Li Lu

Multi-label Medical Image Segmentation using Convolutional Neural Networks

Master's thesis in Simulation and Visualization
Supervisor: Kjell-Inge Gjesdal, Robin Trulssen Byeand
June 2020

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of ICT and Natural Science

Abstract

Deep learning algorithms, in particular convolutional neural networks, are becoming a promising research tool in medical image segmentation. This thesis attempted to use annotated knee MRI images provided by Sunnmøre MR-Klinikk to study three architectures of convolutional neural networks including 3D U-net, DeepLab and a type of combined neural network and decide a system that achieves high accuracy with regard to a segmentation task which has 13 extremely imbalanced classes. For 3D U-net architecture, four advanced blocks including residual blocks, residual SE blocks, dense blocks and dense SE blocks were used to replace the standard convolution blocks in it. Because the two SE structures performed better than the other two in 3D U-net in the experiments, 3D U-net with residual SE blocks and 3D U-net with dense SE blocks were chosen as the basic networks for the other two architectures. The experiments show that DeepLab architecture is the most efficient one among the three architectures. It can achieve relatively high accuracies with both loss functions used in this thesis including dice loss and weighted dice loss.

Preface

The technology of Artificial Intelligence (AI) has been widely applied in various industries and achieved great success in recent decades. AI also promises to bring innovation to the field of medicine. With the development of increased computing power and deep learning algorithms, convolutional neural networks have the potential to become a popular technique for medical image analyses. Doctors have been overwhelmed with the increased amount of diagnostic data: MRI, CT, X-ray, etc. Convolutional neural networks may be able to effectively process these images and then reduce the labor force involved. This work aims to study several convolutional neural networks used for medical image segmentation in order to choose the architecture which can achieve higher segmentation accuracy on a specific dataset. The sampling of human (knee) data was approved by the Regional Committee for Medical and Health Research Ethics (REK nr. 61225).

This thesis is the final work for the master degree in the Simulation and Visualization program at the Norwegian University of Science and Technology (NTNU), Department of ICT and Natural Sciences. I have written this thesis in an order that I progressively learnt and understood the algorithms of convolutional neural networks, and then developed the neural networks based on the knowledge. I think this order is clear for me to explain the work in this thesis, which might be a little different from other theses. For example, I keep the chapter Introduction short and describe lots of neural networks in the chapter Related work, which some researchers may prefer to put it in the chapter Introduction.

As an international student in NTNU, I met lots of problems during the period of my study life. I would like to thank all the people who helped me when I was confused, sad, and helpless. Their kindness encourages me to be myself no matter what difficulties may come my way.

Contents

1. Introduction 1
1.1 Background & motivation 2
1.2 Scope 3
1.3 Objectives 4
2. Theory 5
2.2 Deep learning 6
2.2.1 Neural networks 6
2.2.2 Convolutional neural networks 8
2.2.2.1 Convolution layer 8
2.2.2.2 Pooling layer 10
2.2.2.3 Fully-connected layer 11
2.2.2.4 2D convolutional neural networks 11
2.2.2.5 3D convolutional neural networks 13
2.2.3 Improving neural networks 14
2.2.3.1 Optimization algorithms 14
Mini-batch gradient descent 14
Gradient descent with Momentum 15
RMSProp 16
Adam 17
2.2.3.2 Hyperparameter tuning 17
Learning rate 18
Batch-size 18
Epochs 19
Activation function 19
2.2.3.3 Regularization 20
Data augmentation 20
L1 and L2 regularization 21
Dropout 22
Early stopping 22
2.2.3.4 Normalization 23
2.3 MRI image 24
3. Related work 26

3.1 Computer vison 27
3.1.1 Image classification 27
3.1.2 Object detection 29
3.1.3 Object tracking 31
3.1.4 Semantic segmentation 32
3.1.5 Instance segmentation 33
3.2 Semantic segmentation 33
3.2.1 Neural networks 34
3.2.1.1 FCN 34
3.2.1.2 SegNet 36
3.2.1.3 DeepLab 37
3.2.1.4 Unet 38
3.2.1.5 Other networks 39
3.2.2 Loss functions 39
3.2.2.1 Cross-entropy 39
3.2.2.2 Weighted cross-entropy 40
3.2.2.3 Focal loss 40
3.2.2.4 Dice loss 40
3.2.2.5 Generalized Dice loss 41
3.2.2.6 IOU loss 41
3.2.2.7 Combined loss 42
4. Method 43
4.1 Neural Networks 44
4.1.1 3D U-net variants 45
4.1.1.1 Residual Block 46
4.1.1.2 Residual SE Block 46
4.1.1.3 Dense Block 48
4.1.1.4 Dense SE Block 49
4.1.2 DeepLab variants 49
4.1.2.1 Atrous convolution 50
4.1.2.2 Depthwise separable convolution 51
4.1.2.3 ASPP 52
4.1.2.4 Network 54
4.1.3 Combined neural networks 55

4.1.3.1 Edge detection networks 56
4.1.3.2 Network 57
4.2 Loss functions 58
4.2.1 For 3D U-net variants and DeepLab variants 58
4.2.2 For combined network 59
4.3 Metrics 60
5. Experiments and results 62
5.1 Datasets 63
5.1.1 Dataset1 (with 10 labels) 63
5.1.2 Dataset2 (with 13 labels) 65
5.2 Training and Results 66
5.2.1 Experiments on dataset1 67
5.2.1.1 Preprocessing 67
Cutting image into patches 67
Data format 68
5.2.1.2 Training of 3D U-net and its variants 69
Preliminary models 69
Model with residual SE blocks 71
Models with Dense, Dense SE blocks 72
Final result 73
5.2.2 Experiments on dataset2 76
5.2.2.1 Preprocessing 76
Downsampling 76
Cutting image into patches 77
Data format 78
Generate ground truth of edges 78
5.2.2.2 Training with downsampling images 79
Larger Patch size vs. larger batch size 79
3D U-net variants 80
Combined network 82
Traning time: DeepLab variants vs. other networks 83
5.2.2.3 Training with original images 84
DeepLab variants 85
Other networks 88

