. Price, T. R. Raviv, S. M. S. Reza, M. Ryan, D. Sarikaya, L. Schwartz, H. -C. Shin, J. Shotton, C. A. Silva, N. Sousa, N. K. Subbanna, G. Szekely, T. J. Taylor, O. M. Thomas, N. J. Tustison, G. Unal, F. Vasseur, M. Wintermark, D. H. Ye, L. Zhao, B. Zhao, D. Zikic, M. Prastawa, M. Reyes, and K. Van Leemput, "The multimodal brain tumor image segmentation benchmark (BRATS)," *IEEE Transactions on Medical Imaging*, vol. 34, no. 10, pp. 1993–2024, Oct. 2015.

[8] V. Cherukuri, P. Ssenyonga, B. C. Warf, A. V. Kulkarni, V. Monga, and S. J. Schiff, "Learning based segmentation of CT brain images: Application to postoperative hydrocephalic scans," *IEEE Trans Biomed Eng*, vol. 65, no. 8, pp. 1871–1884, Dec. 2017.

[9] S. Wang, M. Zhou, Z. Liu, Z. Liu, D. Gu, Y. Zang, D. Dong, O. Gevaert, and J. Tian, "Central focused convolutional neural networks: Developing a data-driven model for lung nodule segmentation," *Med Image Anal*, vol. 40, pp. 172–183, Jun. 2017.

[10] *A Study on Heart Segmentation Using Deep Learning Algorithm for MRI Scans*, 2019.

[11] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results," http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

[12] M. Forouzanfar, A. Afshin, L. Alexander, H. Anderson, Z. Bhutta, S. Biryukov, M. Brauer, R. Burnett, K. Cercy, F. Charlson, A. Cohen, L. Dandona, K. Estep, A. Ferrari, J. Frostad, N. Fullman, P. Gething, W. Godwin, M. Griswold, and C. Murray, "Global, regional, and national comparative risk assessment of 79 behavioural, environmental and occupational, and metabolic risks or clusters of risks, 1990–2015: a systematic analysis for the global burden of disease study 2015," *The Lancet*, vol. 388, pp. 1659–1724, 10 2016.

[13] R. Erbel, V. Aboyans, C. Boileau, E. Bossone, R. D. Bartolomeo, H. Eggebrecht, A. Evangelista, V. Falk, H. Frank, O. Gaemperli, M. GrabenwÃ¶ger, A. Haverich, B. Iung, A. J. Manolis, F. Meijboom, C. A. Nienaber, M. Roffi, H. Rousseau, U. Sechtem, P. A. Sirnes, R. S. von Allmen, and C. J. Vrints, "2014 esc guidelines on the diagnosis and treatment of aortic diseases," *Kardiologia Polska (Polish Heart Journal)*, vol. 72, no. 12, pp. 1169 – 1252, 2014. [Online]. Available: https://journals.viamedica.pl/kardiologia_polska/article/view/KP.2014.0225

[14] "Aortic aneurysm fact sheet," Jun 2016. [Online]. Available: http://medbox.iiab.me/modules/en-cdc/www.cdc.gov/dhdsp/data_statistics/fact_sheets/fs_aortic_aneurysm.htm#socialMediaShareContainer

[15] G. R. Upchurch, Jr and T. A. Schaub, "Abdominal aortic aneurysm," *Am Fam Physician*, vol. 73, no. 7, pp. 1198–1204, Apr. 2006.

[16] M. Wortmann, R. Klotz, E. Kalkum, S. Dihlmann, D. Böckler, and A. S. Peters, "Inflammasome targeted therapy as novel treatment option for aortic aneurysms and dissections: A systematic review of the preclinical evidence," *Frontiers in Cardiovascular Medicine*, vol. 8, 2022. [Online]. Available: https://www.frontiersin.org/article/10.3389/fcvm.2021.805150

[17] T. Siriapisith, W. Kusakunniran, and P. Haddawy, "Outer wall segmentation of abdominal aortic aneurysm by variable neighborhood search through intensity and gradient spaces," *Journal of Digital Imaging*, vol. 31, no. 4, pp. 490–504, Aug. 2018.

[18] A. Comelli, N. Dahiya, A. Stefano, V. Benfante, G. Gentile, V. Agnese, G. M. Raffa, M. Pilato, A. Yezzi, G. Petrucci *et al.*, "Deep learning approach for the segmentation of aneurysmal ascending aorta," *Biomedical Engineering Letters*, vol. 11, no. 1, pp. 15–24, 2021.

