VIETNAM NATIONAL UNIVERSITY, HANOI

UNIVERSITY OF ENGINEERING AND TECHNOLOGY

Le Ngoc Tuan Khang

AUTOMATIC SEGMENTATION OF LEFT VENTRICLE IN 2D ECHOCARDIOGRAM USING DEEP LEARNING

Major: Computer Science

HANOI - 2019

VIETNAM NATIONAL UNIVERSITY, HANOI

UNIVERSITY OF ENGINEERING AND TECHNOLOGY

Le Ngoc Tuan Khang

AUTOMATIC SEGMENTATION OF LEFT VENTRICLE IN 2D ECHOCARDIOGRAM USING DEEP LEARNING

Major: Computer Science

Supervisor: Dr. Tran Quoc Long

Co-Supervisor: Assoc. Prof. Dr. Le Sy Vinh

HÀ NỘI - 2019

AUTHORSHIP

"I hereby declare that the work contained in this thesis is of my own and has not been previously submitted for a degree or diploma at this or any other higher education institution. To the best of my knowledge and belief, the thesis contains no materials previously published or written by another person except where due reference or acknowledgement is made."

Signature:

SUPERVISOR'S APPROVAL

"I hereby approve that the thesis in its current form is ready for committee examination as a requirement for the Bachelor of Computer Science degree at the University of Engineering and Technology."

Signature:

ACKNOWLEDGMENT

First and foremost, I would like to express my genuine gratitude to my supervisor, Dr. Tran Quoc Long. Without his technical advice and dedicated guidance throughout the process, this thesis would have never been accomplished. I am also grateful to my co-supervisor, Assoc. Prof. Dr. Le Sy Vinh, for his helpful insight on various obstacles arisen along the way.

Next, I would like to thank all lecturers at the Faculty of Information Technology, VNU University of Engineering and Technology, for their assistance and valuable lessons over my student years at the university.

Last but by no means least, I take this opportunity to thank my family and my friends, for their emotional support that have encouraged me at difficult times in life.

ABSTRACT

Ejection fraction (EF) is a heart measurement of how much blood the left ventricle (LV) pumps out with each contraction, which can help doctors diagnose and track heart failure. Calculating this value requires the segmentation of LV in two-dimensional (2D) echocardiography, which is usually done manually by medical experts or specialized devices. While there are paid software available in large hospitals that can perform this segmentation automatically, many local faculties do not have access to such utilities, resulting in the requirement of manual segmentation, which is professional-dependent and biased. This raises the need of a mobile application that can support local doctors to segment LV and calculate EF, and in turn of a segmentation method that is able to work on noisy images taken from smartphones with a practical speed and an acceptable accuracy. This is a new problem, as LV segmentation methods proposed in the literature usually work on original images taken directly from 2D echocardiogram output, thus less subject to environmental factors such as lighting conditions and noises. With the rise of deep learning in the field computer vision in general and medical image processing in particular, which has shown exceptional performances on key problems of the field and far surpassed traditional machine learning and image processing methods, it is a natural question to ask whether deep learning approach is suitable for the proposed problem. In this thesis, some deep learning architectures on the segmentation task will be experimented, along with some other processing techniques and practical strategies, on a data set of LV images from 2D echo taken by mobile devices and annotated by experts. Results showed that a deep learning model with carefully chosen architectures and suitable processing methods can give a prediction of LV region on a noisy image in a reasonable time with high accuracy, which is feasible for mobile applications.

Keywords: left ventricle segmentation, deep learning, ejection fraction, Unet, Linknet, MobileNetV2, ResNet, active shape model, data augmentation, pseudo-labeling.

Contents

List of Figures v

List of Tables v

1 Introduction **1**

1.1 Motivation 1

1.2 Related Work 3

1.3 Contributions 5

2 Background **6**

2.1 Problem Formalization 6

2.2 Segmentation 6

2.3 Deep Learning 7

2.3.1 Artificial neural network 7

2.3.2 Convolutional neural network 9

2.4 Deep CNN Architectures for Segmentation 11

2.4.1 Unet 11

2.4.2 Linknet 12

2.5 Encoders and Decoders 12

2.5.1 Encoders . . . 12
2.5.2 Decoders . . . 16
2.6 Loss function . . . 19
2.6.1 Binary cross-entropy . . . 19
2.6.2 Jaccard distance . . . 19
2.6.3 Dice coefficient . . . 20
2.6.4 Combined losses . . . 21
2.7 Active contour model . . . 21

3 Methods **23**

3.1 Overall pipeline . . . 23
3.2 Preprocessing . . . 25
3.2.1 Color spaces . . . 25
3.2.2 Data augmentation . . . 25
3.3 Architectures . . . 26
3.3.1 Segmentation Architectures . . . 26
3.3.2 Encoder Backbones . . . 26
3.3.3 Decoder . . . 28
3.4 Boundary-weighted loss function . . . 28
3.5 Training and Inference Strategy . . . 28
3.5.1 Pseudo-labeling . . . 28
3.5.2 Test-time augmentation . . . 31
3.6 Post processing . . . 31
3.6.1 Active contour model . . . 32

3.6.2 Active shape model 32

4 Experiments **36**

4.1 Data 36

4.2 Experimental Settings 36

4.2.1 Data Augmentation 36

4.2.2 Architectures 37

4.2.3 Training 38

4.2.4 Postprocessing 38

4.3 Metric 38

4.4 Results and Analysis 39

4.4.1 Color spaces 39

4.4.2 Data augmentation 39

4.4.3 Loss function 40

4.4.4 Architectures 40

4.4.5 Strategies 41

4.4.6 Postprocessing 44

4.4.7 Inference time 44

4.4.8 Ejection Fraction 45

5 Conclusions **48**

A Statistical Shape Analysis **50**

A.1 Shapes and Landmarks 50

A.2 Shape Alignment - Procrustes Analysis 50

A.3 Shape Modeling - Principal Component Analysis 51

B Ejection Fraction Calculation using Simpson's Rule 52

List of Figures

1.1 Three typical views obtained during a routine TTE 2
1.2 Visualization of the heart from A4C and A2C view 3
1.3 Biplane Simpson method 4

2.1 Semantic Segmentation and Instance Segmentation 7
2.2 A brain analogy of Artificial Neural Network 8
2.3 ANN in detail. 8
2.4 How the convolution operation works. 10
2.5 Two types of Pooling layers 10
2.6 Segmentation Architectures 11
2.7 ResNet motivation 13
2.8 Residual Connection 14
2.9 Depthwise Convolution 14
2.10 Encoders in MobileNet architectures. 15
2.11 Visualization of some nearest neighbors upsampling methods 17
2.12 Transposed Convolution 18

3.1 Pipeline. 24
3.2 Affine Transformation 25
3.3 Grid Distortion and Elastic Transform 26
3.4 Two experiment encoders 27
3.5 Two experiment decoders 29
3.6 Procedure of making a weight mask 30
3.7 Pseudo Labeling 30
3.8 Test-time Augmentation 31
3.9 Some methods to combine the TTA results 32
3.10 An example of the procedure for Active Shape Model 33

3.11 Active Shape Model Iterations 35

4.1 Dataset 37
4.2 IoU Visualization. 38
4.3 Training histories of best models 42
4.4 Three best models comparison 42
4.5 Some good predictions of the best model. 43
4.6 Some bad predictions of the best model. 44
4.7 Some results of active contour model and active shape model. 45
4.8 EF Visualization 46

B.1 Simpson rule for calculating LV volume. 52

List of Tables

1.1 LVEF ranges by gender 3

2.1 TP, FP, FN in a confusion matrix. 20

4.1 Experiment results of color spaces. 39
4.2 Experiment results of augmentation methods. 40
4.3 Experiment results of loss functions. 40
4.4 Experiment results of architectures. 40
4.5 Experiment results of some strategies. 43
4.6 Experiment results of postprocessing techniques. 44
4.7 Experiment results of EF calculation. 47
4.8 Statistics of EF calculation (mean ± std). 47

Abbreviations

ANN	Artificial Neural Network
CNN	Convolution Neural Network
LV	Left Ventricle
A2C	Apical Two Chamber
A3C	Apical Three Chamber
A4C	Apical Four Chamber
EF	Ejection Fraction
2D	Two-dimensional
DL	Deep Learning
ReLU	Rectified Linear Unit
ACM	Active Contour Model
ASM	Active Shape Model
DT	Distance Transform
TTA	Test-time Augmentation
GD	Gradient Descent
IoU	Intersection-over-Union
BCE	Binary Cross-entropy
PCA	Principal component analysis
EC	Encoder
DC	Decoder

Chapter 1

Introduction

1.1 Motivation

Segmentation of the left ventricle (LV) in two-dimensional (2D) echocardiography is an essential step for calculating Ejection Fraction (EF), a measurement that assists doctors in diagnosing heart problems. In practice, this segmentation task is performed manually by medical experts or automatically with the help of expensive specialized devices that are only available in large hospitals. This raises a need for an alternative method that is cheap, automatic and reliable to be used in local clinics. With considerable developments in the field of deep learning, which has shown exceptional performances on fundamental computer vision problems and far surpassed traditional machine learning and image processing methods, this thesis aims to explore how deep learning approaches can be applied to this task. This section will introduce some basic concepts to understand the medical aspect of the problem.

Echocardiography Ultrasonography is a technique that makes use of high-frequency sound waves bouncing off internal structures to create a moving image [1]. The ultrasonography of the heart is called **echocardiography** (also known as *echocardiogram* or *echo*). It is a test that takes pictures of the heart's components such as chambers, walls, valves and attachments such as the blood vessels (arteries, aorta, veins) [2]. As a noninvasive, painless and widely available method that provides images of high quality with a reasonable price, it is a popular choice for diagnosing heart problems like high blood pressure, leaky heart valves, heart failure or even blood clots and tumors [3].

2D echocardiography Main types of echocardiography are *transthoracic echocardiography*, *stress echocardiography*, *transesophageal echocardiography*, and *three- di-*

[1] https://www.msdmanuals.com/home/heart-and-blood-vessel-disorders/diagnosis-of-heart-and-blood-vessel-disorders/echocardiography-and-other-ultrasound-procedures

[2] https://www.heart.org/en/health-topics/heart-attack/diagnosing-a-heart-attack/echocardiogram-echo

[3] https://www.nhlbi.nih.gov/health-topics/echocardiography

mensional echocardiography. The subject of this thesis is **two-dimensional (2D) images** in *transthoracic echocardiography*, the most common type of echocardiogram test. This test involves the use of a device called a transducer which is placed on the chest or abdomen to get various views of the heart. Three typical views will be experimented: apical four chamber (A4C), apical three chamber (A3C) and apical two chamber (A2C). Those views are useful to when it comes to the quantitation of left ventricular volumes, ejection fraction, four segmental wall motions, and wall thickness [1].

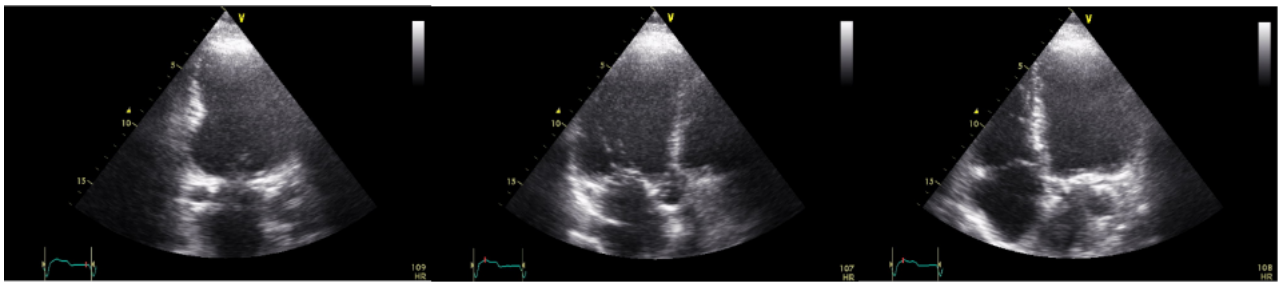


Figure 1.1: Three typical views obtained during a routine TTE. From left to right: Apical two chamber (A2C), Apical three chamber (A3C) and Apical four chamber (A4C).

Left Ventricle The left ventricle (LV) is a large chamber toward the bottom of the heart, located below the left atrium and separated from it by the mitral valve (see figure 1.2 for a positional visualization; note that these images are upside-down views of the heart). When the heart shrinks, blood moves back into the left atrium, and then through the mitral valve and later enters the left ventricle. The left ventricle is the thickest of the heart's chambers and holds responsibility for pumping blood enriched with oxygen to tissues all over the body [4].

An echocardiogram can be used to assess left ventricular function measurements, quantifying the efficiency of the left ventricle to pump blood through the body in each heartbeat. One of the parameters for quantification of left ventricular function (LVF) which can be obtained from 2D (transthoracic) echocardiography is Ejection Fraction. *Left Ventricular Ejection Fraction* (LVEF) is the central measure of left ventricular systolic function. If the volumes of blood left in a ventricle at the end of diastole and of systole are the end-diastolic volume (EDV) and the end-systolic volume (ESV) respectively, and the difference between these two is stroke volume, then EF is the fraction of stroke volume in relation to the end-diastolic volume:

$$EF(\%) = \frac{EDV - ESV}{EDV} \times 100 \tag{1.1}$$

Ranges for assessing two-dimensional echocardiography obtained LVEF as per American Society of Echocardiography and the European Association of Cardiovascular Imaging are given in Table 1.1.

[4] https://www.healthline.com/human-body-maps/left-ventricle

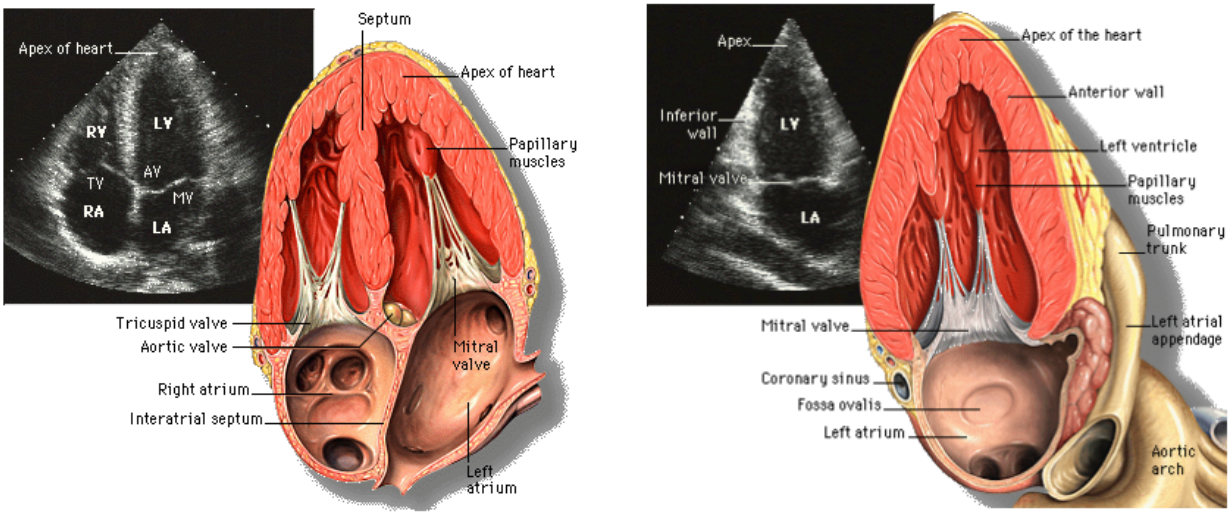


Figure 1.2: Visualization of the heart from A4C (right) and A2C (left) view.

Gender \ Range (%)	Severely abnormal	Moderately abnormal	Mildly abnormal	Normal
Male	<30	30-40	41 - 51	52-72
Female	<30	30-40	41 - 53	54-74

Table 1.1: LVEF ranges by gender.

Segmentation of the Left Ventricle in 2D Echocardiogram To calculate the volume of the left ventricle from a 2D echocardiogram, the first step is detecting the LV region. This detection process, where the boundary of LV is drawn, is called *left ventricle segmentation*. In practice, the expert will run a result video frame by frame from an echocardiogram and choose the frame that corresponds to the systolic and diastolic moment. Then there will be an automatic software that calculates the EF by some methods (such as the modified Simpson method). This is a time-consuming and expertise-required process. There are many methods that aim to automatize this process, which includes traditional and modern deep learning approaches, but they do not tackles the problem this thesis is facing. To the best of my knowledge, this is the first intense experimental conduction that solves the described problem.

1.2 Related Work

Automatic (and semi-automatic) segmentation of LV is a well-studied problem in the literature of medical image analysis. This section will briefly discuss two main approaches: image processing-based and deep learning-based.

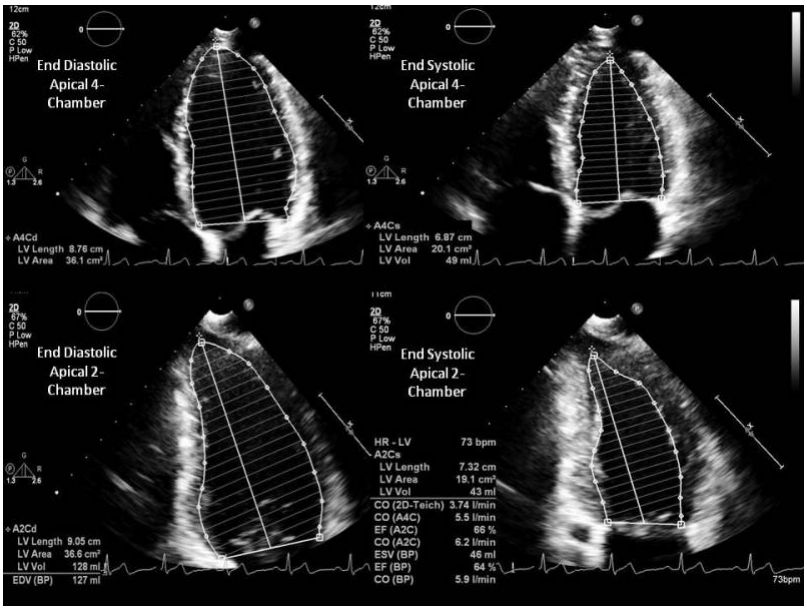


Figure 1.3: Biplane Simpson method to calculate LVEF.

Image Processing Approaches

Before the dawn of deep learning, there are many proposed methods for 2D left ventricle segmentation, such as active contours, level sets, active shape models and Kalman filter. A thorough review of methods belong to this approach can be referred to Noble and Boukerroui's survey [5].