6. Discussions 91
6.1 Conclusions 92
6.2 Contributions 93
6.3 Future work 95

List of Figures

Figure 2-1. AI, machine learning, and deep learning 6
Figure 2-2. Neural network 7
Figure 2-3. The calculation of 2D CNN 9
Figure 2-4. Padding 9
Figure 2-5. Max pooling 11
Figure 2-6. An example of 2D CNN 12
Figure 2-7. Multi-channels 2D CNN 12
Figure 2-8. 3D convolution 13
Figure 2-9. Exponentially Weighted averages 15
Figure 2-10. Gradient descent with momentum 16
Figure 2-11. Large batch size vs. small batch size 19
Figure 2-12. Sigmoid and tanh 20
Figure 2-13. Using conditional GANs for data augmentation 21
Figure 2-14. Dropout 22
Figure 2-15. T1, PD and FS MRI images 25
Figure 3-1. An example of image classification 27
Figure 3-2. An example of object detection 30
Figure 3-3. The implementation stages of R-CNN 30
Figure 3-4. Object tracking 32
Figure 3-5. An example of Semantic segmentation. 33
Figure 3-6. Instance segmentation and semantic segmentation 33
Figure 3-7. Fully Convolutional Network 34
Figure 3-8. Deconvolution 35
Figure 3-9. Fusing information from layers with different strides 35
Figure 3-10. Results of refining FCN 36
Figure 3-11. SegNet 36
Figure 3-12. DeepLabV3Plus 37
Figure 3-13. U-net 38
Figure 4-1. The network uses 3D U-net as the backbone 45
Figure 4-2. Residual SE block 47
Figure 4-3. Dense SE block 48
Figure 4-4. The architecture of encoder-decoder with atrous convolution 50

Figure 4-5. Atrous convolution 51
Figure 4-6. Depthwise separable convolution 51
Figure 4-7. ASPP (Atrous Spatial Pyramid Pooling) 53
Figure 4-8. Atrous separable convolution 53
Figure 4-9. The architecture of DeepLab variants 54
Figure 4-10. CASENet 56
Figure 4-11. Components of CASENet 57
Figure 4-12. Combined neural network 58
Figure 5-1. An example of segmentation on dataset1 64
Figure 5-2. The frequency of voxels for each class on dataset1 64
Figure 5-3. An example of segmentation on dataset2 65
Figure 5-4. The frequency of voxels for each class on dataset2 66
Figure 5-5. Patch-wise training due to the limitation of GPU memory 67
Figure 5-6. Segmentation results of V-net with dice loss 69
Figure 5-7. Performance matrix for V-net 70
Figure 5-8. Segmentation results of V-net with Weighted dice loss 70
Figure 5-9. Segmentation results of U-net with residual SE 71
Figure 5-10. Performance matrix for V-net with residual SE blocks 71
Figure 5-11. The process to get the final result 75
Figure 5-12. Performance matrix for the final result 76
Figure 5-13. The method of cutting patches 77
Figure 5-14. Generating edge ground truth 79
Figure 5-15. Larger Patch size vs. larger batch size 80
Figure 5-16. the segmentation restults on downsampling dataset 81
Figure 5-17. Convergence line of the combined neural network 83
Figure 5-18. Convergence curve of DeepLab no downsampling dataset 84
Figure 5-19. Performance matrix for the DenseDeeplab with dice loss 86
Figure 5-20. Segmentation results of networks on dataset2 90

List of Tables

Table 3-1. A rough development history of neural networks 28
Table 4-1. The numbers of convolution blocks in grey blocks 46
Table 5-1. The abbreviations and values of classes (organs) on dataset1 63
Table 5-2. The hyperparameters of dense blocks 72
Table 5-3. Performances of 3 dense structures with SE and without SE 73
Table 5-4. Performances of U-net and its variants 74
Table 5-5. Training time on different resolutions 77
Table 5-6. The restults of 3D U-net variants on downsampling dataset 81
Table 5-7. Training results of two combined network 82
Table 5-8. Training time of an epoch on downsampling dataset 84
Table 5-9. Performance of DeepLab variants 85
Table 5-10. Performance of ResidualDeeplab with more data 87
Table 5-11. Performance of networks on original dataset2 88

1. Introduction

In the recent decade the development of computational power has made deep learning algorithms used for analyzing medical images possible. Segmentation is a common task in medical image analysis. For the task of knee MRI image segmentation in this thesis, three types of architectures have been developed including 3D U-net variants, DeepLab variants, and a type of neural network which combines the 3D U-net variants with an edge detection neural network. This chapter will introduce the background and motivation for this thesis, and also declare the scope and objectives of our work.

1.1 Background & motivation

Accurate segmentation of organs is essential to support clinical workflows in multiple domains, including diagnostic interventions, treatment planning etc. However, manual segmentation of anatomical structures is labor-intensive and therefore expensive, which motivates automated segmentation researches [1].

According to Geert et al. [2], artificial intelligence technologies have been applied in automated medical image analysis since 1970s. Initially, researchers used low-level pixel processing techniques such as edge and line detector filters and mathematical modelling to build rule-based systems which have been described as GOFAI (good old-fashioned artificial intelligence). These systems developed based on many if-then-else statements for particular tasks are often brittle. Supervised techniques were introduced to construct a system at the end of the 1990s. Building a computer model and then training it by using related datasets is the crucial idea to develop such a system. Models based on deep learning algorithms can extract features from the images efficiently, and therefore have been widely used in computer vision.

Among various types of neural networks built for medical image analysis, the most successful type is convolutional neural networks (CNNs). The advances in computational power in recent decades made it possible to train complicated neural networks such as deep convolutional neural networks (DCNNs) with large datasets, which has high potential in medical image segmentation. Such a system can increase the segmentation accuracy, and also decrease the time and labor force involved.

