Segmentation of Breast Cancer Ultrasound Images

Segmentation of Breast Cancer Ultrasound Images

By Mingjie Jiang, B.Eng.

A Thesis Submitted to the School of Graduate Studies in the Partial Fulfillment of the Requirements for the Degree Master of Science

McMaster University

McMaster University
Master of Science (2019)
Hamilton, Ontario (School of Computational Science and Engineering)

TITLE: Segmentation of Breast Cancer Ultrasound Images

AUTHOR: Mingjie Jiang (McMaster University)
SUPERVISOR: Dr. Ned Nedialkov

NUMBER OF PAGES: x, 90

Abstract

Breast cancer is the most common cancer occurring in women. Breast-conserving surgery is a desirable choice for an early-stage breast cancer. An intra-operative margin assessment of excised breast lesion tissue can help avoid additional surgeries. An essential problem in intra-operative margin assessment is how to extract an accurate boundary of the excised lesion automatically and quickly. To solve this problem, we segment breast cancer ultrasound (US) images and then generate boundaries based on the segmentation results.

In this research, we propose a new convolutional neural network model, named IU-Net, to segment breast cancer US images. IU-Net combines inception blocks and the well-known U-Net model. We train IU-Net with US images and corresponding manually segmented images provided by Dr. Jeffery Carson and his research group of the Lawson Health Research Institute, London, Ontario, Canada. We also apply an autoencoder in training IU-Net. The experimental results show that IU-Net achieves slightly more accurate results than U-Net and uses 3.8x fewer parameters than U-Net.

Acknowledgements

Firstly, I would like to thank my supervisor Prof. Ned Nedialkov of the department of computing and software at McMaster University. He gave me a lot of advice on my research and thesis writing.

I would also like to thank Dr. Jeffery Carson at Lawson Health Research Institute. He and his research group provided the breast cancer ultrasound images dataset and also gave me some suggestion on my research.

Finally, I would like to thank my parents and my girlfriend for their continuous support and encouragement throughout my years of study.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Breast cancer is the most common cancer among women [8]. It is also diagnosed among men, but the incidence is low [22]. The statistics of DeSantis et al. [17] shows that each American woman has a 12.29% risk of being diagnosed with breast cancer during her lifetime, and the incidence of in situ breast cancer increased from 1975 to 2010. Breast cancer has five stages (0-4), which are identified by a TNM[1] staging system. For an early-stage, breast-conserving surgery (BCS) is usually an effective method of treatment [31]. It consists of removing the part of the breast that contains a tumor. Typically, some healthy tissue around the cancer is also excised. According to [20], the survival rates of BCS and mastectomy are at the same level. However, BCS can provide better cosmetic effects than mastectomy. One of the main goals of BCS is to obtain tumor-free resection margins, since positive margins are in association to an increased risk of local recurrence [19]. Intra-operative margin assessment of excised breast lesion tissue can help avoid additional surgery. According to the research of Olsha et al. [43], intra-operative ultrasound (US) imaging can help maintain a low level of re-excision rate after BCS.

[1] T–tumor, N–(lymph) nodes, M–metastasis.

An essential problem in intra-operative US is how to extract an accurate boundary of the excised lesion automatically and quickly. Some artifacts due to the presence of bags and shadows appear in US images, and these artifacts would degrade performance of some classic segmentation methods (thresholding, region growing, or watershed [23]). An accurate boundary of an excised lesion can help surgeons make better decisions.

The structure of this chapter is as follows. Section 1.1 states the segmentation problem that is the subject of this thesis. Section 1.2 summarizes our method to solve this problem and summarizes our experimental results. Section 1.3 is the contribution of this thesis. Section 1.4 gives an outline of this thesis.

1.1 Problem statement

We are interested in automatically determining the boundary of a tissue in a US image of breast cancer. We have a data set of 2D slices of 3D US images and corresponding manually segmented images. Our goal is to employ deep learning to train a neural network so it can automatically segment the US images and extract accurate boundaries.

The data set and the segmented images are provided by Dr. J. Carson and his research group from the Lawson Health Research Institute, London, Ontario, Canada. They are building a photoacoustic tomography (PAT) imaging system and produce PAT images of excised breast lesion. However, in a PAT image, the boundary of a tissue is hard to determine, while such a boundary is clearer in a US image. They expect that the segmentation result of our networks helps to determine the boundary of a tissue in a PAT image.

1.2 Summary of methodology

We build a neural network based on the original U-Net [45] model and the inception network [57]. We refer to our model as IU-Net, from Inception U-Net. We have experimented with U-Net, our IU-Net, and two smaller versions of it, small and mini IU-Net. We also employ an autoencoder in our models.

Our training data set consists of 1912 images of 52 samples, where for each image we have two manual segmentations results. They are performed by two summer students of Dr. Jeffery Carson's research group independently. We choose 358 images of 10 samples as test set, and use data augmentation to enlarge the test set to 8234 images. Then we measure the accuracy of our models on this test set. The images are 2D slices in the z direction and we perform data augmentation on all these images. Then we train our models with augmented training dataset.

To measure the accuracy of our models, we use both the Dice similarity coefficient (DSC) [18, 55] and the mean-squared-error (MSE) metrics. The results are given in Table 1.1, where +encoder means training using an encoder.

model	DSC		MSE		# of parameters
	mean	std	mean	std	
U-Net	91.0%	0.15	2.02×10^{-2}	2.22×10^{-2}	7.76×10^{6}
U-Net+encoder	90.6%	0.16	2.08×10^{-2}	2.29×10^{-2}	7.76×10^{6}
IU-Net	91.5%	0.12	2.04×10^{-2}	2.22×10^{-2}	2.05×10^{6}
IU-Net+encoder	**92.0%**	**0.11**	$\mathbf{2.01 \times 10^{-2}}$	$\mathbf{2.17 \times 10^{-2}}$	2.05×10^{6}
small IU-Net	91.2%	0.14	2.06×10^{-2}	2.20×10^{-2}	5.10×10^{5}
small IU-Net+encoder	90.9%	0.14	2.07×10^{-2}	2.25×10^{-2}	5.10×10^{5}
mini IU-Net	89.7%	0.16	2.40×10^{-2}	2.43×10^{-2}	$\mathbf{1.24 \times 10^{5}}$
mini IU-Net+encoder	89.7%	0.17	2.29×10^{-2}	2.38×10^{-2}	$\mathbf{1.24 \times 10^{5}}$

TABLE 1.1: Accuracy and number of parameters

From Table 1.1, we see that all models achieve similar accuracies. IU-Net+encoder achieves the highest DSC accuracy and the smallest MSE. The number of trainable parameters of mini IU-Net and mini IU-Net+encoder is the smallest among

all models.

Furthermore, we also apply multiple post-processing techniques (largest connected component extraction, small hole filling, boundary extraction) to improve the visualization of boundaries.

1.3 Contribution

In this study, we build a new CNN model (IU-Net) based on the existed model U-Net and inception network. In addition, the number of trainable parameters of IU-Net is smaller than the number of trainable parameters of U-Net. We also train an autoencoder with the manual segmentation results. We use the encoder part to compute the internal representations of the prediction of the network and corresponding manual segmentation. Then we construct a new regularization term in the loss function by computing the differences between the internal representations of the prediction and the corresponding manual segmentation. Our experimental results show that this regularization term can improve the segmentation results and all models (U-Net, IU-Net and its two smaller version) achieve the similar and good accuracies.

1.4 Thesis outline

This thesis is organized as follows. In Chapter 2, we introduce related work. Chapter 3 presents the basic principles and the architecture of convolutional neural networks (CNNs). Chapter 4 introduces our models, training, prediction and post-processing techniques. Chapter 5 shows our experimental results. Finally, Chapter 6 is the conclusion and future research directions.

Chapter 2

Related work

In this chapter, we overview previous breast US images segmentation methods. Various breast US image segmentation approaches have been proposed in the last two decades. Xian et al. [68] classified such methods into six categories: (1) graph-based, (2) deformable models, (3) learning-based, (4) thresholding, (5) region growing, and (6) watershed. According to the statistics from [68], (4), (5) and (6) are not popular in breast US segmentation, we only summarize (1), (2) and (3) here.

Graph-based approaches apply graph algorithms on a graph associated with the image to be segmented. Ashton et al. [3] segmented speckle-laden US images by obtaining the maximum a posteriori (MAP) of the Markov random field (MRF) associated with the images to be segmented. Boukerroui et al. [7] modified the method in [3] and segmented breast tumor US images. Xiao et al. [69] combined MAP and MRF to segment B-mode US images. Xian et al. [67] proposed a breast US image segmentation framework based on graph cuts. Chiang et al. [11] applied graph cuts on sonographic breast images. Aleman-Flores et al. [2] segmented breast US images using normalized cuts [53].

A deformable model (DM) [58] describes a curve (in 2D) or a surface (in 3D),

whose shape can be changed to fit the object boundary under the impact of forces (determined by information from the input image). According to the statistics of Xian et al. [68], DM is the most popular breast US segmentation approach. DMs can be classified into two categories based on different curve (surface) representations: parametric deformable models (PDMs) and geometric deformable models (GDMs). Madabhushi et al. [41] introduced a fully automatic approach for breast cancer US image segmentation using PDMs. Sahiner [50] segmented 3D breast US image with PDMs. Yezzi et al. [72] segmented Magnetic Resonance Imaging (MRI), Computed Tomography (CT) and US with GDMs.

Learning-based approaches are becoming popular in breast cancer US segmentation recently. These approaches are classified into two categories based on whether there is labelled data (ground truth) in training learning models: supervised learning, and unsupervised learning which does not have ground truth.

Boukerroui et al. [7] segmented breast lesion using 2D adaptive k-mean clustering algorithm. Liu et al. [35] trained a support vector machine (SVM) for breast cancer US tumor segmentation. Huang et al. [28] built a neural network to segment 3D breast cancer US images. Huang et al. [27] combined a neural network and watershed to extract the contour of a breast tumor. Xu et al. [71] segmented breast US images into four classes (skin, fibroglandular tissue, mass, and fatty tissue) using CNNs.

In recent years, many deep learning approaches have been introduced to solve the segmentation problem. Here we introduce two classical neural networks for segmentation, fully convolutional network (FCN) [38] and U-Net [45].

First we introduce the architecture of the FCN. The FCN has three different architectures, FCN-32s, FCN-16s and FCN-8s, see Figures 2.1, 2.2 and 2.3. In these Figures, conv refers to a convolution layer; maxpool is a max pooling layer;

for more details of convolution and pooling layers, see Chapter 3; upsample(k) is to enlarge the size of the input image k times; + denotes element-wise addition.

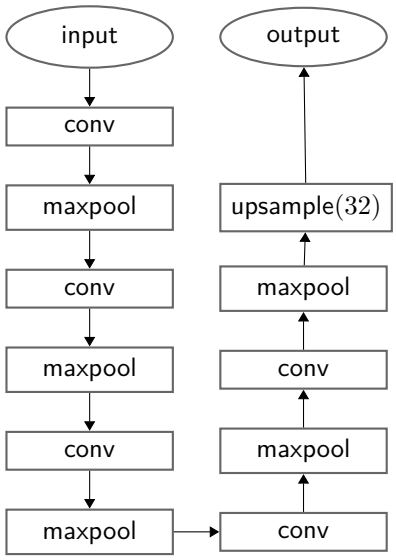


FIGURE 2.1: FCN-32s architecture.

There is no dense (or fully-connected) layer in a FCN. This architecture reduces the number of trainable parameters compared to previous convolutional neural networks, which usually have dense layers. The FCN consists of downsampling path, upsampling path and skip connection (or shortcut). The downsampling path consists of convolution layers and pooling layers, and their purpose is to extract semantic features of input data. The upsampling path consists of upsampling layers, and its purpose is to recover spatial information. The skip connection transfers information from the downsampling path to the upsampling path; this helps recover spatial information that may be lost in the pooling layers.

The architecture of U-Net is based on the FCN and also consists of downsampling path, upsampling path and skip connection, see Figure 2.4, where concat. denotes concatenation. The most significant difference between U-Net and FCN is whether the downsampling path and upsampling path are symmetric. The architecture of U-Net may generate more accurate segmentation results [66] than

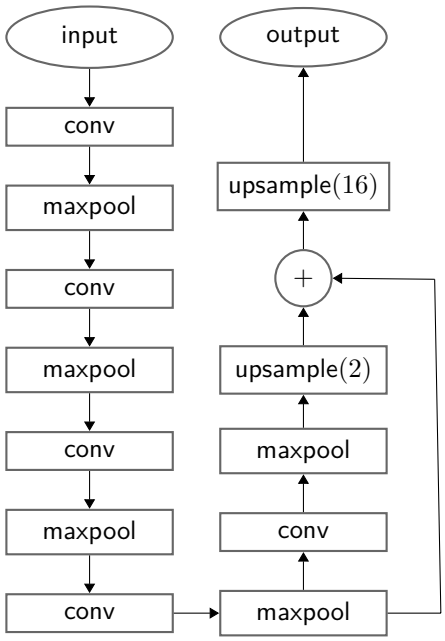


FIGURE 2.2: FCN-16s architecture.

FCN due to the presence of skip connections and convolution layers in upsampling path. The main drawback of U-Net is the large number of trainable parameters. In our research, we modify some parts of U-Net to reduce the number of trainable parameters, see Chapter 4.

Here is the summary of advantages and disadvantages of FCN and U-Net.

FCN has two advantages over previous CNN: FCN has no dense layer, and FCN employs downsampling, upsampling, and skip-connection architectures. The disadvantage of FCN is the coarse segmentation result [73], especially the segmentation result of FCN-32s.

U-Net has three advantages over previous CNN: U-Net has no dense layer; U-Net also applies downsampling, upsampling, and skip-connections; The result of U-Net may be finer than FCN [66], since U-Net has more convolution layers in the upsampling path and more skip connections between upsampling and downsampling. However, U-Net has a obvious disadvantage: U-Net has a large number of

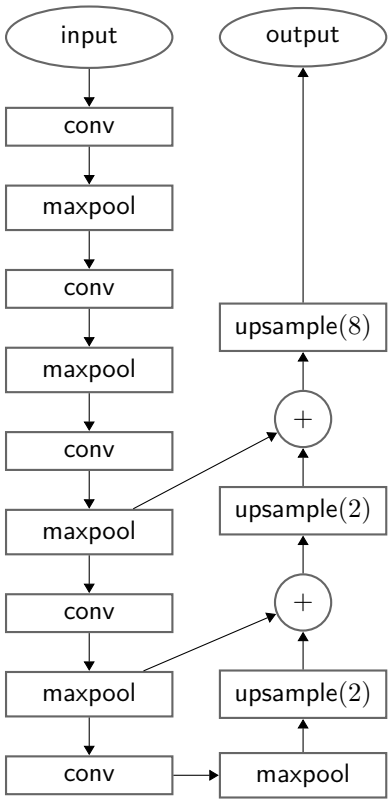


FIGURE 2.3: FCN-8s architecture.

trainable parameters.

There are also other segmentation networks. Badrinarayanan et al. [5] built a convolutional encoder-decoder architecture SEGNET. Luc et al. [39] proposed a segmentation network using adversarial networks [24]. Visin et al. [63] combined recurrent neural networks (RNNs) and CNNs to do segmentation. LaLonde et al. [33] introduced a segmentation network based on capsule networks [49].

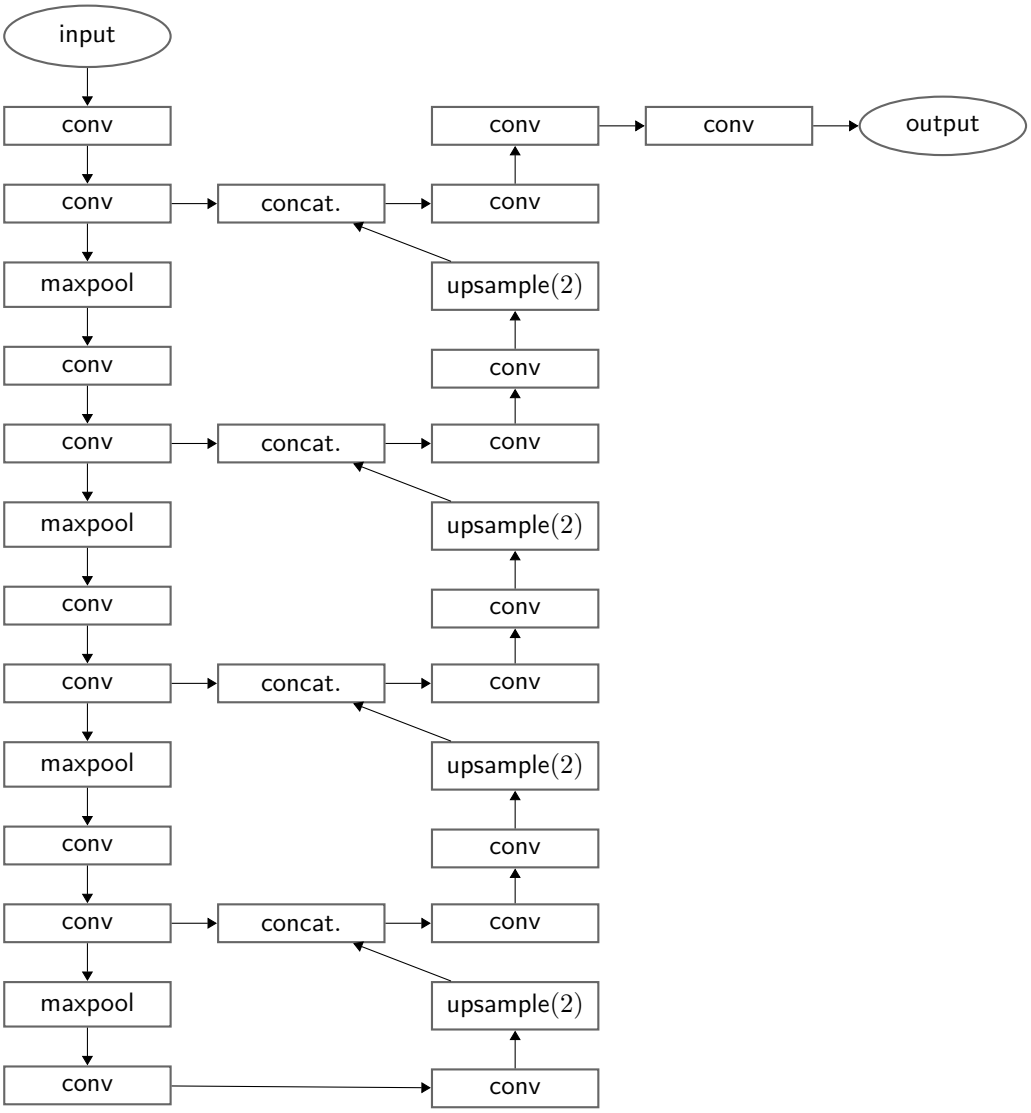


FIGURE 2.4: U-Net architecture.

Chapter 3

Background

This chapter introduces some of the history and theory underlying CNNs. Section 3.1 gives a brief introduction to the history of CNNs and lists some medical applications of CNNs. Section 3.2 overviews underlying theory. Section 3.3 describes convolution and pooling layers in a CNN. Section 3.4 presents the inception block which is used in our models. Section 3.5 presents the back propagation algorithm, which is one of the most important algorithms in training neural networks. Section 3.6 explains how to update the trainable parameters of a CNN after back propagation.

3.1 History of CNNs

The research on CNNs developed rapidly in recent years due to the development of computer hardware such as graphics processing units (GPUs). Lecun et al. [34] developed a CNN model LeNet-5 to classify handwritten characters. LeNet-5 is a small network compared to recent CNNs. Chellapilla et al. [10] trained a CNN using an NVIDIA GeForce 7800 Ultra graphics card and obtained 3-4× speedup compared to without GPUs. Krizhevsky et al. [32] built a deep CNN AlexNet to classify 1.2 million high-resolution images in the ImageNet LSVRC-2010 contest [6]

into 1000 different classes. The architecture of ALEXNET is similar to LENET-5, but ALEXNET is deeper and has a larger number of trainable parameters than LENET-5. Simonyan et al. [54] proposed a deeper CNN model VGG-16 (compared to ALEXNET), which achieved higher accuracy than ALEXNET. Szegedy et al. [57] built an inception network. He et al. [25] proposed a CNN architecture residual network (RESNET). The experimental results show that RESNET is easier to train than previous neural networks and RESNET achieves the highest accuracy on the ImageNet LSVRC-2015 contest. Veit et al. [62] illustrated that RESNET can be viewed as a collection of many relatively shallow networks.

Many researchers have applied CNNs to medical applications. Ciresan et al. [14] built a 13-layer CNN and an 11-layer CNN to detect mitosis in breast histology images. Roth et al. [46] presented a method for organ anatomical classification of medical images using an 11-layer CNN. Pereira et al. [44] studied CNNs for brain tumor segmentation using magnetic-resonance images. Fakoor et al. [21] applied deep learning to detect and classify cancer types based on gene expression data. Su et al. [56] did region segmentation in histopathological breast cancer images using a simplified ALEXNET.

3.2 Fundamental knowledge

In this section, we illustrate the theory of neural networks following a simple example from [26]. The architecture of this simple network is given in Figure 3.1. This network is designed to classify two-dimensional points into two classes.

The basic concepts of neural networks are as follows.

Unit: Unit is the basic component of a neural network, and each unit has an input and an output. In Figure 3.1, each circle refers to a unit.

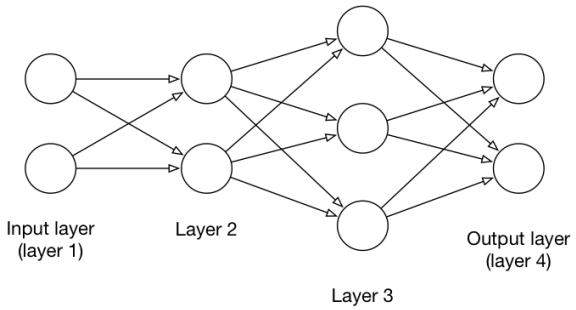


FIGURE 3.1: A simple neural network

Layer: There are various types of layers in neural networks. Different layers have different architectures and have different ways to compute the output of units in a layer. In this example, the network uses fully-connected layers, that is, every unit in layer $l+1$ is connected with all units in layer l, for $l = 1, 2, 3$. Here, the input of each unit in layer $l+1$ is a weighted sum of the outputs of all units in layer l. The neural network in Figure 3.1 has four layers (input layer, layer 2, layer 3 and output layer).

Throughout this thesis we denote the number of layers in a network by L.

Weight $W_{ij}^{(l)}$: weight $W_{ij}^{(l)}$ describes how much influence unit j in layer $l-1$ has on unit i in layer l.

Weights matrix $W^{(l)}$: $W^{(l)} = \left(W_{ij}^{(l)}\right) \in \mathbb{R}^{N_l \times N_{l-1}}$, where N_{l-1} and N_l are the number of units in layer $l-1$ and layer l, respectively.

Bias $b^{(l)}$: $b^{(l)} \in \mathbb{R}^{N_l}$, where N_l is the number of units in layer l. To be specific, $b_i^{(l)}$ describes how much influence layer $l-1$ has on unit i in layer l, regardless of the outputs of units in layer $l-1$.