Deep Learning Approaches

Recent research using deep convolutional neural networks for segmentation of left ventricles in 2D echocardiogram have shown very promising results. Though deep learning is a powerful framework, to make a deep learning model efficiently working is a challenging task, especially in medical image analysis, as this approach requires a large amount of (usually annotated) data. For the LV segmentation problem, this means that experts have to draw the LV border in potentially thousands of images, which is time-consuming and extremely tedious. Smistad et al. [32] adopted an approach that tackled the data-hungry problem of deep learning models [23], where an Unet-inspired convolution neural network (CNN) was pretrained on the "pseudo" data generated from other segmentation methods such as Kalman filter. From experiments, they observed that CNN is able to reach comparable accuracy to that of the automatic method, by only training with artificially generated data. In another research, Smistad et al. also proposed a framework that uses a neural network to identify and separate eight cardiac views and another segmentation network adopted from Unet architecture [15] that was trained to classify pixels in 2D echocardiogram images into four categories (background, left ventricle, myocardium and left atrium) for each view. Arafati et al. [24] fine-tuned a pretrained FCN-VGG Net [14] on a augmented dataset to segment all four chambers of the heart and evaluated the model using multiple evaluation metrics including Dice similarity coefficient and Hausdorff

distance, showing that automatic cardiac segmentation in noisy 2D echocardiograms can be solved as a semantic segmentation problem using latest deep-learning algorithms.

1.3 Contributions

This thesis investigates how a deep learning (DL) approach can be applied for segmenting LV regions in a relatively new data set of noisy images taken from 2D echocardiograms. Some DL architectures will be examined along with some processing techniques and practical strategies to choose the best pipeline for the given data set. Experiments revealed that a deep learning model with carefully chosen architectures and proper processing methods can produce a segmentation of LV region on a noisy image in a reasonable time with high accuracy, that makes it possible for mobile applications.

Chapter 2

Background

2.1 Problem Formalization

The problem of left ventricle segmentation can be formally defined as follows: Given an image represented by a 3-channel matrix $I = (x_{ij}) \in \mathbb{N}^{H \times W \times 3}$ (where $x_{ij} = (r_{ij}, g_{ij}, b_{ij})$ are values of three color channels: red, green, blue), the goal is classifying each pixel x_{ij} into one of two classes: *LV* (i.e. the pixel is in the left ventricular region) or *not LV*. If $y_{ij} = 1$ then x_{ij} belongs to *LV* and $y_{ij} = 0$ otherwise, the prediction mask is given by the 1-channel matrix $P = (y_{ij})$.

The following sections of this chapter will introduce some fundamental background of relevant knowledge to provide readers with a more thorough understanding of proposed methods.

2.2 Segmentation

Introduction Image segmentation is the process of subdividing an image into its constituent regions or segments (sets of pixels) of similar functional or characteristic properties [6]. The aim of this process is to change the way an image is represented, making it more meaningful and easier to examine. Semantic segmentation, where each pixel within the image is classified to an object category (e.g. background, person, cat, dog, ...), is one of the fundamental problems in the area of computer vision. Another type of segmentation is instance segmentation (see figure 2.1 [19]), where each pixel is identified with an object instance (e.g. two different people are different instances, though they both belong to category "person"), which requires more fine-grained inferences compared to coarse-grained results of semantic segmentation. Accurate and efficient segmentation mechanisms are needed in fundamental phases of many applications such as autonomous driving, video surveillance, and even medical imaging (locating tumors, providing organic measurements or virtual surgery stimulation). As images of interest contain only a single object (LV) within a background, we simply consider the current problem a binary semantic segmenta-

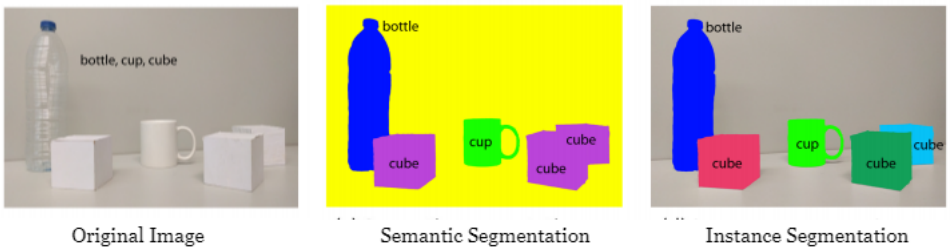


Figure 2.1: Semantic Segmentation and Instance Segmentation

tion task.

Medical Image Segmentation Medical images play a vital role in medical diagnosis and treatment. Over the years, medical image processing has contributed a lot in medical applications; for example, methods of image segmentation, image registration, and image-guided surgery are widely used in surgical procedures [9]. The segmentation of tissue classes, organs, pathologies, or other biologically relevant structures in medical images allows quantitative analysis of clinical parameters related to volume and shape (for example, in cardiac or brain analysis [22]) or simulations based on the extracted boundary information. Some other medical problems demanding segmentation can be listed such as surgical planning, tumor detection, brain development study, etc. [9]. Challenges in medical image segmentation usually come from low contrast, noises, and other imaging ambiguities. With the growth of deep learning, especially in the realm of computer vision, many segmentation problems in the medical field can be tackled using deep architectures, usually Convolutional Neural Networks (CNN), which are exceeding other approaches by a huge margin in terms of precision and sometimes even efficiency.

2.3 Deep Learning

2.3.1 Artificial neural network

An *artificial neural network (ANN)* composes of connected units or nodes called artificial neurons that are connected to each other in some specified way. The field of neural networks has originally been motivated by the goal of modeling biological neural systems (depicted in figure 2.2 [1]), but has since diverged and become a matter of engineering and obtaining good results in machine learning tasks [2]. There are various types of neural networks, from the simple feedforward neural network to more complex recurrent neural network which models temporal dynamic behavior or convolutional neural network used to analyze visual imagery.

A basic neural network comprises layers, which in turn comprises neurons (figure 2.3a). Each neuron in a layer will be connected to all neurons of the former layer

[1] Image taken from http://cs231n.github.io/neural-networks-1/
[2] http://cs231n.github.io/neural-networks-1/

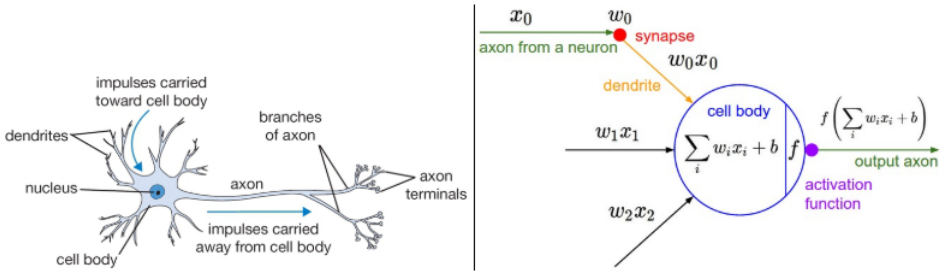
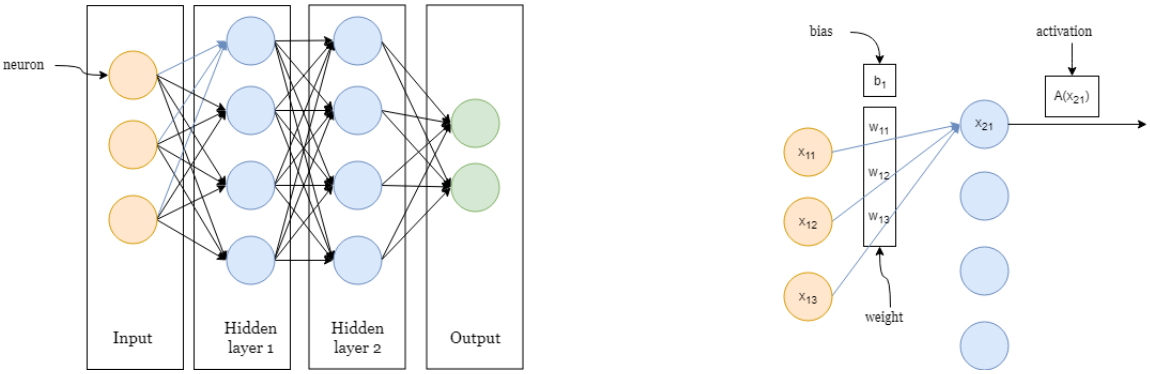


Figure 2.2: A brain analogy of Artificial Neural Network



(a) A simple ANN

(b) Connection between neurons

Figure 2.3: ANN in detail.

by a linear combination operation:

$$x_{i,1} = w_{i-1,1}x_{i-1,1} + w_{i-1,2}x_{i-1,2} + \cdots + w_{i-1,n}x_{i-1,n} + b_{i-1} \tag{2.1}$$

where $x_{i,1}$ is the first neuron of i-th layer, terms $w_{i-1,j}$, for $j = 1..n$ are called weights and b_{i-1} is the bias term. The latter two are trainable parameters of the network, which can be updated to make the network perform better in mapping input to output.

After each neuron in hidden layers, there is an activation function applied to the value of that neuron. As the combination of weights and neurons from the previous layer is just a linear operation, the activation can bring a non-linear mapping to the network, therefore makes the network more complicated and consequently be able to learn complicated structures from the data. Some popular activation functions are:

- Sigmoid function: $f(x) = \frac{1}{1+e^{-x}}$, which is a nonlinear operation mapping a real value to another real value between $(0,1)$.
- ReLU (Rectified Linear Unit): $f(x) = \max(x,0)$, which is also a nonlinear mapping. Because ReLU is computationally inexpensive and is able to make the network sparse (as negative values are mapped to 0), it is widely used in deep networks.

Note that there exist linear activation functions, such as the identity mapping $f(x) = x$, but using this type of activations inside a neural network is pointless, as a

series of layers with linear activations is simply equivalent to a linear combination, which can be represented by a single layer.

Deep learning usually refers to the usage of deep neural networks (which is artificial neural networks with multiple layers between the input and output layers). It allows models to learn representations of data with multiple levels of abstraction (this is one reason why deep learning is powerful when it comes to analysis images, which has a hierarchical structure, e.g. faces are made up of eyes, which are made up of edges, etc.).

2.3.2 Convolutional neural network

A *convolutional neural network* (CNN, or ConvNet) is a class of deep neural networks, where each layer is a three-dimensional representation of neurons, and each neuron from the current layer has a spatial connection to a region of neurons in the previous layer. CNN is very similar to ordinary Neural Networks as they compose of neurons that have learnable weights and biases; the difference is that CNN architectures explicitly assume that the inputs are images, which allows us to encode certain properties into the architecture, thus help the forward function more efficient to implement and greatly reduce the number of parameters in the network. Each layer of a convolutional neural network has neurons arranged in 3 dimensions: width, height, depth. There are three main types of layers, they are Convolutional Layer, Pooling Layer, and Fully-Connected Layer. For simplicity, the following part will describe the mechanism of those layers with the input of depth 1, but it can easily generalized into any depth.

- **Convolutional Layer** This layer's mechanism is based on the convolution operation in the signal processing, which is defined over two one-dimensional discrete functions $f[x]$ and $g[x]$ as:

$$(f*g)[x] = \sum_{m=-\infty}^{\infty} f[m]g[x-m] \tag{2.2}$$

  or in 2D case:

$$(f*g)[x,y] = \sum_{m=-\infty}^{\infty}\sum_{n=-\infty}^{\infty} f[x,y]g[x-m,y-n] \tag{2.3}$$

  Figure 2.4 shows a visualization of how a convolution operation works on a 2D image.

- **Pooling Layer** This layer's function is to progressively reduce the spatial size of the representation to reduce the number of parameters and computation in the network, and hence to also control overfitting. Some widely used pooling layers are:

  – Max Pooling: As figure 2.5 implies, the max pooling operating over a region will return the max element of that region. For example, a filter over the

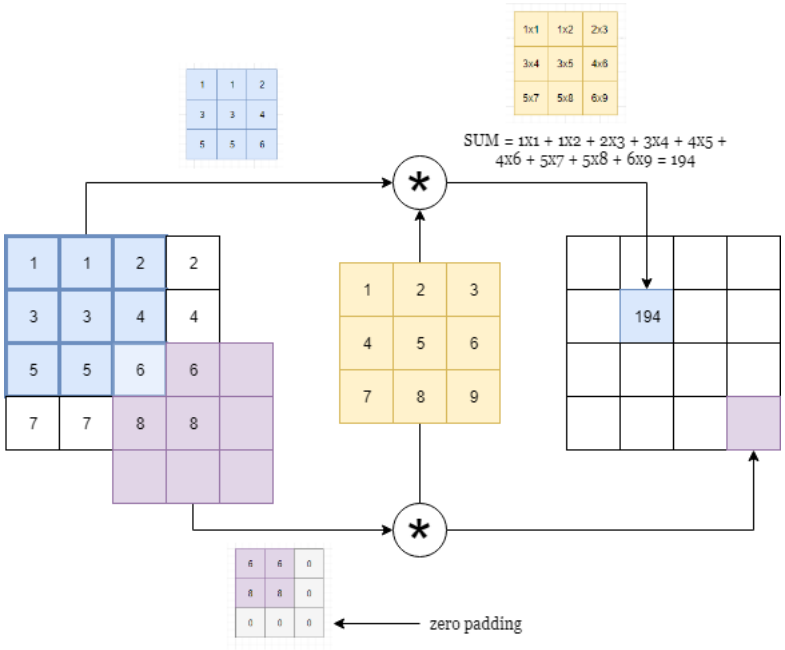


Figure 2.4: How the convolution operation works.

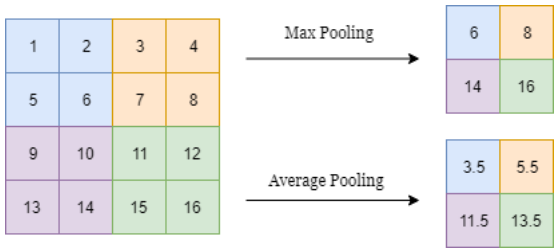


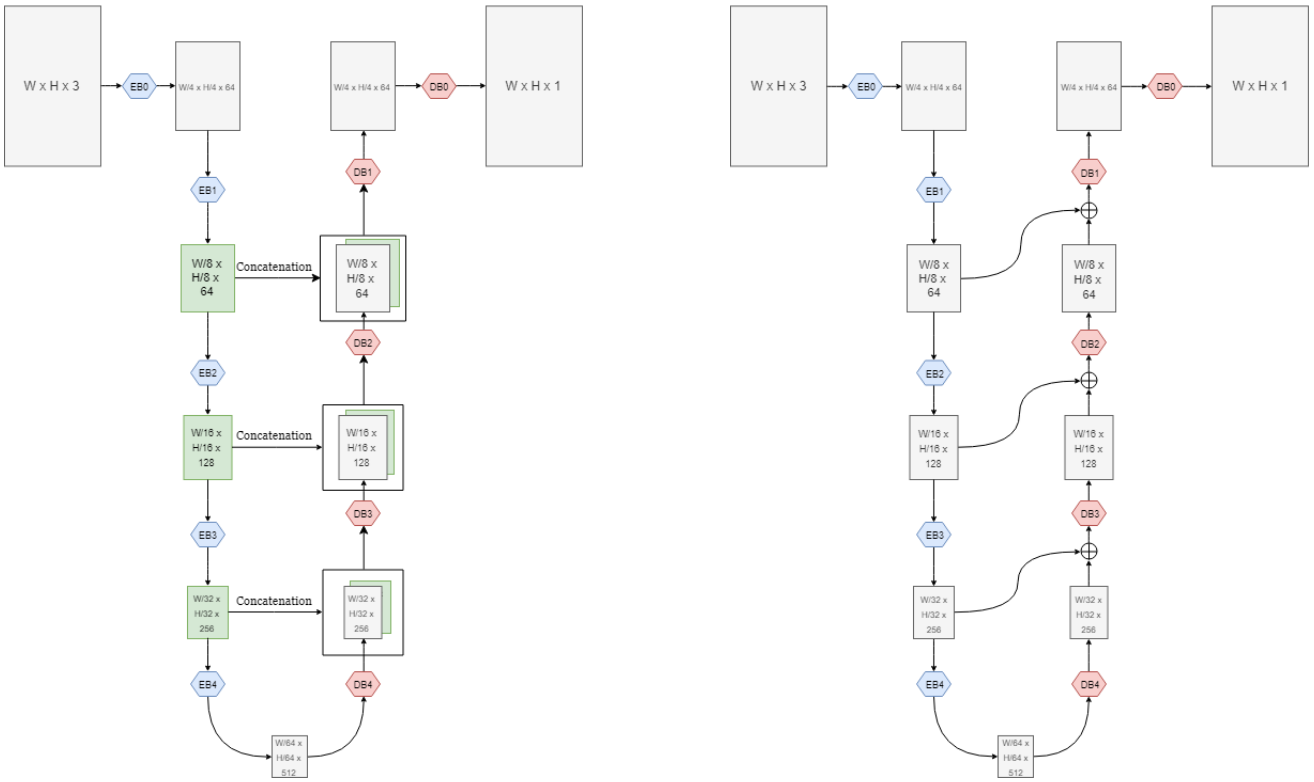
Figure 2.5: Two types of Pooling layers

region of $(1,2,5,6)$ will have the output being $6 = \max(1,2,5,6)$.

– Average Pooling This layer functions similarly to Max Pooling layer apart from the max operator which is replaced by the average operator. For instance, $3.5 = \frac{1+3+5+6}{4}$ is the output of filtering through $(1,2,5,6)$.

As we can infer from mechanisms described of two pooling layers, the input going through this layer will only be reduced in height and width size when keeping the same the number of channels.

- **Fully-Connected Layers** Those layers are similar to hidden layers in basic ANN architecture, where a neurons is connected to all the neurons in the previous layer.



(a) Unet (b) Linknet

Figure 2.6: Segmentation Architectures

2.4 Deep CNN Architectures for Segmentation

2.4.1 Unet

Unet [15] employs an encoder-decoder architecture, in which there are blocks of layers that encode the input in different feature space, from dense (which is related to high resolution and high-level feature) to coarse (which is related to low resolution and low-level feature). Letter "U" in Unet comes from the “U” shape given by an encoding path (“contracting”) paired with a decoding path (“expanding”) in the architecture. There are skip-connections that concatenate feature maps from each level of the contracting path over with feature maps precedeing the analogous level in the expanding path, serving as a mean for information passing which supports decoder path in the upsampling process and capturing fine features. They also allow the network to examine features at various scales and complexities to make its decision. In the paper, Unet consisting of 4 main encoders and 4 corresponding main decoders, each downsamples and upsamples input by a factor of 2, while simultaneously doubles and halves the number of channels, respectively. The high-level architecture of Unet is depicted in figure 2.6a.

2.4.2 Linknet

Also based on the encoder-decoder architecture similar to Unet, the difference between Linknet and Unet is that it reduces the number of parameters by replacing concatenation with adding operation. Similar to Unet, the input of each encoder layer is bypassed to the output of its corresponding decoder. This will help to recover lost spatial information that can be used by the decoder and its upsampling operations. In addition, since the decoder is sharing knowledge learned by the encoder at every layer, the decoder can use fewer parameters [26]. For a visualization of Linknet architecture, see figure 2.6b.

2.5 Encoders and Decoders

The need for encoders and decoders arise from the importance of representations in machine learning (i.e. how features are represented makes a substantial difference in the performance of our learning algorithms). While in the past most features are hand-crafted (which also called feature engineering, where designer explicitly chose features and their representation), the present witnesses the rise of feature learning methods, where an algorithm is able to learn the feature representation itself. Deep learning uses a deep neural network to achieve this learning task. Specifically, the first few layers of a neural network extract features from the data, or in other words map raw inputs to useful feature representations; and the next few layers combine these features to produce the desired output.