Knee joint is one of the most important joints of the human body, and is frequently injured in sports and accidents. The Magnetic resonance imaging (MRI) is a widely used technique to image patients' knee. Automated knee segmentation can assist orthopedists in examination and treatment of various kinds of knee lesions.

1.2 Scope

The scope of this thesis is to study how to apply technologies of deep learning, especially convolutional neural networks (CNN), for medical image segmentation. Details about knee joints or MRI technique in medical domain will not be discussed unless it is necessary for explaining the deep learning techniques used in this thesis.

Neural networks in deep learning will be introduced first, and then we will discuss the most commonly used components and techniques in convolutional neural networks. How to choose the best neural network and how to improve the neural network are two of significant research domains in deep learning. These techniques will be introduced next in Chapter 2.

Computer vision is the hottest research area in deep learning. Besides semantic segmentation, which is the domain of this thesis, several other common tasks in computer vision will be also introduced in Chapter 3 including image classification, object detection, object tracking, and instance segmentation. The techniques used in these tasks can inspire innovative ideas in semantic segmentation. Then we will focus on the related work in semantic segmentation. Neural networks and loss functions are the two of most crucial parts which decide the performance of the deep learning system, so more details in these two areas will be introduced.

In Chapter 4, the methods of building the three architectures including 3D U-net, DeepLab and combined neural network will be introduced. Firstly, we will discuss how to build the four variants of 3D U-net using residual blocks, residual squeeze-and-excitation (SE) blocks, dense blocks and dense SE blocks respectively. Neural networks developed based on DeepLabv3plus will be introduced next. To improve the segmentation accuracy, we will also discuss a type of combined neural networks which combine an edge detection neural network with the segmentation neural network. The details of these three architectures will be explained in this chapter.

In Chapter 5, we will discuss the details of the experiments including the datasets used to train the neural networks described in Chapter 4, some implementation details of the deep learning system and the performances of these neural networks. Plenty of experiments have been designed to compare the performance of the neural networks

in this chapter. The reason behind the performance will be discussed based on the results as well.

The last chapter will summarize the experiments and discussions of the neural networks used in this thesis, and also try to generalize the conclusion from the experiments. In the end, we will discuss the future work based on the work finished in this thesis.

1.3 Objectives

Small organ segmentation is always a challenge for medical image segmentation. To address this problem, various types of neural networks and loss functions have been proposed in recent years. However, for different datasets, their performances are different. The purpose is to find the best segmentation neural network and the appropriate loss function on the specific dataset used in this thesis.

Another big challenge for this thesis is that there are multiple classes on our datasets. Their frequencies are extremely imbalanced, which will be introduced in chapter 5.1. The neural network needs to achieve high performance on all classes including both smaller and larger tissues. If unweighted loss function is used, it is possible that the neural networks intend to ignore the segmentation of small organs. However, if more attentions are paid on small targets, the neural networks may be apt to sacrifice the accuracy of large organs to increase the accuracy of small ones. The problem is how to find a method or a neural network which can balance the performance on both small and large organs. We will also discuss this problem in this thesis.

2. Theory

In this chapter, we will initially discuss the definitions and scopes of artificial intelligence, machine learning and deep learning. Deep learning is the emphasis of this thesis, and neural networks are the crucial part to understand deep learning algorithms. Subsequently, neural networks will be introduced next. Convolutional neural network (CNN) is the most successful neural network used in computer vision, which is the hottest research area in deep learning, and also the area of this thesis. Ultimately, we will discuss more about details regarding CNN that will be helpful for explanation of segmentation neural networks.

There are plenty of researches relating to improvement of neural networks. It is one of the crucial parts to improve the training efficiency of CNN. So, more details in this area will be introduced as well. The basic concepts of MRI images used on this thesis will be introduced last.

2.2 Deep learning

In the past decades, artificial intelligence, machine learning and deep learning have been applied in a wide range of industries and achieved great success. Countless articles and researches are about them, but how can we define them clearly and what is the difference between them?

Figure 2-1 shows the relations of artificial intelligence, machine learning and deep learning. Artificial intelligence (AI) means any technique that enables computers to mimic human intelligence. Machine learning is a subset of AI including algorithms which enable machines to improve their performances on some specific tasks with experiences. Deep learning is a subset of machine learning and excels at recognizing patterns. The algorithms of deep learning typically require a large number of data to train neural networks. Sometimes we use the terminology deep learning to indicate the process of training neural networks.

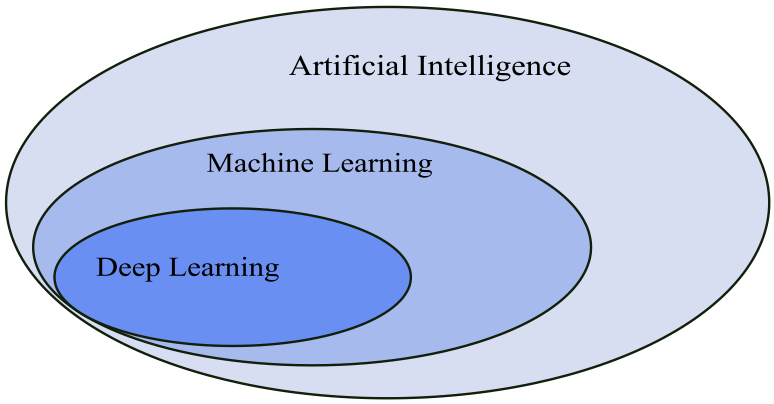


Figure 2-1. AI, machine learning, and deep learning

2.2.1 Neural networks

As we discussed above, the basic idea of deep learning is using large datasets to train neural networks. Neural Network can be understood as a computational model that works in a similar way to the neurons in human brains. They are designed to recognize patterns in the datasets which are used to train them.