Activation function: The output of a unit is determined by the input and activation function which is normally nonlinear. The sigmoid function $\sigma(x) = 1/(1+e^{-x})$ and the rectified linear unit (ReLU), $\text{ReLU}(x) = \max(x, 0)$

are common choices. Denote by f an activation function. For a vector x, $(f(x))_i = f(x_i)$.

Training dataset: Training dataset consists of pairs of an input vector and an target output vector, that is, $\{(x^{(1)}, y^{(1)}), ..., (x^{(M)}, y^{(M)})\}$, where $x^{(k)}$ is the kth input vector; $y^{(k)}$ is the kth target output vector; The target output vector is also named as the ground truth (GT). In this example, $x^{(k)} \in \mathbb{R}^2$, $y^{(k)} \in \mathbb{R}^2$.

Forward propagation: To produce the output of a neural network, we have to compute the outputs of units layer by layer. This process is forward propagation.

Cost function J: $J = \frac{1}{M}\sum_{k=1}^{M} \mathcal{L}(\widehat{y}^{(k)}, y^{(k)})$ and $\mathcal{L} : \mathbb{R}^{N_L} \times \mathbb{R}^{N_L} \to \mathbb{R}$ measures the difference between the prediction of a neural network and its corresponding ground truth (GT); N_L is the number of units in the output layer. In this example, $N_L = 2$; $\widehat{y}^{(k)}$ is the kth prediction; $y^{(k)}$ is the GT of $\widehat{y}^{(k)}$.

Trainable parameter: Trainable parameter refers to the parameter which will be updated by gradient descent or other optimization algorithms in training a neural network. In this example, $W_{ij}^{(l)}$, $b_i^{(l)}$ are trainable parameters.

Gradient descent: Gradient descent [47] is an iterative optimization algorithm to find the local minimum of the cost function. The purpose of training a neural network is to compute

$$\underset{w}{\operatorname{argmin}}\, J = \underset{w}{\operatorname{argmin}}\, \frac{1}{M}\sum_{k=1}^{M} \mathcal{L}\left(\widehat{y}^{(k)}, y^{(k)}\right),$$

where $w = (w_i) \in \mathbb{R}^{n_p}$ (n_p is the number of trainable parameters) is a vector containing all trainable parameters of the network. To achieve this goal, gradient descent is a common choice. In each iteration of the gradient descent,

we compute the partial derivatives of the cost function J with respect to each trainable parameter $\frac{\partial J}{\partial w_i}$ and then update this parameter by $w_i \leftarrow w_i - \alpha \frac{\partial J}{\partial w_i}$, where $\alpha > 0$ is learning rate. See Algorithm 1 for how gradient descent works in this example.

Back propagation: With gradient descent or some other optimization algorithms applied in deep learning, we need to compute the partial derivatives of the loss function with respect to all trainable parameters. Back propagation [48] is an efficient way to calculate these partial derivatives. In this example, back propagation is to compute $\frac{\partial \mathcal{L}}{\partial W_{ij}^{(l)}}$ and $\frac{\partial \mathcal{L}}{\partial b_i^{(l)}}$, $l = 2, 3, 4$.

The process of training the neural network in Figure 3.1 with gradient descent is given in Algorithm 1.

Algorithm 1 Training with gradient descent

Input:
$W^{(l)}$, $b^{(l)}$: initial values for trainable parameters
n_{iter}: number of iterations
α: learning rate
training dataset: $\{(x^{(1)}, y^{(1)}), ..., (x^{(M)}, y^{(M)})\}$
for iter $= 1 : n_{\text{iter}}$ **do**
  for $k = 1 : M$ **do**
    do forward propagation on $x^{(k)}$
    compute $\frac{\partial \mathcal{L}}{\partial W_{ij}^{(l)}}$ and $\frac{\partial \mathcal{L}}{\partial b_i^{(l)}}$ by back propagation, $l = 2, 3, 4$.
  compute $\frac{\partial J}{\partial W_{ij}^{(l)}}$ and $\frac{\partial J}{\partial b_i^{(l)}}$, $l = 2, 3, 4$.
  update $W^{(l)}$ and $b^{(l)}$ by gradient descent, that is,
  $W_{ij}^{(l)} \leftarrow W_{ij}^{(l)} - \alpha \frac{\partial J}{\partial W_{ij}^{(l)}}$, $b_i^{(l)} \leftarrow b_i^{(l)} - \alpha \frac{\partial J}{\partial b_i^{(l)}}$, $l = 2, 3, 4$.

The key part of Algorithm 1 is the forward and back propagation for each training data. Here we explain how to perform forward propagation on an input data x, and how to compute $\frac{\partial \mathcal{L}}{\partial W_{ij}^{(l)}}$ and $\frac{\partial \mathcal{L}}{\partial b_i^{(l)}}$ by back propagation, $l = 2, 3, 4$.

First we perform forward propagation for an input $x = (x_i) \in \mathbb{R}^2$ of this network and $y = (y_i) \in \mathbb{R}^2$ is the GT, where x_i is the input of the ith unit in the input layer; y_i is the target output of the ith unit in the output layer.

To compute the outputs of units in layer 2:

$$\begin{aligned} a_1^{(2)} &= f\left(W_{11}^{(2)} x_1 + W_{12}^{(2)} x_2 + b_1^{(2)}\right) \\ a_2^{(2)} &= f\left(W_{21}^{(2)} x_1 + W_{22}^{(2)} x_2 + b_2^{(2)}\right). \end{aligned}$$

To compute the outputs of units in layer 3:

$$\begin{aligned} a_1^{(3)} &= f\left(W_{11}^{(3)} a_1^{(2)} + W_{12}^{(3)} a_2^{(2)} + b_1^{(3)}\right) \\ a_2^{(3)} &= f\left(W_{21}^{(3)} a_1^{(2)} + W_{22}^{(3)} a_2^{(2)} + b_2^{(3)}\right) \\ a_3^{(3)} &= f\left(W_{31}^{(3)} a_1^{(2)} + W_{32}^{(3)} a_2^{(2)} + b_3^{(3)}\right). \end{aligned}$$

To compute the outputs of units in layer 4, output layer:

$$\begin{aligned} \widehat{y}_1 &= f\left(W_{11}^{(4)} a_1^{(3)} + W_{12}^{(4)} a_2^{(3)} + W_{13}^{(4)} a_3^{(3)} + b_1^{(4)}\right) \\ \widehat{y}_2 &= f\left(W_{21}^{(4)} a_1^{(3)} + W_{22}^{(4)} a_2^{(3)} + W_{23}^{(4)} a_3^{(3)} + b_2^{(4)}\right). \end{aligned}$$

In vector form:

$$\begin{aligned} a^{(2)} &= f\left(W^{(2)} x + b^{(2)}\right) \\ a^{(3)} &= f\left(W^{(3)} a^{(2)} + b^{(3)}\right) \\ \widehat{y} &= f\left(W^{(4)} a^{(3)} + b^{(4)}\right), \end{aligned}$$

where $W^{(2)} \in \mathbb{R}^{2\times 2}$; $W^{(3)} \in \mathbb{R}^{3\times 2}$; $W^{(4)} \in \mathbb{R}^{2\times 3}$; $b^{(2)} \in \mathbb{R}^2$; $b^{(3)} \in \mathbb{R}^3$; $b^{(4)} \in \mathbb{R}^2$.

Then we compute the partial derivatives of the loss function $\mathcal{L}$ with respect to $W_{ij}^{(l)}$ and $b_i^{(l)}$, $\frac{\partial \mathcal{L}}{\partial W_{ij}^{(l)}}$ and $\frac{\partial \mathcal{L}}{\partial b_i^{(l)}}$ $(l = 2, 3, 4)$ by back propagation.

Denote

$$z^{(l)} = W^{(l)} a^{(l-1)} + b^{(l)}, \text{ for } l = 2, 3, ..., L$$

where $a^{(1)} = x$ for consistent notation; L is the number of layers.

Let

$$\delta_i^{(l)} = \frac{\partial \mathcal{L}}{\partial z_i^{(l)}}, \text{ for } l = 2, 3, ..., L \text{ and } i = 1, ..., N_l$$

This expression is important in the computation of $\frac{\partial \mathcal{L}}{\partial W_{ij}^{(l)}}$ and $\frac{\partial \mathcal{L}}{\partial b_i^{(l)}}$.

Below, $\circ$ is the Hadamard product (element-wise multiplication).

The lemma from [26] gives

$$\begin{aligned}
\delta^{(L)} &= \frac{\partial \mathcal{L}}{\partial \widehat{y}} \circ f'(z^{(L)}) \\
\delta^{(l)} &= \left(W^{(l+1)}\right)^T \delta^{(l+1)} \circ f'(z^{(l)}), \text{ for } l = 2, ..., L-1 \\
\frac{\partial \mathcal{L}}{\partial b_i^{(l)}} &= \delta_i^{(l)}, \text{ for } l = 2, ..., L \qquad (3.1) \\
\frac{\partial \mathcal{L}}{\partial W_{ij}^{(l)}} &= \delta_i^{(l)} a_j^{(l-1)}, \text{ for } l = 2, ..., L, \qquad (3.2)
\end{aligned}$$

where $\delta^{(l)} = \left(\delta_i^{(l)}\right) \in \mathbb{R}^{N_l}$.

We define

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial b^{(l)}} &= \left(\frac{\partial \mathcal{L}}{\partial b_i^{(l)}}\right) \in \mathbb{R}^{N_l} \\
\frac{\partial \mathcal{L}}{\partial W^{(l)}} &= \left(\frac{\partial \mathcal{L}}{\partial W_{ij}^{(l)}}\right) \in \mathbb{R}^{N_l \times N_{l-1}},
\end{aligned}$$

where $l = 2, 3, 4$.

With above definition, we rewrite equations 3.1 and 3.2 in vector and matrix forms:

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial b^{(l)}} &= \delta^{(l)}, \text{ for } l = 2, ..., L \\
\frac{\partial \mathcal{L}}{\partial W^{(l)}} &= \delta^{(l)} \left(a^{(l-1)}\right)^T, \text{ for } l = 2, ..., L,
\end{aligned}$$

These forms are useful in the implementation of back propagation of this example, see [26] for the implementation code.

Therefore, we compute $\frac{\partial \mathcal{L}}{\partial W^{(l)}}$ and $\frac{\partial \mathcal{L}}{\partial b^{(l)}}$, $l = 2, 3, 4$ using above equations,

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial b^{(4)}} &= \delta^{(4)} = \frac{\partial \mathcal{L}}{\partial \widehat{y}} \circ f'(z^{(4)}) \\
\frac{\partial \mathcal{L}}{\partial W^{(4)}} &= \delta^{(4)} \left(a^{(3)}\right)^T \\
\frac{\partial \mathcal{L}}{\partial b^{(3)}} &= \delta^{(3)} = \left(W_{ij}^{(4)}\right)^T \delta^{(4)} \circ f'(z^{(3)}) \\
\frac{\partial \mathcal{L}}{\partial W^{(3)}} &= \delta^{(3)} \left(a^{(2)}\right)^T \\
\frac{\partial \mathcal{L}}{\partial b^{(2)}} &= \delta^{(2)} = \left(W^{(3)}\right)^T \delta^{(3)} \circ f'(z^{(2)}) \\
\frac{\partial \mathcal{L}}{\partial W^{(2)}} &= \delta^{(2)} x^T.
\end{aligned}$$

The basic principles of CNNs are similar with this simple network, but the architecture of a CNN is much more complex and suitable for image analysis (such as classification, segmentation and localization). Generally speaking, a CNN consists of convolution layers, pooling layers, batch normalization layers, etc. We introduce how these layers work in the next section.

3.3 Layers in CNNs

In practice, the input data of convolution and pooling layers is regarded as a 3D tensor [26]. The input data is stacked by D matrices of size $H \times W$, where H, W and D refer to the width, height and depth (or the number of slices) of the input, respectively. When $D = 1$, the input data is a matrix. An example of input data X with $H = 3$, $W = 3$ and $D = 2$ is given as follows.

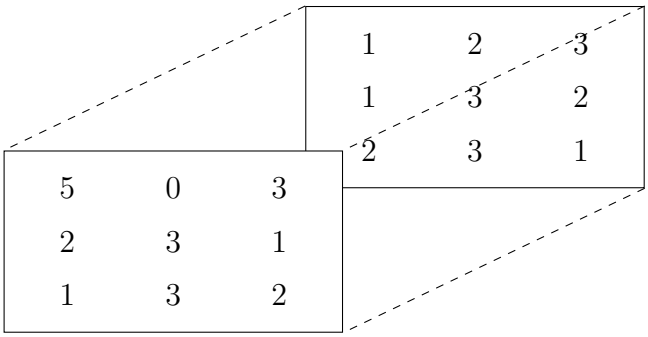


FIGURE 3.2: Input data of convolution and pooling layers

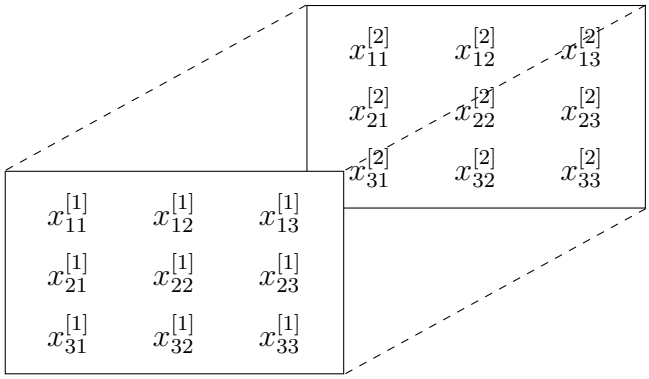
In Figure 3.2, the input data has two slices: the first slice is $\begin{bmatrix} 5 & 0 & 3 \\ 2 & 3 & 1 \\ 1 & 3 & 2 \end{bmatrix}$ and the second slice is $\begin{bmatrix} 1 & 2 & 3 \\ 1 & 3 & 2 \\ 2 & 3 & 1 \end{bmatrix}$.

For convenience, we denote $x_{ij}^{[s]}$ as the (i, j) element of the sth slice of X. In the example above, $x_{11}^{[1]} = 5$, $x_{12}^{[1]} = 0$, $x_{13}^{[1]} = 3$.

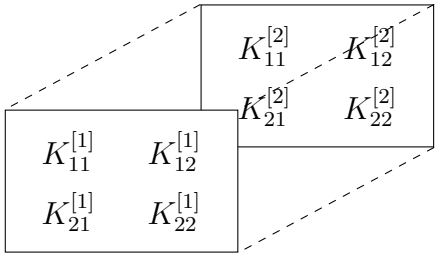
3.3.1 Convolution and pooling layers

The purpose of a convolution layer is to apply small filters (or kernels, where the weights in the kernels are trainable) across all the input data. Here we illustrate how a convolution layer works through a simple example.

Let the input data X with $H = 3$, $W = 3$ and $D = 2$ be



and consider a kernel K



Notice that the depth of filter must be equal to the depth of input data. In this example, the depth of the input data is 2, so the depth of filter is also 2.

The kernel starts from the top-left corner of the input data, and we do pixel-wise multiplication between the kernel and the part of the input data where the filter covers; then we sum up these multiplications and add bias b,

$$y^{[1]}_{11} = K^{[1]}_{11}x^{[1]}_{11} + K^{[1]}_{12}x^{[1]}_{12} + K^{[1]}_{21}x^{[1]}_{21} + K^{[1]}_{22}x^{[1]}_{22} + K^{[2]}_{11}x^{[2]}_{11} + K^{[2]}_{12}x^{[2]}_{12} + K^{[2]}_{21}x^{[2]}_{21} + K^{[2]}_{22}x^{[2]}_{22} + b,$$

where $y^{[1]}_{11}$ is the top-left value of the first slice of the output.

We move the filter and repeat above procedure and obtain all values of the first slice in the output.

$$y^{[1]}_{12} = K^{[1]}_{11}x^{[1]}_{12} + K^{[1]}_{12}x^{[1]}_{13} + K^{[1]}_{21}x^{[1]}_{22} + K^{[1]}_{22}x^{[1]}_{23} + K^{[2]}_{11}x^{[2]}_{12} + K^{[2]}_{12}x^{[2]}_{13} + K^{[2]}_{21}x^{[2]}_{22} + K^{[2]}_{22}x^{[2]}_{23} + b$$

$$y_{21}^{[1]} = K_{11}^{[1]}x_{21}^{[1]} + K_{12}^{[1]}x_{22}^{[1]} + K_{21}^{[1]}x_{31}^{[1]} + K_{22}^{[1]}x_{32}^{[1]} + K_{11}^{[2]}x_{21}^{[2]} + K_{12}^{[2]}x_{22}^{[2]} + K_{21}^{[2]}x_{31}^{[2]} + K_{22}^{[2]}x_{32}^{[2]} + b$$

$$y_{22}^{[1]} = K_{11}^{[1]}x_{22}^{[1]} + K_{12}^{[1]}x_{23}^{[1]} + K_{21}^{[1]}x_{32}^{[1]} + K_{22}^{[1]}x_{33}^{[1]} + K_{11}^{[2]}x_{22}^{[2]} + K_{12}^{[2]}x_{23}^{[2]} + K_{21}^{[2]}x_{32}^{[2]} + K_{22}^{[2]}x_{33}^{[2]} + b$$

So far we obtain the first slice of the output data, $y^{[1]}$. From the above equations, we summarize the computation of $y^{[1]}$ as:

$$y_{ij}^{[1]} = \sum_{s=1}^{2}\sum_{m=1}^{2}\sum_{n=1}^{2} x_{i+m-1\ j+n-1}^{[s]} K_{mn}^{[s]} + b.$$

Then we apply another kernel on the input data and obtain the second slice of the output, $y^{[2]}$. If we employ n_f filters in a convolution layer, the output data has n_f slices.

Here are some important parameters in a convolution layer.

Filter (kernel) size: In a convolution layer, a filter K is also a 3D tensor. $K \in \mathbb{R}^{H_f \times W_f \times D_f}$, where $H_f \times W_f \times D_f$ is the filter size; H_f, W_f and D_f refer to the width, height and depth of the filter, respectively. Normally, $H_f = W_f$ and they are small integers such as 1, 2, 3. In above example, $H_f = W_f = 2$. In addtion, $D_f = D$.

Stride (S_H, S_W): S_H and S_W refer to the number of pixels we move a filter at a time in vertical and horizontal directions, respectively. In the example above, $S_H = S_W = 1$.

Number of filters n_f: We sometimes apply multiple filters in one convolution layer since we expect one convolution layer to extract various features of the input data.

Padding p: p refers to the number of elements we pad around the input data. Since a convolution without padding always reduces the width and the height of the input data, we usually keep the width and the height of the output

data the same with the width and the height of input data by padding around the input before a convolution operation.

The number of trainable parameters of a convolution layer is $H_f \times W_f \times D \times n_f + n_f = (H_f W_f D + 1) n_f$.

The relation between the size of the output and the size of the input of a convolution layer is given as follow:

$$\begin{aligned} H' &= \left\lfloor \frac{H + 2p - H_f}{S_H} + 1 \right\rfloor \\ W' &= \left\lfloor \frac{W + 2p - W_f}{S_W} + 1 \right\rfloor \\ D' &= n_f, \end{aligned}$$

where $\lfloor \cdot \rfloor$ is the floor function; $H \times W \times D$ is the size of the input data and $H' \times W' \times D'$ denotes the size of the output data.

Compared to a fully-connected (FC) layer, a convolution layer has its own advantages.

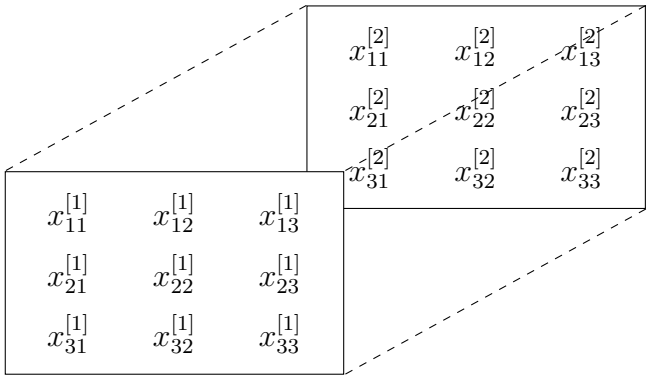
First, the number of trainable parameters of a convolution layer is smaller than the number of parameters in a FC layer. For example, assume that the input size of a convolution layer and a FC layer is $128 \times 128 \times 1$ and the output size is $126 \times 126 \times 16$. If we use a FC layer, the number of trainable parameters is $n_{\text{FC}} = (128 \times 128 \times 1) \times (126 \times 126 \times 16) + 126 \times 126 \times 16 \approx 4.16 \times 10^9$. If we employ a convolution layer with 16 filters of size 3×3, stride $(1, 1)$ and no zero padding, the number of trainable parameters is $n_{\text{conv}} = (3 \times 3 \times 1 + 1) \times 16 = 160$. That is, $n_{\text{FC}} >> n_{\text{conv}}$.

In addition, every slice of the output data is obtained by the convolution of different filters, that is, each filter is to extract specific feature of the input data.

Since we usually employ multiple filters in a convolution layer, a convolution layer may extract multiple features at the same time.

In general, a pooling layer follows a convolution layer. One of main purposes of a pooling layer is to perform downsampling on the input, that is, to reduce H and W of the input data. Here we illustrate how a pooling layer works by a simple example.

Let the input data X be



and consider 2×2 pooling.

The pooling layer operates independently on every depth of the input data. First we do pooling on the first slice of the input data, $x^{[1]}$. The kernel starts from the top-left corner of $x^{[1]}$ and maps the region where the kernel covers to a single number, that is,

$$y_{11}^{[1]} = \text{pool}(x_{11}^{[1]}, x_{12}^{[1]}, x_{21}^{[1]}, x_{22}^{[1]}),$$

where the pool function is determined by the type of the pooling layer. If we employ a max pooling layer, the pool function is to find the largest number among $x_{11}^{[1]}$, $x_{12}^{[1]}$, $x_{21}^{[1]}$ and $x_{22}^{[1]}$.

We move the filter and repeat the above procedure, and then obtain the first slice in the output, $y^{[1]}$. Then we do pooling on the second slice of the input data

and obtain the second slice of the output, $y^{[2]}$. We repeat the above procedure to obtain the entire output data.

The size of the output of a pooling layer is:

$$\begin{aligned} H' &= \left\lfloor \frac{H + 2p - H_p}{S_H} + 1 \right\rfloor \\ W' &= \left\lfloor \frac{W + 2p - W_p}{S_W} + 1 \right\rfloor \\ D' &= D, \end{aligned}$$

where $H_p \times W_p$ is the pooling size. In the example above, $H_p = W_p = 2$. The most common setting of a pooling layer is $H_p = W_p = 2$, $S_H = S_W = 2$, $p = 0$. With this setting, we have $H' = \left\lfloor \frac{H}{2} \right\rfloor$ and $W' = \left\lfloor \frac{W}{2} \right\rfloor$.

Another purpose of a pooling layer is to extract position invariant features. In the example above, when we do max pooling on $x_{11}^{[1]}$, $x_{12}^{[1]}$, $x_{21}^{[1]}$, $x_{22}^{[1]}$ and obtain $y_{11}^{[1]}$, pooling layer extracts the dominant feature $y_{11}^{[1]}$ from $x_{11}^{[1]}$, $x_{12}^{[1]}$, $x_{21}^{[1]}$, $x_{22}^{[1]}$ irrespective of the position of $x_{11}^{[1]}$, $x_{12}^{[1]}$, $x_{21}^{[1]}$, $x_{22}^{[1]}$. In addition, pooling layer is also used to reduce the height and width of input data.