Some network architectures are specially designed to leverage this ability of neural networks to learn efficient representations. One of those is encoder-decoder network, where an encoder network is used to map raw inputs to feature representations, and a decoder network will receive this feature representation as input, learn how to map this input back to the original representation, thus further understand the meaning of the feature mapping. Unet and Linknet introduced above are two examples of this architecture type.

This section presents some encoder and decoder blocks that are in active research in academia and also widely used in practice.

2.5.1 Encoders

A simple encoder

A simple encoder that maps an input onto a lower-dimensional feature space can be a achieve with a normal convolution layer with stride greater than 1. While simple as it sounds, this naive approach has many problems such as being hard to effectively design, difficult to optimize (which is the problem that ResNet solves) and the number of parameters is scaled considerably with the depth of the networks (which is tackled by depthwise convolution blocks of MobileNetV2).

Residual block

Residual block is a component in the famous ResNet [12], where a skip connection is introduced to tackle the problem of vanishing gradient when training deep neural networks. Vanishing gradient is an optimization problem in deep learning where the gradients become smaller and smaller after several applications of the chain rule and finally become zeroes, meaning that there are weights whose values will never be updated and therefore no learning is being performed.

Although the network depth has been showed of crucial importance in the literature ([10], [16]), experiment pointed out that deeper models are harder to optimize. A typical example is showed in figure 2.7 [12], where adding more layer to a suitably deep model leads to higher training and testing error. This seems apparently counter-intuitive because theoretically the deep networks must not be worse than similar shallow networks, for there exists a solution by construction to the deeper model: the first layers are taken from the learned shallower model, and the subsequent layers are identity mappings.

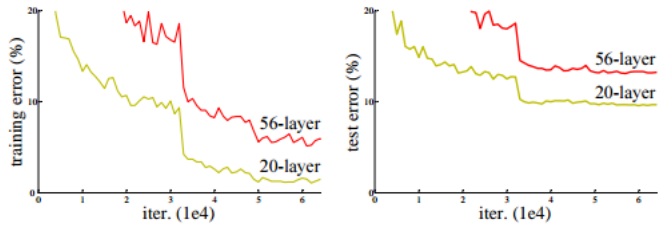


Figure 2.7: ResNet motivation: An example of a deeper network giving higher training error than a shallow network.

ResNet tackles the degradation problem by introducing a deep residual learning framework. In ResNet, an input x, after going through a series of weigh layer mapping it to $\mathcal{F}(x)$, is added with the original x by a shortcut connection (see figure 2.8). A mapping from x to $\mathcal{F}(x)+x$ by this mechanism is called residual mapping. Experimental results showed that the residual mapping is easier to optimize than the original, unreferenced mapping. As a small note, this is a kind of shortcut connections which simply perform *identity mapping*.

Depthwise separable convolution block

This block type (figure 2.9) was first introduced in Xception [17] and used in MobileNetV1 architecture [20], composing of a 3x3 depthwise convolution and a 1x1 pointwise convolution, each followed by a batch normalization [13] and ReLU nonlinearity. More specifically, ReLU6 is used, which is similar the popular ReLU but it prevents activations from becoming too large:

$$y = \min(\max(0, x), 6) \tag{2.4}$$

The authors of the MobileNet paper found that ReLU6 is more robust than ReLU when using low-precision computation.

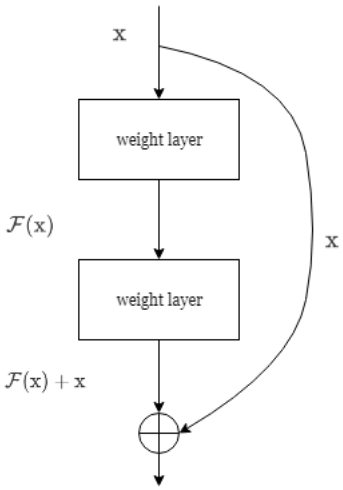


Figure 2.8: Residual Connection.

The goal of the depthwise separable convolution block is to reduce the number of parameters while retaining the complexity of the mapping. It does approximately the same thing as traditional convolution but is much faster because of using fewer parameters (see figure 2.9). To reduce the resolution of input, instead of using pooling layers like in traditional CNN, MobileNetV1 makes use of depthwise layers with stride 2 while simultaneously doubles the number of output channels by pointwise layers. The design of a depthwise separable convolution block is depicted in figure 2.10a.

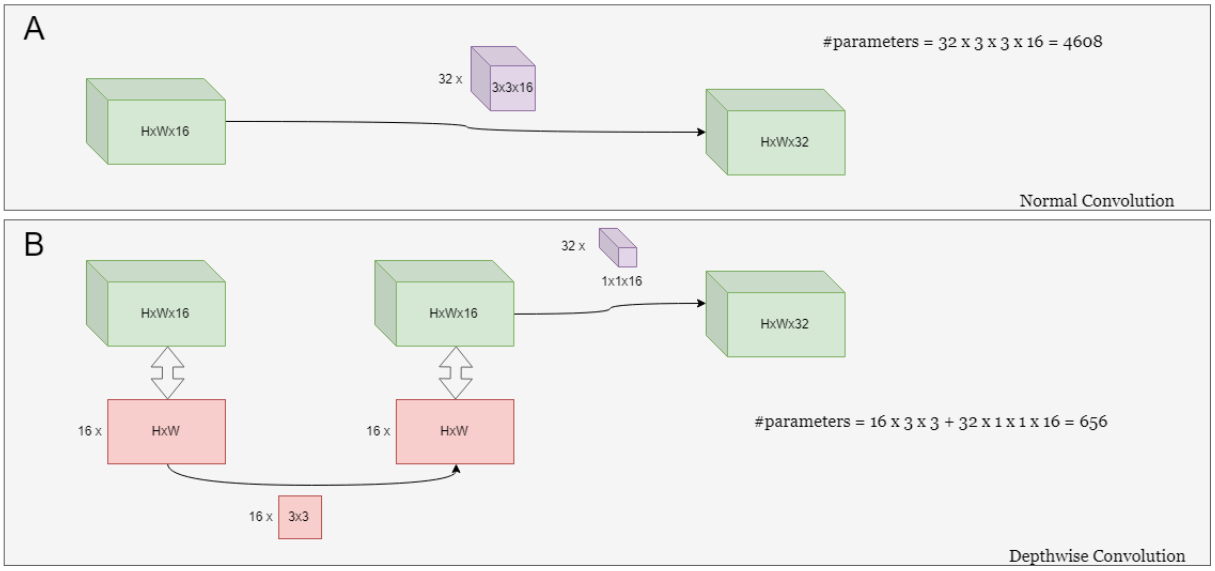
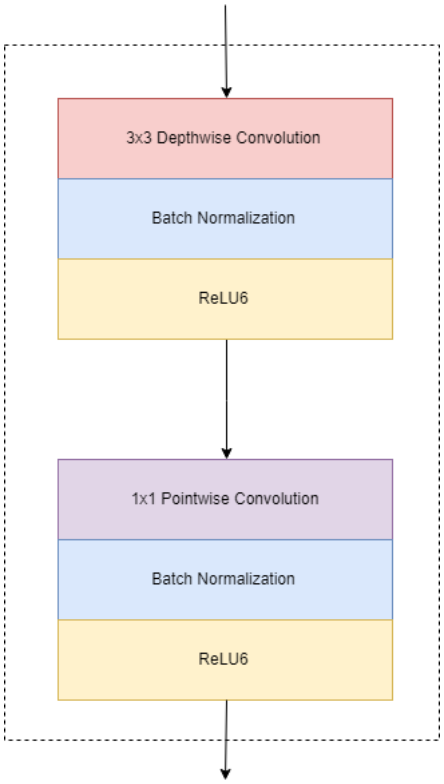


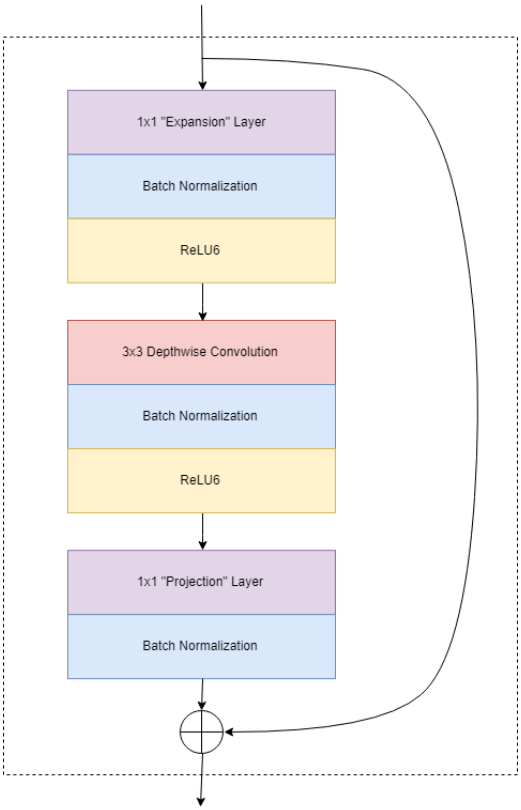
Figure 2.9: **Depthwise Convolution**: Block A is a standard convolution layer with 32 filters of size 3×3 and depth 16. Block B is a depthwise Convolution layer, which consists of 16 3×3 with depth 1 and 32 filters of size 1×1 and depth 16, producing an output of a same size as in block A with much fewer parameters.

Bottleneck residual block.

Using low-dimension tensors is an approach to reducing the number of computations as the smaller the tensor, the fewer multiplications the convolutional layers



(a) **Depthwise Separable Convolution Block**.



(b) **Bottleneck Residual Block**.

Figure 2.10: Encoders in MobileNet architectures.

have to do. Nevertheless, low-dimensional tensors will not be able to extract much information. If the low-dimensional data that passes between the blocks can be considered as being a compressed version of the real data, we can design layers that act like a decompressor restoring data to its original form. Employing this idea, MobileNetV2 [28] introduces Bottleneck Residual Block (figure 2.10b), which consists of three convolutional layers in the block: a depthwise convolution that filters the inputs between two 1x1 pointwise convolution layers - a compressor and a decompressor.

Specifically, this block first takes as input a low-dimensional compressed representation, then passes this representation into a decompressor (1×1 convolution) that expands it to high dimension, filters it by a depthwise convolution, and finally uses a compressor (1×1 convolution) to project it back to the original space of lower dimension. The final step, which maps a low-dimensional tensor to a high-dimensional layer, is called a bottleneck, thus gives the name of this module. How much the data gets expanded is control by a hyperparameter called the *expansion factor*. For example, with an expansion factor of 6, the input tensor of depth 24 will be expanded to a new tensor with $24 \times 6 = 144$ channels. So the input and the output of the block are low-dimensional tensors, while the filtering step that happens inside the block is done on a high-dimensional tensor.

Another new component in this block compared to depthwise Separable Block is the residual connection to help with the flow of gradients through the network. As usual, each layer has batch normalization and the activation function is ReLU6. However, the output of the projection layer does not have an activation function applied to it. As the output of this layer is in low-dimensional space, experiments showed that using a non-linearity subsequent to this layer actually destroyed useful information.

2.5.2 Decoders

A decoder component receives a low-resolution image and returns a high-resolution image. The need for this type of convolution rises from the need for encoder in image analysis: as an encoder projects input into lower-dimensional spaces to capture the hierarchical structure in images, in the segmentation problem (which is formulated as a pixel classification problem), those lower-dimensional spaces need to be converted back to the original resolution. This can be performed by at least two types of operations [3]:

Nearest neighbors upsampling

This algorithm is an interpolation method which, like convolution, performs a mathematical operation on each pixel (and its neighbors) within the image to enlarge the image size. In the simplest case, the value of each pixel is replicated and placed nearby. More complex combinations of neighboring information such as bilinear, cubic, and Lanczos interpolation can produce smoother and realistic upsampled out-

[3] https://www.intel.ai/biomedical-image-segmentation-u-net/

put. Figure 2.11 provides visualizations of how nearest neighbors upsampling is done by replicating and by bilinear interpolation.

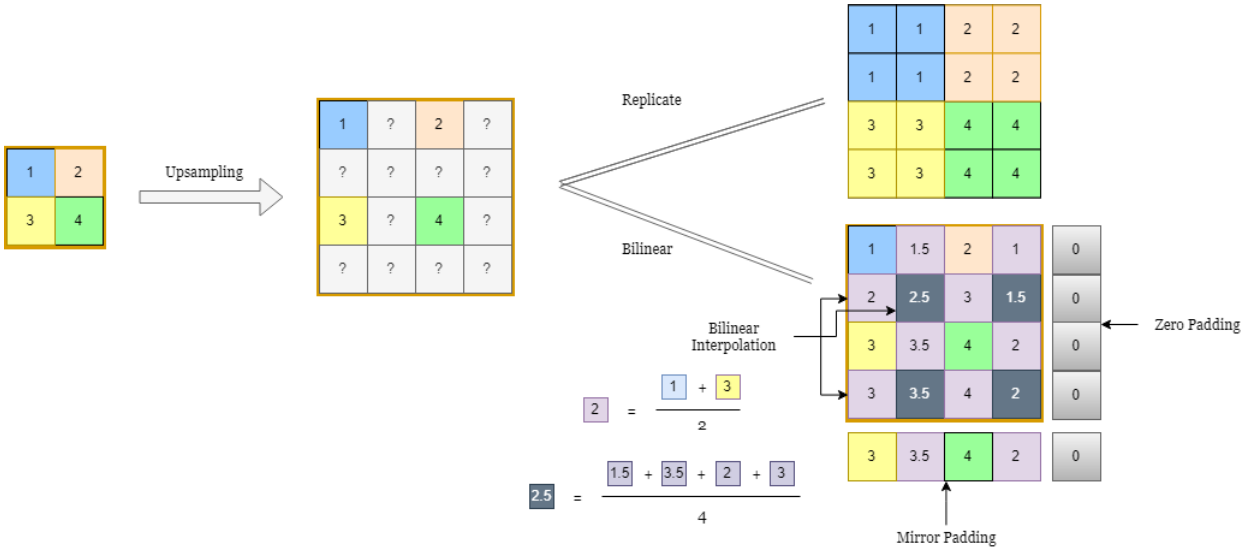


Figure 2.11: Visualization of some nearest neighbors upsampling methods

Transposed convolution

To understand the concept of transposed convolution, first we can see convolution as a matrix operation [18]. Take a simple 1D convolution as an intuitive example, the normal convolution between a 4×1 matrix and a 3×1 filter with stride 1 and padding 1:

$$\overrightarrow{a} * \overrightarrow{x} = \begin{bmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{bmatrix} * \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} ay + bz \\ ax + by + cz \\ bx + cy + dz \\ bx + dy \end{bmatrix} \tag{2.5}$$

This convolution can be expressed as a sparse matrix multiplication as follows:

$$\mathbf{X} \times \overrightarrow{a} = \begin{bmatrix} x & y & z & 0 & 0 & 0 \\ 0 & x & y & z & 0 & 0 \\ 0 & 0 & x & y & z & 0 \\ 0 & 0 & 0 & x & y & z \end{bmatrix} \times \begin{bmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{bmatrix} = \begin{bmatrix} ay + bz \\ ax + by + cz \\ bx + cy + dz \\ bx + dy \end{bmatrix} \tag{2.6}$$

The **transposed convolution** can be defined as the matrix multiplication of the

transpose of $\mathbf{X}$ with $\overrightarrow{a}$ (note that there is no padding here):

$$\mathbf{X}^T \times \overrightarrow{a} = \begin{bmatrix} x & 0 & 0 & 0 \\ y & x & 0 & 0 \\ z & y & x & 0 \\ 0 & z & y & x \\ 0 & 0 & z & y \\ 0 & 0 & 0 & z \end{bmatrix} \times \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} ax \\ ay + bx \\ az + by + cx \\ bz + cy + dx \\ cz + dy \\ dz \end{bmatrix} \tag{2.7}$$

Look at the result of the above transpose convolution, we can see that it is similar to a regular convolution (with different padding rules). This is due to the trivial stride of 1. But when we increase stride to 2, we can see that the transposed convolution is not longer a normal convolution and it really does upsampling:

$$\mathbf{X} \times \overrightarrow{a} = \begin{bmatrix} x & y & z & 0 & 0 & 0 \\ 0 & 0 & x & y & z & 0 \end{bmatrix} \times \begin{bmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{bmatrix} = \begin{bmatrix} ay + bz \\ bx + cy + dz \end{bmatrix} \tag{2.8}$$

$$\mathbf{X}^T \times \overrightarrow{a} = \begin{bmatrix} x & 0 \\ y & 0 \\ z & x \\ 0 & y \\ 0 & z \\ 0 & 0 \end{bmatrix} \times \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} ax \\ ay \\ az + bx \\ by \\ bz \\ 0 \end{bmatrix} \tag{2.9}$$

Figurative illustration of $2D$ transposed convolution can be seen in figure 2.12.

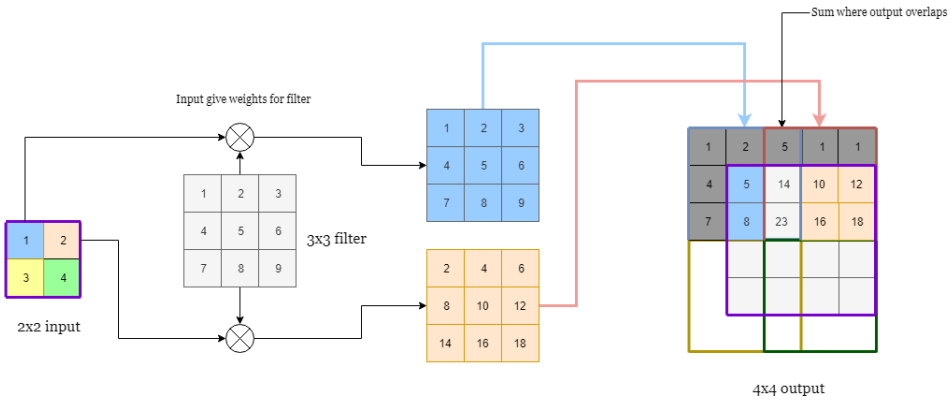


Figure 2.12: Transposed Convolution

2.6 Loss function

2.6.1 Binary cross-entropy

The prediction mask $P = (p_{ij}) \in [0,1]^{H \times W \times C}$ and the groundtruth $T = (t_{ij}) \in [0,1]^{H \times W \times C}$ can be described as matrices of probability distribution, where $p_{ij} = (x_1, x_2, \ldots, x_C)$ and $t_{ij} = (y_1, y_2, \ldots, y_C)$ are the categorical probability distributions of pixel at the position (i, j)). The cross-entropy formula calculates how close those pairs of distribution are to each others:

$$H(p_{ij}, t_{ij}) = \sum_{i=1}^{C} x_i \log y_i \tag{2.10}$$

In the binary case, the formula is:

$$\mathscr{L}_{BCE} = \sum_{\substack{1 \le i \le H \\ 1 \le j \le W}} -t_{ij} \log(p_{ij}) - (1 - t_{ij}) \log(1 - p_{ij}) \tag{2.11}$$

where $p_{ij} \in [0,1]$ is the value (probability of class 1) at position (i, j) in the prediction and $t_{ij} \in \{0,1\}$ is the ground truth at position (i, j).