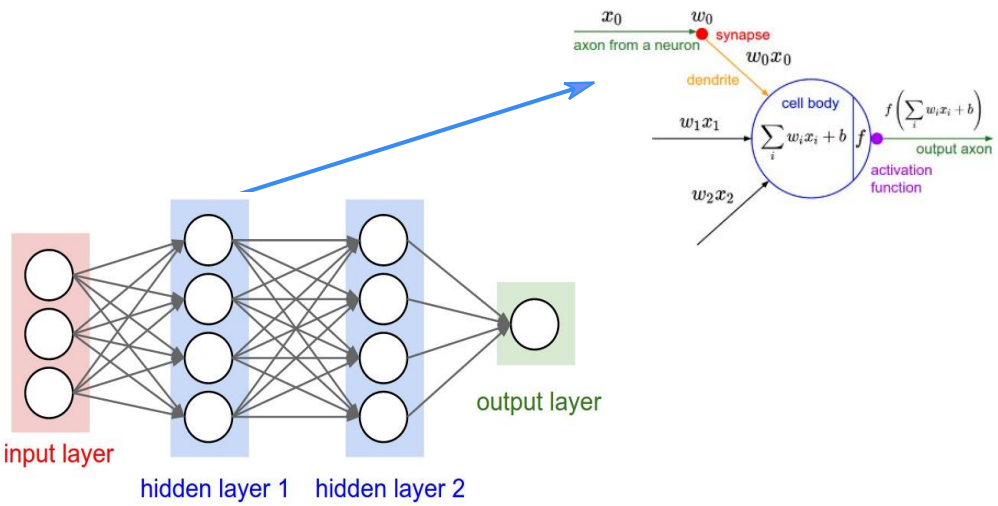


Figure 2-2. Neural network

Figure 2-2 [3] shows a shallow neural network which has two hidden layers. Each hidden layer has four neurons, which is the basic computational unit of neural network. The calculating process of each neuron is shown at the top right corner of the figure. The output is calculated according

$$y=f(\sum_{i=1}^{n} w_i x_i+b) \tag{2.1}$$

where x_i is the input data from previous layer, w_i is the weight which is learnable during training, b is the bias which is also learnable, f is activation function. The example shown at the top right corner has three inputs because the last layer of hidden layer 1, which is the input layer, has three neurons. For hidden layer 2, it has four inputs for each neuro. Bias can be deleted if normalization is chosen.

The calculations before activation function are linear. The representation capacity of a neural network will be limited if we only use linear calculation in it. So activation function is adopted to introduce non-linear features in order to increase its representation capacity, which enables the neural network to implement more complicated functions and then recognize the more intricate patterns behind the dataset.

In order to increase the representation capacity of neural networks we can increase the depth (use more hidden layers) or width (use more neuros in each layer) of the neural networks. However, it will bring a handful of problems. For example, larger memory

will be needed to train the neural networks. Actually lots of researches have been working on improving the representation capacity of neural networks without bringing problems at least without bringing more problems.

If we want to build a neural network from scratch, there are lots of elements we need to consider about such as the number of layers, the number of neuros for each layer and the activation function. However, there are dozens of existing neural networks which have various architectures and advantages we can choose or use as references. There are few researchers choosing to build a neural network from scratch now. They prefer to use the existing neural networks as the backbone network and then adjust them or add more advanced components according the feature of datasets they use.

2.2.2 Convolutional neural networks

The most popular research areas in deep learning are computer vision and natural language processing. Computer vision has wider applications in the fields of astronomy, medicine, transportation and navigation, and military industry. The most successful neural network in computer vision is convolutional neural networks (CNN), which use combinations of convolutions, pooling, and other techniques to extract features from images to recognize patterns behind the training data.

2.2.2.1 Convolution layer

Most studies in computer vision are dealing with two dimensional images. Understanding the computational process of 2D convolutions is extremely important for understanding the architectures of CNN and how it works. The convolutional calculation on 2D image is shown Figure 2-3.

In this example, the size of input image is 6×6, and the size of convolutional kernel is 3x3. The kernel is slipped on the image in a method as the colorful square frames in the input image show. Use the data in the kernel to multiply the corresponding data in the slipping windows, and then use the summary of them as the result in the output image. For example, for the first slipping window, the result -5 is calculated through $3\times1+1\times1+2\times1+0\times0+5\times0+7\times0+1\times(-1)+8\times(-1)+2\times(-1)=-5$.

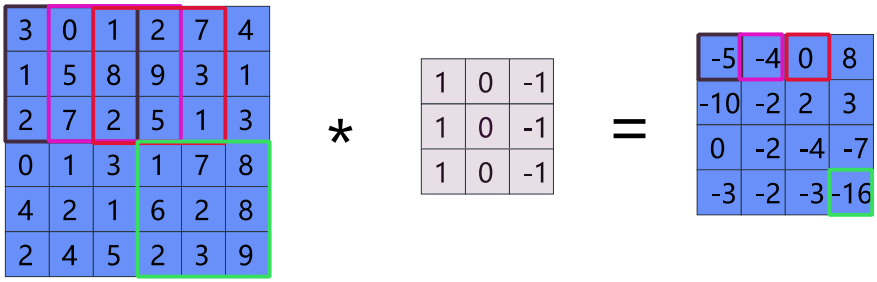


Figure 2-3. The calculation of 2D CNN

After this calculation the image is changed from 6×6 to 4×4. Sometimes we don't want to change the size of input image, so padding can be used. As shown in figure below, padding is adopted to enlarge the size of input image to 8×8, and then the size of output image is still 6×6 after convolutional calculation.