[19] "Cs231n: Deep learning for computer vision." [Online]. Available: http://cs231n.stanford.edu/

[20] L. Chen, P. Bentley, and D. Rueckert, "Fully automatic acute ischemic lesion segmentation in dwi using convolutional neural networks," *NeuroImage: Clinical*, vol. 15, pp. 633–643, 2017.

[21] L. Liu, S. Chen, F. Zhang, F.-X. Wu, Y. Pan, and J. Wang, "Deep convolutional neural network for automatically segmenting acute ischemic stroke lesion in multi-modality mri," *Neural Computing and Applications*, vol. 32, no. 11, pp. 6545–6558, 2020.

[22] J. N. Stember, P. Chang, D. M. Stember, M. Liu, J. Grinband, C. G. Filippi, P. Meyers, and S. Jambawalikar, "Convolutional neural networks for the detection and measurement of cerebral aneurysms on magnetic resonance angiography," *Journal of digital imaging*, vol. 32, no. 5, pp. 808–815, 2019.

[23] D. Ueda, A. Yamamoto, M. Nishimori, T. Shimono, S. Doishita, A. Shimazaki, Y. Katayama, S. Fukumoto, A. Choppin, Y. Shimahara *et al.*, "Deep learning for mr angiography: automated detection of cerebral aneurysms," *Radiology*, vol. 290, no. 1, pp. 187–194, 2019.

[24] M. Havaei, A. Davy, D. Warde-Farley, A. Biard, A. Courville, Y. Bengio, C. Pal, P.-M. Jodoin, and H. Larochelle, "Brain tumor segmentation with deep neural networks," *Medical image analysis*, vol. 35, pp. 18–31, 2017.

[25] R. Ranjbarzadeh, A. Bagherian Kasgari, S. Jafarzadeh Ghoushchi, S. Anari, M. Naseri, and M. Bendechache, "Brain tumor segmentation based on deep learning and an attention mechanism using mri multi-modalities brain images," *Scientific Reports*, vol. 11, no. 1, p. 10930, May 2021. [Online]. Available: https://doi.org/10.1038/s41598-021-90428-8

[26] Z. Xiong, V. V. Fedorov, X. Fu, E. Cheng, R. Macleod, and J. Zhao, "Fully automatic left atrium segmentation from late gadolinium enhanced magnetic resonance imaging using a dual fully convolutional neural network," *IEEE transactions on medical imaging*, vol. 38, no. 2, pp. 515–524, 2018.

[27] X.-F. Han, H. Laga, and M. Bennamoun, "Image-based 3d object reconstruction: State-of-the-art and trends in the deep learning era," *IEEE transactions on pattern analysis and machine intelligence*, vol. 43, no. 5, pp. 1578–1604, 2019.

[28] H. Xie, H. Yao, X. Sun, S. Zhou, and S. Zhang, "Pix2vox: Context-aware 3d reconstruction from single and multi-view images," *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, Oct 2019. [Online]. Available: http://dx.doi.org/10.1109/ICCV.2019.00278

[29] X.-F. Han, H. Laga, and M. Bennamoun, "Image-based 3d object reconstruction: State-of-the-art and trends in the deep learning era," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 5, p. 1578–1604, May 2021. [Online]. Available: http://dx.doi.org/10.1109/TPAMI.2019.2954885

[30] N. Heller, F. Isensee, K. H. Maier-Hein, X. Hou, C. Xie, F. Li, Y. Nan, G. Mu, Z. Lin, M. Han *et al.*, "The state of the art in kidney and kidney tumor segmentation in contrast-enhanced ct imaging: Results of the kits19 challenge," *Medical Image Analysis*, p. 101821, 2020.

[31] N. Heller, N. Sathianathen, A. Kalapara, E. Walczak, K. Moore, H. Kaluzniak, J. Rosenberg, P. Blake, Z. Rengel, M. Oestreich *et al.*, "The kits19 challenge data: 300 kidney tumor cases with clinical context, ct semantic segmentations, and surgical outcomes," *arXiv preprint arXiv:1904.00445*, 2019.

[32] P. Bilic, P. F. Christ, E. Vorontsov, G. Chlebus, H. Chen, Q. Dou, C. Fu, X. Han, P. Heng, J. Hesser, S. Kadoury, T. K. Konopczynski, M. Le, C. Li, X. Li, J. Lipková, J. S. Lowengrub, H. Meine, J. H. Moltz, C. Pal, M. Piraud, X. Qi, J. Qi, M. Rempfler, K. Roth, A. Schenk, A. Sekuboyina, P. Zhou, C. Hülsemeyer, M. Beetz, F. Ettlinger, F. Grün, G. Kaissis, F. Lohöfer, R. Braren, J. Holch, F. Hofmann, W. H. Sommer, V. Heinemann, C. Jacobs, G. E. H. Mamani, B. van Ginneken, G. Chartrand, A. Tang, M. Drozdzal, A. Ben-Cohen, E. Klang, M. M.