2.6.2 Jaccard distance

In the segmentation problem, a segmented object can be considered as a set of pixel positions where the object resides. The Jaccard distance, which is calculated from Jaccard index (or Intersection Over Union), measures the dissimilarity between 2 sets:

$$\mathscr{L}_{Jaccard} = 1 - J(P, T) \tag{2.12}$$

where P is a set of predicted pixel positions of an object, T is a set of true pixel positions of that object, and the Jaccard Index $J(P, T)$ is defined as the size of the intersection divided by the size of the union of the two sets:

$$J(P, T) = \frac{P \cap T}{P \cup T} \tag{2.13}$$

When P and T are two binary masks, Jaccard index can be defined using the definition of true positive (TP), false positive (FP), and false negative (FN) as:

$$J(P, T) = \frac{TP}{TP + FP + FN} \tag{2.14}$$

where TP is the number of pixel positions belong to the object in both the predicted mask and the groundtruth, FP is the number of pixel positions belong to the object in the predicted mask but belong to the background in the groundtruth, and FN is the number of pixel positions belong to the object in the groundtruth but belong to the background in the predicted mask (see table 2.1).

In the binary segmentation task, a deep learning model will give the prediction

	$y_{true}(i,j)=1$	$y_{true}(i,j)=0$
$y_{pred}(i,j)=1$	TP	FP
$y_{pred}(i,j)=0$	FN	TN

Table 2.1: TP, FP, FN in a confusion matrix.

in form of a matrix $P=(p_{ij})\in[0,1]^{H\times W}$ whose each pixel is a value p_{ij} of the probability for that pixel belonging to the object. On the other hand, the groundtruth is a matrix $T=(t_{ij})\in\{0,1\}^{H\times W}$ of the same size in which each element is 0 or 1. The formula for Jaccard Index between these two matrices is:

$$J(P,T)=\frac{1}{H\times W}\sum_{i,j}\frac{p_{ij}\times t_{ij}}{p_{ij}+t_{ij}-p_{ij}\times t_{ij}} \tag{2.15}$$

This equation is derived from the formula 2.15 with below specifications:

$$TP_{ij}=p_{ij}\times t_{ij} \tag{2.16}$$

$$FP_{ij}=p_{ij}-p_{ij}\times t_{ij} \tag{2.17}$$

$$FN_{ij}=t_{ij}-p_{ij}\times t_{ij} \tag{2.18}$$

Using the formula 2.15 makes the loss differentiable, which is a requirement to perform optimization techniques such as Stochastic Gradient Descent.

2.6.3 Dice coefficient

This coefficient is similar in form to the Jaccard index.

$$\mathscr{L}_{Dice}=1-D(P,T) \tag{2.19}$$

in which

$$D(P,T)=\frac{2|P\cap T|}{|P|+|T|} \tag{2.20}$$

When applied to boolean matrices, using the definition of TP, FP, FN in section 2.6.2, equation 2.20 can be written as:

$$D(P,T)=\frac{2TP}{2TP+FP+FN} \tag{2.21}$$

To make $D(P,T)$ differentiable, the following formula will be used:

$$D(P,T)=\frac{1}{H\times W}\sum_{i,j}\frac{2\times TP_{ij}}{2\times TP_{ij}+FP_{ij}+FN_{ij}} \tag{2.22}$$

where

$$TP_{ij}=p_{ij}\times t_{ij} \tag{2.23}$$

$$FP_{ij} = p_{ij} - p_{ij} \times t_{ij} \tag{2.24}$$

$$FN_{ij} = t_{ij} - p_{ij} \times t_{ij} \tag{2.25}$$

2.6.4 Combined losses

As being characteristically different measures for same goal, BCE and Jaccard distance (or Dice coefficient) can be combined into a single loss in order to tackle the characteristic variability of objects:

$$\mathscr{L}_{BCE+Jaccard} = \alpha \mathscr{L}_{BCE} + \mathscr{L}_{Jaccard} \tag{2.26}$$

or similarly

$$\mathscr{L}_{BCE+Dice} = \alpha \mathscr{L}_{BCE} + \mathscr{L}_{Dice} \tag{2.27}$$

where α is a weight of importance assigned to BCE loss.

2.7 Active contour model

Active contour model [2] is a method to fit open or closed outlines (called snakes) to lines or edges in an image. A snake is an energy minimizing, deformable spline influenced by constraint and image forces that pull it towards object contours and internal forces that resist deformation.

Formally, a snake (or contour) is defined be a set of n points $V = \{v_1, v_2, \ldots, v_n\}$, where $v_i = (x_i, y_i)$ for $i = 1..n$. The energy function of the snake is the total sum of its internal energy,and external energy:

$$E_{snake} = \sum_{i=1}^{n} (E_{internal}(v_i) + E_{external}(v_i)) \tag{2.28}$$

Then fitting the snake (or contour) onto the image is equivalent to the optimization problem of finding $V*$:

$$V* = \underset{V=\{v_1,\ldots,v_n\}}{\arg\min}\ E_{snake} \tag{2.29}$$

Now we will look in detail two principal components of the energy: internal energy and external energy. The internal energy is a term that only depends on the shape of the snake. It composed of 2 energy parts, i.e. continuity energy which keeps the snake from stretching or contracting along its length and curvature energy which keeps the snake from bending:

$$E_{internal} = E_{cont} + E_{curv} \tag{2.30}$$

where E_{cont} and E_{curv} are defined based on the first and the second derivative respectively:

$$E_{cont} = \alpha \sum_{i} |v_{i+1} - v_i|^2 \tag{2.31}$$

$$E_{curv} = \beta \sum_i |v_{i+1} - 2v_i + v_{i-1}|^2 \tag{2.32}$$

In contrast, the external energy is defined based image features, where strong features have low energy and weak features (or no features) have high energy. Specifically, it consists of three terms, that are E_{line} corresponding to lines in the images, E_{edge} sensitive to edges and E_{term} relating to line terminations and corners:

$$E_{external} = w_{line}E_{line} + w_{edge}E_{edge} + w_{term}E_{term} \tag{2.33}$$

in which $E_{line} = I(v_i)$ is the intensity of pixels in the image, $E_{edge} = -|\nabla I(v_i)|^2$ is the gradient, and E_{term} is computed from curvature of level lines in a slightly smoothed image:

$$E_{term} = \frac{C_{yy}C_x^2 - 2C_{xy}C_xC_y + C_{xx}C_y^2}{(C_x^2 + C_y^2)^{\frac{3}{2}}} \tag{2.34}$$

where $C(x,y) = G_\sigma * I(x,y)$ is the image smoothed by a Gaussian kernel.

In active contour model, there are hyperparameters such as α, β, w_{line}, w_{edge}, w_{term} to control the affect of each energy term on the fitting of the snake.

Chapter 3

Methods

3.1 Overall pipeline

The overall pipeline of conducting experiments in a deep learning manner for the LV segmentation problem in noisy 2D echocardiography is described in figure 3.1. First, annotated data will be split into a training set and a testing set, each consisting of images and corresponding annotated masks. Next, the images will be converted to a proper color space, and the masks are used to generate a boundary-weight image if necessary. Subsequently, both images and masks will go through an augmenter; they are first resized to a chosen shape depending on the model and then are applied basic affine transformation such as translation, scaling, rotation and flipping or other non-linear transformation such as grid distortion or elastic transform to create new data. This augmentation process is done in real-time in training (i.e. while the data is being fed into the model). The testing set is processed similarly, except for augmentation where its images are only resized, and is used as a validation set to pick up the best model. The final phase is refinement, where the prediction on the test set can be further improved by inference strategies such as test-time augmentation (where predictions are made on differently modified versions of an image, and the final output is a combination of those predictions) or statistical shape methods such as active shape model.

Due to the limitation of computing power, not all preprocessing and postprocessing methods, architectures, training strategies and the combination of those can be tested to decide which will form the best pipeline. As a result, the experiment procedure will be constructed in a bottom-up manner. It means that methods will be examined in a logical order, and after each method, settings that associate with the best result will be chosen and fixed for subsequent trials. For example, if a color space is shown experimentally to give the best model performance, all the following experiments will use that color space. Hence, the methods in this chapter will be structured according to the order of experiment: first, I describe some preprocessing techniques, then move to the architecture, training and inference strategies, and finally post-processing mechanisms. In each section, I introduce those techniques and describe how they were used in the experiments. The detail results will be showed and analyzed in Chapter 4.

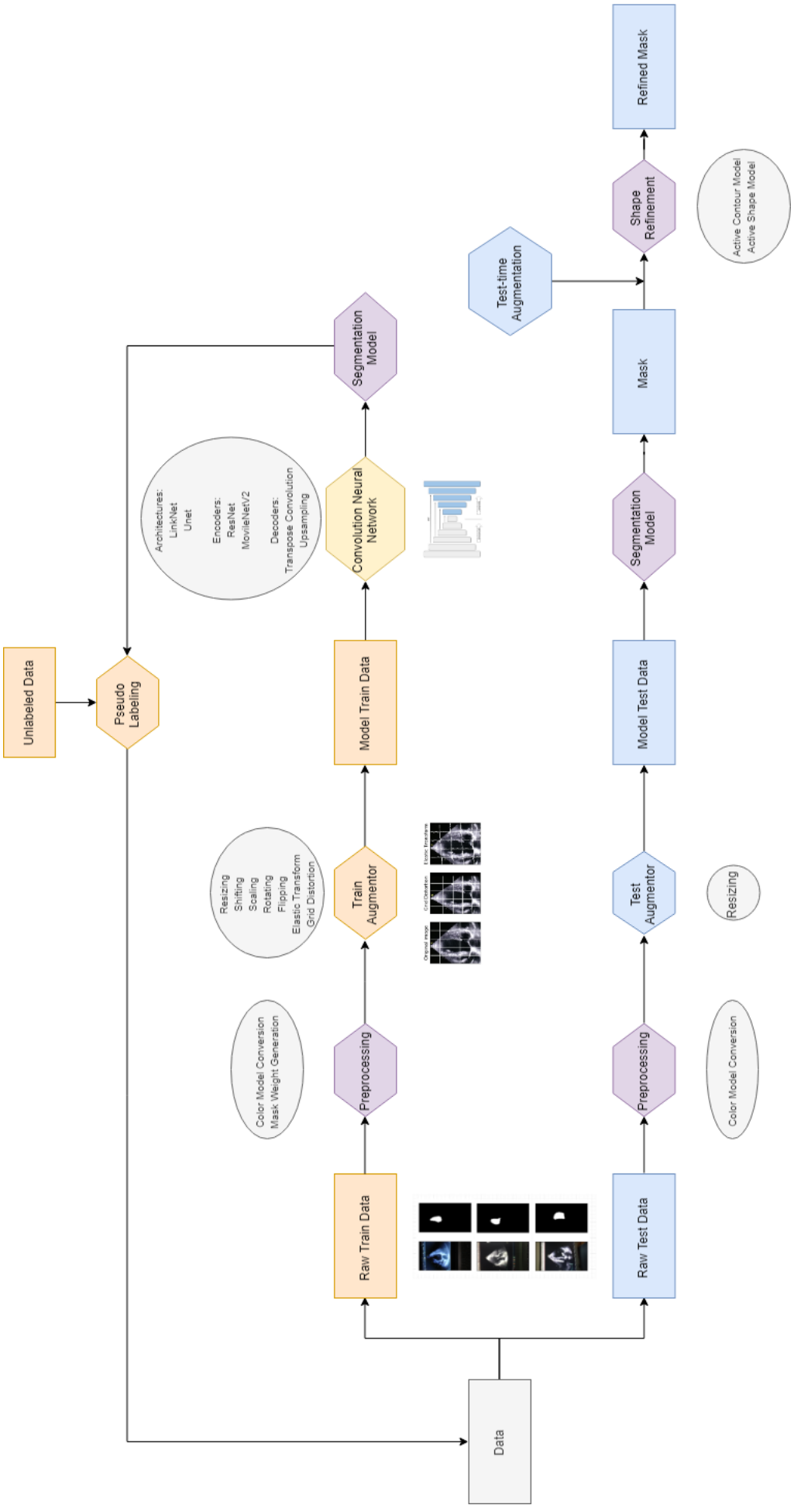


Figure 3.1: Pipeline.

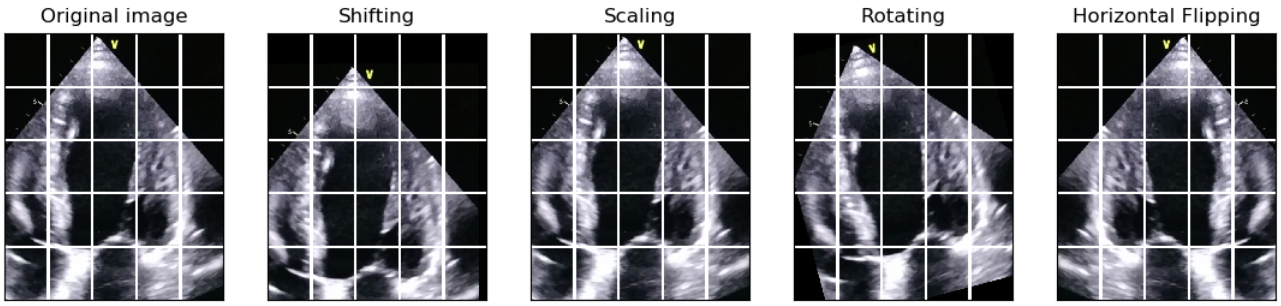


Figure 3.2: Affine Transformation. White grid lines are added similarly to all images for better visualization.

3.2 Preprocessing

3.2.1 Color spaces

The thesis examines different color spaces for preprocessing images in order to figure what works best in practice for the available dataset. Five color space are considered: *RGB* (red, green, blue), *gray-scale*, *HSV* (hue, saturation, value), *YUV* (Y stands for the luminance component (the brightness) and U and V are the chrominance (color) components), and *YCbCr* (Cb, Cr are the blue-difference and red-difference chroma). Experimental results indicated that *HSV* gives a slightly higher IoU score compared to other color spaces. This can be explained as *HSV* space separates color information from intensity or lighting, so the neural network can learn regardless of lighting changes in the environment. As stated above, from now on, all the data will be converted to *HSV* before further processing.

3.2.2 Data augmentation

Data augmentation is a technique to create new data from the available data. As it is common knowledge that the performance of deep learning models improves in concert with the size of the training dataset, data augmentation is an essential step when adopting a DL approach, especially when the available dataset is small. Besides, the modified versions of the images in the training dataset aid the model in extracting and learning characteristic features and make the model invariant to lighting condition, position, noise, etc., thus provide it with the ability of generalization. Common methods of augmenting data are affine transformations (translation, scale, rotation, horizontal/vertical flipping ...) (see figure 3.2). In addition to these commonly used transforms, operations like grid distortion and elastic transform often can be helpful, see figure 3.3, since medical imaging is often dealing with non-rigid structures that have shape variations. [25].

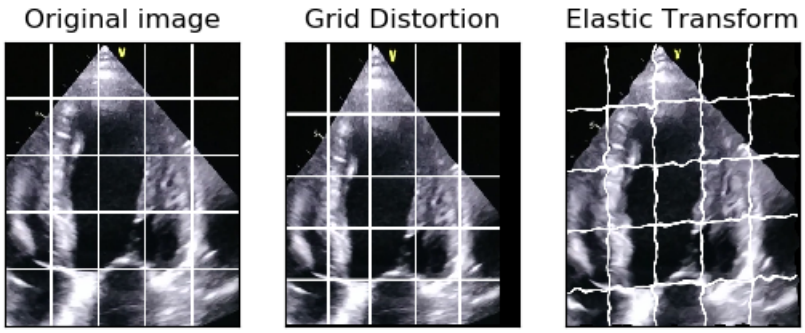


Figure 3.3: Grid Distortion and Elastic Transform. White grid lines are added to the original image and transformed accordingly for better visualization.

3.3 Architectures

3.3.1 Segmentation Architectures

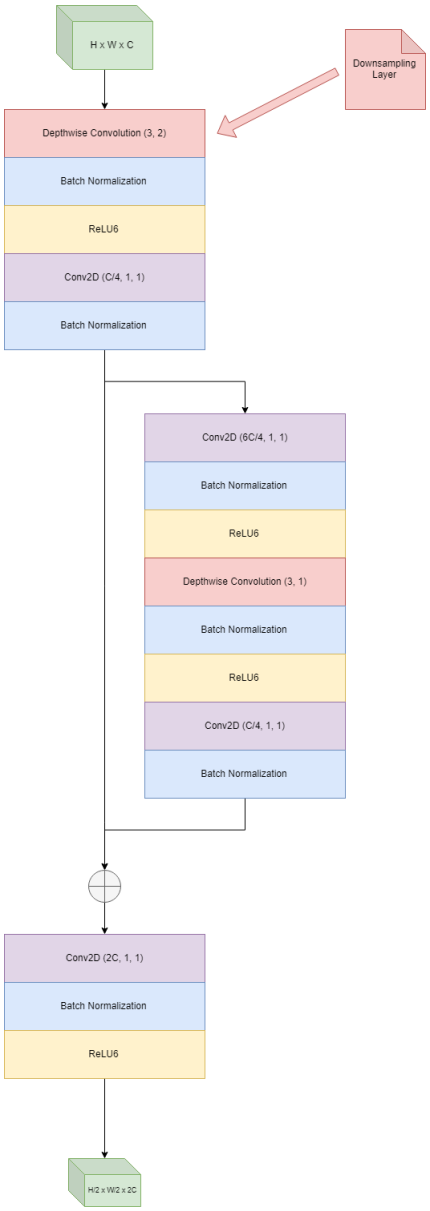
This thesis tested two architectures Unet and Linknet with two different encoder backbones and two different decoder types.