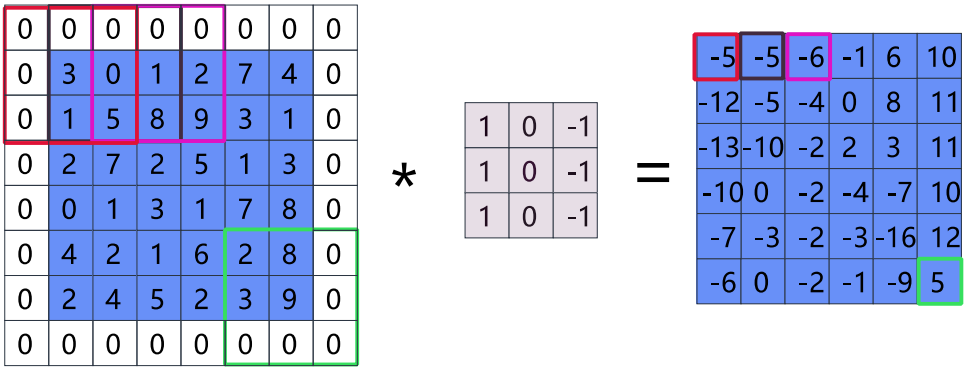


Figure 2-4. Padding

There is another advantage of padding. The pixels at the corners and on the edges can contribute more information to the output with padding compared with these without padding. For example, the pixel at the right top corner is used only once in convolutional calculation if we don't use padding, but the pixel in the middle can be used nine times at most. With padding this pixel will be used four times, which means the output image contains more information about this pixel. So, researchers usually choose to use padding when they build a convolutional neural network.

The most frequently used padding mode is same padding, which means the output image will have the same size as the input image. In such situation, the number of paddings added on each edge should be $(f - 1)/2$, where f is the size of convolutional kernel.

When same padding is used, strides can be used to reduce the image's size. Stride means the step of slipping the kernel window on images. For example, if we set the stride to be two, the output image will be reduced half in size when we use same padding. Strides can also be used for downsampling to increase the receptive field of an image. Another common operation called valid padding means there is no padding when employing convolution.

If we employ padding and strides together in convolutions, the size of output images will be changed from n×n to $(\frac{n + 2\times p - f}{s} + 1) \times (\frac{n + 2\times p - f}{s} + 1)$, where n is the input size of images, p is the padding number, f is the size of convolutional kernel, s is the stride.

The convolutional kernel is also called filter in convolutions. The filter in the example above is to detect the vertical edge in the image. Different filters can learn different features from the image. What the filter should be is the key point to extract features in convolution layer, and it is also what the convolutional neural network is designed to learn. Multiple convolutional filters can be used in a convolution layer. It is obvious that the number of filters used in the convolution decides how many feature maps will be output in this layer. We often call the different filters used in a convolution layer as channels. For example, 50 filters are adapted in a convolution layer, and then the size of output of this layer should be 50×n×n (50 is the number of filters, n×n is the size of an output feature map).

2.2.2.2 Pooling layer

Downsampling operations enable neural networks to get the features of higher levels, which can help the neural networks learn more about the input images. In convolutional neural networks, there are many techniques to implement downsampling. Using stride in convolution that we discussed above is one of them. Max pooling is another technique used for downsampling.

Figure 2-5 shows an example how to do max pooling. Same as convolution, a kernel window is used to slip on the input image. In this example, its size is 3×3. Choose the maximum one in the window as the result directly. The calculation involved in max pooling is simpler than convolution, but the result contains less information, which can be seen obviously in the example. However, it has fewer. There are other methods

to do pooling operation such as average pooling, which uses the average value of the pixels in the slipping window as the result.

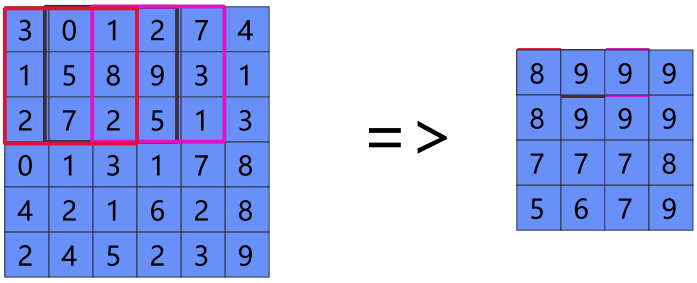


Figure 2-5. Max pooling

2.2.2.3 Fully-connected layer

For some computer vision tasks such as image classification, the output should be vectors (one-dimensional tensors) rather than images (two- dimensional tensors). In such situation, we can use fully-connected layer to flatten the output of convolution layer or pooling layer to vectors. Connecting multiple fully-connected layers with different numbers of neuros, which is same as neural network operations described in Chapter 2.2.1, can be used to generate vectors in different lengths.

2.2.2.4 2D convolutional neural networks

Convolutional neural networks are combinations of these operations including convolution layers, pooling layers, and fully-connected layer etc. Figure 2-6 [4] is an example of CNN used for classifying handwritten digits. The size of input image is 28×28×1. The first layer is convolution layer, which uses n1 convolutional filters. For each filter, the size is 5×5 with valid padding on the input image. The output of this layer is 24×24×n1 (n1 is the number of channels). Then max-pooling layer is adopted to reduce the size of images to 12×12×n1. Similar operations are employed in next two layers of conv_2 convolution and max-pooling. There is a tricky problem with regard to the filters used in the convolution layer. We will talk about it very soon. The two fully-connected layers are used to get the final output which is a vector whose length is 10 because there are 10 digits we need to classify. Dropout is used in the last layer, which is a way of regularization. More details about it technique will be introduced in the chapter that deals with improving neural network.