Amitai, E. Konen, H. Greenspan, J. Moreau, A. Hostettler, L. Soler, R. Vivanti, A. Szeskin, N. Lev-Cohain, J. Sosna, L. Joskowicz, and B. H. Menze, "The liver tumor segmentation benchmark (lits)," *CoRR*, vol. abs/1901.04056, 2019. [Online]. Available: http://arxiv.org/abs/1901.04056

[33] S. M. Moore, D. R. Maffitt, K. E. Smith, J. S. Kirby, K. W. Clark, J. B. Freymann, B. A. Vendt, L. R. Tarbox, and F. W. Prior, "De-identification of medical images with retention of scientific research value," *Radiographics*, vol. 35, no. 3, pp. 727–735, May 2015.

[34] T. Tseng, A. Stent, and D. Maida, "Best practices for managing data annotation projects," *arXiv preprint arXiv:2009.11654*, 2020.

[35] SimpleITK, "Simpleitk/simpleitk: Simpleitk: A layer built on top of the insight toolkit (itk), intended to simplify and facilitate itk's use in rapid prototyping, education and interpreted languages." [Online]. Available: https://github.com/SimpleITK/SimpleITK

[36] D. Müller and F. Kramer, "Miscnn: a framework for medical image segmentation with convolutional neural networks and deep learning," *BMC medical imaging*, vol. 21, no. 1, pp. 1–11, 2021.

[37] L. Perez and J. Wang, "The effectiveness of data augmentation in image classification using deep learning," *arXiv preprint arXiv:1712.04621*, 2017.

[38] M. Yeung, E. Sala, C.-B. Schönlieb, and L. Rundo, "Unified focal loss: Generalising dice and cross entropy-based losses to handle class imbalanced medical image segmentation," *Computerized Medical Imaging and Graphics*, vol. 95, p. 102026, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0895611121001750

[39] P. C. Earle, "The great semiconductor shortage," Jul 2021. [Online]. Available: https://www.aier.org/article/the-great-semiconductor-shortage/

[40] J. B. Gilmour, A. W. Lui, and D. C. Briggs, "Emr," 2022. [Online]. Available: https://aws.amazon.com/emr/pricing/

Appendix A

Python implementation of UNet

```
def build_model(input_layer, start_neurons):
    conv1 = Conv2D(start_neurons*1, (3,3), activation="relu", padding="same")(input_layer)
    conv1 = Conv2D(start_neurons*1, (3,3), activation="relu", padding="same")(conv1)
    pool1 = MaxPooling2D((2,2))(conv1)
    pool1 = Dropout(0.25)(pool1)

    conv2 = Conv2D(start_neurons*2, (3,3), activation="relu", padding="same")(pool1)
    conv2 = Conv2D(start_neurons*2, (3,3), activation="relu", padding="same")(conv2)
    pool2 = MaxPooling2D((2,2))(conv2)
    pool2 = Dropout(0.5)(pool2)

    conv3 = Conv2D(start_neurons*4, (3,3), activation="relu", padding="same")(pool2)
    conv3 = Conv2D(start_neurons*4, (3,3), activation="relu", padding="same")(conv3)
    pool3 = MaxPooling2D((2,2))(conv3)
    pool3 = Dropout(0.5)(pool3)

    conv4 = Conv2D(start_neurons*8, (3,3), activation="relu", padding="same")(pool3)
    conv4 = Conv2D(start_neurons*8, (3,3), activation="relu", padding="same")(conv4)
    pool4 = MaxPooling2D((2,2))(conv4)
    pool4 = Dropout(0.5)(pool4)

    # Middle
    convm = Conv2D(start_neurons*16, (3,3), activation="relu", padding="same")(pool4)
    convm = Conv2D(start_neurons*16, (3,3), activation="relu", padding="same")(convm)

    deconv4 = Conv2DTranspose(start_neurons*8, (3,3), strides=(2, 2), padding="same")(
    convm)
    uconv4 = concatenate([deconv4, conv4])
    uconv4 = Dropout(0.5)(uconv4)
    uconv4 = Conv2D(start_neurons*8, (3,3), activation="relu", padding="same")(uconv4)
    uconv4 = Conv2D(start_neurons*8, (3,3), activation="relu", padding="same")(uconv4)

    deconv3 = Conv2DTranspose(start_neurons * 4, (3,3), strides=(2,2), padding="same")(
    uconv4)
    uconv3 = concatenate([deconv3, conv3])
    uconv3 = Dropout(0.5)(uconv3)
    uconv3 = Conv2D(start_neurons*4, (3,3), activation="relu", padding="same")(uconv3)
    uconv3 = Conv2D(start_neurons*4, (3,3), activation="relu", padding="same")(uconv3)

    deconv2 = Conv2DTranspose(start_neurons * 2, (3,3), strides=(2,2), padding="same")(
    uconv3)
    uconv2 = concatenate([deconv2, conv2])
    uconv2 = Dropout(0.5)(uconv2)
    uconv2 = Conv2D(start_neurons*2, (3,3), activation="relu", padding="same")(uconv2)
    uconv2 = Conv2D(start_neurons*2, (3,3), activation="relu", padding="same")(uconv2)

    deconv1 = Conv2DTranspose(start_neurons * 1, (3,3), strides=(2,2), padding="same")(
    uconv2)
    uconv1 = concatenate([deconv1, conv1])
    uconv1 = Dropout(0.5)(uconv1)
    uconv1 = Conv2D(start_neurons*1, (3,3), activation="relu", padding="same")(uconv1)
    uconv1 = Conv2D(start_neurons*1, (3,3), activation="relu", padding="same")(uconv1)

    output_layer = Conv2D(1, (1,1), padding="same", activation="sigmoid")(uconv1)

    return output_layer

input_layer = Input((img_size_target, img_size_target, 1))
output_layer = build_model(input_layer, 16)
```