3.3.2 Batch normalization Layer

Batch normalization (BN) [29] is used to normalize and rescale the input data. The computation of BN's output is shown as follows.

Assuming BN's inputs are $x_1, x_2, ..., x_m$,

$$\begin{aligned}
\mu &= \frac{1}{m}\sum_{i=1}^{m} x_i \\
\sigma^2 &= \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu)^2 \\
x_i^{\text{norm}} &= \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}} \\
x_i^{\text{BN}} &= \gamma x_i^{\text{norm}} + \beta,
\end{aligned}$$

where γ and β are trainable parameters; ϵ is a small number (e.g. $\epsilon = 10^{-5}$) to avoid the denominator to become zero; x_i^{BN} is the output of a BN layer.

BNs are widely used in deep learning, but the underlying theory of BNs' effectiveness is still poorly understood [51]. Currently, a popular explanation is that BNs can reduce the internal covariate shift (ICS), where ICS refers to the change in the distribution of the input of internal layers (hidden layers) in a network. This change is caused by the update of trainable parameters in the network. In addition, reducing ICS is also the original purpose of BNs [29]. The experiment results in [29] show that BNs reduce the ICS and improve accuracy. However, Santurkar et al. [51] demonstrated that the effectiveness of BNs has little to do with the reduction of ICS. Santurkar et al. proved that the BSs can make the landscape of optimization smoother and gradients are more predictive (that is, the l_2 norms of gradients change in a smaller range) with BNs and thus we can use a higher learning rate and speed up the convergence.

Some researchers applied BNs in their models and their experimental results prove the effectiveness of BNs. Liu et al. [36] detected P300 signals from electroencephalography using a CNN containing BNs. When BNs are removed, the performance of CNN is reduced considerably. Wang et al. [64] add into the CNN

architecture of Zhang et al. [74] and the experimental results show that the performance of CNN is improved.

3.4 Inception block

One drawback of a convolution layer is: a convolution layer can only adopt filters with the same size, such as 1×1, 2×2 and 3×3. In practice, we do not know which filter size is a good choice. One solution is to employ filters with different sizes at the same time.

The purpose of an inception [57] block is to extract features of input data in four branches with different filter sizes (e.g. 1×1, 3×3, ...) and then concatenate the outputs of the four branches along the direction of depth to obtain the result of an inception block.

The inception block in our research in given in Figure 3.3. Here $\mathsf{conv}(f_s, n_f)$ denotes a 2D convolution layer with size $f_s \times f_s$, stride 1, zero padding, and number of filters n_f; batch norm. is a batch normalization layer; $\mathsf{avepool}(f_s)$ denotes average pooling with pool size of $f_s \times f_s$ and zero padding; and concat. is a concatenation layer. In our inception block, we employ 1×1 and 3×3 convolution layers with stride 1 and zero padding to ensure the output size the same as the input size, and these convolution layers are followed by batch normalization layers (except the 4th branch). Each convolution layer has the same number of filters, denoted here by n_f. We use ReLU as activation function.

In addition, the number of trainable parameters of the inception block in our research is smaller than the double 3×3 2D convolution layers in U-Net [45]. To explain this, we assume the size of the input to an inception block to be $H \times W \times D$ and the size of the output to be $H' \times W' \times D'$.

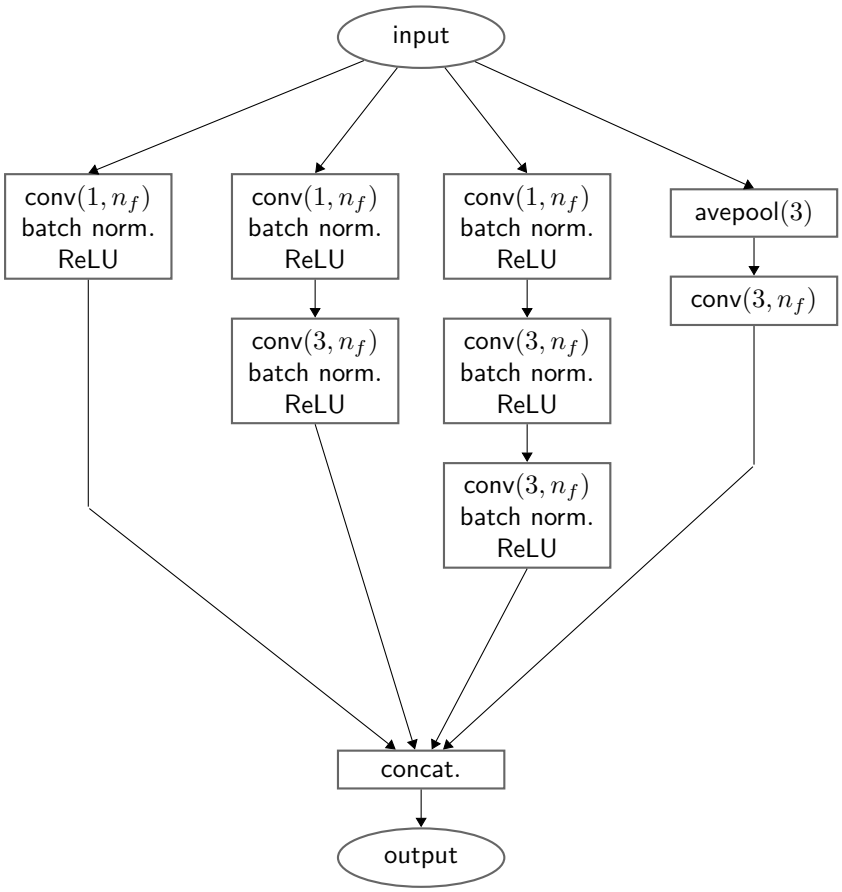


FIGURE 3.3: Inception block structure. n_f denotes number of filters.

For the double 3×3 convolution layers in U-Net, the number of trainable parameters is $(3 \times 3 \times D + 1) \times D' + (3 \times 3 \times D' + 1) \times D' = 9DD' + 9D'^2 + 2D'$.

For the inception block, since the depth of output is D', $n_f = \frac{D'}{4}$. Since the structure of inception is complex, we compute the number of trainable parameters branch by branch.

The number of trainable parameters of the first branch is

$$(1 \times 1 \times D + 1) \times \frac{D'}{4} = \frac{1}{4}DD' + \frac{D'}{4}.$$

The number of trainable parameters of the second branch is

$$(1 \times 1 \times D + 1) \times \frac{D'}{4} + (3 \times 3 \times \frac{D'}{4} + 1) \times \frac{D'}{4} = \frac{1}{4}DD' + \frac{9}{16}D'^2 + \frac{D'}{2}.$$

The number of trainable parameters of the third branch is

$$(1\times1\times D+1)\times\frac{D'}{4}+(3\times3\times\frac{D'}{4}+1)\times\frac{D'}{4}+(3\times3\times\frac{D'}{4}+1)\times\frac{D'}{4} = \frac{1}{4}DD'+\frac{9}{8}D'^2+\frac{3}{4}D'.$$

The number of trainable parameters of the fourth branch is

$$(3 \times 3 \times D + 1) \times \frac{D'}{4} = \frac{9}{4}DD' + \frac{D'}{4}.$$

Therefore, the total number of trainable parameters of our inception block is

$$3DD' + \frac{27}{16}D'^2 + \frac{7}{4}D'.$$

Obviously, $9DD' + 9D'^2 + 2D' > 3DD' + \frac{27}{16}D'^2 + \frac{7}{4}D'$.

For the above reasons, we replace the double 3×3 2D convolution layers by inception blocks to construct our models, see Chapter 4. However, the advantages of the inception can not guarantee that it certainly achieves higher accuracy than the convolution layer does.

3.5 Back propagation

In this chapter, we describe how back propagation performs on CNNs through a small CNN example.

First, we introduce how back propagation works in a convolution layer. For concreteness of our exposition, let the input data be

$$X = \begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix},$$

let the filter be

$$K = \begin{bmatrix} K_{11} & K_{12} \\ K_{21} & K_{22} \end{bmatrix},$$

and denote the output by

$$Y = \begin{bmatrix} y_{11} & y_{12} \\ y_{21} & y_{22} \end{bmatrix}.$$

We compute

$$\begin{aligned} y_{11} &= K_{11}x_{11} + K_{12}x_{12} + K_{21}x_{21} + K_{22}x_{22} \\ y_{12} &= K_{11}x_{12} + K_{12}x_{13} + K_{21}x_{22} + K_{22}x_{23} \\ y_{21} &= K_{11}x_{21} + K_{12}x_{22} + K_{21}x_{31} + K_{22}x_{32} \\ y_{22} &= K_{11}x_{22} + K_{12}x_{23} + K_{21}x_{32} + K_{22}x_{33}. \end{aligned}$$

Suppose we have computed the partial derivative of a cost function $\mathcal{L}$ with respect to y_{ij}, $\frac{\partial \mathcal{L}}{\partial y_{ij}}$. Applying the chain rule,

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial K_{11}} &= \frac{\partial \mathcal{L}}{\partial y_{11}}x_{11} + \frac{\partial \mathcal{L}}{\partial y_{12}}x_{12} + \frac{\partial \mathcal{L}}{\partial y_{21}}x_{21} + \frac{\partial \mathcal{L}}{\partial y_{22}}x_{22} \\ \frac{\partial \mathcal{L}}{\partial K_{12}} &= \frac{\partial \mathcal{L}}{\partial y_{11}}x_{12} + \frac{\partial \mathcal{L}}{\partial y_{12}}x_{13} + \frac{\partial \mathcal{L}}{\partial y_{21}}x_{22} + \frac{\partial \mathcal{L}}{\partial y_{22}}x_{23} \\ \frac{\partial \mathcal{L}}{\partial K_{21}} &= \frac{\partial \mathcal{L}}{\partial y_{11}}x_{21} + \frac{\partial \mathcal{L}}{\partial y_{12}}x_{22} + \frac{\partial \mathcal{L}}{\partial y_{21}}x_{31} + \frac{\partial \mathcal{L}}{\partial y_{22}}x_{32} \\ \frac{\partial \mathcal{L}}{\partial K_{22}} &= \frac{\partial \mathcal{L}}{\partial y_{11}}x_{22} + \frac{\partial \mathcal{L}}{\partial y_{12}}x_{23} + \frac{\partial \mathcal{L}}{\partial y_{21}}x_{32} + \frac{\partial \mathcal{L}}{\partial y_{22}}x_{33}. \end{aligned}$$

In matrix form:

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial K} &= \begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix} * \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial y_{11}} & \frac{\partial \mathcal{L}}{\partial y_{12}} \\ \frac{\partial \mathcal{L}}{\partial y_{21}} & \frac{\partial \mathcal{L}}{\partial y_{22}} \end{bmatrix} \\ &= X * \frac{\partial \mathcal{L}}{\partial Y},\end{aligned}$$

where $*$ refers to the convolution operator.

Next we introduce how back propagation works in a max pooling layer. For concreteness of our exposition, let the input data be

$$X = \begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} \\ x_{21} & x_{22} & x_{23} & x_{24} \\ x_{31} & x_{32} & x_{33} & x_{34} \\ x_{41} & x_{42} & x_{43} & x_{44} \end{bmatrix},$$

let $H_f = W_f = 2$ and $S_H = S_W = 2$, and denote the output by $Y = \begin{bmatrix} y_{11} & y_{12} \\ y_{21} & y_{22} \end{bmatrix}$.

We have

$$\begin{aligned} y_{11} &= \max(x_{11}, x_{12}, x_{21}, x_{22}) \\ y_{12} &= \max(x_{13}, x_{14}, x_{23}, x_{24}) \\ y_{21} &= \max(x_{31}, x_{32}, x_{41}, x_{42}) \\ y_{22} &= \max(x_{33}, x_{34}, x_{43}, x_{44}). \end{aligned}$$

For convenience, here we assume x_{11}, x_{13}, x_{31} and x_{33} are the maximums among $\{x_{11}, x_{12}, x_{21}, x_{22}\}$, $\{x_{13}, x_{14}, x_{23}, x_{24}\}$, $\{x_{31}, x_{32}, x_{41}, x_{42}\}$ and $\{x_{33}, x_{34}, x_{43}, x_{44}\}$, respectively. We also have a logical matrix $\mathrm{d}M$ to record the positions of these

maximums, here

$$\mathrm{d}M = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

Since there is no trainable parameter in a max pooling layer, the purpose of back propagation is to compute $\frac{\partial \mathcal{L}}{\partial X}$ based on $\frac{\partial \mathcal{L}}{\partial Y}$.

Obviously, we have

$$\begin{aligned} y_{11} &= x_{11} \\ y_{12} &= x_{13} \\ y_{21} &= x_{31} \\ y_{22} &= x_{33}, \end{aligned}$$

therefore,

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial x_{11}} &= \frac{\partial \mathcal{L}}{\partial y_{11}} \\ \frac{\partial \mathcal{L}}{\partial x_{13}} &= \frac{\partial \mathcal{L}}{\partial y_{12}} \\ \frac{\partial \mathcal{L}}{\partial x_{31}} &= \frac{\partial \mathcal{L}}{\partial y_{21}} \\ \frac{\partial \mathcal{L}}{\partial x_{33}} &= \frac{\partial \mathcal{L}}{\partial y_{22}}. \end{aligned}$$

In matrix form:

$$\frac{\partial \mathcal{L}}{\partial X} = \text{repeat}\left(\frac{\partial \mathcal{L}}{\partial Y}\right) \circ \mathrm{d}M,$$

where $\circ$ is the Hadamard product (element-wise multiplication); the repeat function maps each single number x in input data to $\begin{bmatrix} x & x \\ x & x \end{bmatrix}$, e.g.

$$\text{repeat}\left(\begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix}\right) = \begin{bmatrix} x_{11} & x_{11} & x_{12} & x_{12} \\ x_{11} & x_{11} & x_{12} & x_{12} \\ x_{21} & x_{21} & x_{22} & x_{22} \\ x_{21} & x_{21} & x_{22} & x_{22} \end{bmatrix}.$$

Now, we build a simple CNN and illustrate how to compute derivatives of all trainable parameters. The architecture is shown in Figure 3.4, where $\mathsf{conv}(f_s, n_f)$ is a 2D convolution layer with size $f_s \times f_s$, stride $(1, 1)$, no padding, and number of filters n_f; $\mathsf{maxpool}(f_s)$ denotes 2D max pooling with pool size $f_s \times f_s$, no padding and stride (f_s, f_s); the input data size is $6 \times 6 \times 1$; the output is a scalar; the reshape function converts a matrix to a vector row-wisely, for example,

$$\text{reshape}\left(\begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix}\right) = \begin{bmatrix} x_{11} & x_{12} & x_{21} & x_{22} \end{bmatrix}^T.$$

The loss function is $\mathcal{L} = \frac{1}{2}(\widehat{y} - y)^2$, where $\widehat{y}$ is the prediction (output) of this CNN, and y is the GT. Here $\widehat{y}, y \in \mathbb{R}$.

We perform forward propagation and compute $\widehat{y}$, where $x \in \mathbb{R}^{6\times 6}$ is the input data; $K \in \mathbb{R}^{3\times 3}$ is the kernel; σ is the sigmoid function; for a matrix X, $(\text{ReLU}(X))_{ij} = \text{ReLU}(X_{ij})$; W_{FC} refers to the weights matrix of fully-connected

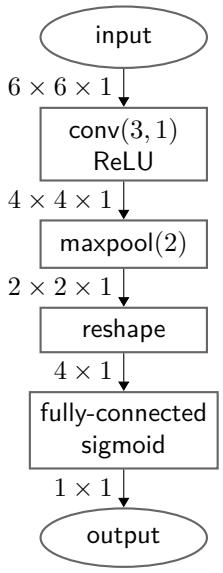


FIGURE 3.4: A simple CNN architecture. The numbers represent the output of each layer.

layer; $*$ refers to convolution operator; L_i is the output of ith layer:

$$\begin{aligned} L_1 &= x * K \\ L_2 &= \mathrm{ReLU}(L_1) \\ L_3 &= \mathrm{maxpool}(L_2) \\ L_4 &= \mathrm{reshape}(L_3) \\ L_5 &= W_{\mathrm{FC}} L_4 \\ \widehat{y} &= \sigma(L_5). \end{aligned}$$

For this network, W_{FC} and K are trainable parameters. Then we perform back propagation to compute $\frac{\partial \mathcal{L}}{\partial K}$ and $\frac{\partial \mathcal{L}}{\partial W_{\mathrm{FC}}}$, where $\mathrm{reshape}^{-1}$ is the inverse function of reshape, that is,

$$\mathrm{reshape}^{-1}\left(\begin{bmatrix} x_{11} & x_{12} & x_{21} & x_{22} \end{bmatrix}^{\mathrm{T}}\right) = \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix},$$

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \widehat{y}} &= \widehat{y} - y \\
\frac{\partial \mathcal{L}}{\partial L_5} &= \frac{\partial \mathcal{L}}{\partial \widehat{y}} \circ \sigma'(L_5) = \frac{\partial \mathcal{L}}{\partial \widehat{y}}\big(\widehat{y}(1-\widehat{y})\big) \\
\frac{\partial \mathcal{L}}{\partial W_{\mathrm{FC}}} &= \frac{\partial \mathcal{L}}{\partial L_5} L_4^T \\
\frac{\partial \mathcal{L}}{\partial L_4} &= W_{\mathrm{FC}}^{\mathrm{T}} \frac{\partial \mathcal{L}}{\partial L_5}.
\end{aligned}$$

So far we obtain $\frac{\partial \mathcal{L}}{\partial L_4}$. Since reshape function only transforms L_3 to a vector L_4 and does not change the value of L_3, we only need to transform $\frac{\partial \mathcal{L}}{\partial L_4}$ to a matrix to compute $\frac{\partial \mathcal{L}}{\partial L_3}$,

$$\frac{\partial \mathcal{L}}{\partial L_3} = \mathrm{reshape}^{-1}\left(\frac{\partial \mathcal{L}}{\partial L_4}\right).$$

Then we compute $\frac{\partial \mathcal{L}}{\partial L_2}$ by

$$\frac{\partial \mathcal{L}}{\partial L_2} = \mathrm{repeat}\left(\frac{\partial \mathcal{L}}{\partial L_3}\right) \circ \mathrm{d}M.$$

The derivative of ReLU is [40]

$$\mathrm{ReLU}'(x) = \begin{cases} 1 & x > 0 \\ 0 & \text{otherwise,} \end{cases}$$

although ReLU is non-differentiable at 0. Then we compute $\frac{\partial \mathcal{L}}{\partial L_1}$ by

$$\frac{\partial \mathcal{L}}{\partial L_1} = \mathrm{ReLU}'(L_1) \circ \frac{\partial \mathcal{L}}{\partial L_2}.$$

Finally, we compute $\frac{\partial \mathcal{L}}{\partial K}$ by

$$\frac{\partial \mathcal{L}}{\partial K} = x * \frac{\partial \mathcal{L}}{\partial L_1}.$$

3.6 Optimization algorithm

In Section 3.2, we employ gradient descent to update trainable parameters. However, when the size of a training dataset is large, each iteration of training can be time-consuming since it needs to compute the gradient for entire training dataset before computing the partial derivatives of the cost function with respect to the trainable parameters. In addition, the gradient descent also has high demand of computer memory [47].

The mini-batch gradient descent [47] is a better choice where the cost function J is computed by one mini batch (one mini-batch is a small subset of training dataset) instead of the entire training dataset. The mini-batch gradient descent approximates the gradient by taking a small subset of the entire training dataset. Normally, the batch size is $4, 8, 16, 32, 64, 128$. In mini-batch gradient descent, when all mini-batches are fed to the network exactly once, the algorithm completes one epoch. The mini-batch gradient descent works as shown in Algorithm 2.

Algorithm 2 Mini-batch gradient descent

Input:

w: initial trainable parameters
n_{epoch}: number of epochs
n_{iter}: number of iterations for each epoch
batch size: m
α: learning rate
training dataset: $\{(x^{(1)}, y^{(1)}), ..., (x^{(M)}, y^{(M)})\}$
for each epoch
    shuffle the training dataset
    divide training set into mini-batches b_1,...,b_m
    for $i = 1 : m$
        choose mini-batch b_i
        do forward propagation and compute the cost function J
        compute the gradient of all trainable parameters $\frac{\partial J}{\partial w_i}$ by back propagation, where $i = 1, ..., n_p$; n_p is the number of trainable parameters.
        update $w_i \leftarrow w_i - \alpha \frac{\partial J}{\partial w_i}$ for $i = 1, ..., n_p$

In mini-batch gradient descent, the way to update trainable parameters is the same with gradient descent. Some researchers proposed other approaches to update trainable parameters in training neural networks. In this thesis, we use the Adam [30] optimization algorithm to update all trainable parameters.

In Adam, there are two variables V and S to estimate the first and the second moment of the gradients of the cost function with respect to the trainable parameter, w_i $(i = 1, ..., n_p)$. Denote by $V^{\{j\}}$ the estimate of the first moment of the gradients and denote by $S^{\{j\}}$ the estimate of the second moment of the gradients. The initial values of V and S are 0, that is, $V^{\{0\}} = S^{\{0\}} = 0$.

In the jth iteration, Adam computes

$$V^{\{j\}} = \beta_1 V^{\{j-1\}} + (1-\beta_1)\frac{\partial J}{\partial w_i}$$
$$S^{\{j\}} = \beta_2 S^{\{j-1\}} + (1-\beta_2)\left(\frac{\partial J}{\partial w_i}\right)^2,$$

where β_1 and β_2 are set before training. A common choice for β_1 and β_2 is $\beta_1 = 0.9$ and $\beta_2 = 0.999$. In fact, $V^{\{j\}}$ is the exponential weighted moving average (EWMA) [65] of the past gradients and $S^{\{j\}}$ is the EWMA of the past squared gradients.

Then we do bias correction on $V^{\{j\}}$ and $S^{\{j\}}$ since $V^{\{j\}}$ and $S^{\{j\}}$ are biased in the first few iteration. That is,

$$V^{\{1\}} = \beta_1 V^{\{0\}} + (1-\beta_1)\frac{\partial J}{\partial w_i} = (1-\beta_1)\frac{\partial J}{\partial w_i}$$
$$S^{\{1\}} = \beta_2 S^{\{0\}} + (1-\beta_2)\left(\frac{\partial J}{\partial w_i}\right)^2 = (1-\beta_2)\left(\frac{\partial J}{\partial w_i}\right)^2,$$

In the first iteration, $V^{\{1\}} = (1-\beta_1)\frac{\partial J}{\partial w_i}$ and $S^{\{1\}} = (1-\beta_2)\left(\frac{\partial J}{\partial w_i}\right)^2$, but $V^{\{1\}}$ and $S^{\{1\}}$ should be $\frac{\partial J}{\partial w_i}$ and $\left(\frac{\partial J}{\partial w_i}\right)^2$ due to the purpose of $V^{\{1\}}$ and $S^{\{1\}}$. To achieve

this goal, Adam does bias correction by

$$V^{\{j\}} \leftarrow \frac{V^{\{j\}}}{1-\beta_1^j}$$
$$S^{\{j\}} \leftarrow \frac{S^{\{j\}}}{1-\beta_2^j}.$$

Then Adam uses $S^{\{j\}}$ to scale the learning rate by

$$\widehat{\alpha} = \frac{\alpha}{\sqrt{S^{\{j\}}}+\epsilon},$$

where α is the initial learning rate set before training; ϵ is a small number to avoid the denominator to become zero. In fact, the method to scale the learning rate in Adam is from [59]. The purpose of scaling the learning rate is to find a proper learning rate for each trainable parameter based on its previous gradients. To be concrete, when $S^{\{j\}}$ is large, $\widehat{\alpha}$ is small; $S^{\{j\}}$ is small, $\widehat{\alpha}$ is large.