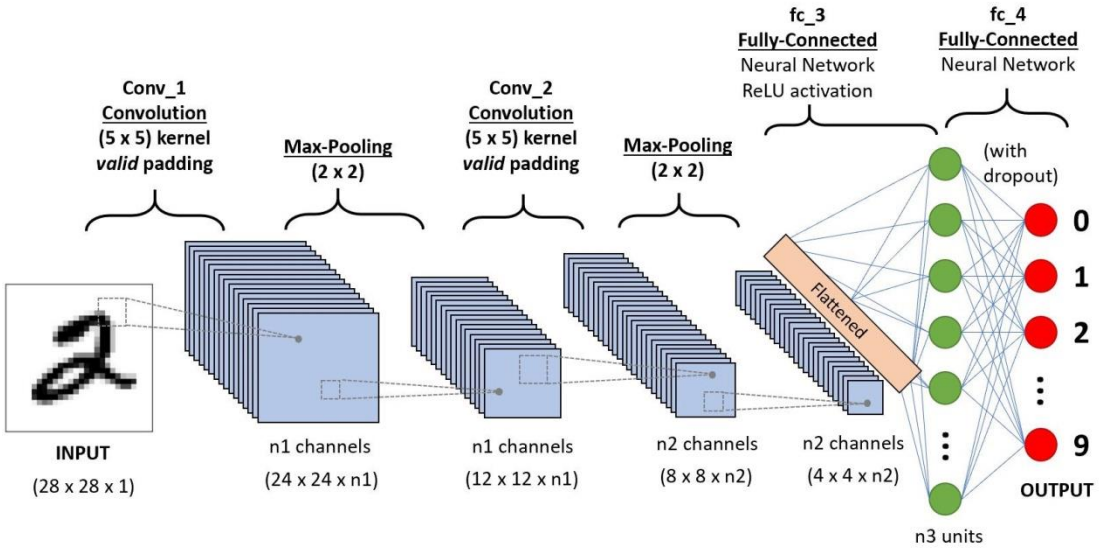


Figure 2-6. An example of 2D CNN

The convolutional neural network we discussed above is 2D convolutions used for dealing with two dimensional images whose pixels are represented by grey values, which means the input channel of the images is 1. However, some 2D images' pixels are represented by multiple values such as RGB images whose pixels are comprised by three values, which means the input channel of the images is 3. To deal with such an image, we use multi-channels 2D convolutions.

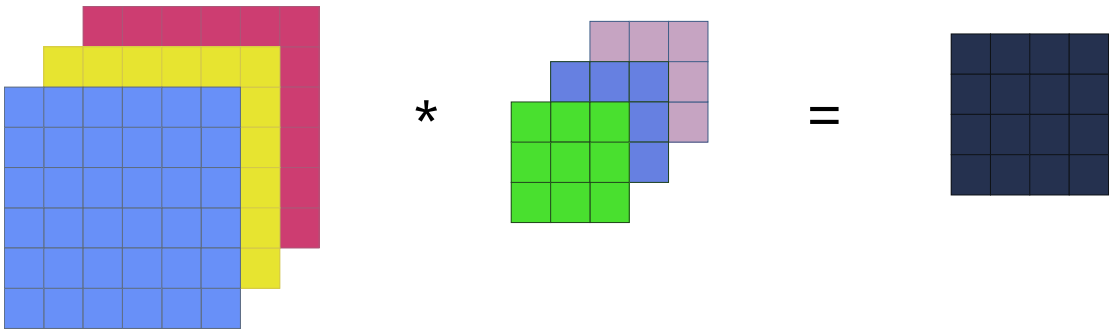


Figure 2-7. Multi-channels 2D CNN

Figure 2-7 shows the process how to do multi-channels convolution calculation. The size of the filter we use in this operation should have the same number of channels as the input image. In this example, the input channel is 3, so the filter's channel is 3 as well. We slip the filter on the input image in the same way as we deal with one

channel 2D image. The only difference is that we need to add 27 (9×3) addends for the summary instead of 9 addends.

The tricky problem mentioned when explaining the neural network above is that the filter's channel number must be same as the channel number of input image. When we put this rule in the middle of a neural network, it becomes the filter's channel number used in the convolution layer should be same as the number of filters used in the last convolution layer. For example, the filters in conv_2 convolution layer should be n1. It is the most important rule to understand how to use filters in different layers. 2D convolution is actually the operation on 3-dimension tensors, which is (channel, height, width) or (height, width, channel). Adding the number of samples, the input should be 4-dimension tensors.

2.2.2.5 3D convolutional neural networks

Some of images are 3 dimensions such as MRI or CT medical images, and videos. 3D convolution is used to analyze these types of images. The figure below shows the calculation of 3D convolution. Similar with 2D convolution, it can handle images with multiple channels. In the example below the input image has 3 channels. The filter used here also should have the same number of channels as the input images, which is 3. 3D convolution is actually the operation on 4-demensional tensors, which is (channel, height, width, depth) or (height, width, depth, channel). The convolutional filter is also 4D, and will be slipped on three directions. Adding the number of samples, the input should be 5-demensional tensors.

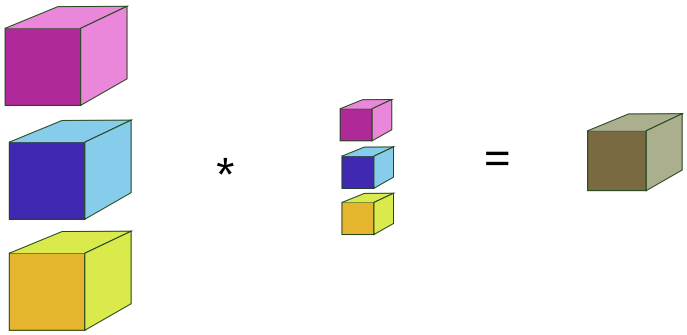


Figure 2-8. 3D convolution

3D convolutional neural networks are also similar as 2D convolutional neural networks. They are the different combinations of convolutional layer, pooling layer etc. But these layers are all in 3D. Because this thesis focuses on the analysis of MRI 3D knee images, the neural networks used are 3D CNN. More examples of 3D CNN will be introduced in Chapter 4.

2.2.3 Improving neural networks

When we have a neural network and the datasets, we can start to train the neural network. It is rare that the neural network and hyperparameters we used at the beginning are the best. How to choose or adjust the neural network and how to tune the hyperparameters are definitely very important problems, even the most important one in some cases, to get a better result.

2.2.3.1 Optimization algorithms

As we discussed above, we need to choose the most accurate neural network for the thesis, and also need to find the most suitable hyperparameters for the dataset. It depends on experiences in some extents, but we still need to train various neural networks and try different hyperparameters. It is substantially an iterative process. Optimization algorithms can speed up the training and reduce the time spent on iteration. Gradient descent is one of the most popular algorithms to optimize neural networks.