Listing A.1: input image size 2×2

Appendix B

Aorta Segmenting Function

```python
def circle_filter(i, centre, image_type = "reg"):
    # set slice
    if(image_type == "255"):
        imgSlice = images_255[ct_number][:,:,i]
    elif(image_type == "cropped"):
        imgSlice = cropped_images[ct_number][:,:,i]
    elif(image_type == "cropped_255"):
        imgSlice = cropped_images_255[ct_number][:,:,i]
    elif(image_type == "reg"):
        imgSlice = images[ct_number][:,:,i]

    # re-normalize
    if(normalize):
        imgSlice = sitk.Cast(sitk.RescaleIntensity(imgSlice), sitk.sitkUInt8)

    # make new image for putting seed in
    seg_2d = sitk.Image(imgSlice.GetSize(), sitk.sitkUInt8)
    seg_2d.CopyInformation(imgSlice)

    # add original seed and additional seeds three pixels apart
    spacing = 3
    for j in range(-1,2):
        one = centre[0] + spacing*j
        two = centre[1]
        seg_2d[(one,two)] = 1

    seg_2d = sitk.BinaryDilate(seg_2d, [3]*2)

    # determine threshold values based on seed location
    stats = sitk.LabelStatisticsImageFilter()
    stats.Execute(imgSlice, seg_2d)

    factor = 3.5
    lower_threshold = stats.GetMean(1)-factor*stats.GetSigma(1)
    upper_threshold = stats.GetMean(1)+factor*stats.GetSigma(1)

    # use filter to apply threshold to image
    init_ls = sitk.SignedMaurerDistanceMap(seg_2d, insideIsPositive=True, useImageSpacing=
    True)

    # segment the aorta using the seed values and threshold values
    lsFilter = sitk.ThresholdSegmentationLevelSetImageFilter()
    lsFilter.SetLowerThreshold(lower_threshold)
    lsFilter.SetUpperThreshold(upper_threshold)
    lsFilter.SetMaximumRMSError(0.02)
    lsFilter.SetNumberOfIterations(1000)
    lsFilter.SetCurvatureScaling(.5)
    lsFilter.SetPropagationScaling(1)
    lsFilter.ReverseExpansionDirectionOn()
    ls = lsFilter.Execute(init_ls, sitk.Cast(imgSlice, sitk.sitkFloat32))

    # assign segmentation to fully_seg_slice
    if(white_circles):
        fully_seg_slice = ls>0
    else:
        if(image_type == "255" or image_type == "reg"):
            fully_seg_slice = sitk.LabelOverlay(images_255[ct_number][:,:,i], ls>0)
        else:
            fully_seg_slice = sitk.LabelOverlay(cropped_images[ct_number][:,:,i], ls>0)

    # get array from segmentation
    nda = sitk.GetArrayFromImage(ls>0)

    # calculate average x and average y values, to get the new centre value
    avg_x = 0
    avg_y = 0
    total_coord = 0

    for x in range(len(nda)):
        for y in range(len(nda[0])):
            if(nda[x][y]==1):
                avg_x += x
                avg_y += y
                total_coord+=1

    centre_new = (int(avg_y/total_coord),int(avg_x/total_coord))

    return fully_seg_slice, total_coord, centre_new, lower_threshold, upper_threshold
```