3.3.2 Encoder Backbones

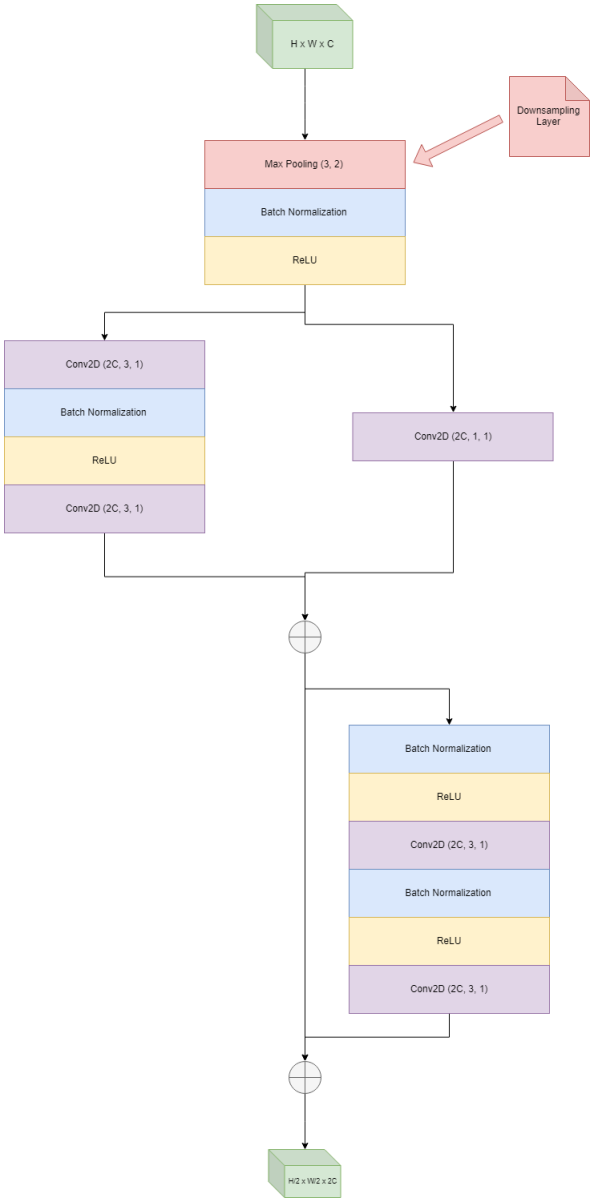
Every sequence of layers that projects an input onto a smaller dimension can be used as an encoder. In this thesis, two types of encoders were experimented, which is based on designs in ResNet and MobileNetV2 architectures. More specifically, we adopt blocks from those two architectures that downsample input to put in our Unet/Linknet encoder positions.

In terms of ResNet (figure 3.4a), a tensor of shape $H \times W \times C$ will be downsampled by a factor of 2 with a max pooling operation. The output tensor will be fed into two blocks, one is two consecutive normal convolution layers and the other is a pointwise convolution. The two output tensor will be added together, and finally, go through a block of 2 normal convolution layers with the residual connection. The final output has the shape of $\frac{H}{2} \times \frac{W}{2} \times 2C$, which means the shape is halved on each side and the channel is doubled.

With regards to MobileNetV2 (figure 3.4b), the first max pooling is replaced by a depthwise convolution with stride 2, followed by a pointwise convolution that projecting the input into the lower dimension of depth $\frac{C}{4}$. After that, the output will go through a bottleneck residual block with expansion factor of 6, and finally be expanded by a 1×1 convolution layer. The final output also have the shape of $\frac{H}{2} \times \frac{W}{2} \times 2C$.



(a) ResNet



(b) MobileNetV2

Figure 3.4: **Two experiment encoders**. Conv2D(c, k, s) represents a 2D convolution with c filters of $k \times k$ with stride s. Depthwise Convolution(k, s) denotes the use of $k \times k$ filters with stride s.

3.3.3 Decoder

The decoder blocks used in experiments are similar to ones proposed in Linknet: it consists of a transposed convolution (or upsampling) layer between two pointwise convolution layers (this design resembles bottleneck design of block in MobileNetV2 but in the opposite). Specifically, if the input tensor is of shape $H \times W \times C$, first it is projected to lower-dimensional space of $\frac{C}{4}$ by a pointwise convolution, then doubly upsampled to shape $2H \times 2W$ by a transposed convolution layer or a upsampling layer followed by a normal convolution, and finally expanded to higher-dimensional of $\frac{C}{2}$ by another pointwise convolution (figure 3.5). Hence there is only one setting here, which are the choice of how the input is upsampled, by nearest neighbor methods (Upsampling2D) or by transposed convolution (TransposedConv2D).

3.4 Boundary-weighted loss function

Segmentation in medical imaging requires a high accuracy (especially in the boundary region) in order to be practical. To address this problem, a weight mask is created for each ground truth mask which places greater importance to pixels at the boundary, leading to the model forced to learn to accurately classify each boundary pixel. This is a preprocessing applied to masks, based on distance transform.

Formally, given a binary mask $M = (m_{ij})$, the weight mask $W = (w_{ij})$ is defined as:

$$w_{ij} = \begin{cases} 0 & \text{if } m_{ij} = 0 \\ d_{ij} & \text{if } m_{ij} = 1 \end{cases} \tag{3.1}$$

where d_{ij} is the Euclidean distance from position (i, j) to its nearest boundary pixel location.

A visualization of the procedure is depicted in figure 3.6. First, the contour of the image is estimated. It is represented as a list of points along the boundary of the mask object. Connecting these points with lines of thickness 1 gives a 1-pixel wide boundary of the mask (called it a contour image). Then the distance transform (DT) image corresponding to the mask is calculated based on the contour image. Each pixel of the DT image is the distance (in Euclidean space) from that pixel's location to the location of the nearest non-zero pixel. Finally, the weight mask is given by inverting the DT image, removing the region outside the mask (or the contour).

3.5 Training and Inference Strategy

3.5.1 Pseudo-labeling

A pseudo-labeling [7] is in the realm of semi-supervised learning, in which a model makes use of both labeled and unlabeled data (typically a small amount of labeled data with a large amount of unlabeled data). As annotating data is a time-consuming,

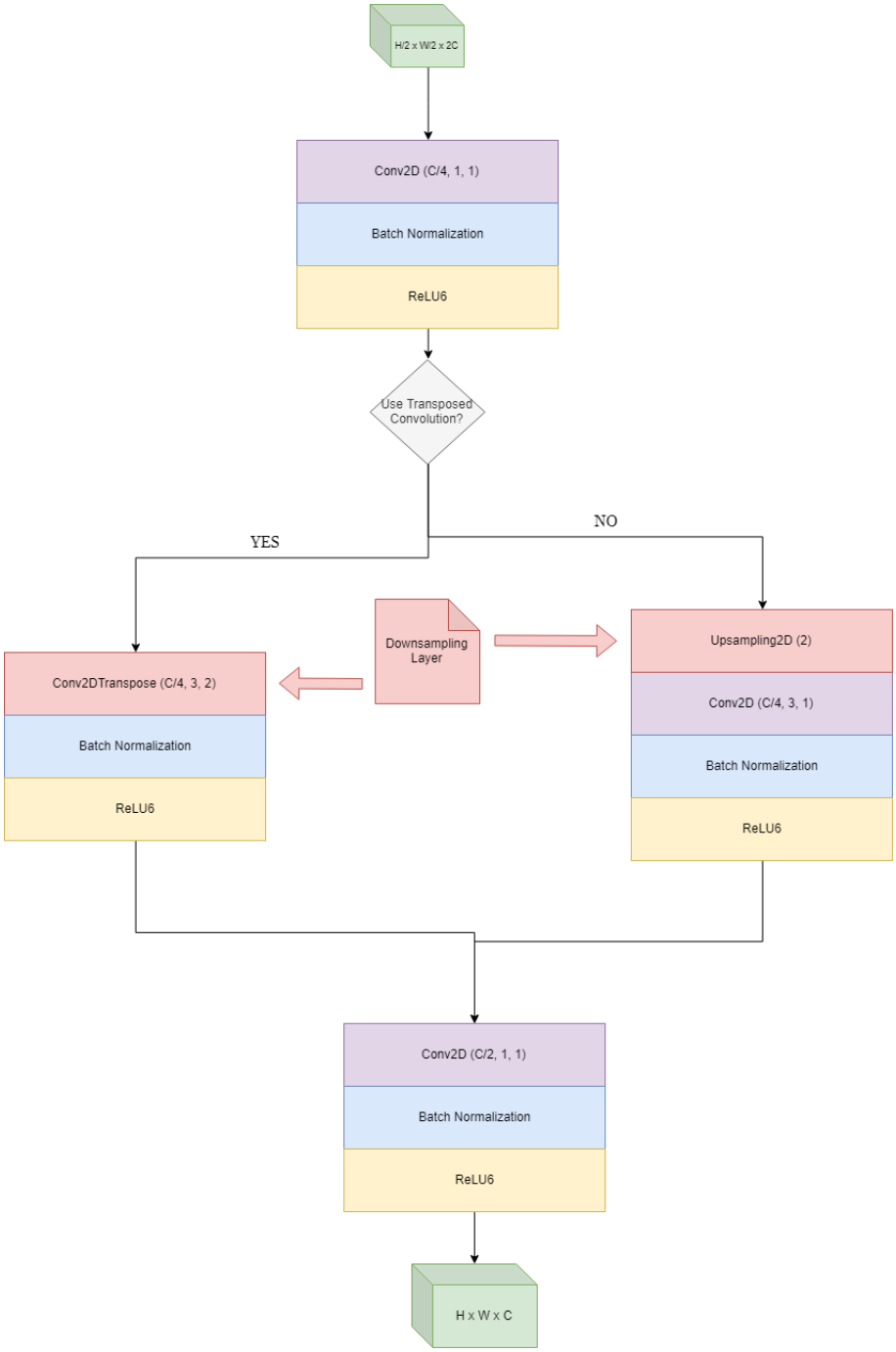


Figure 3.5: Two experiment decoders corresponding to two choices: transposed convolution or upsampling. Note that Upsampling2D(s) means the use of Upsampling method with upsample factor of s.

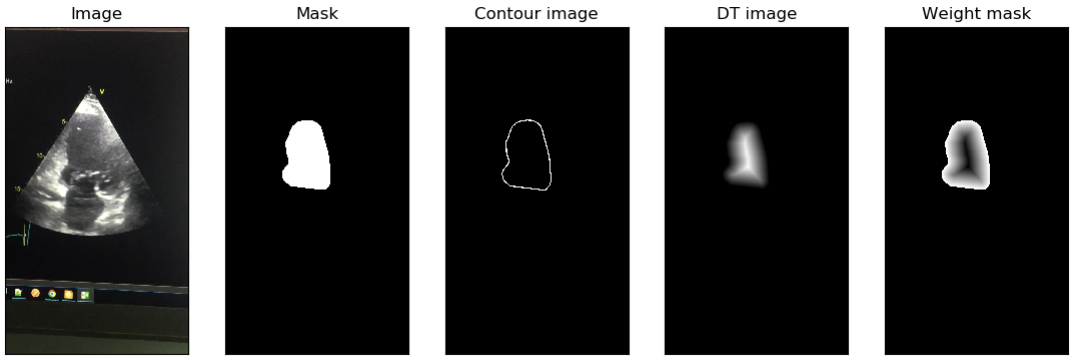


Figure 3.6: **Procedure of making a weight mask**: The contour is extracted from the binary mask to create a contour image, then this image is used to compute distance image and subsequently weigh mask.

laborious and expert-dependent process, there arises a situation where there are a great number of images unannotated. Using annotated data as guiding points, pseudo-labeling takes advantages of those unannotated data to help model learn about the underlying patterns in the images.

Here is how pseudo-labeling works: First, a model is trained on a small amount of labeled data. Next, the trained model is used to predict label (or mask in the segmentation problem) for unlabeled data; those will be called pseudo-labeled data. After that, pseudo-labeled data is filtered by some predefined confidence metrics (as we only want to take data that the model is highly confident about), and merge them with expert-labeled data. Finally, the model is retrained on this dataset, and the procedure is repeated until the model accuracy does not improve or some other criteria.

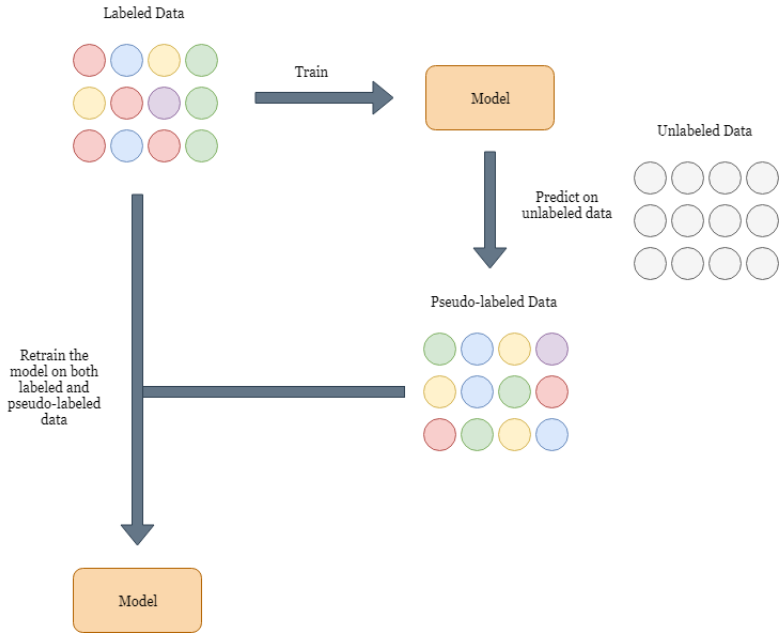


Figure 3.7: Pseudo Labeling.

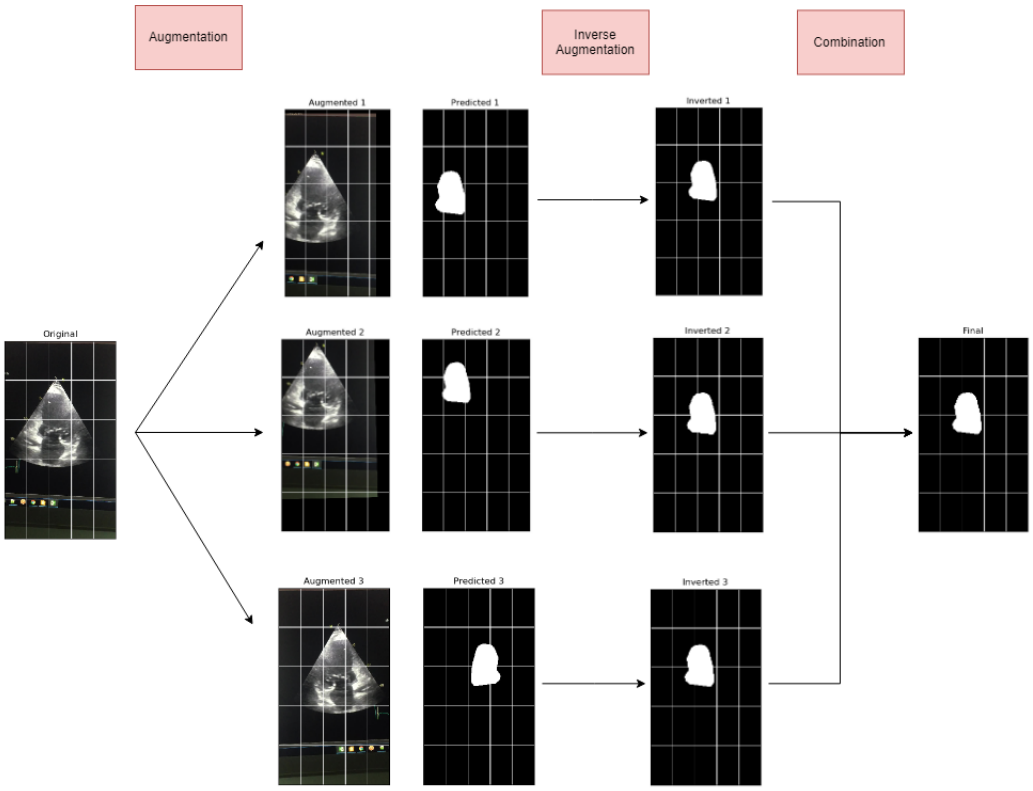


Figure 3.8: Test-time Augmentation

3.5.2 Test-time augmentation

Test-time augmentation (TTA) is a widely used technique to boost the accuracy of segmentation models ([31], [30], [11], [29], [21], [27]). In TTA, different transformations are applied to test images such as translations, scaling, rotations, and flipping, then these different transformed images are fed into the trained model and the final step is ensembling the results to get a more accurate prediction. This technique comes from the intuition that changing the test image in different ways will give the model more opportunities to make inference based on various views of the same image, which increases the chance of producing accurate predictions. While test-time augmentation takes advantage of transforms used in train-time augmentation, it increases the computational costs because it requires making predictions several times.

3.6 Post processing

This section explores the usage of the active shape model and the active contour model on the refinement of segmentation result from deep learning model.

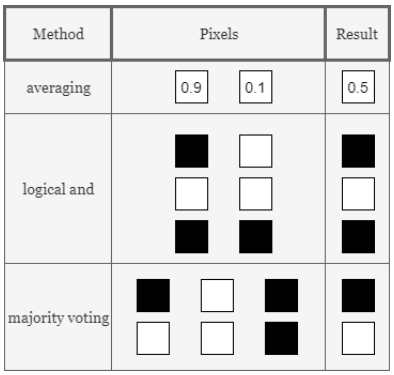


Figure 3.9: Some methods to combine the TTA results

3.6.1 Active contour model

The active contour model algorithm described in section 2.7 will be used with the initial snake initialized as an eclipse that completely contains the contour.

3.6.2 Active shape model

Left ventricular shapes in 2D echocardiogram usually vary in appearance significantly from one image to another, but still, retain some characteristic features. Active shape models (ASM), developed by Tim Cootes and Chris Taylor [3], are statistical models representing the shape of objects, in which an initialized shape is iteratively deformed to fit an instance of the object in a new image.

In this thesis, a simplified active shape model was implemented to improve the segmentation output of the deep learning model, which is a binary image where white pixels belong to the object and black pixels belong to the background. This step is necessary because the shape of LV from segmentation output is quite arbitrary (as the segmentation network is a per-pixel classification network, so it does not give much attention on the information of shape). Given a binary image with a single object, the active shape model aims to properly initialize a shape and fit it to that object so that the final shape of the object is recognizable and acceptable by experts.

The algorithm of the active shape model can be described step by step as follows:

1. **Training a model of shape from data** Given groundtruths of training images (i.e. binary masks), a contour extraction algorithm is used to extract the boundary (or shape) of left ventricles. After that, 3 pivot points of the LVs (one tipping point and two bottom points) will be specified manually or by a trained objection detection model, and $n-3$ other points will be automatically generated on three segments created by those three pivot points, all are equally distributed on each segment by angle. This process produces training data of shape, each is composed of n landmarks. Then, a dimensionality reduction method (here is Principal component analysis - see appendix A) is adopted to model those

shapes from the training set. This model is used in later steps to control the deformed shape which will be fit to the object in a new image.

The result of this step is a PCA model is represented by a mean shape $m = (x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_n) \in \mathbb{R}^{2n}$ (relating to n landmarks), k largest eigenvalues $\{\theta_1, \theta_2, \ldots, \theta_n\}, \theta_i \in \mathbb{R}$ chosen to cover 99% variance of the dataset and k corresponding eigenvectors $\{v_1, v_2, \ldots, v_k\} \subset \mathbb{R}^{2n}$. Those vectors form a basis of a subspace V where the data is projected on.

2. **Generating a distance image** A distance image is an image corresponding to a binary image where intensities of pixels on the corresponding object boundary are the highest, and the further a pixel from the boundary, the smaller its intensity. This type of image is used to compute the difference between an arbitrary shape and the object: the higher total intensity sum of shape pixels, the fitter the shape to the object.