Mini-batch gradient descent

In machine learning, there are three ways in terms of how much data should be used to calculate gradients to update parameters in a neural network. The first one is batch gradient descent which is to feed all the training data to the neural network to calculate the loss and gradient, and then update parameters. For this method, the computation cost is large, the speed is slow. The second one is stochastic gradient descent (SGD). It uses a randomly selected subset of the data to update parameters. The computation burden is reduced, and the iteration speed is faster. But the convergence performance may be degraded; it may result in a sharp oscillation on the

convergence curve. To overcome the drawbacks of the two methods, mini-batch gradient descent was proposed, which makes a compromise between the performance and computation burden. It divides the dataset into several batches and updates the parameters by each batch.

However, mini-batch gradient descent, does not guarantee good convergence, there are still a few challenges that need to be addressed. Choosing a proper learning rate can be difficult. In the algorithms we discussed above the same learning rate is applied to update all parameters. If the features on the dataset have extremely different frequencies, we might not want to update all of them in the same extent. Instead we hope to perform a larger update for rarely occurring features [5].

Gradient descent with Momentum

Gradient descent with momentum is an algorithm used to address these problems. It uses exponentially weighted averages (also known as exponential moving averages) of gradients to update the parameters. Exponentially weighted averages are calculated according

$$v_t=\beta v_{t-1}+(1-\beta)\theta_t \tag{2.2}$$

where v_t is the exponentially weighted average of the first t data in the dataset, v_{t-1} is the exponentially weighted average of the first t-1 data in the dataset, β is the weight, θ_t is the t-th data in the dataset.

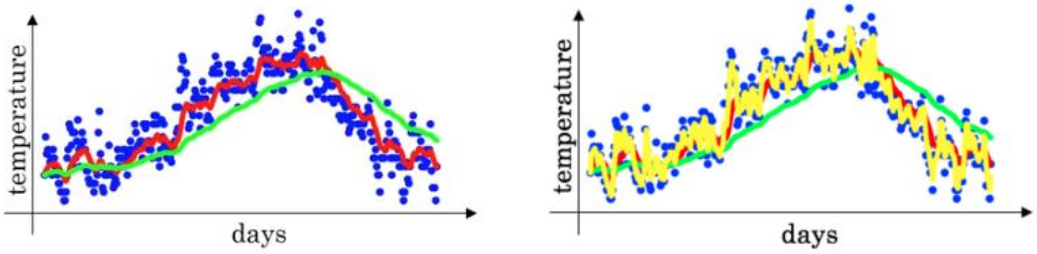


Figure 2-9. Exponentially Weighted averages

Figure 2-9 [6] shows an example of exponentially weighted averages of the temperature of a year in London. If we set β=0.9, the exponentially weighted averages are shown on the red line; If we set β=0.98, the exponentially weighted averages are

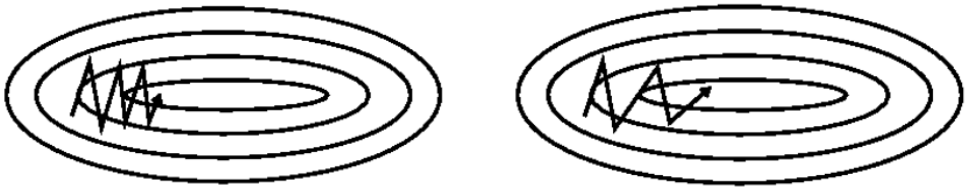
shown on the green line; If we set β=0.5, the exponentially weighted averages are shown on the yellow line. We can see that when β is larger, or close to 1, the line is smoother because it contains more historical information. And when β is smaller, the line is oscillating because less data is used to calculate the average and it will be more influenced by individual data.

Gradient descent with momentum uses exponentially weighted averages of gradients to update the parameters (equation 2.4) instead of using gradients directly (equation 2.3).

$$w:=w-\alpha*dw \tag{2.3}$$

$$w:=w-\alpha*v_{dw} \tag{2.4}$$

In these equations, dw is the differential of w (weight). α is learning rate, and v_{dw} is the exponentially weighted averages of dw. If we set the weight β in equation 2.2 close to 1, the exponentially weighted averages of gradients will become smoother according its historical records. It will accelerate gradient descent in the relevant direction and dampen oscillations, which can be seen in the figure below [5]. In the right graph of gradient descent with momentum, the oscillations on the vertical axis are smaller than on the horizontal axis. So, for gradient descent with momentum there are two hyperparameters, learning rate α and weight β to calculate exponentially weighted average, which usually is set to 0.9.



a)Gradient descent without momentum b)Gradient descent with momentum

Figure 2-10. Gradient descent with momentum

RMSProp

The algorithm of gradient descent with momentum enables users to apply different learning rates to different features, but the learning rates are changed in the same

direction. However, in some cases we want to speed up the descent in a direction and slow down the descent in another direction. Root mean square prop (RMSProp) enables users to do so. RMSProp updates weights according

$$w:=w-\alpha\frac{dw}{\sqrt{s_{dw}}} \tag{2.5}$$

where $s_{dw}:=\beta s_{dw}+(1-\beta)*dw^2$.

Adam

Adaptive Moment Estimation (Adam) [7] is a method that computes adaptive learning rates. The basic idea of it is to combine the algorithm of gradient descent with momentum with RMSprop.

$$v_{dw}=\beta_1 v_{dw}+(1-\beta_1)*dw \tag{2.6}$$

$$s_{dw}=\beta_2 s_{dw}+(1-\beta_2)*dw^2 \tag{2.7}$$

$$v_{dw}^{corrected}=\frac{v_{dw}}{1-\beta_1^t} \tag{2.8}$$

$$s_{dw}^{corrected}=\frac{s_{dw}}{1-\beta_2^t} \tag{2.9}$$

$$w:=w-\alpha\frac{v_{dw}^{corrected}}{\sqrt{s_{dw}^{corrected}}+\varepsilon} \tag{2.10}$$

The update process is shown by equation 2.6-2.10. $v_{dw}^{corrected}$ and $s_{dw}^{corrected}$ are bias correction of v_{dw} and s_{dw} . So, there are four hyperparameters here. We usually set β_1 to 0.9, β_2 to 0.999, and ε to 10^{-8}. For learning rate α, it needs to be tuned for different cases.