Listing B.1: Aorta Segmenting Function

Appendix C

SimpleITK for Aorta segmentation

This is work from Kailin, modified her work to generate training data.

Segmentation of Thoracic Aorta

Overview

The goal of this algorithm is to fully segment the thoracic aorta. A diagram of the thoracic aorta is shown below.

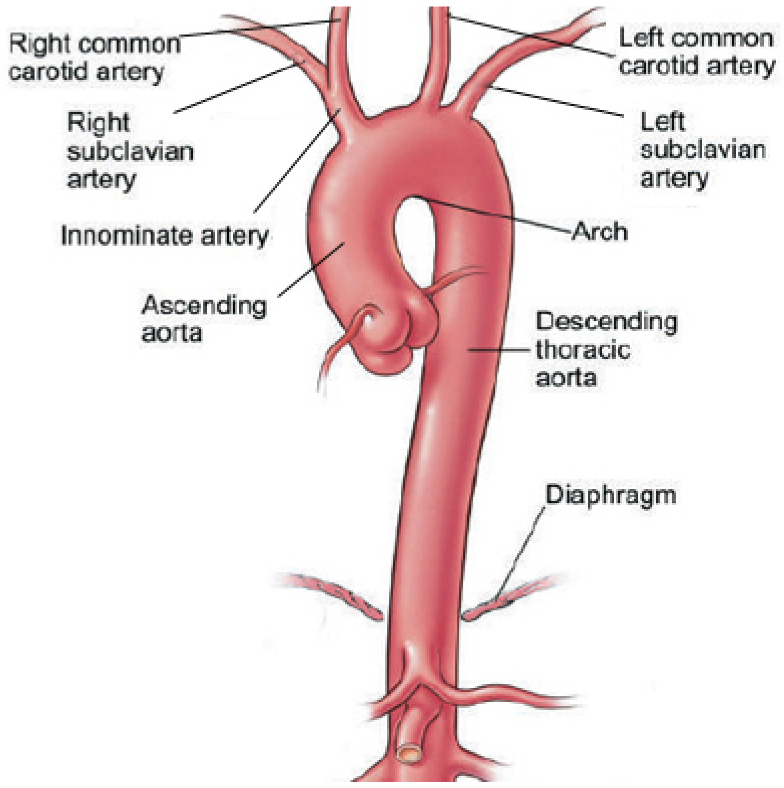


Image from the [Cleveland Clinic]()

This algorithm consists of five main sections:

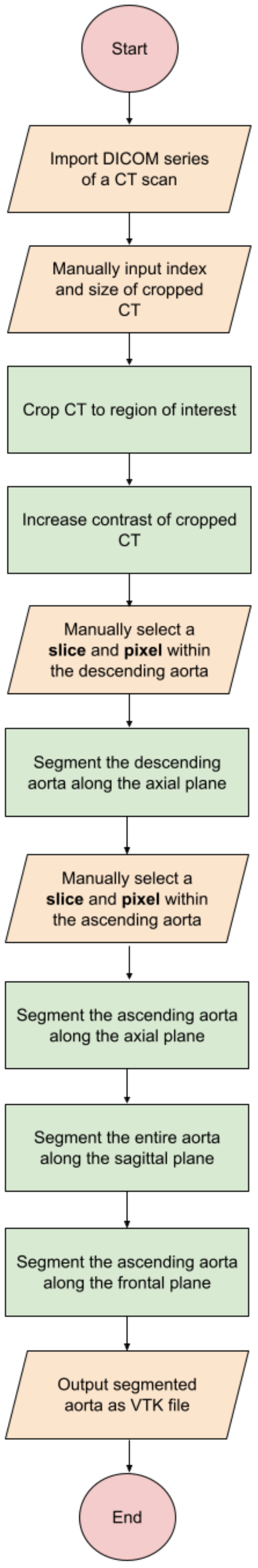
1. Setup - cropping the image and performing contrast enhancement
2. Axial Segmentation - segmenting the aorta axially. This is a two step process.
    a. The descending aorta is segmented first.
    b. The ascending aorta and aortic arch are segmented second.