   To compute the distance image, first, from the binary image, a contour-finding algorithm is conducted, which returns a list of points on the object boundary. Connecting consecutive points in the contour gives a thin boundary, then from this boundary image, a distance transform image is calculated. The inverse of the distance transform image gives the distance image.

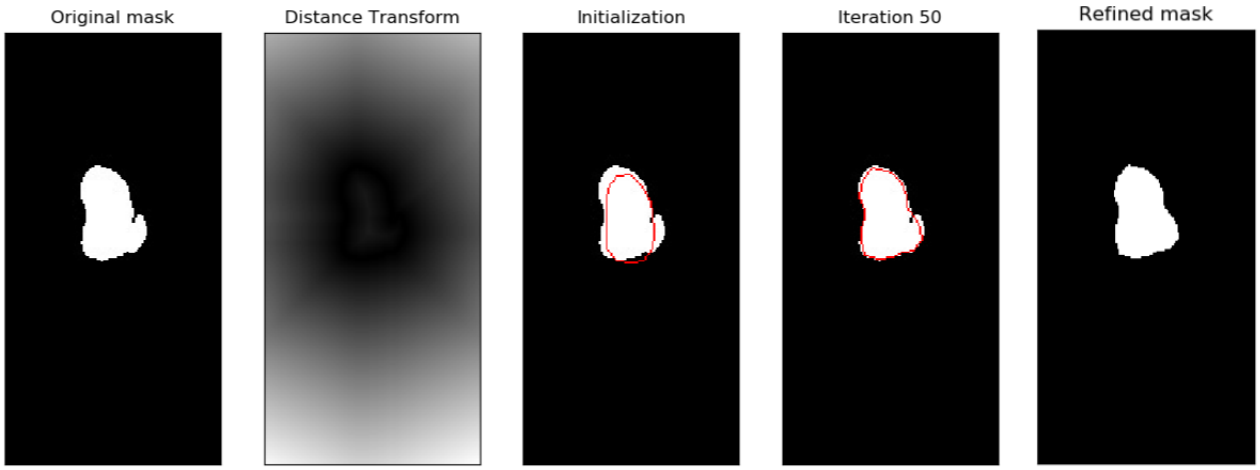


Figure 3.10: An example of the procedure for Active Shape Model

3. **Defining affine transformation** Given a shape $S = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$, the transformation T_{s,θ,t_x,t_y} that translates S by (t_x, t_y), rotates S by angle θ and scales S by a factor s can be represented by a matrix multiplication:

$$T_{s,\theta,t_x,t_y}(S) = S' = \begin{bmatrix} x'_1 & y'_1 \\ x'_2 & y'_2 \\ \vdots & \vdots \\ x'_n & y'_n \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ \vdots & \vdots & \\ x_n & y_n & 1 \end{bmatrix} \times \begin{bmatrix} s\cos\theta & s\sin\theta \\ -s\sin\theta & s\cos\theta \\ t_x & t_y \end{bmatrix} = [S, 1] \times M \quad (3.2)$$

where

$$M = \begin{bmatrix} s\cos\theta & s\sin\theta \\ -s\sin\theta & s\cos\theta \\ t_x & t_y \end{bmatrix} \tag{3.3}$$

is called transformation matrix.

4. **Shape transformation from PCA model** Given a subspace V whose basis is k eigenvectors $\{v_1, v_2, \ldots, v_k\}$, any shape in that subspace is a linear combination of vectors in the basis. Formally, any shape $v_s \in V$ can be expressed as:

$$v_s = v_1 a_1 + v_2 a_2 + \cdots + v_k a_k = V\mathbf{a} \tag{3.4}$$

where

$$V = \begin{bmatrix} | & | & \ldots & | \\ v_1 & v_2 & \ldots & v_k \\ | & | & \ldots & | \end{bmatrix} \tag{3.5}$$

$$\mathbf{a} = [a_1 \quad a_2 \quad \ldots \quad a_k]^T \tag{3.6}$$

The shape to fit onto the object in a new image then is defined by the transformation from the subspace:

$$\mathscr{T}_{\mathbf{a},s,\theta,t_x,t_y} = T_{s,\theta,t_x,t_y}(V\mathbf{a}) \tag{3.7}$$

Here $\mathbf{a}, s, \theta, t_x, t_y$ are variables to optimize in order to find the best fit.

5. **Initializing the mean shape on the distance image** Initialization is an important step in the active shape model. A good initialization has a positive effect on the convergence of the model, while a poor one may make it hard to optimize even if the algorithm is sophisticated. In the experiment using ASM, the mean shape from PCA with proper scaling and translation will be chosen to be the initialized shape. More specifically, the mean shape will be translated so that its center is matched with the object's center, and its area is 1.5 times larger than the object's area. This heuristic aims to put the object approximately fit in the initialized shape, which can lead to faster and more accurate convergence.

6. **Defining loss function** Given a shape represented by a list of points $\{(x_i, y_i)\}_{i=1}^n$ and a distance image $\mathscr{D}$, minimizing the loss function $\mathscr{L}$ is equivalent to minimizing the difference between the shape and object boundary:

$$\mathscr{L} = \sum_{i=1}^{n} \mathscr{D}(x_i, y_i) \tag{3.8}$$

where :

$$\mathscr{T}_{\mathbf{a},s,\theta,t_x,t_y} = T_{s,\theta,t_x,t_y}(V\mathbf{a}) = [x_1 \quad y_1 \quad x_2 \quad y_2 \quad \ldots \quad x_n \quad y_n]^T \tag{3.9}$$

As showed above, $\mathscr{L}$ is a function of transformation parameters (s, θ, t_x, t_y) and

PCA model parameter $\mathbf{a}$. We aims to find $(\mathbf{a}, s, \theta, t_x, t_y)$ that minimize $\mathscr{L}$:

$$(\hat{\mathbf{a}}, \hat{s}, \hat{\theta}, \hat{t}_x, \hat{t}_y) = \underset{\mathbf{a},s,\theta,t_x,t_y}{\arg\max}\, \mathscr{L}(\mathbf{a}, s, \theta, t_x, t_y) \tag{3.10}$$

7. **Fitting by Gradient Descent** To find the minimum of $\mathscr{L}$, we use an iterative optimization algorithm called Gradient Descent.

$$\mathbf{a} = \mathbf{a} - \alpha \frac{\delta \mathscr{L}}{\delta \mathbf{a}} \tag{3.11}$$

where the derivatives can be calculated by the chain rule:

$$\frac{\delta \mathscr{L}}{\delta \mathbf{a}} = \sum_{i=1}^{n} \frac{\mathscr{D}(x_i, y_i)}{\delta \mathbf{a}} = \sum_{i=1}^{n} \frac{\mathscr{D}(x_i, y_i)}{x_i} \frac{x_i}{\delta \mathbf{a}} + \sum_{i=1}^{n} \frac{\mathscr{D}(x_i, y_i)}{y_i} \frac{y_i}{\delta \mathbf{a}} \tag{3.12}$$

The derivative of distance image with respect to x-coordinate and y-coordinate (i.e. $\frac{\mathscr{D}(x_i,y_i)}{x_i}$ and $\frac{\mathscr{D}(x_i,y_i)}{y_i}$) can be calculated by computing Sobel derivative image.

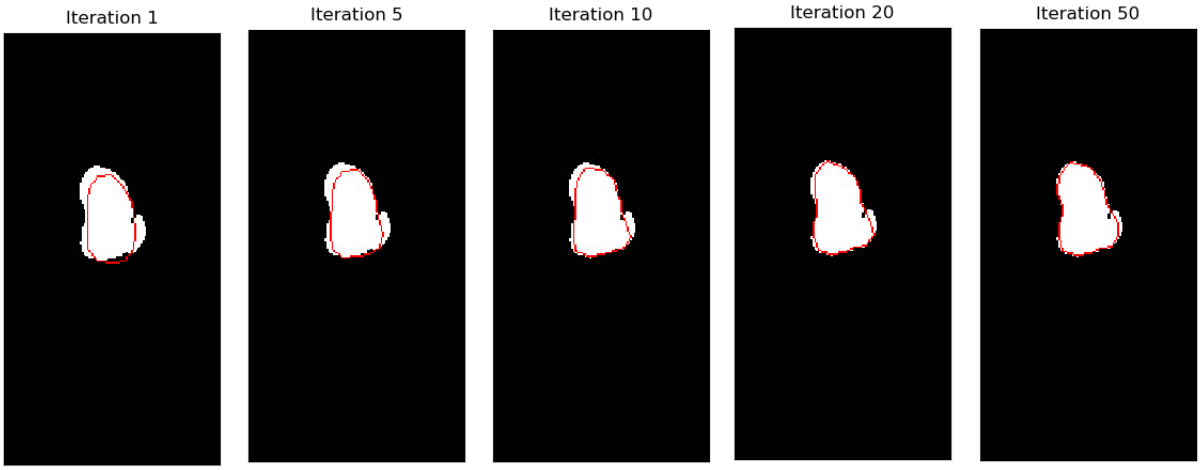


Figure 3.11: Active Shape Model Iterations

Chapter 4

Experiments

4.1 Data

Images from the dataset are 2D echocardiography images that is displayed on different computers and captured under different environmental factors such as lighting conditions or noises. Boundaries of left ventricular regions in those images then were annotated by a domain expert. After that, images of binary masks were generated from those annotated boundaries, and the final raw dataset composed of those masks paired with corresponding images.

Quantitatively, there are total **2418** (image, mask) pairs from **78** videos. Those were split by video into a training set and a testing set, consisting of **1909** (from **62** videos) and **509** (from **16** videos), respectively.

4.2 Experimental Settings

All the code to produce experiments is implemented in Python [1], and the training procedure was conducted on Google Collaboratory [2] with Tesla K80 GPU. Other information about experiments specified to each method will be described below in detail.

4.2.1 Data Augmentation

All the augmentation methods are performed using *albumentations* library [3]. They are applied with the probability of **0.5**, the border values are set to be **zero constant** and interpolation method is **bilinear interpolation**. Besides, all the images (in both training and testing set) will be resized to have a shape of **(512, 256)**. Specific settings

[1] https://www.python.org/
[2] https://colab.research.google.com
[3] https://github.com/albu/albumentations

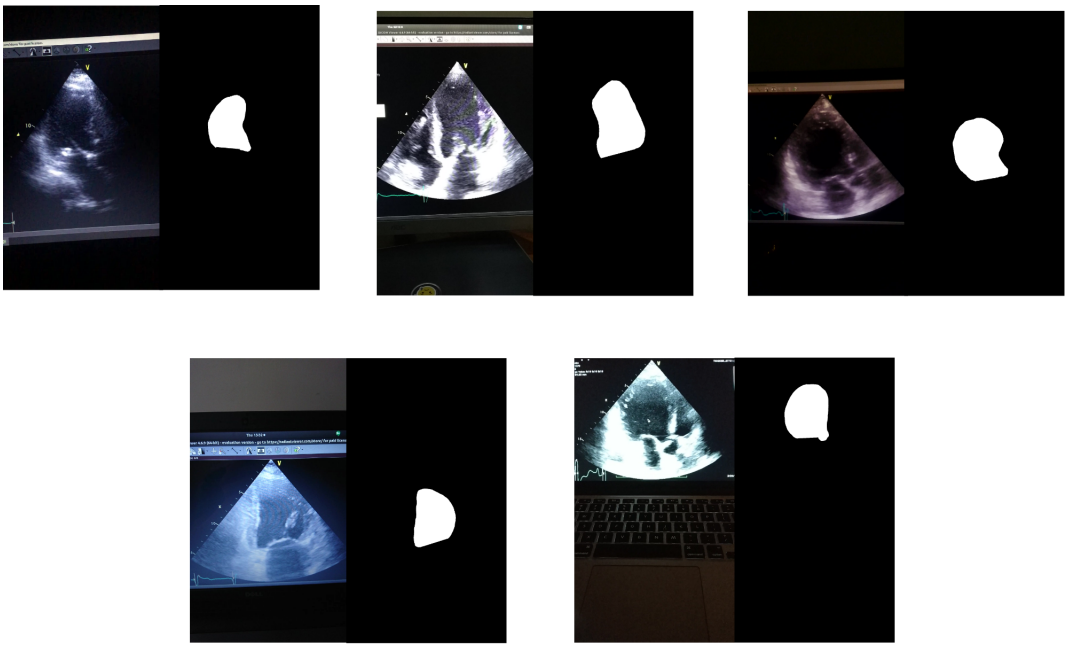


Figure 4.1: A sample view of dataset: the upper row contains sample pairs from the training set, the lower row contains sample pairs from the testing set.

for each technique are:

Affine Transformation The translation limit is set to **0.1** (i.e. the image will be randomly translated, horizontally and vertically in the range equals to 10% of width and height respectively). Similarly, the scale limit is **0.1** (equivalent to $0.9-1.1$ range), the rotation limit is **5º**.

Grid Distortion This technique will be performed in **5** steps, with distortion limit set to **0.3**.

Elastic Transform $\alpha = 100$ and $\sigma = 10$.

4.2.2 Architectures

The number of channels before EC1, EC2, EC3 and EC4 (and similarly after DC4, DC3, DC2 and DC1) in Linknet and Unet architecture are set to be **32**, **64**, **128**, **256** respectively, and the tensor after EC4 and before DC4 has a depth of **512**. Each encoder downsamples and each decoder upsamples the input by a factor of **2**.

All the models is implemented with *Tensorflow Keras* [4].

[4] https://www.tensorflow.org/guide/keras

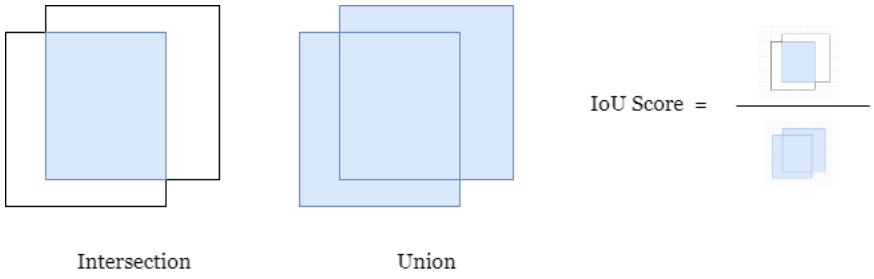


Figure 4.2: IoU Visualization.

4.2.3 Training

The optimization algorithm used in training CNN models is **RMSprop** [5] with initial learning rate of 0.001.

4.2.4 Postprocessing

Active contour model Parameters are set as follows: $\alpha = 0.005$, $\beta = 200$, $w_{line} = 1$ and $w_{edge} = 1$.

Active shape model Each shape is represented by **20** points (or 40 values). Optimization method is **Schocastic Gradient Descent** (implemented in *PyTorch* [6]). Optimization procedure is conducted as follows. First only model parameter a is updated with learning rate of 5×10^{-6} and momentum of 0.9 while transformation parameters s, θ, t_x, t_y are freezed. After 20 epochs, the learning rate of a is reduced to 10^{-6}, and that of θ is set to 10^{-6} while those of s, t_x, t_y are set to 10^{-3}. The momentum value is still 0.9. The total number of epoch is 50.

4.3 Metric

Intersection over Union (IoU) is an evaluation metric used to measure the accuracy of a segmentation result. It is defined by the ratio between the area of overlap between the ground truth mask and the predicted mask of the same object and the area of union between those two masks. Specifically, if the ground truth mask is the binary matrix T, and the predicted mask is the binary matrix P, where white pixels represent the object and black pixels belongs to the background, then **IoU** score can be computed as:

$$\textbf{IoU} = \frac{\textbf{Area of overlap}}{\textbf{Area of union}} = \frac{\textbf{sum}(T \textbf{ and } P)}{\textbf{sum}(T \textbf{ or } P)} \tag{4.1}$$

[5] Geoffrey Hinton's *Neural Networks for machine learning* (https://www.coursera.org/learn/neural-networks)

[6] https://pytorch.org/

Color model	mIoU	stdIoU
RGB	0.8462	0.0916
HSV	**0.8501**	**0.0797**
YCbCr	0.8498	0.0831
YUV	0.8371	0.1170
Gray-scale	0.8466	0.0908

Table 4.1: Experiment results of color spaces.

4.4 Results and Analysis

As mentioned before, due to the fact that the computational power is limited, experiments will be conducted in chronological order of methods introduced in chapter 3, and each method (except the first) will take the best of the previous method as a baseline. We will use *Linknet* architecture with *MobileNetV2* encoder and *upsampling*-type decoder and *BCE Jaccard* loss as the default setting.

4.4.1 Color spaces

Table 4.1 shows the experiments on five color spaces: *RGB*, *HSV*, *YCbCr*, *YUV* and *gray-scale* (here **mIoU** and **stdIoU** means the mean and standard deviation of IoU scores computed from test data). The result shows that *HSV* colors gives the best result on both mIoU (0.8501) and stdIoU (0.0797), which followed closely by *YCbCr* and leaving the popular *RGB* by a margin of 0.004 and 0.022 in mIoU and stdIoU respectively. A reason for this slightly improvement could be that the images in dataset were taken under different light conditions, so color spaces which separating brightness component such as *HSV*, *YCbCr* (note that *YCbCr* is in fact *HSV* in another coordinate) will be given more starting advantage to adapt with the lighting variability of dataset. The similarity in the result of *RGB* and *gray-scale* is also understandable, as regardless of being taken in color, the images of 2D echocardiogram in the dataset are still covered by a large region of original display of 2D echocardiogram, which is in *gray-scale*.

4.4.2 Data augmentation

It can be seen from Table 4.2 that while performing non-linear transformations such as *grid distortion* and *elastic transform* on training set only improve the result from by approximately 0.008 from 0.8501 (Table 4.1) to 0.8587 and 0.8582 respectively, *affine transformation* (with and without *horizontal flipping*) improve much further to 0.8648 (and 0.8714). This can be explained as the nonlinearity greatly deforms the shapes of left ventricles compared to linear transform, then makes the features of left ventricles harder to learn. Finally, the combination of affine transformation, horizontal flipping and grid distortion give the best mIoU score (0.8744) and stdIoU score (0.0710) (note that the combination of affine and elastic transform are not experimented, as in elastic transform the image is also slightly translated and rotated).

Color mode	Augmentation Method	mIoU	stdIoU
HSV	Affine	0.8648	0.0828
HSV	Affine, Hflip	0.8714	0.0711
HSV	Grid Distortion	0.8587	0.0897
HSV	Elastic Transform	0.8582	0.0956
HSV	Affine, Hflip, Grid Distortion	**0.8744**	**0.0710**

Table 4.2: Experiment results of augmentation methods.

Loss function	mIoU	stdIoU
BCE	0.8651	0.0827
Dice	0.8737	0.0676
BCE Dice	0.8726	0.0736
Jaccard	0.8726	0.0637
BCE Jaccard	0.8744	0.0710
Weighted BCE Jaccard	**0.8805**	**0.0633**

Table 4.3: Experiment results of loss functions.