Finally, Adam updates the trainable parameter by

$$w_i \leftarrow w_i - \widehat{\alpha} V^{\{j\}}.$$

In each update, Adam uses the estimation of first moment of the gradients, $V^{\{j\}}$ instead of $\frac{\partial J}{\partial w_i}$ in gradient descent. This method may avoid getting stuck on local minimum when we minimize the cost function and speed up training. For example, if the previous gradients are positive but the gradient is negative in this iteration, $V^{\{j\}}$ may still be positive.

To summarize Adam, we give

Algorithm 3 Adam

Initialize $V^{\{0\}} = S^{\{0\}} = 0$

On jth iteration:

$V^{\{j\}} = \beta_1 V^{\{j-1\}} + (1-\beta_1)\frac{\partial J}{\partial w_i}$

$S^{\{j\}} = \beta_2 S^{\{j-1\}} + (1-\beta_2)\left(\frac{\partial J}{\partial w_i}\right)^2$

$V^{\{j\}} \leftarrow \frac{V^{\{j\}}}{1-\beta_1^j}$

$S^{\{j\}} \leftarrow \frac{s^{\{j\}}}{1-\beta_2^j}$

$\widehat{\alpha} = \frac{\alpha}{\sqrt{S^{\{j\}}}+\epsilon}$

$w_i \leftarrow w_i - \widehat{\alpha} V^{\{j\}}$

Chapter 4

Segmentation networks

This chapter describes our segmentation networks which includes the architecture of our models, how we train these models, how we use them to segment US images and the post-processing techniques we apply. Section 4.1 introduces the autoencoder. Section 4.2 presents the CNN models in detail. Section 4.3 explains the loss function, regularization and data preprocessing used in training our models. Section 4.4 introduces the prediction methods. Finally, Section 4.5 presents the post-processing methods used after prediction.

4.1 Autoencoder

An autoencoder neural network consists of an encoder network and a decoder network. The encoder network reduces the dimension of the input data and extracts important features from the input data. These features, that is, the output of the encoder network, is the internal representation of an autoencoder. The decoder tries to recover the input data from the internal representation. Our autoencoder network architecture is shown in Figure 4.1, where $\mathsf{upsample}(a)$ denotes $a \times a$ upsampling layer; $\mathsf{maxpool}(f_s)$ denotes 2D max pooling with pool size $f_s \times f_s$, no padding and stride f_s.

Normally, the size of the internal representation is smaller than the size of the input (except in sparse autoencoders [37]). Here, the input size is $128 \times 128 \times 1$, and the size of the internal representation is $16 \times 16 \times 8$.

An autoencoder neural network is also trained by the back propagation algorithm, and the GT is equal to the input. That is, the output of a well-trained autoencoder should be almost the same as the input. We train the autoencoder in advance with the GT. During the training of the segmentation network, we use the encoder part of a well-trained autoencoder to compute the internal representation of the GT and the internal representation of the segmentation network's output to help training CNNs, see Section 4.3 for detail.

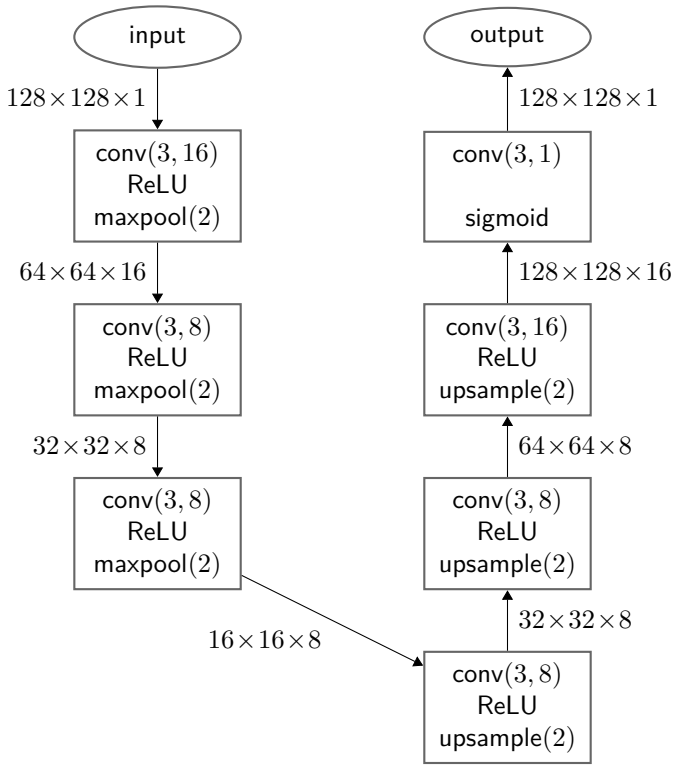


FIGURE 4.1: Autoencoder architecture. The numbers represent the output of each layer.

The autoencoder in our research is from [9], and this architecture uses 3×3

convolution layers, 2×2 max pooling layers and 2×2 upsampling layers. The input size of our autoencoder is $128 \times 128 \times 1$. In the encoder part (down sampling path), there are three 2×2 max pooling layers, that is, the size of input is reduced from $128 \times 128 \times 1$ to $16 \times 16 \times 8$.

4.2 Models

We experiment with the original U-Net [45] and three other models based on it. U-Net is a symmetric encoder-decoder model with shortcuts between the encoder and the decoder. The shortcuts are able to recover information that is lost at the max pooling layers.

Starting with U-Net, we replace in it the double 3×3 2D convolution layers by inception blocks, and the 2×2 up-convolution layers by 2D bilinear up-sampling layers which do bilinear interpolation on the input data. In addition, we add batch normalization layers to improve model's performance. We refer to the resulting model as an inception U-Net, or IU-Net.

To investigate whether an even smaller model exists and achieves better accuracy for our segmentation problem, we derive two smaller versions of IU-Net, small IU-Net, and even smaller, mini IU-Net. We describe these three models in the next subsections.

4.2.1 IU-Net architecture

The architecture of IU-Net is similar to U-Net, where there are five inception blocks and four 2×2 max pooling layers in the downsampling path, and there are four inception blocks and four 2×2 bilinear upsampling layers (U-Net used

deconvolution layers) in the upsampling path. We also use shortcuts between downsampling path and upsampling path to recover information that is lost in the downsampling path, see Figure 4.2 for more details, where inception(n_f) denotes a inception block with n_f filters in every convolution layer.

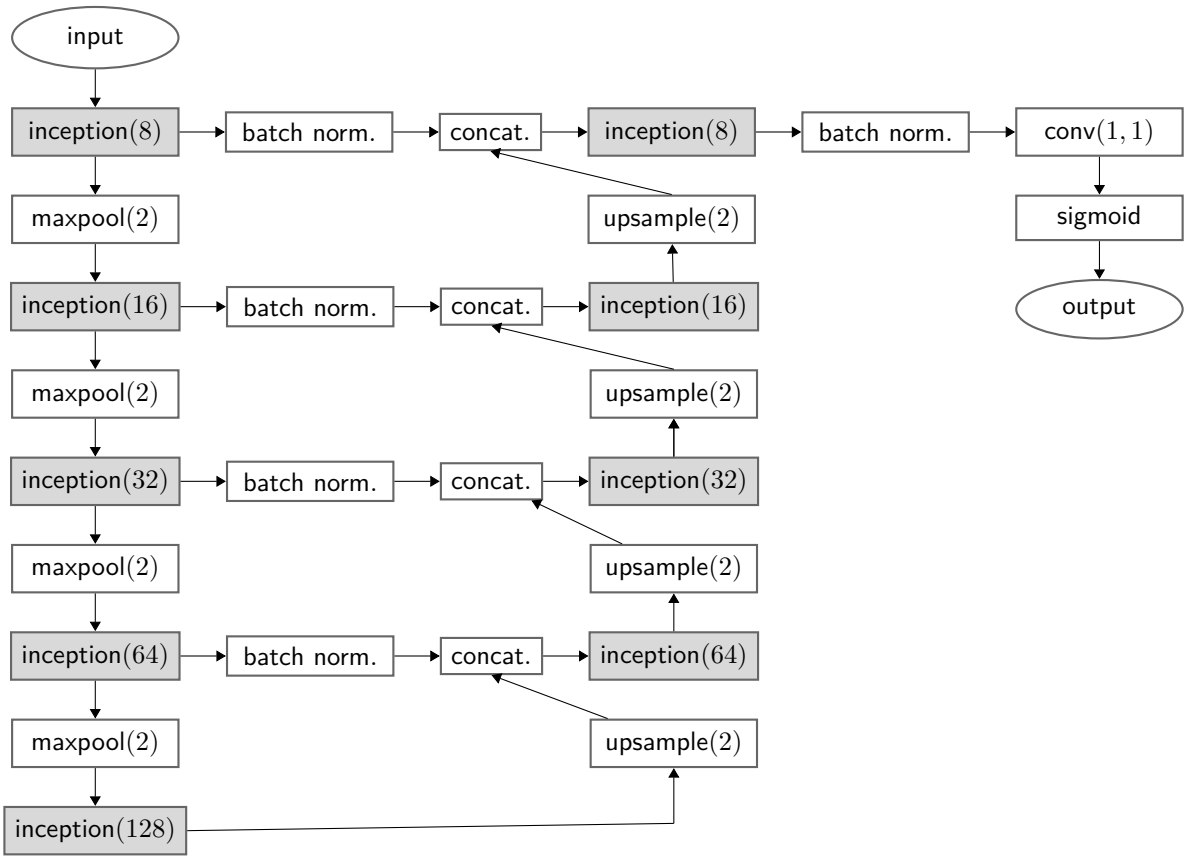


FIGURE 4.2: IU-Net architecture.

4.2.2 Small IU-Net architecture

For the small IU-Net, we remove from IU-Net the 4th max pooling layer and the 5th inception block in the downsampling path, and also remove the 1st upsampling layer and the 1st inception blocks in the upsampling path. Then we connect the

4th inception block in the downsampling path with 2nd upsampling layer in the upsampling path. see Figure 4.3 for more details.

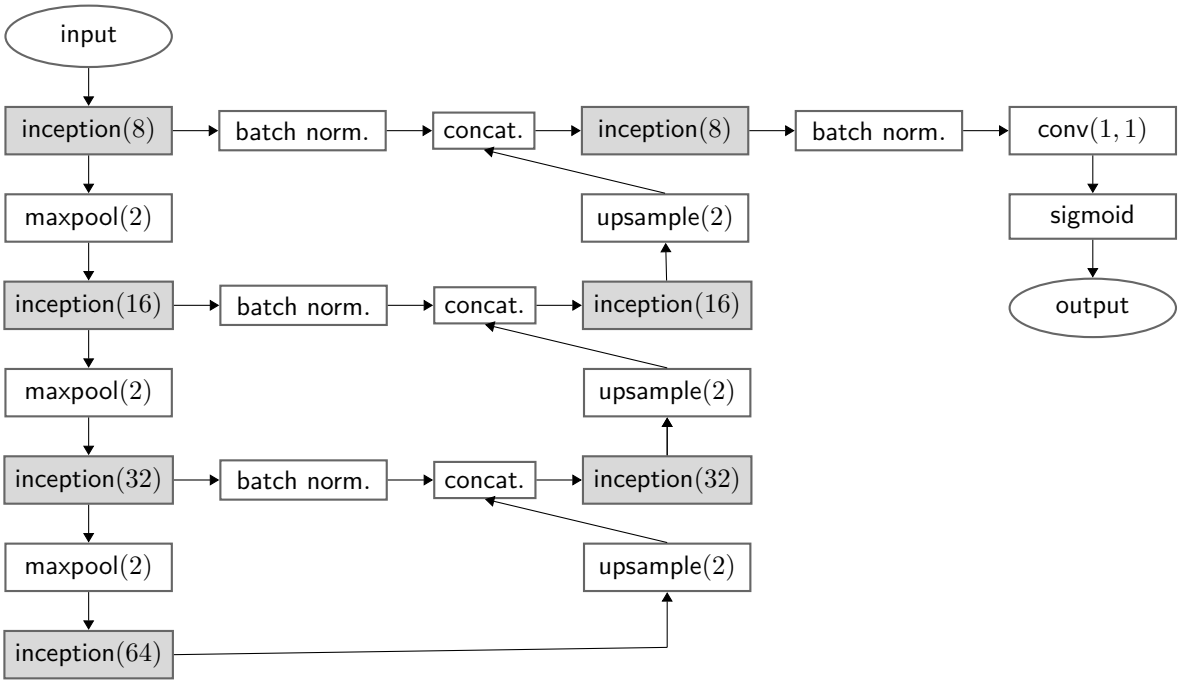


FIGURE 4.3: small IU-Net architecture.

4.2.3 Mini IU-Net architecture

For the mini IU-Net, we remove the 3rd max pooling layer and the 4th inception block in the downsampling path of the small IU-Net, and also remove the 1st upsampling layer and the 1st inception blocks in the upsampling path of the small IU-Net. Then we connect the 3rd inception block in the downsampling path with 2nd upsampling layer in the upsampling path of small IU-Net. see Figure 4.4 for more details.

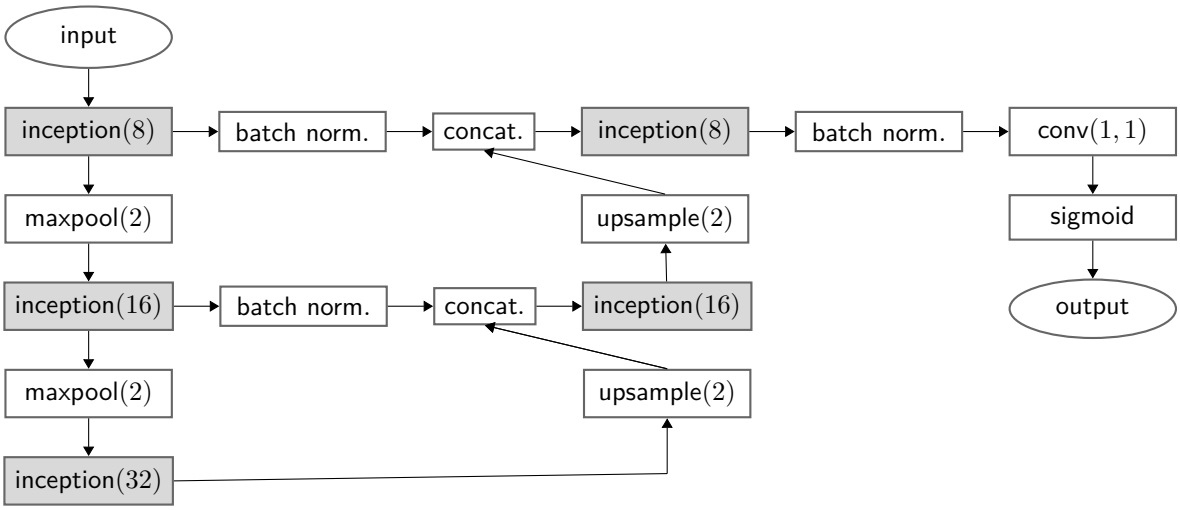


FIGURE 4.4: mini IU-Net architecture.

4.3 Training

In this section, we discuss the loss function, regularization, and data preprocessing which are used in training our models. Section 4.3.1 presents the loss function. Section 4.3.2 introduces a regularization term in the loss function. Section 4.3.3 presents the data preprocessing techniques.

4.3.1 Loss function

First we introduce how the Dice similarity coefficient (DSC) is defined between two binary images. The binary image's pixel value is either 0 or 1. For an image X, denote by x_i its ith pixel, in some predetermined order, which does not matter. Let X and Y be images of the same size $N \times N$. With slight abuse of notation, consider also X and Y as vectors of size N^2. We assume that Y is not the zero vector.

Assume $x_i, y_i \in \{0, 1\}$. Then we compute the DSC between binary images X and Y by

$$\mathcal{D}(X, Y) = 2\frac{\sum_{i=1}^{N^2} x_i y_i}{\sum_{i=1}^{N^2}(x_i + y_i)} = \frac{2Y^T \cdot X}{\mathbf{1} \cdot (X + Y)}, \tag{4.1}$$

where $\cdot$ is dot-product; $\mathbf{1}$ is a row vector of N ones. Obviously $\mathcal{D}(X, Y) \in [0, 1]$. $\mathcal{D}(X, Y) = 0$, if $Y^T \cdot X = 0$. When $X = Y$, that is, $x_i = y_i$ for all i, $\mathcal{D}(X, Y) = 1$.

Then we discuss how to construct the loss function. In training a segmentation neural network, we want to maximize DSC between the prediction $\widehat{y}$ ($\widehat{y}_i \in [0, 1]$) and the ground truth (GT) y ($y_i \in \{0, 1\}$), $\mathcal{D}(\widehat{y}, y)$, that is, to minimize $-\mathcal{D}(\widehat{y}, y)$, so the loss function $\mathcal{L}_{\text{DSC}}$ is defined as

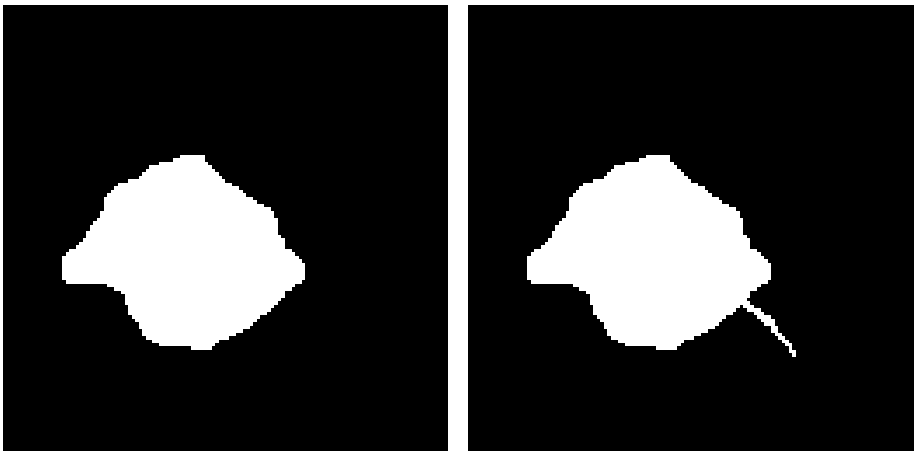
$$\mathcal{L}_{\text{DSC}}(\widehat{y}, y) = -\frac{2\sum_{i=1}^{N^2} \widehat{y}_i y_i + s}{\sum_{i=1}^{N^2}(\widehat{y}_i + y_i) + s} = -\frac{2y^T \cdot \widehat{y} + s}{\mathbf{1} \cdot (\widehat{y} + y) + s} \tag{4.2}$$

where s is smoothing factor which is widely used in training segmentation networks [16, 61], here we pick $s = 1$. When $s = 0$, $\mathcal{L}_{\text{DSC}}(\widehat{y}, y) = -\mathcal{D}(\widehat{y}, y)$.

4.3.2 Regularization

The DSC between two binary images X, Y, $\mathcal{D}(X, Y)$, measures the area of overlap between X and Y. However, DSC is not a perfect metric. In some cases, $\mathcal{D}(X, Y)$ is close to 1 but X and Y differ a lot. For instance, consider the two images X and Y given in Figure 4.5.

From Figure 4.5, even though $\mathcal{D}(X, Y) = 0.992 \approx 1$, images X and Y differ a lot due to the presence of the long needle-like area in Figure 4.5 (b). That is, if we only employ $\mathcal{L}_{\text{DSC}}$ as loss function, the prediction of the network may have this needle-like area, even though the value of the $\mathcal{L}_{\text{DSC}}$ is close to -1. We don't want this needle-like area (or other small area) appear in the segmentation results.



(a) Y (b) X, $\mathcal{D}(X, Y) = 0.992$

FIGURE 4.5: Limitation of DSC

Therefore, we propose a regularization term $\mathcal{L}_{\text{reg}}$,

$$\mathcal{L}_{\text{reg}}(\widehat{y}, y) = \|\text{encoder}(\widehat{y}) - \text{encoder}(y)\|_2, \tag{4.3}$$

where encoder$(\cdot)$ denotes the result of the encoder, that is, the $16 \times 16 \times 8$ output in Figure 4.1; $\|\cdot\|_2$ is the Euclidean norm and λ is a parameter to be set before training. We use $\lambda = 10^{-3}$; $\mathcal{L}_{\text{reg}}$ measures the similarity between the internal representation of $\widehat{y}$ and y. The encoder$(\widehat{y})$ and encoder(y) represent high-level features (could be shape, area or more complex features combined by other features) of $\widehat{y}$ and y, respectively. Hence we expect that $\mathcal{L}_{\text{reg}}$ helps remove these needle-like areas (or other small area) by minimizing the distance between high-level features of $\widehat{y}$ and y. In addition, we expect $\mathcal{L}_{\text{reg}}$ to improve the accuracy of our models.

Then we construct a new loss function $\mathcal{L}_{\text{total}}$ defined as

$$\mathcal{L}_{\text{total}}(\widehat{y}, y) = \mathcal{L}_{\text{DSC}}(\widehat{y}, y) + \lambda \mathcal{L}_{\text{reg}}(\widehat{y}, y). \tag{4.4}$$

Experimental results of IU-Net trained with autoencoder and without autoencoder are given in Chapter 5.

4.3.3 Data preprocessing

The data we have are 3D US images. A 3D image comprises p slices, where each slice is of size $m \times n$, see Figure 4.6. Since the slices are typically of different m and n, with typical values $m = 199, 200$, $n = 215, 221, 226$, and each 3D image has different number of slices p, we resize all of them to $N \times N$ images, where we set $N = 128$. For each slice, we also resize their GT to $N \times N$ images. Then we split the 3D images into 2D images.

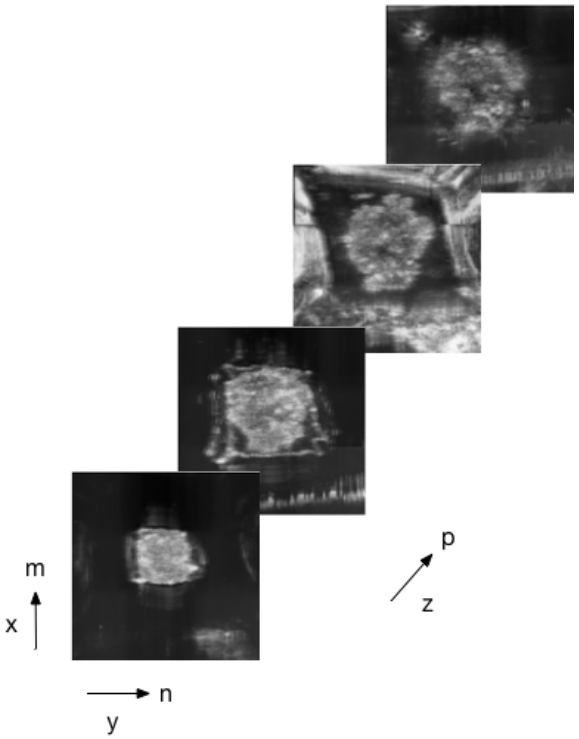


FIGURE 4.6: An example of 3D US image

Next we do data normalization to rescale the range of pixel values to make CNNs easier to train [4].