    Both the descending and ascending segmentations require a user-entered seed value.
3. Sagittal Segmentation - segmenting the aorta sagittally to smooth out the existing segmentation
4. Frontal Segmentation - segmenting the aorta frontally to smooth out the existing segmentation
5. Output - exporting the segmentation as a VTK file that can be viewed in Paraview.

This algorithm was tested in Jupyter Labs using Python 3.9.4. The code can be found in the master branch of the GeomRecon repository under People > Kailin > src > circle-method.ipynb. The following document will outline this algorithm in more detail. You can also follow along using the comments in the Jupyter Notebook file.

General Flowchart

Below is a more visual representation of the algorithm explained above.


Start
Import DICOM series of a CT scan
Manually input index and size of cropped CT
Crop CT to region of interest
Increase contrast of cropped CT
Manually select a **slice** and **pixel** within the descending aorta
Segment the descending aorta along the axial plane
Manually select a **slice** and **pixel** within the ascending aorta
Segment the ascending aorta along the axial plane
Segment the entire aorta along the sagittal plane
Segment the ascending aorta along the frontal plane
Output segmented aorta as VTK file
End

Detailed Description of Processes

Several of the processes in the “General Flowchart” are described below.

Manually input index and size of cropped CT

To determine these values, you can open the cropped CT in ITK-Snap. In ITK-Snap, you can visualize a region of interest by clicking the "active contour" button in the Main Toolbar - it looks like a snake. You can then drag the red dotted line to the appropriate size. The "Position" values correspond with the `indexes` or `index` variable. This is the point closest to the right, anterior, inferior corner. "Size" corresponds to the `sizes` or `size` variable.

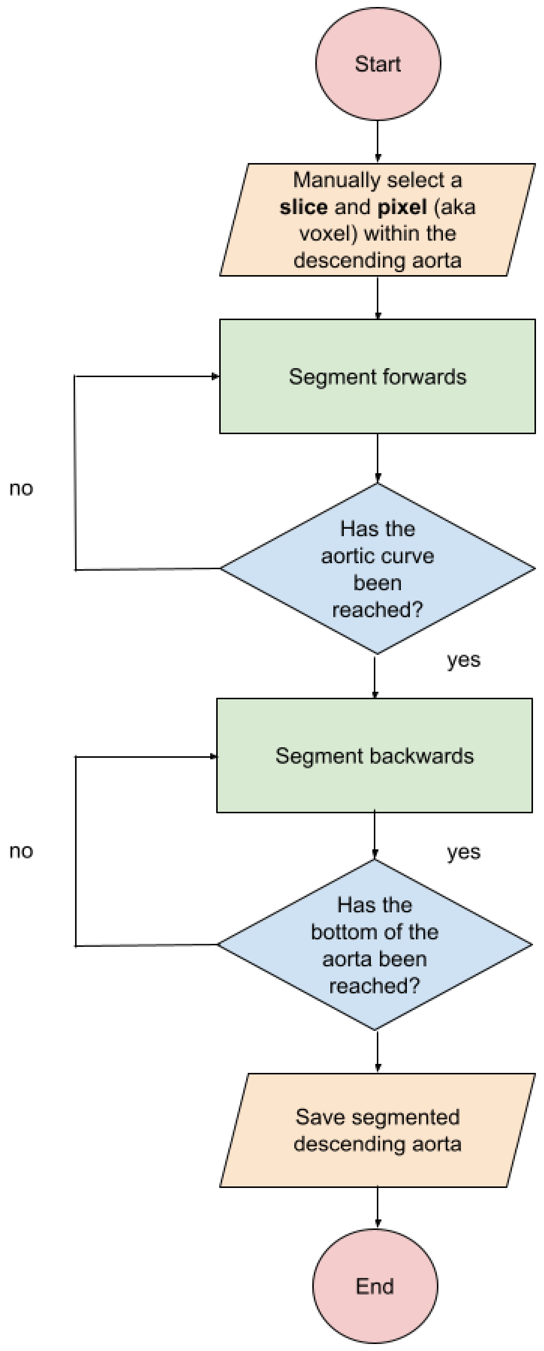
Increase Contrast of Cropped CT

1. Apply histogram equalization (HE) to the cropped CT to increase contrast
   a. The HE algorithm flattens the image into an array. At this point, the elements in the array (i.e., the intensities), are likely within a small range. Next, a mapping table is created. The elements in the array are spread across this table to get a wider range of intensities. The array is turned back into an image. This increases the contrast of the image.