4.4.3 Loss function

In terms of loss function, five loss functions are tested, which are *Binary cross-entropy*, *Dice*, *Jaccard* and the combination of the first with the latter two. The results showed that *BCE-Jaccard* combination produces the highest mIoU score of 0.8744 and the lowest standard derivation of 0.0710 (Table 4.3). After that, this best loss will be weighted by the method proposed in section 3.4, and this gives a nice improvement on both mIoU and stdIoU scores. This strengthens the assumption that the boundary is of greater importance than the region inside the object, and more weights give to the pixel on boundary truly improve the result.

4.4.4 Architectures

Next experiments are about two different architectures (*Linknet* and *Unet*) with two different encoders (*MobileNetV2-based* and *ResNet-based*) and two different decoder options (*transposed convolution* or *upsampling*). To reduce the number of experiment needed (original $2\times2\times2=8$), first two experiments on decoders were conducted, then the best decoder options was used with architectures and encoders ($1+2\times2=5$ in

Architecture	Encoder	Decoder	mIoU	stdIoU
Linknet	MobileNetV2	Upsampling	**0.8805**	**0.0633**
Linknet	MobileNetV2	Transpose	0.8753	0.0846
Linknet	ResNet	Upsampling	0.8720	0.0685
Unet	MobileNetV2	Upsampling	0.8737	0.0673
Unet	ResNet	Upsampling	0.8712	0.0706

Table 4.4: Experiment results of architectures.

total). From the table 4.4, we can see that *upsampling* decoder gives a slightly higher score than *transposed convolution*, and when fixing upsampling decoder, Linknet architecture with *MobileNetV2* encoder gave the best result.

Figure 4.3 shows plots of training and validation losses and mean IoU scores on three models with best setting from each section: *HSV* model from section 4.4.1, HSV with data augmentation (*SCR*, *Hflip* and *Grid Distortion*) model from section 4.4.2, and model with those settings trained with *Weighted BCE Jaccard* loss from section 4.4.3 (note that this is also the best model from section 4.4.4). It can be noticed from plots that *HSV* model subjected to overfitting with extremely low training loss compared to *HSV + Aug* (and similarly extremely high training IoU compared to other two models), while its validation side is considerably inferior to the other two. This is understandable as the data fed into is without augmentation, so the model was hard to learn prominent features. With heavy data augmentation, *HSV + Aug* showed a considerable boost in model performance, and modifying the loss function to concentrate into boundaries will give *HSV + Aug + WBCEJ* model a slightly better IoU score on the validation set.

Figure 4.4 gives two examples that prove the effectiveness of the boundary-weighted loss. *HSV* model, which was only trained on raw data, showed a weak generalization ability, as it failed on low contrast regions and extremely noisy images. While *HSVA* (*HSV + Aug*) presented some improvement as it can trace the boundary more accurately than *HSV* did, there are still some failure regions where the segmentation boundary was highly distorted. *HSVAW* (*HSV + Aug + WBCEJ*), in which more importance was given to pixels at the border, showed its ability to produce smooth boundary and give a better segmentation result. This experimentally proved that weighted-boundary loss truly helps the model to produce a non-distorted and precise boundary, which is very important in medical imaging.

Two figures 4.6 and 4.5 provides some best and worse predictions of *HSVAW* model. Figure 4.6 showed that the model can not give good segmentation results if there are regions with ambiguities (such as those on the right of LV in the left image) or the image is of low contrast (the middle image) or there are many noises (the left image). Although these problems were tackled to some extents (looking at images in figure 4.5, where all those degradation factors exist at a moderate degree), it reveals that the best model still has rooms for development, such as in the ability to cope with high variance in images.

4.4.5 Strategies

Test-time Augmentation

There are four experiments on test-time augmentation. As this technique is only feasible when the transformations are invertible, only affine transformation and horizontal flipping are applied to the original image. In the first experiment, the image is only augmented one time. Then the predicted masked is reverse-transformed to the coordinate corresponding to the original image as the final output. In other three

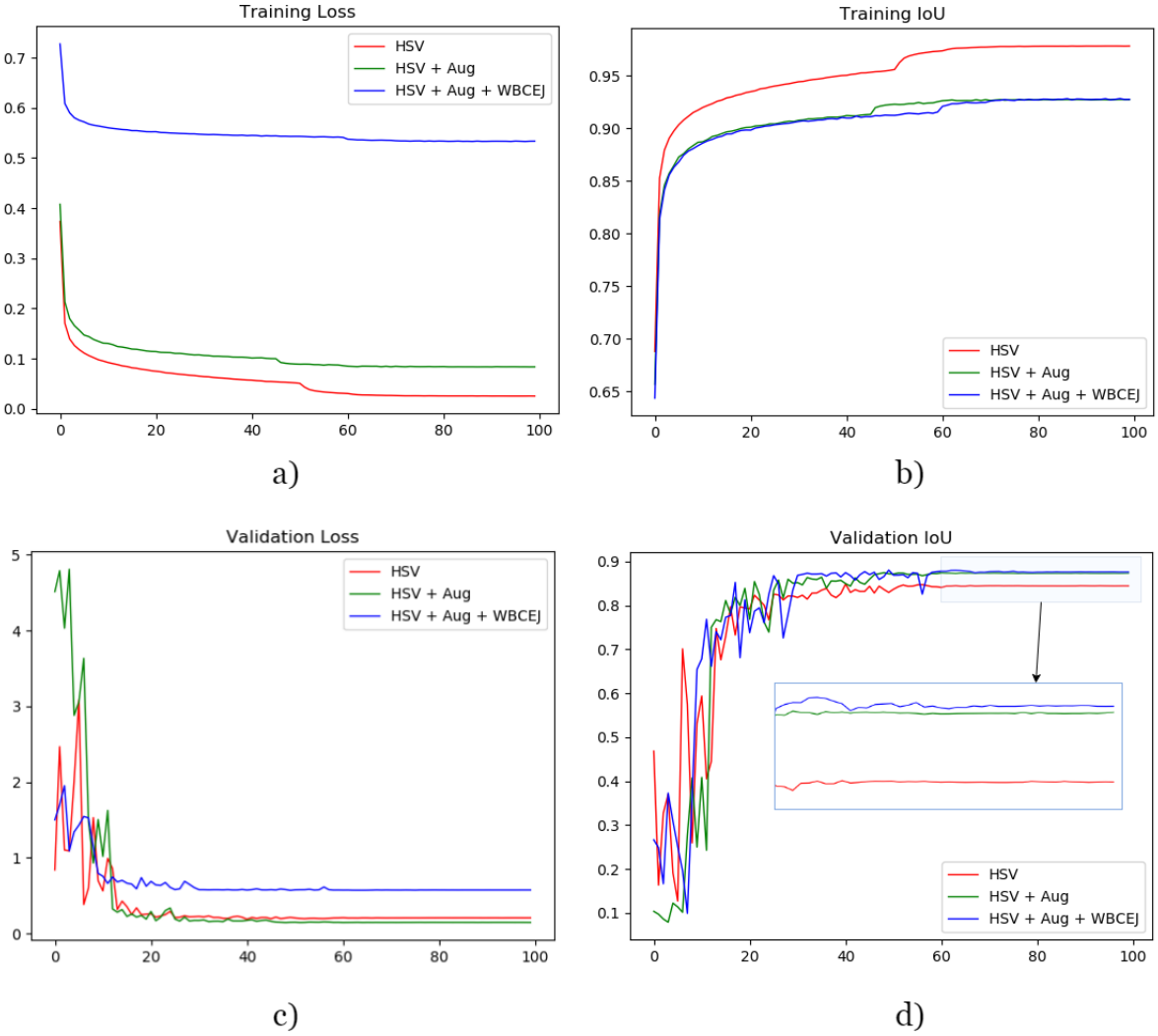


Figure 4.3: Training histories of three best models from each section.

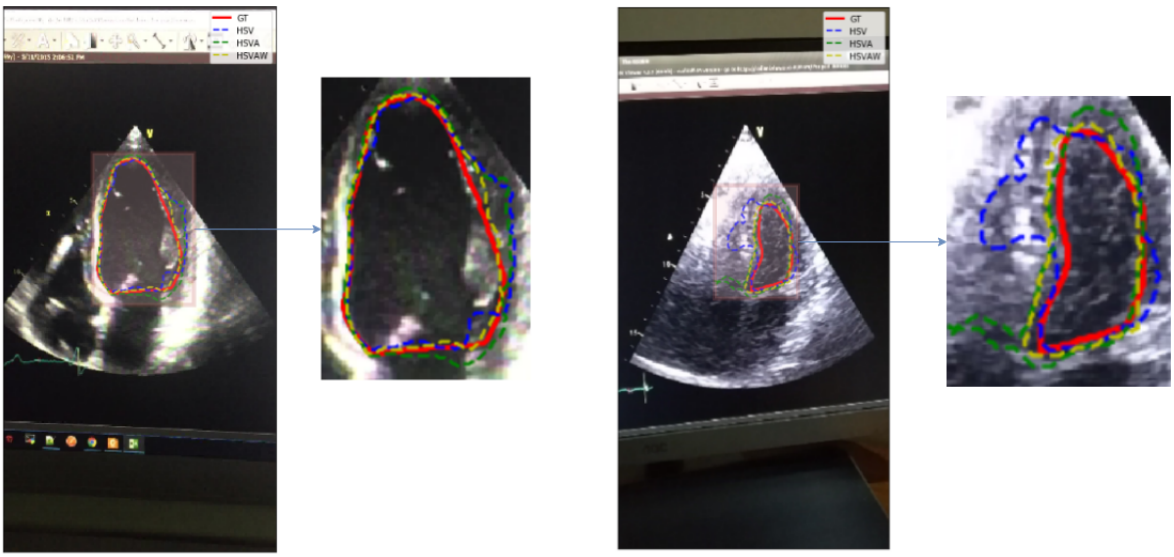


Figure 4.4: Comparison of sample results on the testing set of three best models.

Strategy	Detail	ΔmIoU	ΔstdIoU
Test-time Augmentation	affine + hflip, 1	-0.0330	0.0178
Test-time Augmentation	affine + hflip, 3, logical and	-0.0290	0.0158
Test-time Augmentation	affine + hflip, 3, major voting	-0.0139	0.0053
Pseudo-Labeling	1288 pseudo data pairs	-0.0060	0.0061

Table 4.5: Experiment results of some strategies: ΔmIoU means the difference between IoU score the current method and that of the best model so far.

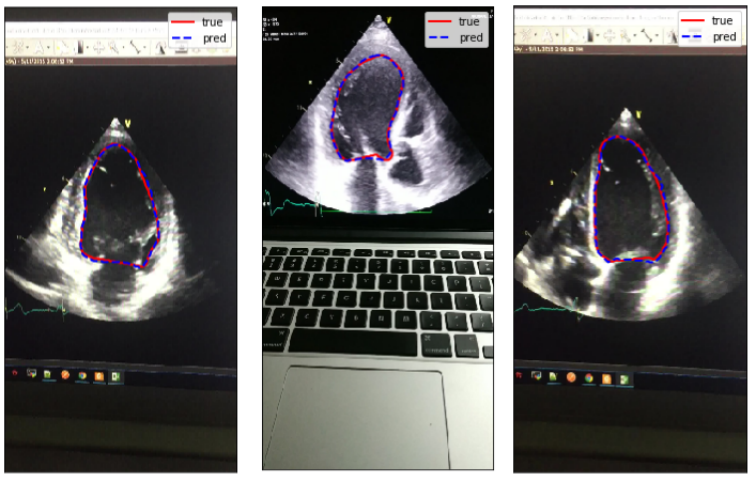


Figure 4.5: Some good predictions of the best model.

experiment, three augmentation are performed on a same image, with only the difference in how to inverted predicted masks are combined (*and* operation, *majority voting* or *averaging*). The results showed in Table 4.5 (here Δ in ΔmIoU and ΔstdIoU means the signed difference in scores after applying the current method to the best model *HSVAW*) indicates that test-time augmentation does not work for this problem (at least on tried methods), as while it increases the computational cost (to three times), there is still a degradation in model performance.

Pseudo Labeling

In this experiment, the best model setting so far (*HSV* space, *Affine transformation-Horizontal Flipping-Grid Distortion*, *Linknet* with *MobileNetV2* encoder and *Upsampling decoder*) was used to give prediction on 1288 unannotated images. The predicted masks paired with the corresponding images created a new pseudo training data. Adding them to the original dataset, there were in total 3197 training pairs, and the model with above settings was trained from scratch on the new dataset. The result in Table 4.5 in this retrain procedure did not enhance the performance of the model, showing that the pseudo labeling also did not work with the problem on current dataset.

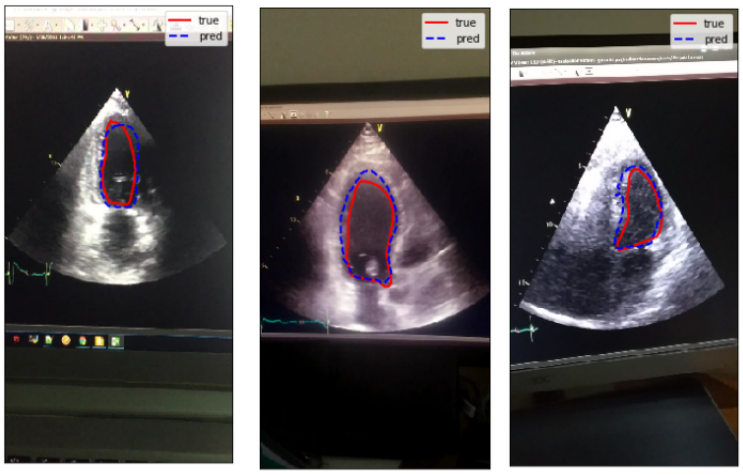


Figure 4.6: Some bad predictions of the best model.

Postprocessing	ΔmIoU	ΔstdIoU
Active contour model	**0.0137**	**-0.0044**
Active shape model	-0.0288	0.0061

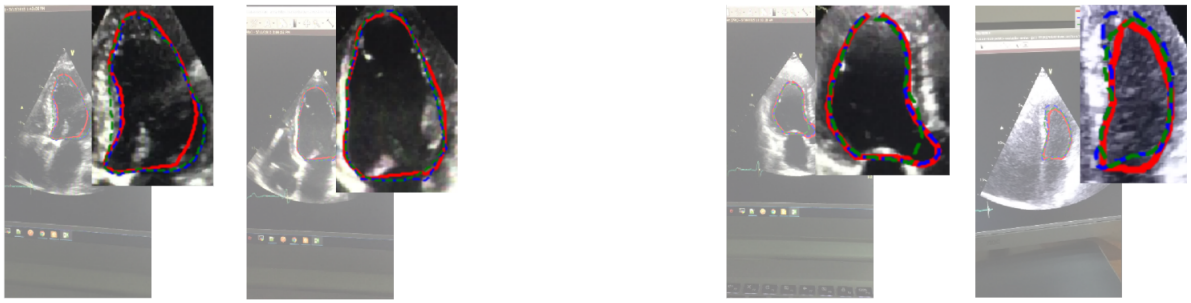
Table 4.6: Experiment results of postprocessing techniques.

4.4.6 Postprocessing

Table 4.6 shows experimental results when applied *active contour model* (ACM) and *active shape model* (ASM) to the best model so far. The results show that while ACM slightly improved the model's performance, ASM produced a considerable degradation. Visualizing results from these two methods indicated that ACM refined noisy and distorted segmentation boundaries in most cases, meanwhile, ASM failed to capture high complexity and variance of testing shapes, which led to poor fittings. Figure 4.7 shows a bad and a good prediction in each postprocessing method. Figure 4.7a (left) represents a case where ACM fails, in which the boundary (on the left side of the object) is over-smoothed; and the good case (right) comes from the refinement of zigzag contour (on the lower right side). In terms of ASM, figure 4.7b on the left shows a case when the fitted shape is too simple, which reduces the accuracy; and the right figure shows a good case where the abnormal shape of model's prediction is improved to have a standard shape which is fitter to the groundtruth.

4.4.7 Inference time

As mentioned above, Linknet architecture was designed to minimize the number of parameters needed in a segmentation architecture, and similarly, the Bottleneck Residual Block in MobileNetV2 also was created to efficiently reduce the size of the model while retaining the ability to produce accurate results. For those reasons, it is not surprising that the proposed model is able to make a fast prediction, of average 20 ms per frame or 50 FPS (compared to 30 ms per frame when replacing Linknet by Unet architecture) on a Tesla K80 GPU.



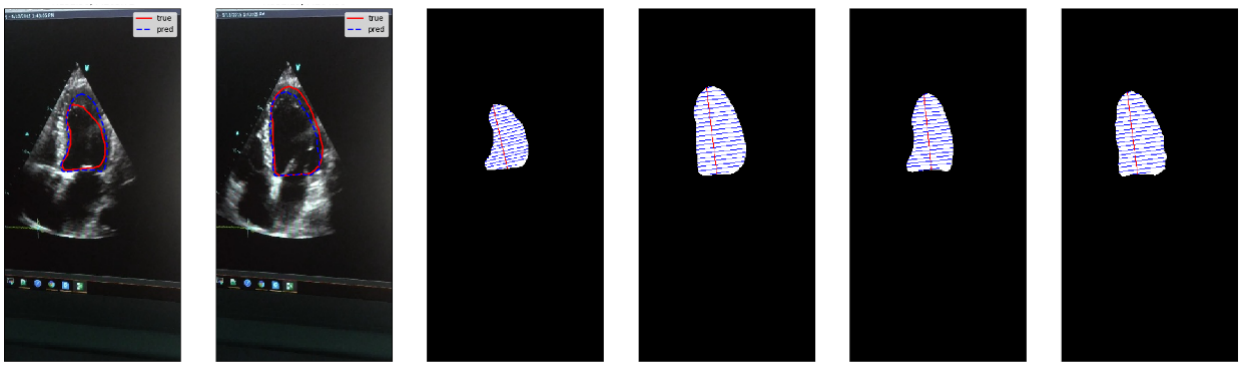
(a) ACM (b) ASM

Figure 4.7: Some results of active contour model and active shape model.