Denote the kth training image by I_k, $k = 1, ..., M$, where M is the number of images in the training set, and its (i, j) pixel by $I_k(i, j)$.

Let

$$\mu = \frac{\sum_{k=1}^{M} \sum_{i=1}^{N} \sum_{j=1}^{N} I_k(i,j)}{N^2 M}$$
$$\sigma = \sqrt{\frac{\sum_{k=1}^{M} \sum_{i=1}^{N} \sum_{j=1}^{N} (I_k(i,j) - \mu)^2}{N^2 M}}.$$

Then we normalize by

$$I_k(i,j) \leftarrow \frac{I_k(i,j) - \mu}{\sigma}.$$

4.4 Prediction

Denote the predicted image by $\widehat{I}$. Then we do thresholding on $\widehat{I}$:

$$\widehat{I}(i,j) \leftarrow \begin{cases} 1 & \widehat{I}(i,j) \geq \text{threshold} \\ 0 & \text{otherwise,} \end{cases}$$

where we set threshold $= 0.5$.

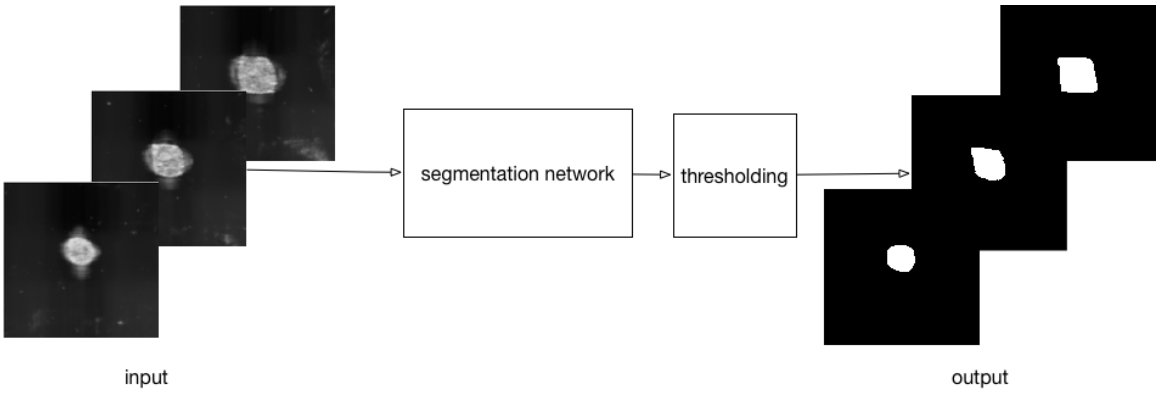


FIGURE 4.7: Prediction

We have experimented with producing segmentations by slicing the images in the x and y directions. This has not led to improving the overall accuracy. Nevertheless, we describe our approach. Since each 3D US image in our dataset has different number of slices p ($p_{\min} = 17, p_{\max} = 66$) in z direction, we pad zero matrices of size $N \times N$ to the top and bottom of each 3D image approximately evenly so that each 3D image contains the same number of slices S. Since there is only one sample with $p > 64$ ($p = p_{\max} = 66$), we remove the first slice and the last slice of this sample. Therefore we set $S = 64$ here. We construct new slices of the resized images and corresponding GT in the x and y direction. In the x direction this results in a set of N slices of size $N \times S$, and similarly in the y direction, a set of N slices of size $N \times S$. For convenience, we denote images in z direction as D_z, images in x direction as D_x and images in y direction as D_y.

We train with the dataset D_z and the constructed sliced images (D_x, D_y) separately, since the former and latter are of different size. We refer to the model trained with D_z as Model 1; the input images are of size $N \times N$. We refer to the model trained with D_x and D_y as Model 2; the input images are of size of $N \times S$. That is, Model 1 is used to segment images with slices in the z direction, and Model 2 is used to segment images with slices in both the x and y directions. After the Model 1 and Model 2 are trained, we input images with slices in the z direction to Model 1 and input images with slices in the x and y directions to Model 2. Finally, we slice the prediction of Model 2 in the z direction. This results in two image sets, each containing p slices of size $N \times N$. The above procedure is shown in Figure 4.8.

We experiment with the following two approaches to combine the segmentation results from the three directions, see results in Chapter 5. The first approach is

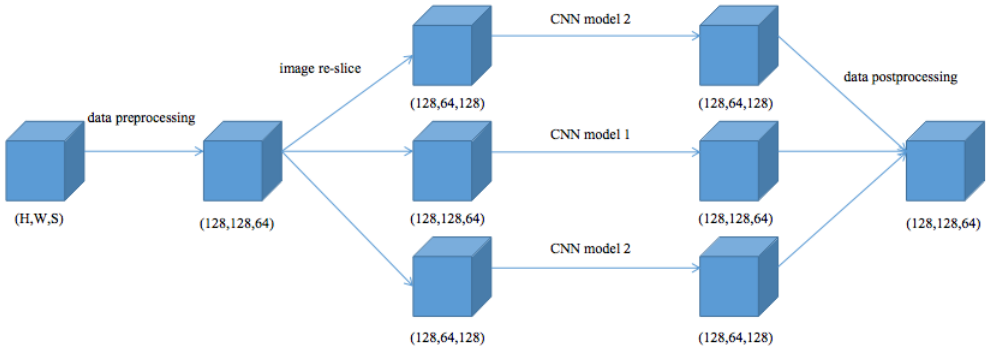


FIGURE 4.8: Segmentation from three direction

equal voting. With equal voting, the final result $\widehat{I}(i,j)$ is computed by

$$\widehat{I}(i,j) = \begin{cases} 1 & \text{if at least two of } \widehat{I}_z(i,j), \widehat{I}_x(i,j), \widehat{I}_y(i,j) \text{ are } 1 \\ 0 & \text{otherwise,} \end{cases}$$

where $\widehat{I}_z$, $\widehat{I}_x$ and $\widehat{I}_y$ are segmentation results from z, x, y direction.

The second approach is weighted voting. Since segmentation from z direction has the highest accuracy among three segmentation results, we give the z direction the largest weight. Here we pick from our observation the weight $\frac{1}{2}$, $\frac{1}{4}$, $\frac{1}{4}$ for the segmentation in z, x, y direction, respectively. The final result $I(i,j)$ is computed by

$$\widehat{I}(i,j) = \widehat{I}_z(i,j) \vee (\widehat{I}_x(i,j) \wedge \widehat{I}_y(i,j)),$$

where $\vee$, $\wedge$ refer to logic OR, logic AND operator respectively.

4.5 Post-processing

The post-processing is to improve the segmentation results, extract boundaries and combine the boundary with original breast cancer US images. These can provide better visualization of segmentation results. We discuss the following post-processing techniques: largest connected component extraction, hole filling and boundary extraction.

4.5.1 Largest connected component extraction

Largest connected component (LCC) extraction is to remove small region (compared to the GT) whose pixel values are 1 in the binary image, see Figure 4.9. These small areas may appear in the outputs of our models.

LCC extraction consists of two steps. The first step is connected component labelling [52].

The purpose of connected component labelling is to label every pixel (except background pixels whose pixel values are 0) with different integers $(1, 2, 3, ...)$. The pixels which belong to the same connected component have the same integer label.

Here we use the LABEL function of the SCIKIT-IMAGE package to perform connected component labelling. This function returns the labelled image and the number of connected components of the input image.

An example of connected component labelling is given as follows,

$$\begin{bmatrix} 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix} \xrightarrow{\text{connected component labelling}} \begin{bmatrix} 1 & 1 & 0 & 2 & 2 \\ 1 & 1 & 0 & 2 & 2 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 3 & 3 \end{bmatrix}.$$

The second step is to compute the area of each connected component and extract the largest one. The pseudocode is given in Algorithm 4

Algorithm 4 LLC extraction

Input: I_{label}: labelled image, n_{cc}: the number of connected component
Output: I_{llc}: image only containing the largest connected component

```
initialize n_max = 0, s_max = 0, (m, n) = size(I_label)
for i = 1 : n_cc do
    count the number of pixels of ith connected component, s
    if s > s_max then
        s_max = s, n_max = i
for i = 1 : m do
    for j = 1 : n do
        if I_label(i, j) ≠ n_max then
            I_label = 0
        else
            I_label = 1
I_llc = I_label
return I_llc;
```

4.5.2 Hole filling

Here a hole is a small region (compared to the area of GT) whose pixel values are 0 and appears in the segmented object whose pixel values are 1. In some cases,

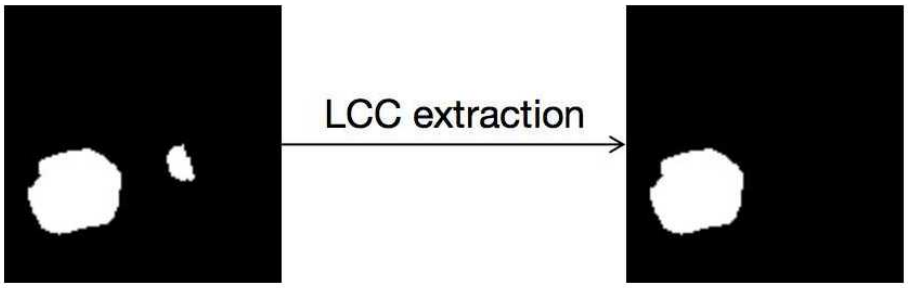


FIGURE 4.9: Largest connected component extraction

some small holes may occur in segmentation results, see Figure 4.10. We need to fill these small holes for better visualization of the boundary.

A hole in the segmentation results is actually a small connected-component of the complement of image I, where the compliment of I is computed by: for each pixel $I(i,j)$ of I, $I(i,j) \leftarrow 1 - I(i,j)$. Therefore, all we have to do is to extract the largest connected-component of the complement of image I so that we remove the small holes.

The algorithm is shown in Algorithm 5, where compliment(I) denotes the compliment of I.

Algorithm 5 Hole filling

Input: I: binary image;
Output: $I_{\text{no hole}}$: binary image without holes
$I_{\text{no hole}} = \text{compliment}(I)$
do connected component labelling on $I_{\text{no hole}}$
extract the LLC of $I_{\text{no hole}}$
$I_{\text{no hole}} \leftarrow \text{compliment}(I_{\text{no hole}})$
return $I_{\text{no hole}}$;

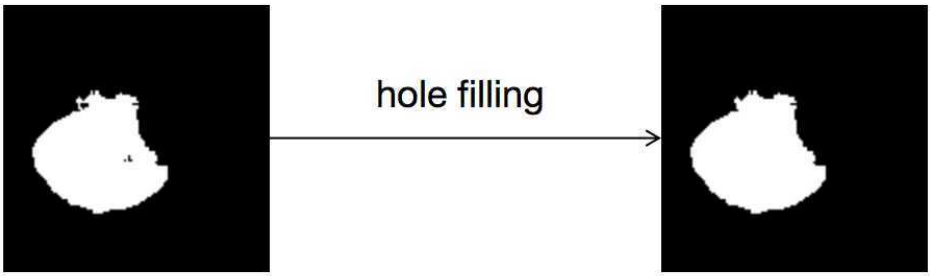


FIGURE 4.10: Hole filling

4.5.3 Boundary extraction

Boundary extraction is a good strategy to improve the visualization of segmentation results. In this research, we extract the boundary by deleting internal pixels. The internal pixels satisfy the following conditions:

(a) its pixel value is equal to 1;

(b) all neighbor pixels' values are equal to 1.

There are two kinds of neighbors, 4-connected neighbors and 8-connected neighbors. 4-connected neighbors of pixel $I(i,j)$ are $I(i-1,j), I(i+1,j), I(i,j-1), I(i,j+1)$. 8-connected neighbors of pixel $I(i,j)$ are $I(i-1,j-1), I(i-1,j), I(i-1,j+1), I(i,j-1), I(i,j+1), I(i+1,j-1), , I(i+1,j), I(i+1,j+1)$, see Figure 4.11.

Here we use 8-connected neighbors to extract a boundary. In our research, the visualizations of boundaries extracted with 4-connected neighbors and with 8-connected neighbors are almost the same, but a boundary extracted with 8-connected neighbors is clearer, see Figure 4.12.

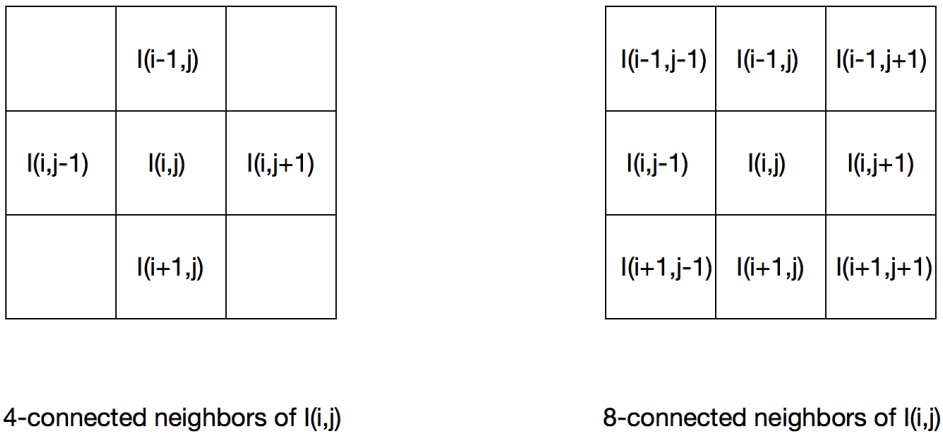


FIGURE 4.11: 4-connected neighbors and 8-connected neighbors

The pseudo code of boundary extraction is shown in Algorithm 6.

Algorithm 6 Boundary extraction

Input: I: binary image;
Output: $\widehat{I}$: binary image only containing the boundary of I;
  initialize $(m,n) = \text{size}(I)$, $\widehat{I} = I$
  for $i = 1 : m$ **do**
    for $j = 1 : n$ **do**
      if neighbors of $I(i,j)$ are all 1s **then**
        $\widehat{I}(i,j) = 0$
  return $\widehat{I}$;

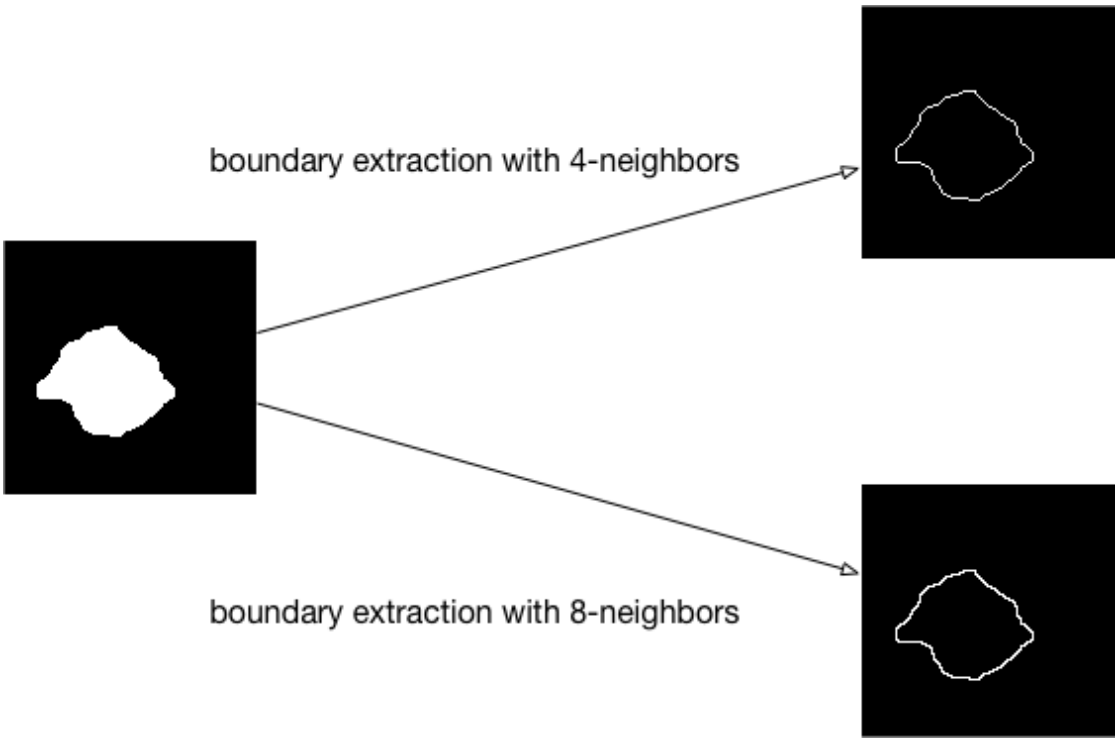


Figure 4.12: Boundary extraction

Chapter 5

Experimental results

This chapter introduces how we do experiments and shows the experimental results. The structure of this chapter is as follows. Section 5.1 presents the programming language and the packages used in this research. Section 5.2 introduces our dataset and the data augmentation techniques we have applied. Section 5.3 presents training parameters setting and learning curves. Section 5.4 shows the results of our segmentation.

5.1 Programming environment

The programming language we use is Python 3.6, and we employed the deep learning framework Keras [12] with TensorFlow [42] as a backend. In addition, we also used the NumPy scientific computing package and scikit-image for image processing.

We train our CNN models on Sharcnet [1] Graham server with an NVIDIA P100 Pascal GPU.

5.2 Dataset

This section introduces original US images data set and data augmentation we applied. Section 5.2.1 presents the original data set. Section 5.2.2 introduces how we do data augmentation to generate more US images.

5.2.1 Original data

The given data are breast cancer US images and manually segmented US images. These data are provided by Dr. Jeffery Carson and his research group at Lawson Health Research Institute, London, Ontario, Canada. The original dataset has 59 tissue specimen. After we remove some samples without manually segmented results, we have 1912 2D images (slices) obtained from 52 breast tissue specimen.

A summary of this data set is given in Table 5.1.

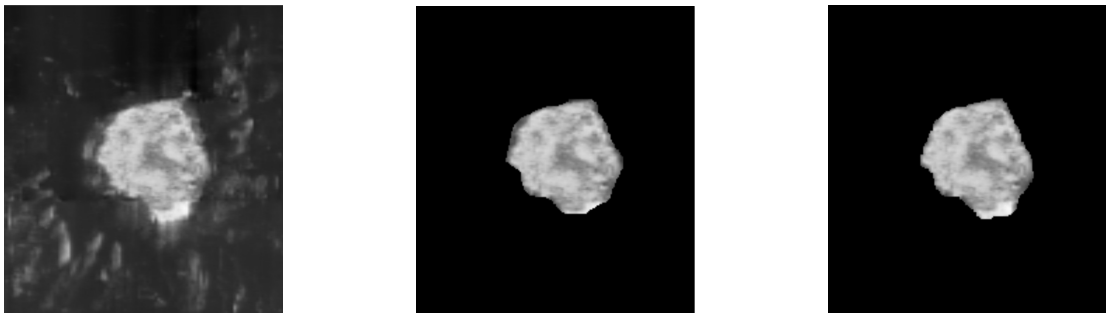
attribute	value
number of specimen	52
mean number of slices per specimen	37
number of 2D images	1912
mean image size	200×222

TABLE 5.1: Statistics about the original dataset

For each image, we also have two manually segmented images, segmentation A and segmentation B, see Figure 5.1. They are performed by two summer students in Dr. Jeffery Carson's research group independently. Before we start training our models, we do thresholding on segmentation A and B to convert these images into binary images.

$$S(i,j) \leftarrow \begin{cases} 1 & S(i,j) > 0 \\ 0 & \text{otherwise,} \end{cases}$$

where S denotes manual segmentation result. Denote by Set A the training set with segmentation A (after thresholding) and Set B the training set with segmentation B (after thresholding).



(a) US image (b) segmentation A (c) segmentation B

FIGURE 5.1: US image with segmentation A and segmentation B

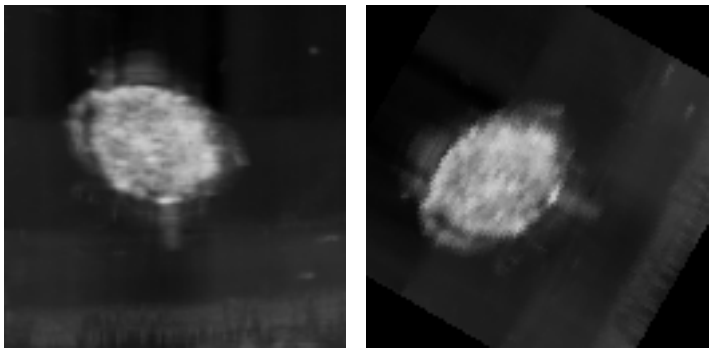
5.2.2 Data augmentation

Data augmentation is a common method to improve the robustness of a neural network model, especially when the training dataset is not large. From the original 1912 images, we remove 358 images (10 specimens) for testing and apply data augmentation on the remaining 1554 images. We performed the following data augmentation techniques.

1. First we rotate each US image and its corresponding segmentation result by 30°, 60°, 90°, 120°, 150°, 180°, 210°, 240°, 270°, 300°, 330°, see Figure 5.2.

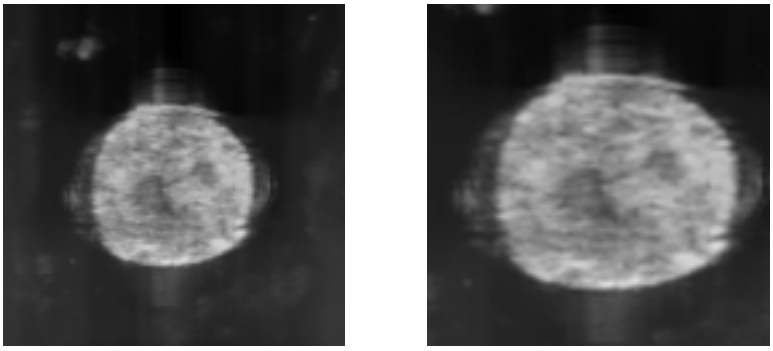
2. Then we perform image cropping on the original and the rotated images. We crop US images and its segmentation result with (a, b, c, d), where a, b, c, d refer to left, upper, right, and lower pixel coordinates, respectively. a, b are chosen randomly from 0 to 20 and c, d are chosen randomly from 100 to 120, see Figure 5.3.

We apply above data augmentation on Set A and Set B, then combine them into a new training set. This dataset is used in training our models.



(a) US image (b) 60° rotation

FIGURE 5.2: Image rotation in data augmentation



(a) US image (b) Cropped image with $(16, 18, 102, 114)$

FIGURE 5.3: Image crop in data augmentation

The number of augmented images for training is shown in Table 5.2

We also apply data augmentation on test images, the number of augmented images for testing is shown in Table 5.3. Since we have a large dataset for testing (we have 8234 images for testing after data augmentation), the accuracy of our models on this dataset is acceptable and trustworthy.

augmentation	# of augmented images
rotation	$17,094$ $(= 11 \times 1,554)$
cropping	$18,648$ $(= 1,554 + 17,094)$
total	$74,448$

TABLE 5.2: Number of augmented images for training. We start with 1554 images

augmentation	# of augmented images
rotation	$3,938$ $(= 11 \times 358)$
cropping	$4,296$ $(= 358 + 3,938)$
total	$8,234$

TABLE 5.3: Number of augmented images for testing. We start with 358 images

We perform the above data augmentation techniques using the PILLOW imaging library [15].