2. Apply a median filter to the cropped CT to reduce noise
   a. Ahmed and Nordin, at the School of Computer Science at University Kebangsaan Malaysia, recommended the use of a median filter to reduce noise after performing contrast enhancement. [https://thescipub.com/pdf/jcssp.2011.1831.1838.pdf]
   b. The aorta segmentation algorithm uses the built-in median filter provided by SimpleITK

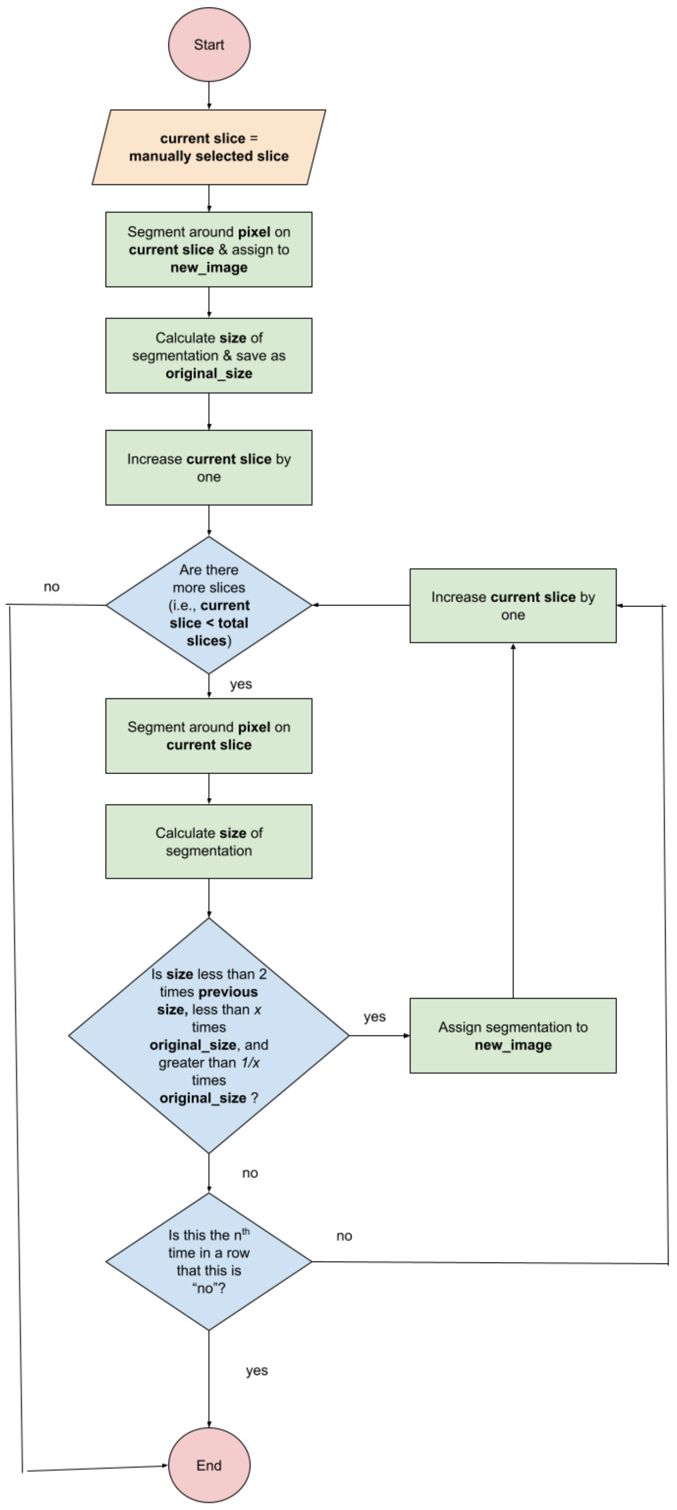
Select Voxel & Segment the *Descending* Aorta Along the Axial Plane

Below is a more detailed flowchart of the descending aorta segmentation algorithm.


Start
Manually select a **slice** and **pixel** (aka voxel) within the descending aorta
Segment forwards
no
Has the aortic curve been reached?
yes
Segment backwards
no
yes
Has the bottom of the aorta been reached?
Save segmented descending aorta
End

Segment forwards & detect whether the aortic curve has been reached

Below is an even more detailed flowchart describing the forward segmentation algorithm.