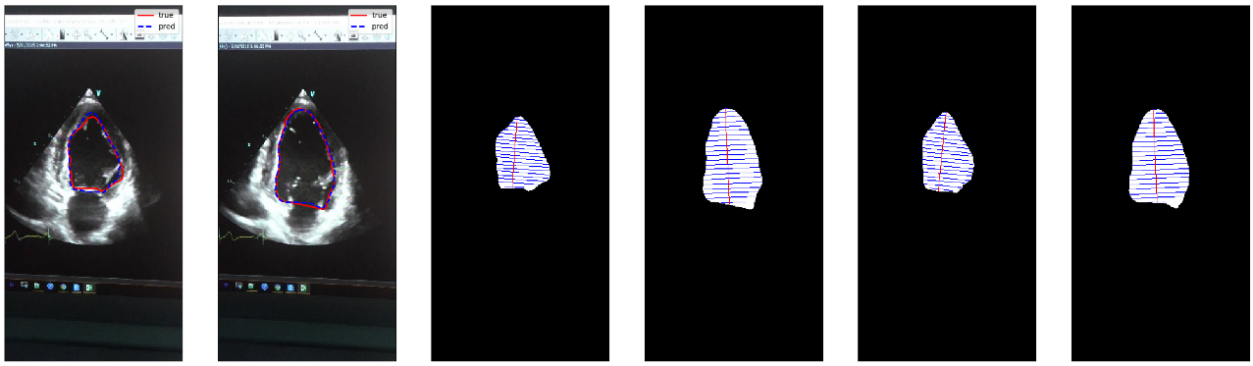
4.4.8 Ejection Fraction

The metric ΔEF in table 4.7 is defined as $\Delta EF = EF_{true} - EF_{pred}$, where EF_{true} is the EF value calculated by applying single-plane Simpson method on two frames that have minimum and maximum LV areas in the groundtruth masks (computed by counting white pixels) and EF_{pred} is the EF value calculated by the same method of two prediction masks corresponding to the two groundtruth masks. IoU_{min} and IoU_{max} are Intersection-over-Union scores between the true masks and the predicted masks of those min and max frames. There are a total of 12 pairs used to test, and the results in table 4.8 revealed a relatively high average difference between expert EF and model EF. It can be observed from the visualization of the best and the worst result (figure 4.8b and 4.8a) that this high difference may result from the poor segmentation of LV at the end-systolic phase, when there are ambiguities near the apex of the heart. Additionally, this may be caused by the calculation error of the single-plane Simpson method, which is relatively large when the LV shape is not similar to an elliptical shape. This method is also highly dependent on how the vertical axis and horizontal segments are drawn. In contrast, the best case shows that the segmentation model can work with images of high quality and LV of good shape, with approximately zero difference from the groundtruth.

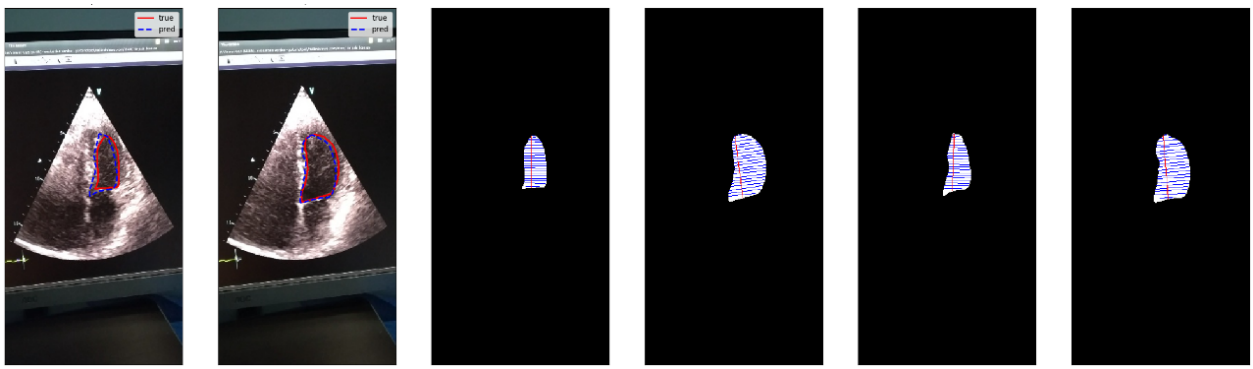
To further backup above arguments, let's take a look at the results of ID 1 and 10 (figure 4.8d and 4.8c respectively). While ID 1 has relatively high IoU scores on both minimum and maximum frames, its ΔEF is pretty large (21%). This contradiction is caused by the difference between two sets of horizontal segments with regards to min frames and the slight distinction between two shapes in max frames. On the other hand, in ID 10, the similarities between shapes and horizontal segments between the poorly segmented mask and the groundtruth gave a quite small ΔEF. In conclusion, while the prediction of the segmentation model is an important factor in the calculation of Ejection Fraction, there are many other tasks that must be achieved in order to get a good EF result.



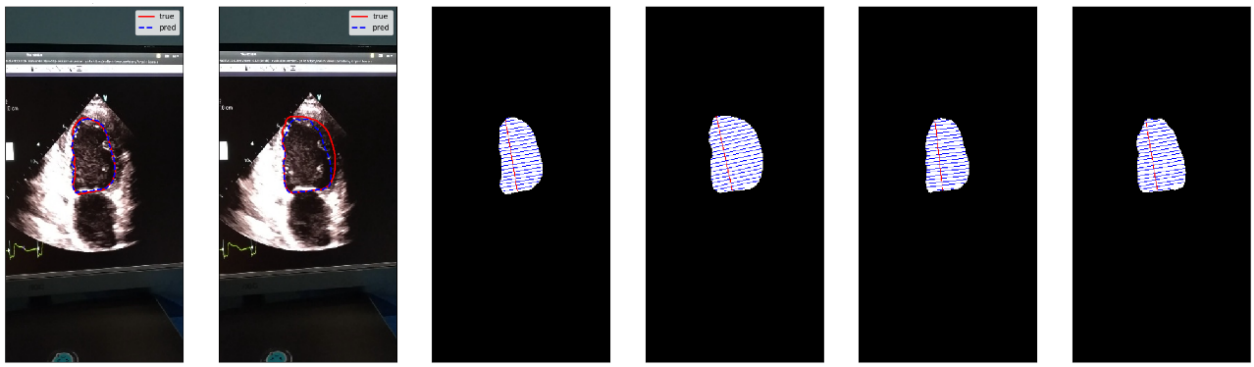
(a) ID 7.



(b) ID 11.



(c) ID 10.



(d) ID 1.

Figure 4.8: Visualization of some EF calculation results.

ID	IoU_{min}	IoU_{max}	EF_{true}	EF_{pred}	ΔEF
1	0.9291	0.8322	40%	19%	21%
2	0.9016	0.9378	27%	20%	7%
3	0.8534	0.7688	59%	45%	14%
4	0.7856	0.8400	69%	65%	4%
5	0.7840	0.7948	44%	14%	30%
6	0.8515	0.8450	57%	33%	24%
7	0.7513	0.8409	60%	27%	*33%*
8	0.8742	0.8799	39%	32%	7%
9	0.9018	0.9074	47%	42%	5%
10	0.6609	0.8280	66%	60%	6%
11	0.8951	0.9468	41%	41%	**0%**
12	0.7955	0.8893	55%	35%	20%

Table 4.7: Experiment results of EF calculation.

IoU_{min}	IoU_{max}	ΔEF
0.8320 ± 0.0749	0.8592 ± 0.0519	$14 \pm 10\%$

Table 4.8: Statistics of EF calculation (mean ± std).

Chapter 5

Conclusions

This thesis explored a deep learning approach to tackle the problem of left ventricle segmentation. Experiments cover a wide range of methods in training and making inference on a deep learning model, from preprocessing data (e.g. color space conversion), artificially generating new data (data augmentation) to picking a suitable loss function. Due to the limitation of computational power, experiments were carried out phase by phase, where each phase used the best experimental settings from the previous phase. The results suggested that *Linknet* with encoders of *Bottleneck Residual Blocks* and *upsampling*-type decoders, training on *HSV*-converted images with *weighted BCE Jaccard* loss and using heavy data augmentation such as *affine transformation* and *grid distortion* and *active contour model* as postprocessing method is the best pipeline for the current data set, with mIoU score of 89.42%. While some methods boost model performance, such as data augmentation and weighted loss function, postprocessing methods such as pseudo-labeling, test-time augmentation, and active shape model have shown some degradation in model accuracy. In terms of EF measurements, the model is able to give an acceptable result on high-quality images with only 2% of error. The model from this best pipeline, which is able to give accurate segmentation result within a reasonable time, have already been deployed in a mobile application of automatically EF calculation.

Although the current model can produce good results on average, there are cases where the images suffer from heavy environmental noises, have low contrast or contain many ambiguities, leading to poor and unsatisfactory prediction outputs. To tackle the above problems, some future trials may include:

- **Getting more data** As data is the key in any deep learning techniques, getting more data is the simplest way to improve a DL-based model. Since the current data set is only annotated by a single expert, obtaining data labeled by different experts can encourage DL model to learn to be invariant to inter- and intra-variability between doctors and make the model more flexible when giving predictions of new images under different environments.
- **Trying new architectures** Beside Unet-like architectures, there are various network designs that can be put into trials, such as FPN and PSPNet or instance

segmentation architecture like Mask-RCNN. Though it has been argued that the LV segmentation problem will be better tackled as a semantic segmentation problem, it is worthwhile to try new methods.

- **Training view-specific models** As images from the same view in TTE have similar shapes and characteristic features, training model on the data belong to the same view will result in a specialized model to this specific view, and combine those model can boost the result overall.
- **Adopting more advanced shape modeling methods** More complex shape systems can be used to model the diversity of shapes in data set, thus making postprocessing a truly refinement step to improve poorly segmented images.

Appendix A

Statistical Shape Analysis

A.1 Shapes and Landmarks

From [8], **shape** is all the geometrical information that remains when location, scale and rotational effects are filtered out from an object. We can describe a shape by points on the contour. This set of points is called landmarks, which have the following definition [8]: a **landmark** is a point of correspondence on each object that matches between and within populations.

A.2 Shape Alignment - Procrustes Analysis

A simple iterative approach is given in [4] as follows:

- Translate each example so that its center of gravity is at the origin.
- Fix one shape S^1 and scale so that $||S^1|| = 1$.
- Align all the other shapes with the current estimate of the mean shape. To align two shape S^1 and S^j, each centered on the origin, we choose a scale s^j and a rotation θ^j that minimizes $||T_{s^j,\theta^j}(S^j) - S^1)||^2$, which is the sum of square distances between landmarks of the two shapes. s^j and θ^j can be computed as follows:

$$a^j = \frac{z^j z^1}{||z^j||^2} \tag{A.1}$$

$$b^j = \sum_{i=1}^{n} \frac{x_i^j y_i^1 - x_i^1 y_i^j}{||z^j||^2} \tag{A.2}$$

$$s^j = \sqrt{(a^j)^2 + (b^j)^2} \tag{A.3}$$

$$\theta^j = \tan^{-1}(\frac{b^j}{a^j}) \tag{A.4}$$

Then the transformation is defined by the transformation matrix M:

$$M = \begin{bmatrix} s^j \cos\theta^j & s^j \sin\theta^j \\ -s^j \sin\theta^j & s^j \cos\theta^j \end{bmatrix} \tag{A.5}$$

and the transformation operation $T_{s^j,\theta^j}(S^j)$ is defined as:

$$T_{s^j,\theta^j}(S^j) = S^j M = \begin{bmatrix} x_1^j & y_1^j \\ x_2^j & y_2^j \\ \vdots & \vdots \\ x_n^j & y_n^j \end{bmatrix} \begin{bmatrix} s^j \cos\theta^j & s^j \sin\theta^j \\ -s^j \sin\theta^j & s^j \cos\theta^j \end{bmatrix} \tag{A.6}$$

A.3 Shape Modeling - Principal Component Analysis

Given the data of shapes S^i represented by a vector $\mathbf{x}_i$, PCA aims to find a lower-dimensional linear space to orthogonally projects the data onto, such that the variance of the projected data is maximized. Step-by-step procedure of modeling shapes by PCA is given as follows:

1. Compute the mean of the shapes:

$$\bar{\mathbf{x}} = \frac{1}{s}\sum_{1}^{s}\mathbf{x}_i \tag{A.7}$$

2. Compute the covariance of the shapes:

$$\mathbf{S} = \frac{1}{s-1}\sum_{i=1}^{s}(\mathbf{x}_i - \bar{\mathbf{x}})((\mathbf{x}_i - \bar{\mathbf{x}})^T \tag{A.8}$$

3. Compute the eigenvectors v_i and corresponding eigenvalues λ_i of $\mathbf{S}$ (λ_i is sorted in an decreasing order).

4. For the pre-specified p%, Choose the first k largest eigenvalues that retain p% of total variance:

$$\sum_{i=1}^{t}\lambda_i \geq \frac{p}{100}\sum_{i=1}^{s}\lambda_i \tag{A.9}$$

Appendix B

Ejection Fraction Calculation using Simpson's Rule

First, three pivots point in the left ventricle boundary are marked (manually or using Deep Learning approaches such as CNN for Object Detection). Then, a vertical axis is computed by connecting to the tipping point to the midpoint of the bottom segment, and 20 horizontal segments are drawn along the vertical axis, equally distributed (figure B.1). If the image from A2C and A4C view are available, we can approximately calculate the volume of LV by Modified Simple's Rule using the following formula:

$$V = \frac{\pi}{4} \sum_{i=1}^{20} a_i b_i \frac{L}{20} \tag{B.1}$$

where a_i is the length of i-th horizontal segment in A2C, b_i is the length of i-th horizontal segment in A4C and L is the length of vertical axis in both images (after being scaled for consistency).

If we have one view of LV, the below formula can be used:

$$V = 0.85 \frac{A^2}{L} \tag{B.2}$$

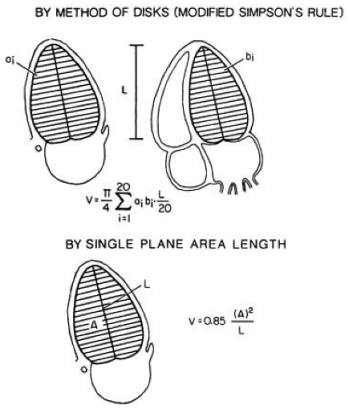


Figure B.1: Simpson rule for calculating LV volume.

where A is the formula of the LV calculated by Simpson's rule:

$$A = \frac{L}{3 \times 20}(a_1 + a_{20} + 4a_2 + 2a_3 + 4a_4 + 2a_5 + \cdots + 2a_{19}) \tag{B.3}$$

Bibliography

[1] P. Gueret, S. Meerbaum, H. L. Wyatt, T. Uchiyama, T. W. Lang, and E. Corday, "Two-dimensional echocardiographic quantitation of left ventricular volumes and ejection fraction. Importance of accounting for dyssynergy in short-axis reconstruction models.," *Circulation*, vol. 62, no. 6, pp. 1308–1318, Dec. 1980.

[2] M. Kass, A. Witkin, and D. Terzopoulos, "Snakes: Active contour models," *International Journal of Computer Vision*, vol. 1, no. 4, pp. 321–331, Jan. 1988.

[3] T. Cootes, C. Taylor, D. Cooper, and J. Graham, "Active Shape Models-Their Training and Application," *Computer Vision and Image Understanding*, vol. 61, no. 1, pp. 38–59, Jan. 1995.

[4] T. Cootes, E. Baldock, and J. G. And, "An introduction to active shape models," *Image processing and analysis*, pp. 223–248, 2000.

[5] N. J Alison and B. Djamal, "Ultrasound image segmentation: a survey.," *IEEE Transactions on Medical Imaging*, vol. 25, no. 8, pp. 987–1010, 2006.

[6] R. C. Gonzalez and R. E. Woods, *Digital image processing*. Prentice Hall, 2008, p. 954.

[7] D.-H. Lee, "Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks," *Workshop on Challenges in Representation Learning, ICML*, vol. 3, 2013.

[8] I. L. Dryden, "Shape Analysis," in *Wiley StatsRef: Statistics Reference Online*, Chichester, UK: John Wiley & Sons, Ltd, Sep. 2014.

[9] A. Norouzi, M. S. M. Rahim, A. Altameem, T. Saba, A. E. Rad, A. Rehman, and M. Uddin, "Medical Image Segmentation Methods, Algorithms, and Applications," *IETE Technical Review*, vol. 31, no. 3, pp. 199–213, May 2014.

[10] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," Sep. 2014.

[11] J. Dai, K. He, and J. Sun, "BoxSup: Exploiting Bounding Boxes to Supervise Convolutional Networks for Semantic Segmentation," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1635–1643.

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," Dec. 2015.

[13] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," Feb. 2015.

[14] J. Long, E. Shelhamer, and T. Darrell, "Fully Convolutional Networks for Semantic Segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 3431–3440, 2015.

[15] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9351, pp. 234–241, 2015.

[16] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, *Going Deeper With Convolutions*, 2015.

[17] F. Chollet, "Xception: Deep Learning with Depthwise Separable Convolutions," Oct. 2016.

[18] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," Mar. 2016.

[19] A. Garcia-Garcia, S. Orts-Escolano, S. Oprea, V. Villena-Martinez, and J. Garcia-Rodriguez, "A Review on Deep Learning Techniques Applied to Semantic Segmentation," Apr. 2017.

[20] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," Apr. 2017.

[21] A. Klibisz, D. Rose, M. Eicholtz, J. Blundon, and S. Zakharenko, "Fast, Simple Calcium Imaging Segmentation with Fully Convolutional Networks," *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support*, pp. 285–293, 2017.

[22] C. I. Sánchez, M. Ghafoorian, T. Kooi, F. Ciompi, G. Litjens, B. E. Bejnordi, A. A. A. Setio, J. A. van der Laak, and B. van Ginneken, "A survey on deep learning in medical image analysis," *Medical Image Analysis*, vol. 42, pp. 60–88, 2017.

[23] E. Smistad, A. Ostvik, B. O. Haugen, and L. Lovstakken, "2D left ventricle segmentation using deep learning," *IEEE International Ultrasonics Symposium, IUS*, 2017.

[24] A. Arafati and H. Jafarkhani, "Multi-label 4-chamber segmentation of echocardiograms using Fully Convolutional Network," 2018.

[25] A. Buslaev, A. Parinov Rickai Samara, E. Khvedchenya ODSai Odessa, V. I. Iglovikov, and A. A. Kalinin, "Albumentations: fast and flexible image augmentations," Tech. Rep., 2018.

[26] A. Chaurasia and E. Culurciello, "LinkNet: Exploiting encoder representations for efficient semantic segmentation," *2017 IEEE Visual Communications and Image Processing, VCIP 2017*, pp. 1–4, 2018.

[27] J. Li, A. Raventos, A. Bhargava, T. Tagawa, and A. Gaidon, "Learning to Fuse Things and Stuff," Tech. Rep., 2018.

[28] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, Jan. 2018.

[29] S. Seferbekov, V. Iglovikov, A. Buslaev, and A. Shvets, "Feature pyramid network for multi-class land segmentation," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, vol. 2018-June, 2018, pp. 272–275.

[30] G. Wang, W. Li, S. Ourselin, and T. Vercauteren, "Automatic Brain Tumor Segmentation using Convolutional Neural Networks with Test-Time Augmentation," *International MICCAI Brainlesion Workshop*, pp. 61–72, 2018.

[31] G. Wang, W. Li, M. Aertsen, J. Deprest, S. Ourselin, and T. Vercauteren, "Aleatoric uncertainty estimation with test-time augmentation for medical image segmentation with convolutional neural networks," *Neurocomputing*, vol. 338, pp. 34–45, Apr. 2019.

[32] E. Smistad, "Fully automatic real-time ejection fraction and MAPSE measurements in 2D echocardiography using deep neural networks," *2018 IEEE International Ultrasonics Symposium (IUS)*,