5.3 Training hyperparameters

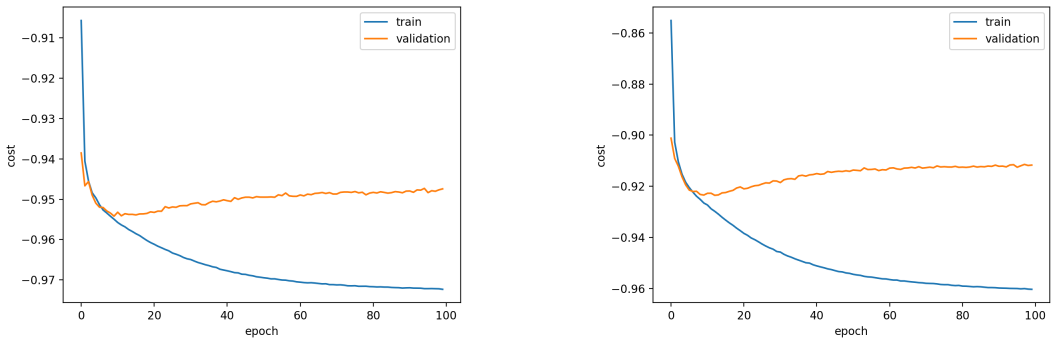
The parameters applied in training are given in Table 5.4.

parameter		value
Adam	α	10^{-4}
	β_1	0.9
	β_2	0.999
	ϵ	10^{-7}
Batch size	m	16
# of epochs	n_{epoch}	100

TABLE 5.4: Training parameters

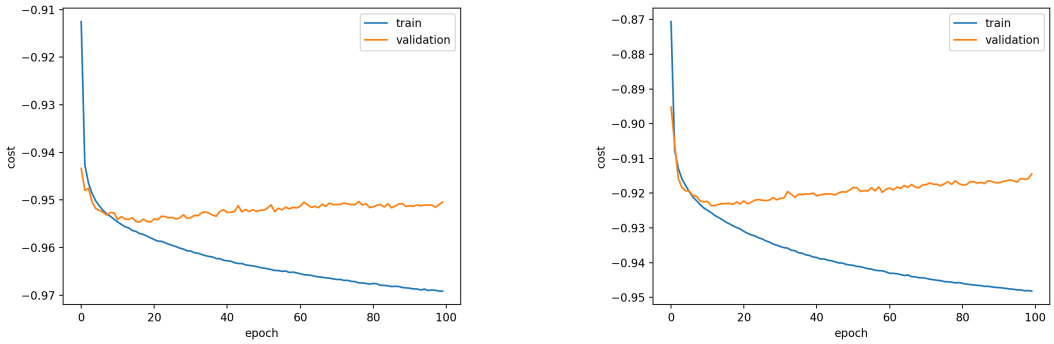
5.4 Results

We train U-Net, IU-Net, IU-Net with encoder, small IU-Net, small IU-Net with encoder, mini IU-Net and mini IU-Net with encoder. For all the networks we use the same hyperparameters setting listed in the previous section. The learning curves of these models are given in Figures 5.4, 5.5, 5.6 and 5.7.



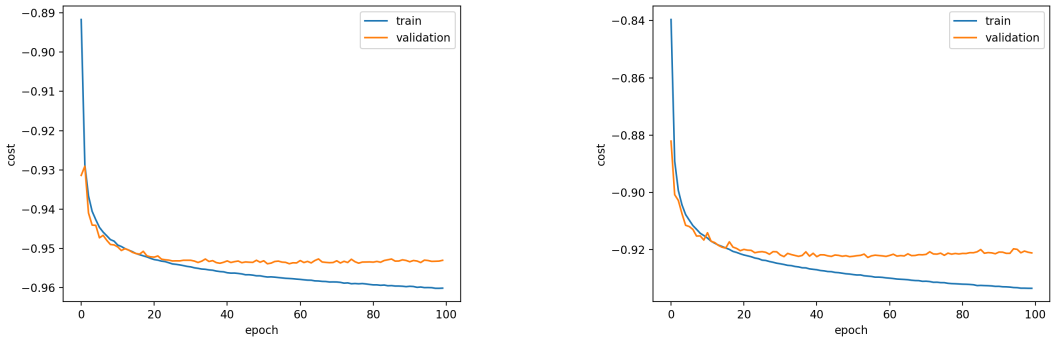
(a) $\lambda = 0$ (b) $\lambda = 10^{-3}$

FIGURE 5.4: Learning curves of U-Net with $\lambda = 0$ and $\lambda = 10^{-3}$



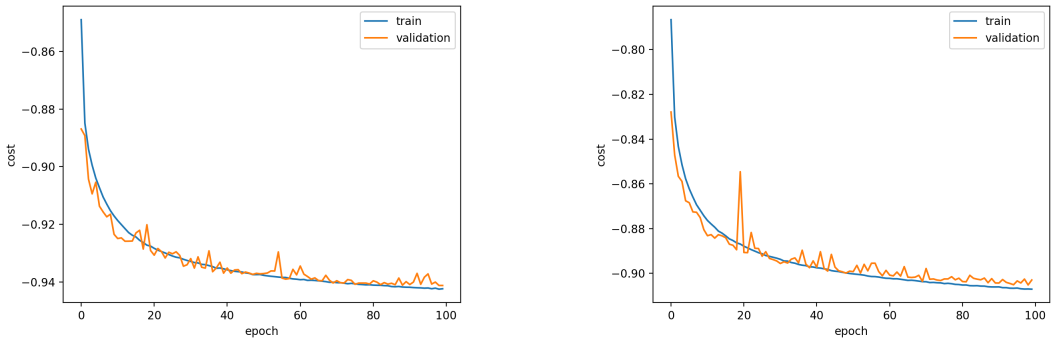
(a) $\lambda = 0$ (b) $\lambda = 10^{-3}$

FIGURE 5.5: Learning curves of IU-Net with $\lambda = 0$ and $\lambda = 10^{-3}$



(a) $\lambda = 0$ (b) $\lambda = 10^{-3}$

FIGURE 5.6: Learning curves of small IU-Net with $\lambda = 0$ and $\lambda = 10^{-3}$



(a) $\lambda = 0$ (b) $\lambda = 10^{-3}$

FIGURE 5.7: Learning curves of mini IU-Net with $\lambda = 0$ and $\lambda = 10^{-3}$

Figures 5.4, 5.5, 5.6 and 5.7 show that the cost of training data decreases, which confirms the training works, since the purpose of training is to minimize the cost function. In addition, Figures 5.4, 5.5, 5.6 show that the cost of validation data does not decrease after a certain number of epochs. That is, overfitting occurs in training U-Net, IU-Net and small IU-Net. However, KERAS can save the model which has the lowest cost of validation data, so training with too many epochs will not reduce the performance of our models.

In our test set we have 358 breast cancer US images. Since we have two manual segmentation results (S^A and S^B) for each US image, we compute the DSC and MSE of the prediction $\widehat{I}$ of our models by:

$$\overline{\mathcal{D}}\left(\widehat{I}_k, S_k^A, S_k^B\right) = \frac{\mathcal{D}\left(\widehat{I}_k, S_k^A\right) + \mathcal{D}\left(\widehat{I}_k, S_k^B\right)}{2}$$
$$\overline{\text{MSE}}\left(\widehat{I}_k, S_k^A, S_k^B\right) = \frac{\text{MSE}\left(\widehat{I}_k, S_k^A\right) + \text{MSE}\left(\widehat{I}_k, S_k^B\right)}{2},$$

where $\widehat{I}_k$ denotes the kth prediction, and S_k^A, S_k^B are its segmentation A and segmentation B, respectively, $k = 1, 2, ..., 358$; $\mathcal{D}$ is the DSC which is defined in Chapter 4; MSE is defined as

$$\text{MSE}\left(\widehat{I}, S\right) = \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N}\left(\widehat{I}(i,j) - S(i,j)\right)^2.$$

First we compute the mean and standard deviation (std) of $\mathcal{D}\left(\widehat{I}_k, S_k^A\right)$ and $\text{MSE}\left(\widehat{I}_k, S_k^A\right)$, and we also calculate the mean and std of $\mathcal{D}\left(\widehat{I}_k, S_k^B\right)$ and $\text{MSE}\left(\widehat{I}_k, S_k^B\right)$. The results are given in Tables 5.5 and 5.6.

Then we calculate the mean and std of $\overline{\mathcal{D}}$ and $\overline{\text{MSE}}$ on the test set, the results are given in Table 5.7.

The training time for above models is given in Table 5.8.

model	$\mathcal{D}\left(\widehat{I}_k, S_k^A\right)$		$\mathrm{MSE}\left(\widehat{I}_k, S_k^A\right)$	
	mean	std	mean	std
U-Net	92.2%	0.13	1.75×10^{-2}	1.28×10^{-2}
U-Net+encoder	91.8%	0.14	1.82×10^{-2}	1.43×10^{-2}
IU-Net	92.8%	0.10	1.76×10^{-2}	1.16×10^{-2}
IU-Net+encoder	**93.4%**	**0.08**	$\mathbf{1.71 \times 10^{-2}}$	$\mathbf{1.06 \times 10^{-2}}$
small IU-Net	92.4%	0.12	1.81×10^{-2}	1.25×10^{-2}
small IU-Net+encoder	92.1%	0.13	1.83×10^{-2}	1.28×10^{-3}
mini IU-Net	90.9%	0.14	2.14×10^{-2}	1.57×10^{-2}
mini IU-Net+encoder	90.8%	0.15	2.03×10^{-2}	1.45×10^{-2}

Table 5.5: Accuracy (with segmentation A as GT)

model	$\mathcal{D}\left(\widehat{I}_k, S_k^B\right)$		$\mathrm{MSE}\left(\widehat{I}_k, S_k^B\right)$	
	mean	std	mean	std
U-Net	89.7%	0.18	2.30×10^{-2}	$\mathbf{3.84 \times 10^{-2}}$
U-Net+encoder	89.5%	0.18	2.33×10^{-2}	3.89×10^{-2}
IU-Net	90.4%	0.16	2.31×10^{-2}	3.91×10^{-2}
IU-Net+encoder	**90.5%**	**0.15**	2.32×10^{-2}	3.86×10^{-2}
small IU-Net	90.1%	0.17	$\mathbf{2.30 \times 10^{-2}}$	3.85×10^{-2}
small IU-Net+encoder	89.8%	0.17	2.32×10^{-2}	3.89×10^{-2}
mini IU-Net	88.4%	0.19	2.65×10^{-2}	4.01×10^{-2}
mini IU-Net+encoder	88.4%	0.19	2.55×10^{-2}	4.01×10^{-2}

Table 5.6: Accuracy (with segmentation B as GT)

model	$\overline{\mathcal{D}}$		$\overline{\mathrm{MSE}}$		# of parameters
	mean	std	mean	std	
U-Net	91.0%	0.15	2.02×10^{-2}	2.22×10^{-2}	7.76×10^6
U-Net+encoder	90.6%	0.16	2.08×10^{-2}	2.29×10^{-2}	7.76×10^6
IU-Net	91.5%	0.12	2.04×10^{-2}	2.22×10^{-2}	2.05×10^6
IU-Net+encoder	**92.0%**	**0.11**	$\mathbf{2.01 \times 10^{-2}}$	$\mathbf{2.17 \times 10^{-2}}$	2.05×10^6
small IU-Net	91.2%	0.14	2.06×10^{-2}	2.20×10^{-2}	5.10×10^5
small IU-Net+encoder	90.9%	0.14	2.07×10^{-2}	2.25×10^{-2}	5.10×10^5
mini IU-Net	89.7%	0.16	2.40×10^{-2}	2.43×10^{-2}	$\mathbf{1.24 \times 10^5}$
mini IU-Net+encoder	89.7%	0.17	2.29×10^{-2}	2.38×10^{-2}	$\mathbf{1.24 \times 10^5}$

Table 5.7: Accuracy and number of parameters

model	training time (s/epoch)
U-Net	**254**
U-Net+encoder	256
IU-Net	574
IU-Net+encoder	578
small IU-Net	501
small IU-Net+encoder	503
mini IU-Net	420
mini IU-Net+encoder	437

TABLE 5.8: Training time

We summarize Tables 5.5, 5.6, 5.7 and 5.8:

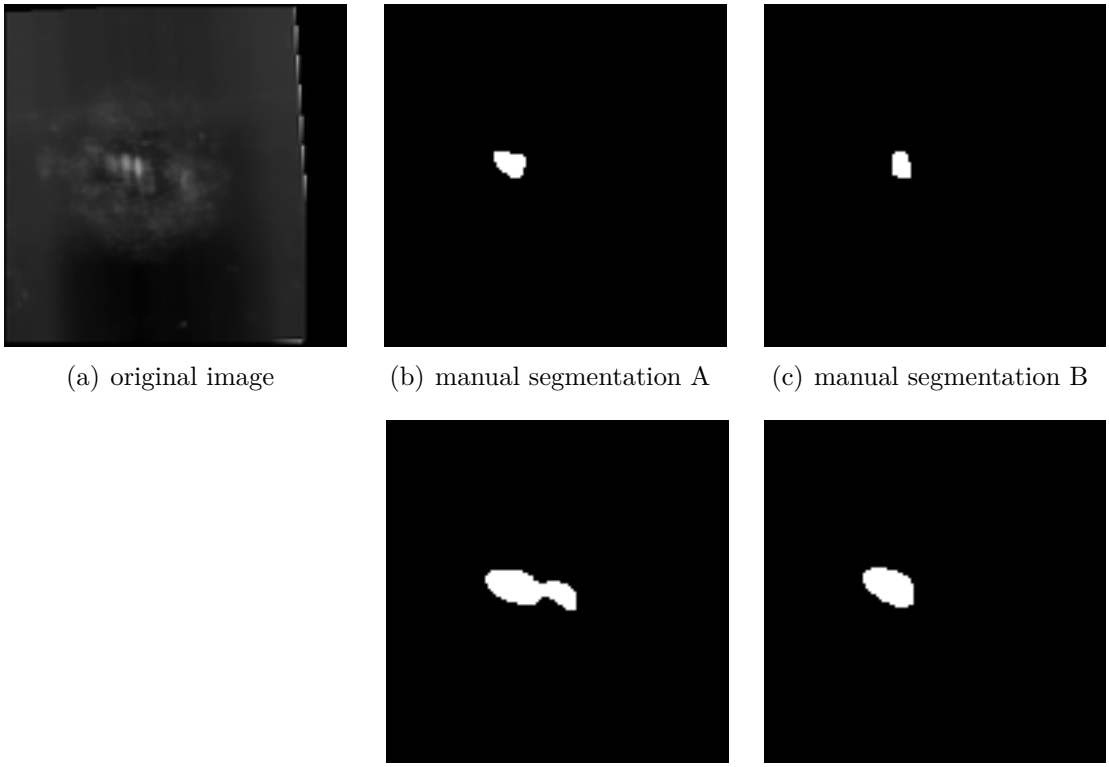
1. All models achieves similar accuracies.
2. All models achieves higher accuracies when the GT is only segmentation A.
3. The encoder improves the accuracy of IU-Net and mini IU-Net, but reduces the accuracy of U-Net and small IU-Net.
4. The encoder increases the training time a little.
5. The number of trainable parameters of U-Net is the largest among all models, but the training time of U-Net is the shortest, since IU-Net, small IU-Net and mini IU-Net have more layers than U-Net has, even though their trainable parameters are fewer than U-Net. See Table 5.9 for the number of layers of each model.

model	number of layers
U-Net	**32**
IU-Net	208
small IU-Net	162
mini IU-Net	116

TABLE 5.9: Number of layers of models

(The number of layers in Table 5.9 is obtained from LAYERS function in Keras.)

Although the encoder does not always improve the accuracy, its effect still exists. We show some examples in Figures 5.8, 5.9, 5.10 and 5.11. The small area in Figures 5.9 and 5.11 can be removed by largest connected component extraction. However, the small area in Figures 5.8 and 5.10 can not be removed by largest connected component extraction, since this small area is connected to the large one.



(a) original image

(b) manual segmentation A

(c) manual segmentation B

(d) prediction, $\lambda = 0$, $\overline{\mathcal{D}} = 35.4\%$

(e) prediction, $\lambda = 10^{-3}$, $\overline{\mathcal{D}} = 48.1\%$

FIGURE 5.8: Effect of encoder on U-Net

Then we illustrate some of the segmentation results in Figures 5.12, 5.13, 5.14, 5.15, 5.16, 5.17, 5.18 and 5.19, where the red boundary represents the segmentation A; the yellow boundary denotes the segmentation B; the blue boundary is the prediction.

(a) original image (b) manual segmentation A (c) manual segmentation B

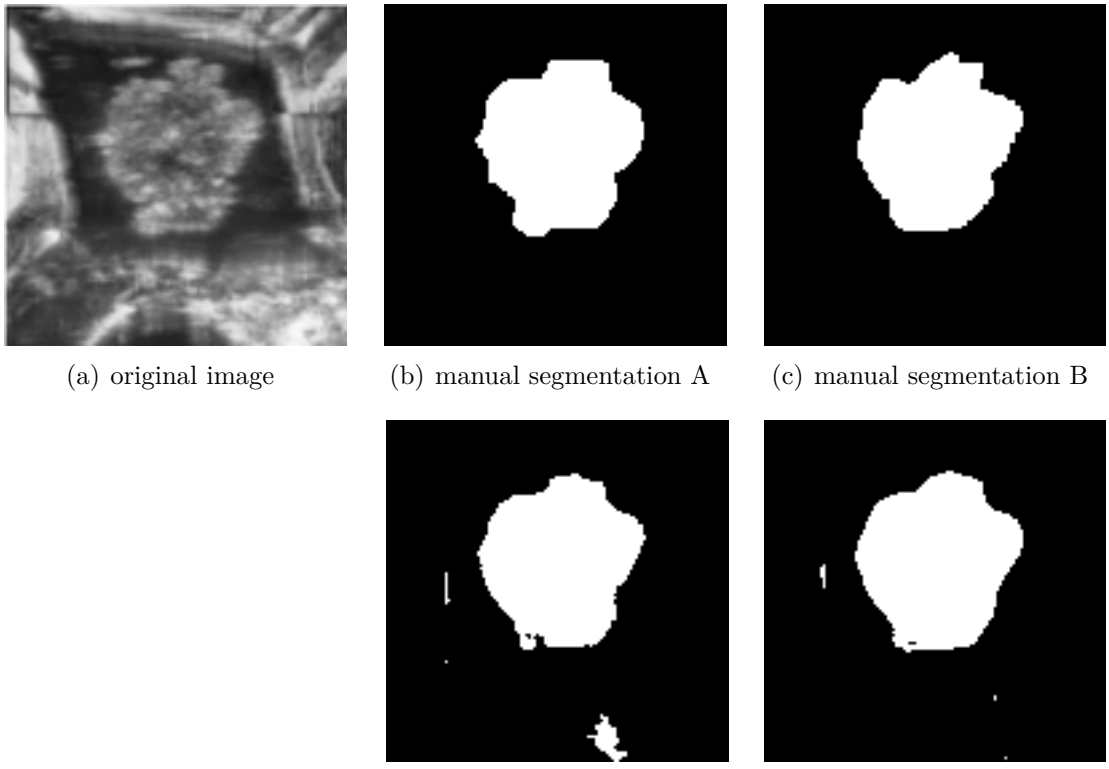
(d) $\lambda = 0$, $\overline{\mathcal{D}} = 92.9\%$ (e) $\lambda = 10^{-3}$, $\overline{\mathcal{D}} = 94.0\%$

Figure 5.9: Effect of encoder on IU-Net

(a) original image (b) manual segmentation A (c) manual segmentation B

(d) $\lambda = 0$, $\overline{\mathcal{D}} = 86.3\%$ (e) $\lambda = 10^{-3}$, $\overline{\mathcal{D}} = 91.9\%$

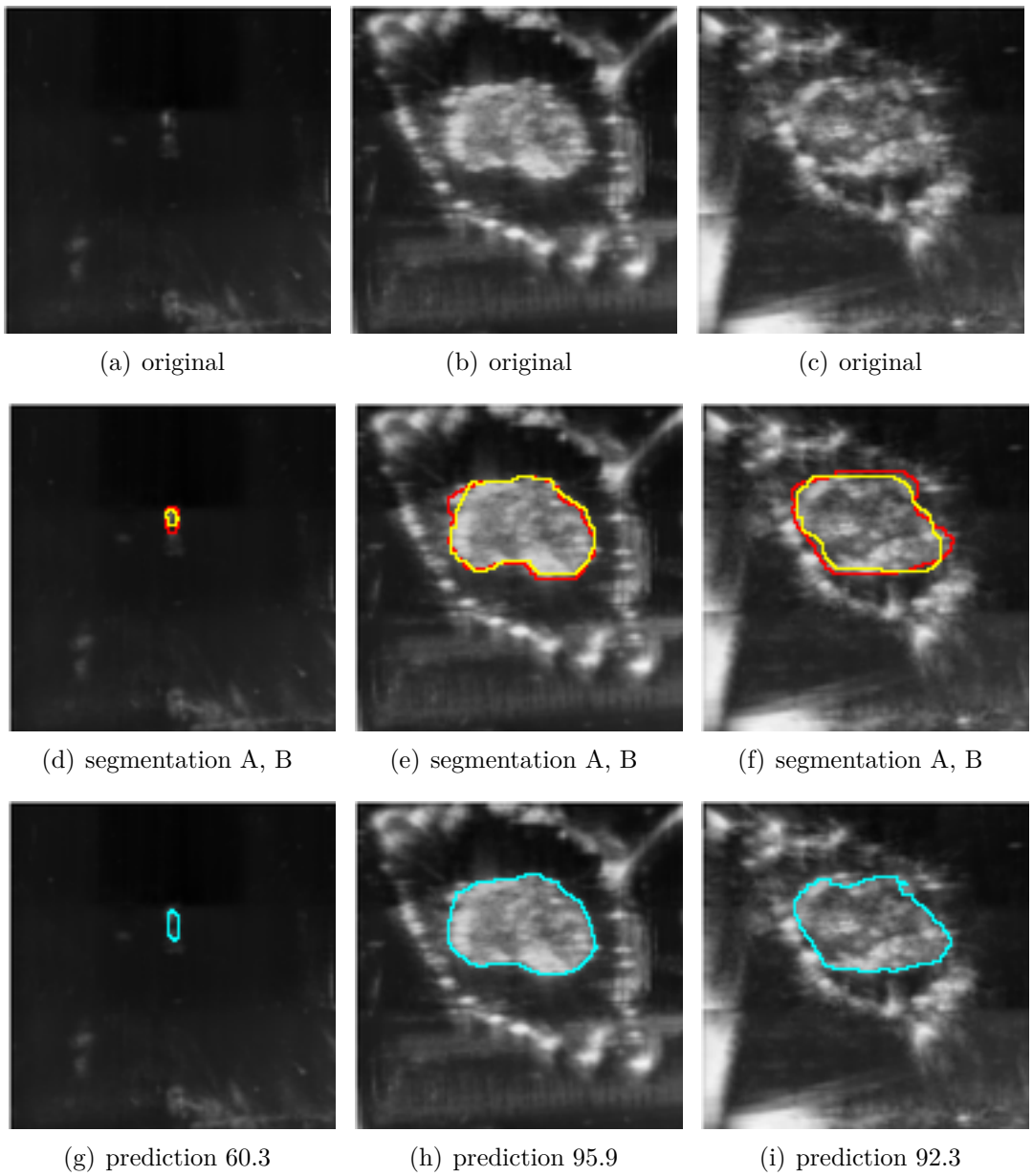
Figure 5.10: Effect of encoder on small IU-Net