Start
current slice = manually selected slice
Segment around pixel on current slice & assign to new_image
Calculate size of segmentation & save as original_size
Increase current slice by one
Are there more slices (i.e., current slice < total slices)
no
yes
Increase current slice by one
Segment around pixel on current slice
Calculate size of segmentation
Is size less than 2 times previous size, less than x times original_size, and greater than 1/x times original_size ?
yes
Assign segmentation to new_image
no
Is this the nth time in a row that this is "no"?
no
yes
End

The "segment around pixel on current slice" represents a function that accepts the image slice and the centre of gravity of the previous image.

The function assigns the centre of gravity as a seed and assigns two other seeds nearby. It calculates the upper and lower threshold based on the mean and sigma intensities of the given slice:

```
    lower_threshold = stats.GetMean(1)-factor*stats.GetSigma(1)
    upper_threshold = stats.GetMean(1)+factor*stats.GetSigma(1)
```

It then creates a Euclidean image of the CT slice using the SignedMaurerDistanceMapImageFilter. Next, it applies the ThresholdSegmentationLevelSetImageFilter on the Euclidean image, using the threshold values to segment around the seeds.

The function calculates the size and centre of gravity of the segmented image by counting the pixels.

The function returns the segmented slice, size, new centre of gravity, and threshold values.

Segment backwards & detect whether the bottom has been reached

Same as segment forwards, except instead of increasing current slice by one, decrease current slice by one.

Select Voxel & Segment the *Ascending* Aorta Along the Axial Plane

This is virtually the same as Select Voxel & Segment the *Descending* Aorta Along the Axial Plane. Here are a few of the differences:

Comparison between Descending and Ascending Aorta Functions

Descending Aorta	**Ascending Aorta**
- To determine the threshold values, `circle_filter` (the descending aorta function) accepts a `centre` value (x,y) for each axial slice. `Centre` acts as a seed value. The function "creates" two additional seeds 3 pixels to the left and right of the `centre`. Thus, `circle_filter` uses 3 closely	- To determine the threshold values, `circle_filter_arch` (the ascending aorta function) accepts a list of seeds, in addition to a `centre` value (x,y) for each axial slice. It uses the seeds and the `centre` value to calculate the threshold values. - After it gets the next segmentation, it

spaced seed values to calculate the threshold values.	finds the next seed values. It does this by getting a point 25% outwards from the `centre` value in the x, y, -x, and -y directions. (By 25% I mean 25% of width/height of segmentation). - I chose to use a more robust seed value approach for the ascending aorta because `circle_filter_arch` is responsible for segmenting the aortic arch. Since we are segmenting axially, the aortic arch is a long, peanut shape. 3 closely spaced seed values would not achieve accurate threshold values. This method gives us seeds that are further spaced apart from one another.
- When segmenting towards the aorta, the descending aorta stops once it is 4 times the size of the original size.	- When segmenting towards the aorta, the ascending aorta stops once it is 4 times the size of the original size. This is because of the long, peanut shape mentioned earlier. Instead, it bases most of the calculations on the previous image size - The algorithm (for the ascending aorta towards the arch) also changes the `factor_size_overlap` value depending on whether it has gotten to the peanut shape section and whether the peanut has started decreasing in size. - If it has gotten to the peanut shape and the ascending aorta segmentation has started to overlap with the descending aorta segmentation, `factor_size_overlap` is increased to accommodate for the rapidly growing size - If the peanut size is decreasing, `factor_size_overlap` will also decrease to avoid over segmentation

Segment the entire aorta along the sagittal plane

This process segments the aorta sagittally to smooth out the existing segmentation. The sagittal segmentation section takes the current segmented image and iterates through every sagittal slice. If there is a decent number of “segmented pixels” under a given slice from the previous axial segmentation, the algorithm uses all the points on that slice as seed values. These seed values are used to determine the threshold values and create a new segmentation. If the new, sagittal segmentation has any new pixels that the axial segmentation did not, the new pixels will be added to the axial segmentation.

Segment the entire aorta along the frontal plane

Segment the aorta frontally to further smooth out the existing segmentation. It uses the same logic as the sagittal segmentation.

* Note: I created a variable called `faulty` that is assigned to 1 if the image is likely inadequate. `Faulty` is assigned to 0 if it is likely adequate. `Faulty` is determined during the ascending aorta section of the algorithm. If the size of the axial peanut reaches 2 times the original size of the aortic circle, the image is likely to be adequate for the algorithm to work. Similarly, if the ascending aorta segmentations meet the descending aorta segmentations, the image is likely to be adequate.