(a) original image (b) manual segmentation A (c) manual segmentation B

(d) $\lambda = 0$, $\overline{\mathcal{D}} = 95.2\%$ (e) $\lambda = 10^{-3}$, $\overline{\mathcal{D}} = 96.3\%$

Figure 5.11: Effect of encoder on mini IU-Net

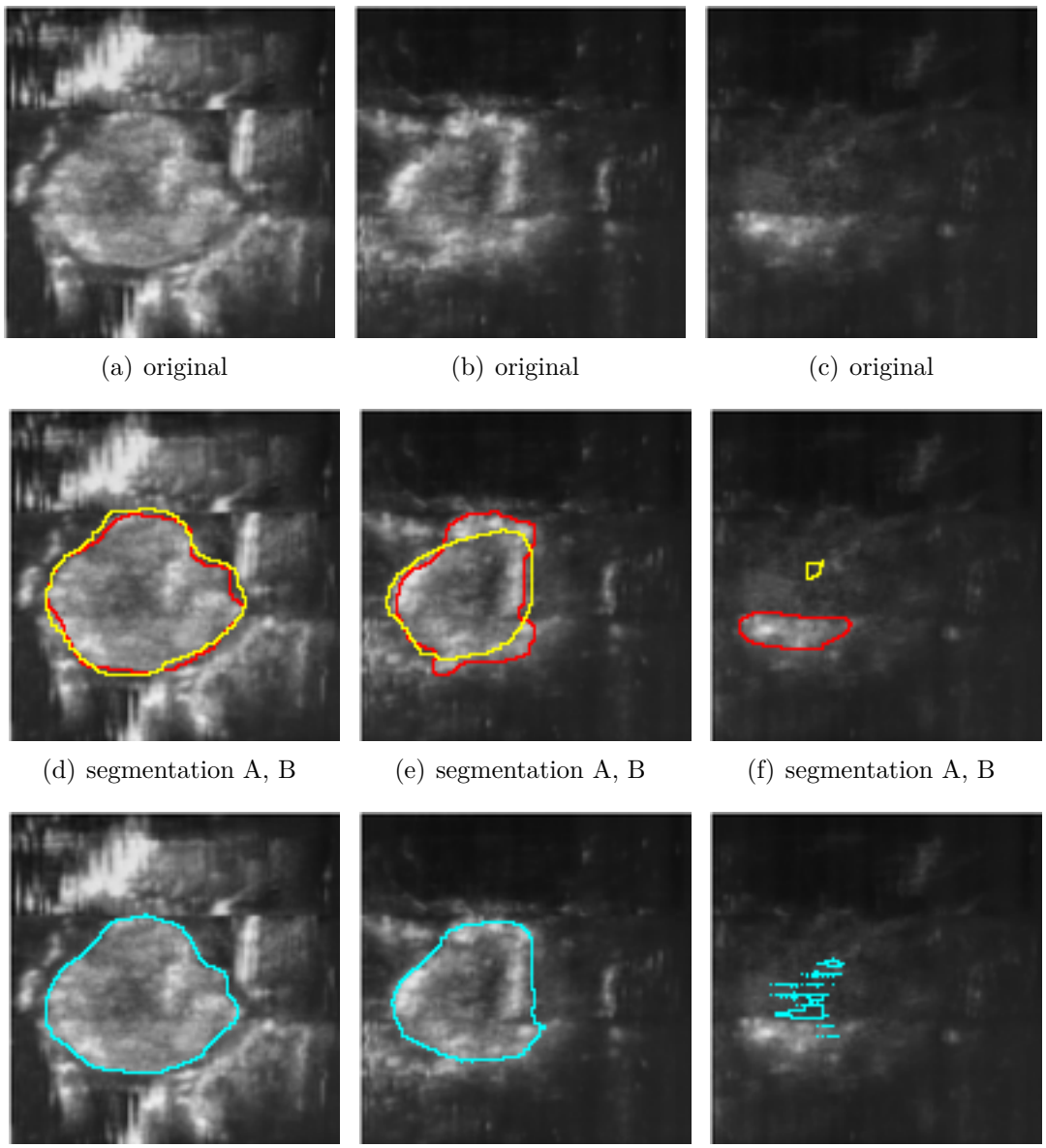


(a) original (b) original (c) original

(d) segmentation A, B (e) segmentation A, B (f) segmentation A, B

(g) prediction 60.3 (h) prediction 95.9 (i) prediction 92.3

Figure 5.12: Segmented US images by U-Net and their achieved DSC in percentages. The red boundary represents segmentation A; the yellow boundary denotes segmentation B; the blue boundary is the prediction

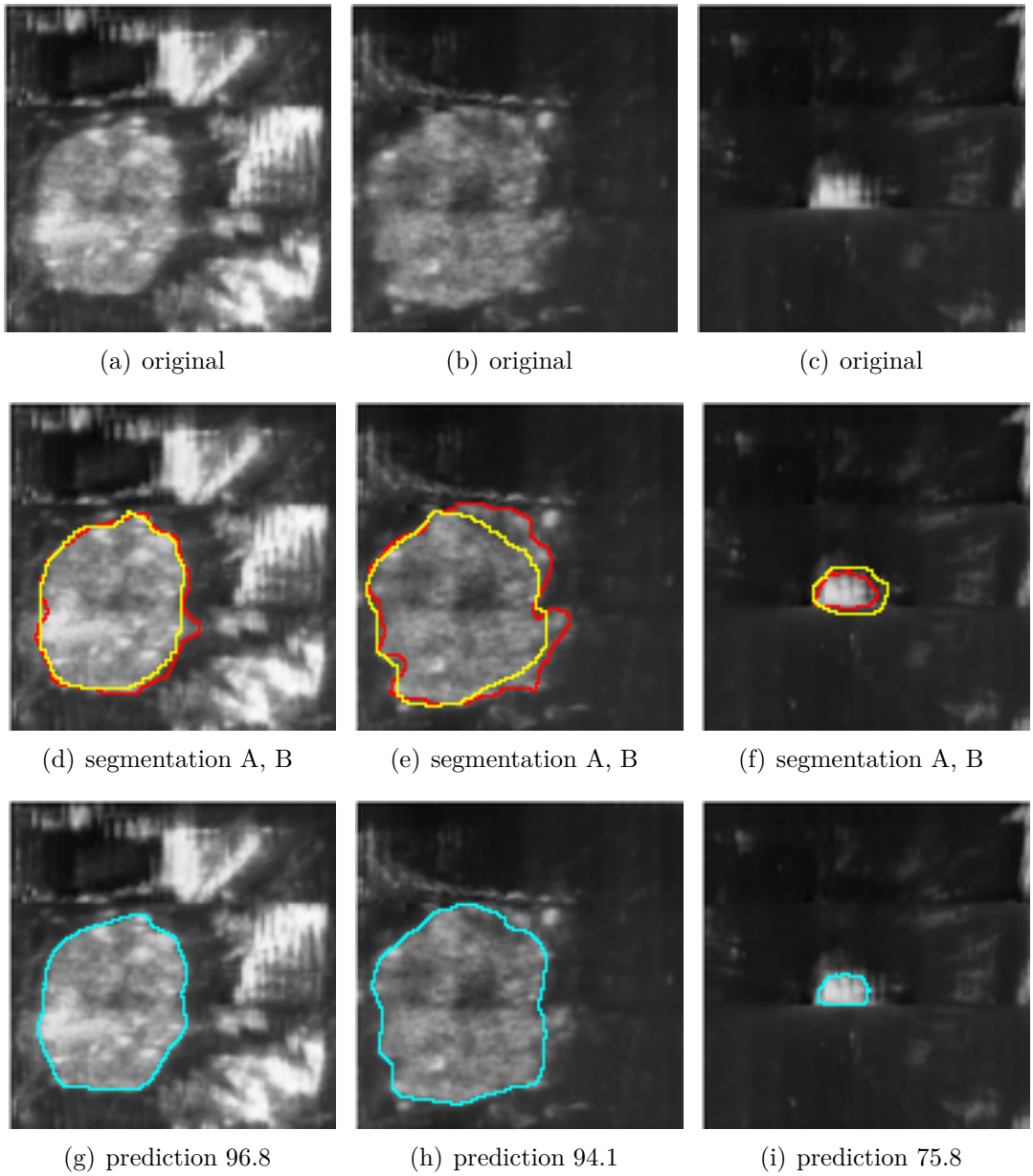


(a) original (b) original (c) original

(d) segmentation A, B (e) segmentation A, B (f) segmentation A, B

(g) prediction 95.6 (h) prediction 90.8 (i) prediction 7.5

Figure 5.13: Segmented US images by U-Net+encoder and their achieved DSC in percentages. The red boundary represents segmentation A; the yellow boundary denotes segmentation B; the blue boundary is the prediction

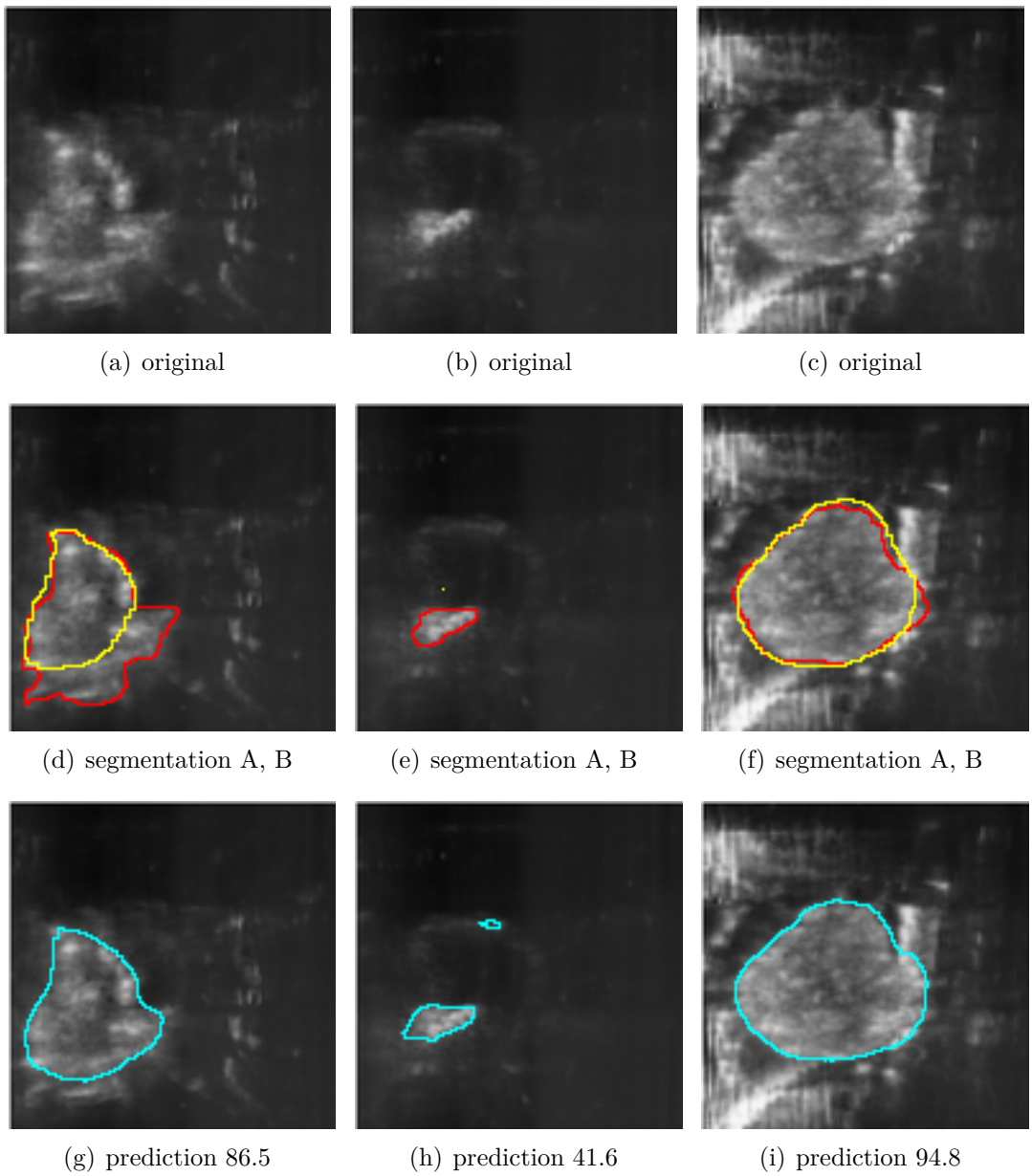


(a) original (b) original (c) original

(d) segmentation A, B (e) segmentation A, B (f) segmentation A, B

(g) prediction 96.8 (h) prediction 94.1 (i) prediction 75.8

Figure 5.14: Segmented US images by IU-Net and their achieved DSC in percentages. The red boundary represents segmentation A; the yellow boundary denotes segmentation B; the blue boundary is the prediction

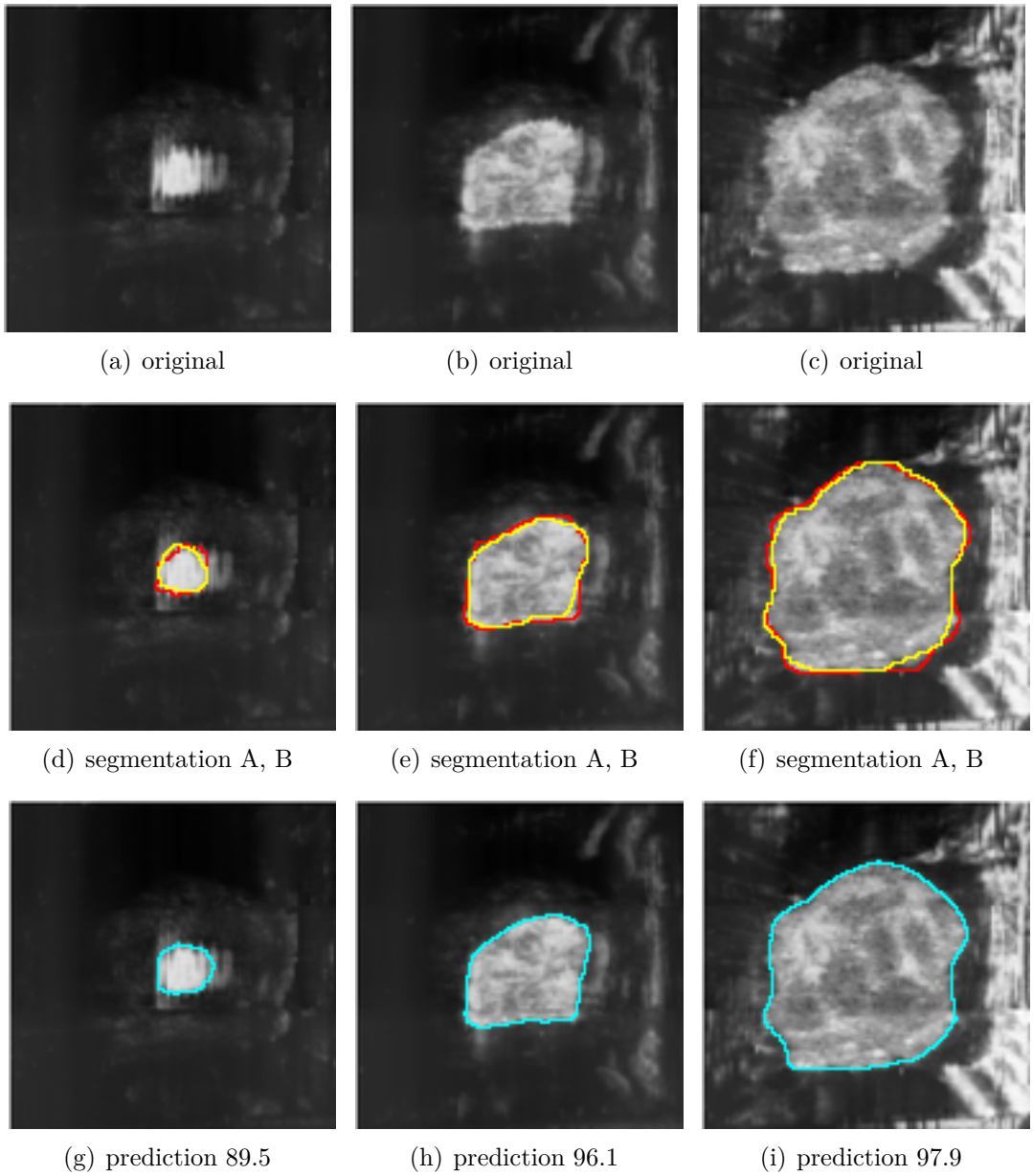


(a) original (b) original (c) original

(d) segmentation A, B (e) segmentation A, B (f) segmentation A, B

(g) prediction 86.5 (h) prediction 41.6 (i) prediction 94.8

FIGURE 5.15: Segmented US images by IU-Net+encoder and their achieved DSC in percentages. The red boundary represents segmentation A; the yellow boundary denotes segmentation B; the blue boundary is the prediction

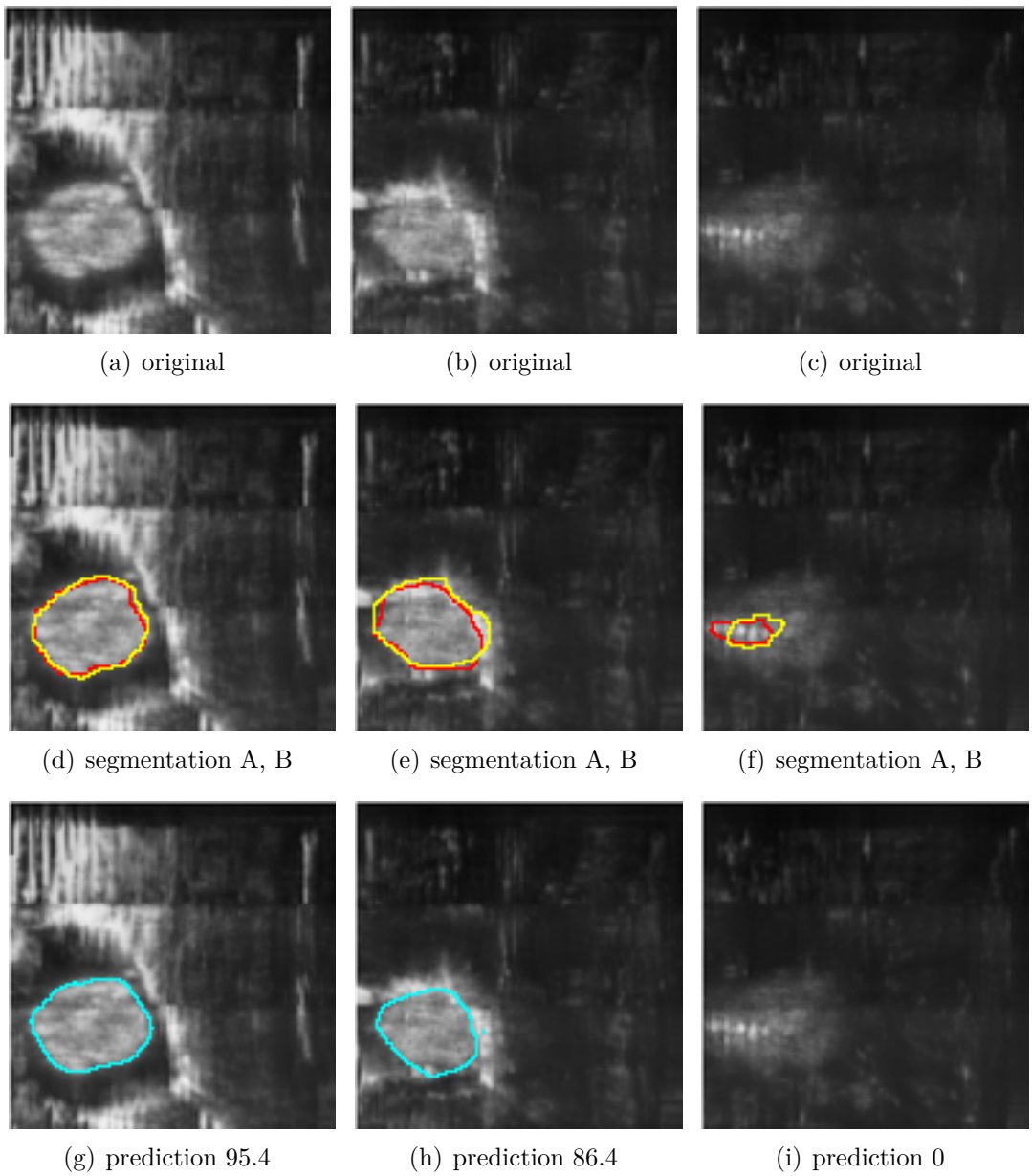


(a) original (b) original (c) original

(d) segmentation A, B (e) segmentation A, B (f) segmentation A, B

(g) prediction 89.5 (h) prediction 96.1 (i) prediction 97.9

Figure 5.16: Segmented US images by small IU-Net and their achieved DSC in percentages. The red boundary represents segmentation A; the yellow boundary denotes segmentation B; the blue boundary is the prediction

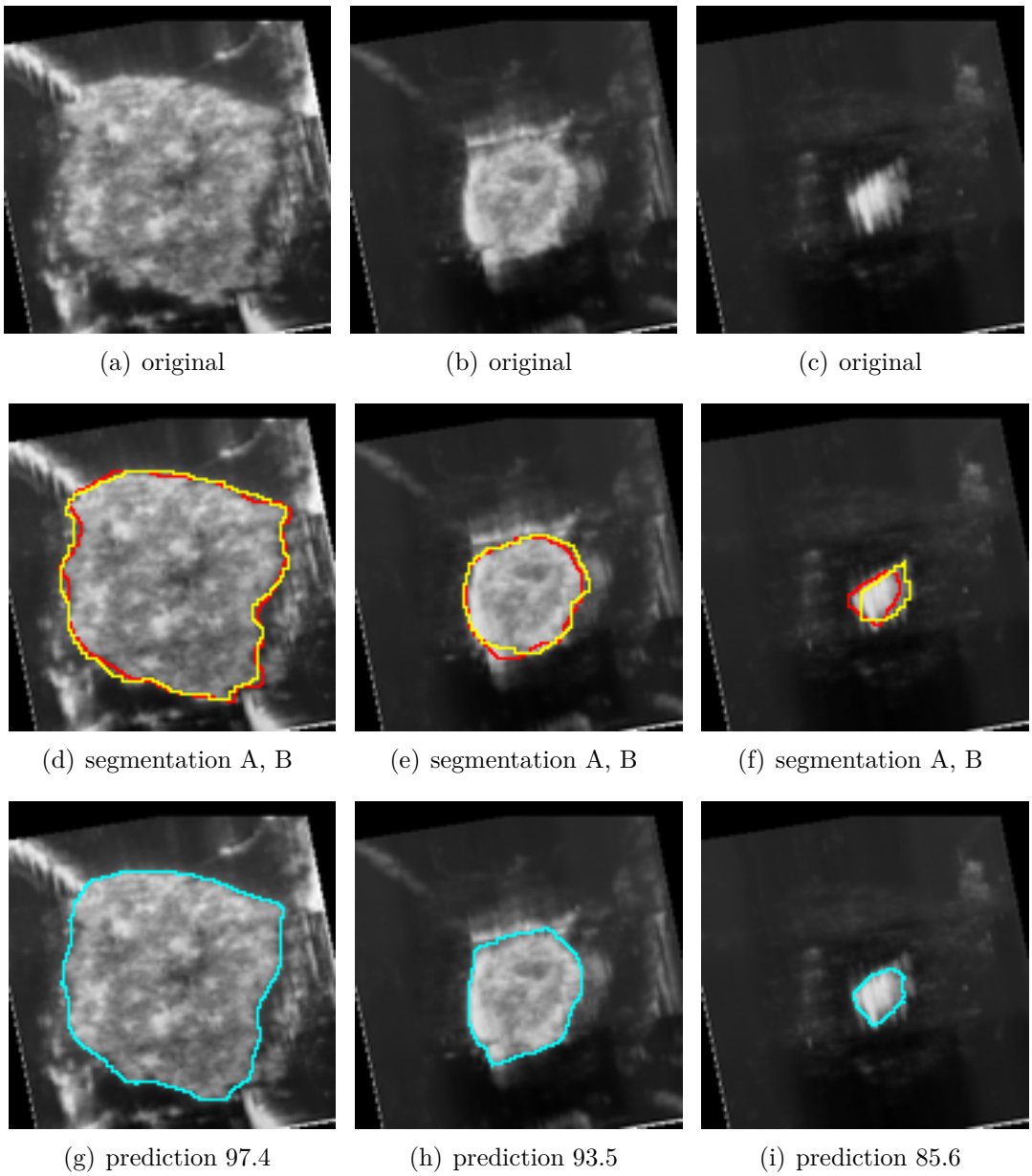


(a) original (b) original (c) original

(d) segmentation A, B (e) segmentation A, B (f) segmentation A, B

(g) prediction 95.4 (h) prediction 86.4 (i) prediction 0

Figure 5.17: Segmented US images by small IU-Net+encoder and their achieved DSC in percentages. The red boundary represents segmentation A; the yellow boundary denotes segmentation B; the blue boundary is the prediction

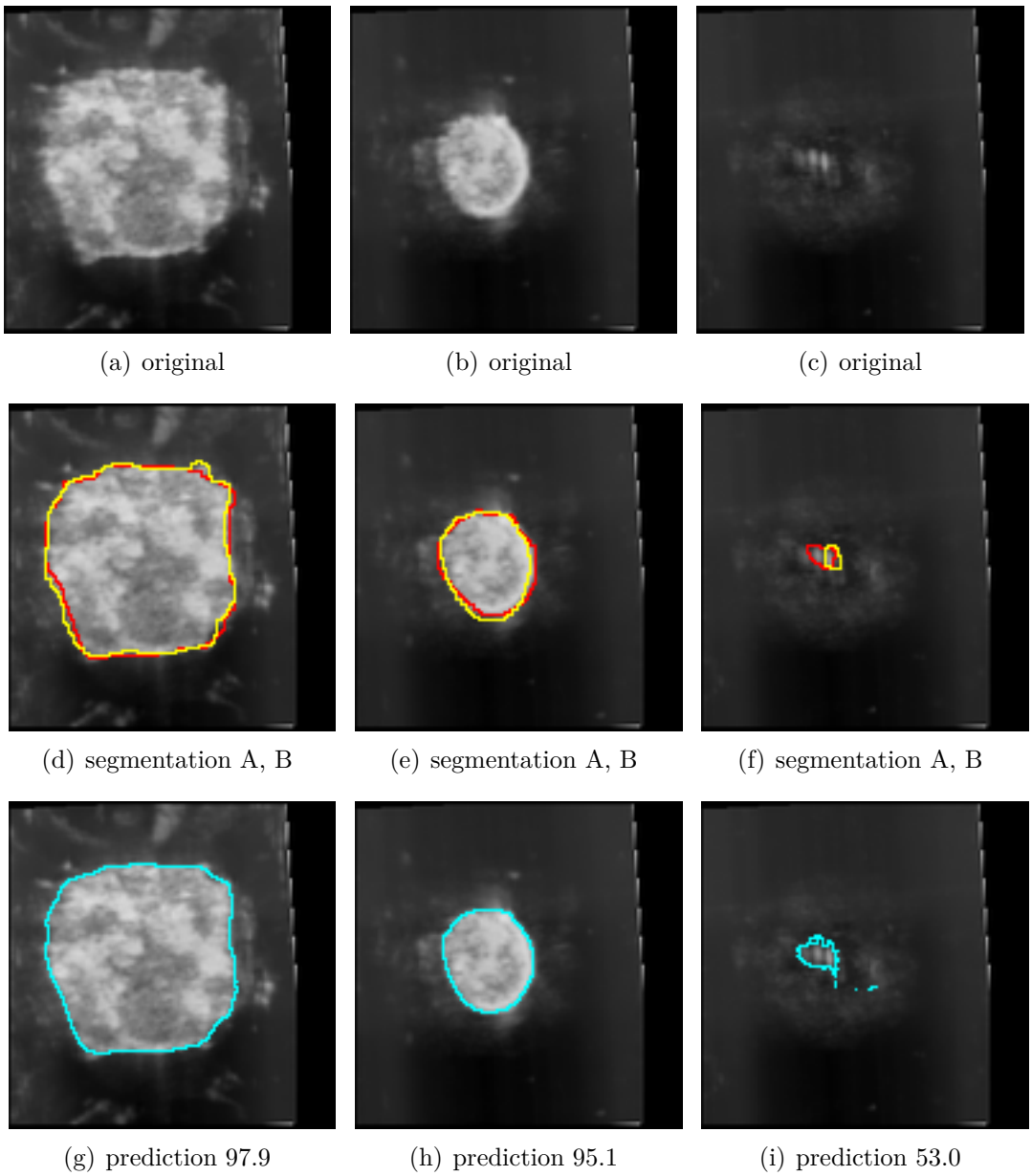


(a) original (b) original (c) original

(d) segmentation A, B (e) segmentation A, B (f) segmentation A, B

(g) prediction 97.4 (h) prediction 93.5 (i) prediction 85.6

Figure 5.18: Segmented US images by mini IU-Net and their achieved DSC in percentages. the red boundary represents segmentation A; The yellow boundary denotes segmentation B; the blue boundary is the prediction



(a) original (b) original (c) original

(d) segmentation A, B (e) segmentation A, B (f) segmentation A, B

(g) prediction 97.9 (h) prediction 95.1 (i) prediction 53.0

Figure 5.19: Segmented US images by mini IU-Net+encoder and their achieved DSC in percentages. The red boundary represents segmentation A; the yellow boundary denotes segmentation B; the blue boundary is the prediction

From Figures 5.12, 5.13, 5.14, 5.15, 5.16, 5.17, 5.18 and 5.19, we draw a conclusion that our models achieve low accuracies, when the tissue is small, see Figure 5.12 (a), (d), (g), Figure 5.13 (c), (f), (i), Figure 5.14 (c), (f), (i), Figure 5.15 (b), (e), (h), Figure 5.16 (a), (d), (g), Figure 5.17 (c), (f), (i), Figure 5.18 (c), (f), (i), Figure 5.19 (c), (f), (i).

When the segmentation A and B are quite different (see Figure 5.14 (e) (h), Figure 5.15 (d), (g)), it appears that the prediction of our models is more accurate than the segmentation A and B.

In addition, we give the accuracy after post processing in Table 5.10.

model	$\overline{\mathcal{D}}$		$\overline{\text{MSE}}$	
	mean	std	mean	std
U-Net	91.0%	0.15	2.02×10^{-2}	2.22×10^{-2}
U-Net+encoder	90.6%	0.16	2.07×10^{-2}	2.29×10^{-2}
IU-Net	91.5%	0.12	2.03×10^{-2}	2.21×10^{-2}
IU-Net+encoder	**92.0%**	**0.10**	$\mathbf{2.00 \times 10^{-2}}$	$\mathbf{2.16 \times 10^{-2}}$
small IU-Net	91.3%	0.14	2.05×10^{-2}	2.19×10^{-2}
small IU-Net+encoder	91.0%	0.14	2.06×10^{-2}	2.25×10^{-2}
mini IU-Net	89.9%	0.16	2.30×10^{-2}	2.40×10^{-2}
mini IU-Net+encoder	89.9%	0.17	2.20×10^{-2}	2.35×10^{-2}

TABLE 5.10: Accuracy after post processing

From Tables 5.7 and 5.10, post processing improves the accuracy a little. For example, DSC of mini IU-Net is improved from 89.7% to 89.9%; MSE of mini IU-Net is improved from 2.40×10^{-2} to 2.30×10^{-2}.

Chapter 6

Conclusion and improvement

We build a new CNN model IU-Net based on inception network and U-Net. We experiment with IU-Net and other two smaller version, small and mini IU-Net. The experimental results show that all models achieve the accuracy at the same level. The highest DSC is achieved by IU-Net which trained with an encoder. In addition, the number of trainable parameters in IU-Net is 3.8× smaller than the number of trainable parameters in U-Net. The experimental results also shows that training with an autoencoder can remove the small area, although autoencoder does not improve the accuracy of all models. In addition, we propose a new segmentation method which merges the segmentation results from three directions. However, this method does not improve the accuracy so we do not report the experimental results in this thesis.

Here are two future research directions:

1. The photoacoustic tomography (PAT) is another method to detect breast lesion. We may combine PAT and US images as the input of the neural network to improve the accuracy of segmentation. For example, we may build a neural network with multiple input images. To achieve this goal, we require the GT of PAT images to train a CNN. We can also employ the GT of US on the premise

that each slice of 3D US and PAT images are matched, that is, the GT of US and PAT images are very similar.

2. In our research, we use a 2D CNN model to segment every slice of 3D US images. We can do the segmentation by a 3D CNN model such as 3D U-Net [13] and 3D fully convolutional network [60]. The problem is that a 3D CNN model is normally large and hard to be trained. We can try to buid a small 3D CNN model or propose a better training method. In addition, we may regard a 3D US image as a time series and build a convolutional RNN [70] to segment 3D US images.

Bibliography

[1] SHARCNET (www.sharcnet.ca) is a consortium of 18 colleges, universities and research institutes operating a network of high-performance computer clusters across south western, central and northern Ontario.

[2] M. Alemán-Flores, L. Álvarez, and V. Caselles. Texture-oriented anisotropic filtering and geodesic active contours in breast tumor ultrasound segmentation. *Journal of Mathematical Imaging and Vision* 28(1) (2007), 81–97.

[3] E. A. Ashton and K. J. Parker. Multiple resolution Bayesian segmentation of ultrasound images. *Ultrasonic imaging* 17(4) (1995), 291–304.

[4] A. Badhe. Using Deep Learning Neural Networks To Find Best Performing Audience Segments. *International Journal of Scientific & Technology Research* 4(8) (2015), 30–31.

[5] V. Badrinarayanan, A. Kendall, and R. Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *arXiv preprint arXiv:1511.00561* (2015).

[6] A. Berg, J. Deng, and L. Fei-Fei. Large scale visual recognition challenge 2010, 2010. *URL http://www. image-net. org/challenges/LSVRC/2010/index* (2011).

[7] D. Boukerroui, O. Basset, N. Guerin, and A. Baskurt. Multiresolution texture based adaptive clustering algorithm for breast lesion segmentation. *European Journal of Ultrasound* 8(2) (1998), 135–144.

[8] *Breast cancer statistics*. http://www.cancer.ca/en/cancer-information/cancer-type/breast/statistics/?region=on.

[9] *Build a simple Image Retrieval System with an Autoencoder*. https://towardsdatascience.com/build-a-simple-image-retrieval-system-with-an-autoencoder-673a262b7921.

[10] K. Chellapilla, S. Puri, and P. Simard. High performance convolutional neural networks for document processing. In: *Tenth International Workshop on Frontiers in Handwriting Recognition*. Suvisoft. 2006.

[11] H.-H. Chiang, J.-Z. Cheng, P.-K. Hung, C.-Y. Liu, C.-H. Chung, and C.-M. Chen. Cell-based graph cut for segmentation of 2D/3D sonographic breast images. In: *Biomedical Imaging: From Nano to Macro, 2010 IEEE International Symposium on*. IEEE. 2010, 177–180.

[12] F. Chollet et al. *Keras*. 2015.

[13] Ö. Çiçek, A. Abdulkadir, S. S. Lienkamp, T. Brox, and O. Ronneberger. 3D U-Net: learning dense volumetric segmentation from sparse annotation. In: *International conference on medical image computing and computer-assisted intervention*. Springer. 2016, 424–432.

[14] D. C. Cireşan, A. Giusti, L. M. Gambardella, and J. Schmidhuber. Mitosis detection in breast cancer histology images with deep neural networks. In: *International Conference on Medical Image Computing and Computer-assisted Intervention*. Springer. 2013, 411–418.

[15] A. Clark. *Pillow (PIL Fork) Documentation*. 2015.

[16] *Deep Learning Tutorial for Kaggle Ultrasound Nerve Segmentation competition, using Keras*. github repository https://github.com/jocicmarko/ultrasound-nerve-segmentation.

[17] C. DeSantis, J. Ma, L. Bryan, and A. Jemal. Breast cancer statistics, 2013. *CA: a cancer journal for clinicians* 64(1) (2014), 52–62.

[18] L. R. Dice. Measures of the amount of ecologic association between species. *Ecology* 26(3) (1945), 297–302.

[19] H. Eggemann, T. Ignatov, S. D. Costa, and A. Ignatov. Accuracy of ultrasound-guided breast-conserving surgery in the determination of adequate surgical margins. *Breast cancer research and treatment* 145(1) (2014), 129–136.

[20] J. Fajdic, D. Djurovic, N. Gotovac, and Z. Hrgovic. Criteria and procedures for breast conserving surgery. *Acta Informatica Medica* 21(1) (2013), 16.

[21] R. Fakoor, F. Ladhak, A. Nazi, and M. Huber. Using deep learning to enhance cancer diagnosis and classification. In: *Proceedings of the International Conference on Machine Learning*. Vol. 28. 2013.

[22] I. S. Fentiman, A. Fourquet, and G. N. Hortobagyi. Male breast cancer. *The Lancet* 367(9510) (2006), 595–604.

[23] R. C. Gonzalez, R. E. Woods, S. L. Eddins, et al. *Digital image processing using MATLAB.* Vol. 624. Pearson-Prentice-Hall Upper Saddle River, New Jersey, 2004.

[24] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In: *Advances in neural information processing systems*. 2014, 2672–2680.

[25] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, 770–778.

[26] C. F. Higham and D. J. Higham. Deep learning: An introduction for applied mathematicians. *arXiv preprint arXiv:1801.05894* (2018).

[27] Y.-L. Huang and D.-R. Chen. Watershed segmentation for breast tumor in 2-D sonography. *Ultrasound in medicine & biology* 30(5) (2004), 625–632.

[28] S.-F. Huang, Y.-C. Chen, and W. K. Moon. Neural network analysis applied to tumor segmentation on 3D breast ultrasound images. In: *2008 5th IEEE International Symposium on Biomedical Imaging: From Nano to Macro*. IEEE. 2008, 1303–1306.

[29] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167* (2015).

[30] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[31] N. M. Krekel, M. H. Haloua, A. M. L. Cardozo, R. H. de Wit, A. M. Bosch, L. M. de Widt-Levert, S. Muller, H. van der Veen, E. Bergers, E. S. d. L. de Klerk, et al. Intraoperative ultrasound guidance for palpable breast cancer excision (COBALT trial): a multicentre, randomised controlled trial. *The lancet oncology* 14(1) (2013), 48–54.

[32] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In: *Advances in neural information processing systems*. 2012, 1097–1105.

[33] R. LaLonde and U. Bagci. Capsules for Object Segmentation. *arXiv preprint arXiv:1804.04241* (2018).

[34] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86(11) (1998), 2278–2324.

[35] B. Liu, H.-D. Cheng, J. Huang, J. Tian, X. Tang, and J. Liu. Fully automatic and segmentation-robust classification of breast tumors based on local texture analysis of ultrasound images. *Pattern Recognition* 43(1) (2010), 280–298.

[36] M. Liu, W. Wu, Z. Gu, Z. Yu, F. Qi, and Y. Li. Deep learning based on Batch Normalization for P300 signal detection. *Neurocomputing* 275 (2018), 288–297.

[37] Y. Liu, X. Hou, J. Chen, C. Yang, G. Su, and W. Dou. Facial expression recognition and generation using sparse autoencoder. In: *2014 International Conference on Smart Computing (SMARTCOMP)*. IEEE. 2014, 125–130.

[38] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, 3431–3440.

[39] P. Luc, C. Couprie, S. Chintala, and J. Verbeek. Semantic segmentation using adversarial networks. *arXiv preprint arXiv:1611.08408* (2016).

[40] A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In: *Proc. icml*. Vol. 30. 1. 2013, 3.

[41] A. Madabhushi and D. Metaxas. Automatic boundary extraction of ultrasonic breast lesions. In: *Biomedical Imaging, 2002. Proceedings. 2002 IEEE International Symposium on*. IEEE. 2002, 601–604.

[42] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Y. Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah,

Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015.

[43] O. Olsha, D. Shemesh, M. Carmon, O. Sibirsky, R. A. Dalo, L. Rivkin, and I. Ashkenazi. Resection margins in ultrasound-guided breast-conserving surgery. *Annals of surgical oncology* 18(2) (2011), 447–452.

[44] S. Pereira, A. Pinto, V. Alves, and C. A. Silva. Brain tumor segmentation using convolutional neural networks in MRI images. *IEEE transactions on medical imaging* 35(5) (2016), 1240–1251.

[45] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In: *International Conference on Medical image computing and computer-assisted intervention*. Springer. 2015, 234–241.

[46] H. R. Roth, C. T. Lee, H.-C. Shin, A. Seff, L. Kim, J. Yao, L. Lu, and R. M. Summers. Anatomy-specific classification of medical images using deep convolutional nets. *arXiv preprint arXiv:1504.04003* (2015).

[47] S. Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747* (2016).

[48] D. E. Rumelhart, G. E. Hinton, R. J. Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling* 5(3) (1988), 1.

[49] S. Sabour, N. Frosst, and G. E. Hinton. Dynamic routing between capsules. In: *Advances in Neural Information Processing Systems*. 2017, 3856–3866.

[50] B. Sahiner, A. Ramachandran, H.-P. Chan, M. A. Roubidoux, L. M. Hadjiiski, M. A. Helvie, N. Petrick, and C. Zhou. Three-dimensional active contour model for characterization of solid breast masses on three-dimensional ultrasound images. In: *Medical Imaging 2003: Image Processing*. Vol. 5032. International Society for Optics and Photonics. 2003, 405–414.

[51] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry. How Does Batch Normalization Help Optimization?(No, It Is Not About Internal Covariate Shift). *arXiv preprint arXiv:1805.11604* (2018).

[52] L. G. Shapiro and G. Linda. stockman, George C. *Computer Vision, Prentice hall. ISBN 0-13-030796-3* (2002).

[53] J. Shi and J. Malik. Normalized cuts and image segmentation. *Departmental Papers (CIS)* (2000), 107.

[54] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[55] T. Sørensen. A method of establishing groups of equal amplitude in plant sociology based on similarity of species and its application to analyses of the vegetation on Danish commons. *Biol. Skr.* 5 (1948), 1–34.

[56] H. Su, F. Liu, Y. Xie, F. Xing, S. Meyyappan, and L. Yang. Region segmentation in histopathological breast cancer images using deep convolutional neural network. In: *Biomedical Imaging (ISBI), 2015 IEEE 12th International Symposium on*. IEEE. 2015, 55–58.

[57] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, 1–9.

[58] D. Terzopoulos. On matching deformable models to images. In: *Topical Meeting on Machine Vision Tech. Digest Series*. Vol. 12. 1987, 160–167.

[59] T. Tieleman and G. Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning* 4(2) (2012), 26–31.

[60] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri. Learning spatiotemporal features with 3d convolutional networks. In: *Proceedings of the IEEE international conference on computer vision*. 2015, 4489–4497.

[61] *Ultrasound nerve segmentation using Keras (1.0.7)*. github repository https://github.com/EdwardTyantov/ultrasound-nerve-segmentation.

[62] A. Veit, M. J. Wilber, and S. Belongie. Residual networks behave like ensembles of relatively shallow networks. In: *Advances in neural information processing systems*. 2016, 550–558.

[63] F. Visin, K. Kastner, K. Cho, M. Matteucci, A. Courville, and Y. Bengio. Renet: A recurrent neural network based alternative to convolutional networks. *arXiv preprint arXiv:1505.00393* (2015).

[64] S.-H. Wang, C. Tang, J. Sun, J. Yang, C. Huang, P. Phillips, and Y.-D. Zhang. Multiple sclerosis identification by 14-layer convolutional neural network with batch normalization, dropout, and stochastic pooling. *Frontiers in neuroscience* 12 (2018).

[65] P. R. Winters. Forecasting sales by exponentially weighted moving averages. *Management science* 6(3) (1960), 324–342.

[66] G. Wu, X. Shao, Z. Guo, Q. Chen, W. Yuan, X. Shi, Y. Xu, and R. Shibasaki. Automatic building segmentation of aerial imagery using multi-constraint fully convolutional networks. *Remote Sensing* 10(3) (2018), 407.

[67] M. Xian, Y. Zhang, and H.-D. Cheng. Fully automatic segmentation of breast ultrasound images based on breast characteristics in space and frequency domains. *Pattern Recognition* 48(2) (2015), 485–497.

[68] M. Xian, Y. Zhang, H.-D. Cheng, F. Xu, B. Zhang, and J. Ding. Automatic breast ultrasound image segmentation: A survey. *Pattern Recognition* 79 (2018), 340–355.

[69] G. Xiao, M. Brady, J. A. Noble, and Y. Zhang. Segmentation of ultrasound B-mode images with intensity inhomogeneity correction. *IEEE transactions on medical imaging* 21(1) (2002), 48–57.

[70] S. Xingjian, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-c. Woo. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. In: *Advances in neural information processing systems*. 2015, 802–810.

[71] Y. Xu, Y. Wang, J. Yuan, Q. Cheng, X. Wang, and P. L. Carson. Medical breast ultrasound image segmentation by machine learning. *Ultrasonics* 91 (2019), 1–9.

[72] A. Yezzi, S. Kichenassamy, A. Kumar, P. Olver, and A. Tannenbaum. A geometric snake model for segmentation of medical imagery. *IEEE Transactions on medical imaging* 16(2) (1997), 199–209.

[73] R. Yu, X. Fu, H. Jiang, C. Wang, X. Li, M. Zhao, X. Ying, and H. Shen. Remote Sensing Image Segmentation by Combining Feature Enhanced with Fully Convolutional Network. In: *International Conference on Neural Information Processing*. Springer. 2018, 406–415.

[74] Y.-D. Zhang, C. Pan, J. Sun, and C. Tang. Multiple sclerosis identification by convolutional neural network with dropout and parametric ReLU. *Journal of computational science* 28 (2018), 1–10.