2.2.3.2 Hyperparameter tuning

Hyperparameters are the knobs to control a deep learning system. There are mainly two types of hyperparameters in deep learning. One is to control neural network structures and loss functions. The other one is to control training efficiency. The hyperparameters we discussed in Chapter 2.1.1 and 2.1.2 such as number of hidden

layers and number of neuros for each layer influence the neural network structure and its representation capacity. The hyperparameters we discussed in Chapter 2.1.3.1 such as learning rate affect the training efficiency. We will discuss the most frequently used hyperparameters and how to adjust them to improve the performance of neural networks here.

Learning rate

Despite the advanced optimization algorithms which were introduced above, we still need to set the learning rate for training. The learning rate controls the step of gradient descent. If we choose a learning rate which is too small, it may result in a long training process that could get stuck in local minimum. While if we choose a learning rate which is too big, it may result in a sub-optimal result or an unstable training process.

The learning rate should be larger at the beginning, and then be reduced in steps. It is recommended to use a learning rate schedule rather than setting a fixed learning rate when training a neural network. For example, there is a learning rate schedule called *ReduceLROnPlateau* provided by *Keras*. It will reduce the learning rate when a plateau in model performance is detected, e.g. no change for a given number of training epochs.

Batch-size

As we discussed above, mini-batch gradient descent is the most popular way to choose how much data should be used in gradient descent. In this way, we need to set the batch size which determines how large a batch should be. Because the computing resource, e.g. GPU memory, is limited, there is a maximum value of batch size for a certain neural network and dataset. To maximize the utilization of memory, we should use as large batch size as possible. However, according to Nitish et al. [8] neural networks converge to sharp minimum with a large batch size, while converge to flat minimum with small batch size as shown in Figure 2-11. The second one has better generalization ability.

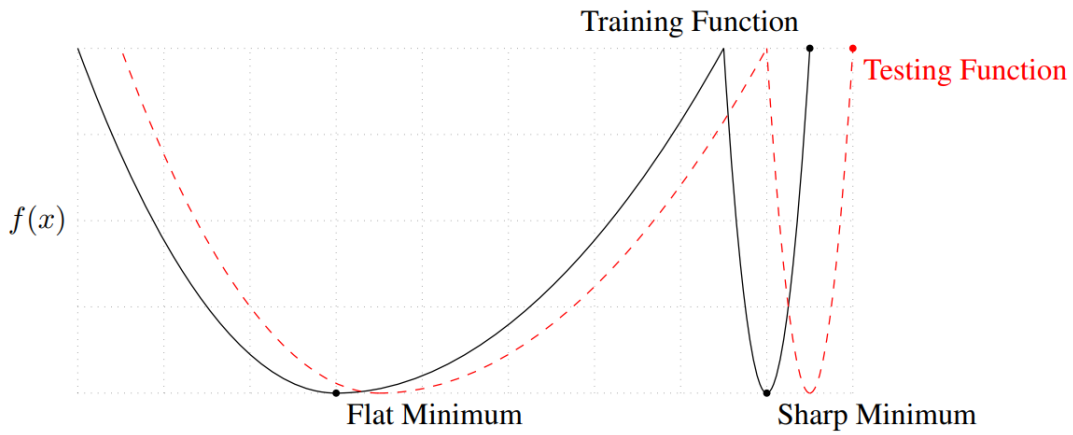


Figure 2-11. Large batch size vs. small batch size

Epochs

An epoch is defined as the time when all training data has been fed to the neural network to update the trainable parameters. For batch gradient descent, an epoch takes an iteration. For SGD and mini-batch gradient descent, an epoch takes a number of iterations, which depends on the size of batch and dataset. It takes at least several epochs to achieve the best result. So we should set the maximum number of epochs to finish the training. Actually, there are other methods to finish the training. One of them will be introduced in the next sub-chapter.

Activation function

As we discussed above, activation function can introduce non-linear features in the neural network. Choosing the appropriate activation functions is also very important to improve the performance of the neural network. The most popular types of activation functions are ReLu (Rectified Linear Unit), Sigmoid, and Tanh.

The expression of ReLu is R(x)=max (0,x), which is very simple, so the computation cost is low. It is introduced to avoid and rectify vanishing gradient problem which can be caused by sigmoid and tanh. It is commonly used in the hidden layers. The output range of ReLu is from 0 to infinity. For the tasks which have different output range, such as classification problems whose output should be from 0 to 1, ReLu can't be used in the output layer. Leaky ReLu is another activation function proposed based on

ReLu in order to address the problem of dead neuros which may be caused by ReLu because the gradient is always zero when x<0.

For image classification, sigmoid can be used in the output layer because its output range is from 0 to 1. Its expression is $S(x)=\frac{1}{1+e^{-x}}$. It is used for binary classification. Another similar activation function is Softmax, which can be used for multiple classes.

Tanh is expressed by $T(x)=\frac{1-e^{-2x}}{1+e^{-2x}}$, and its output range is from -1 to 1. The curves of sigmoid and tanh are shown on figure blew [9]. Compared with sigmoid, tanh is zero-centered.

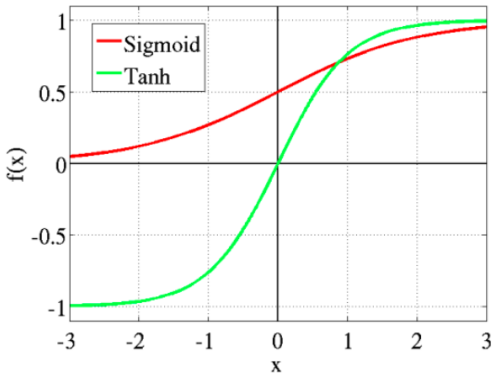


Figure 2-12. Sigmoid and tanh

2.2.3.3 Regularization

Regularization aims to improve the generalization ability of neural networks in order to avoid overfitting, which always happens when the dataset used for training doesn't represent the distribution of the data in the real world. Several techniques which are commonly used for regularization will be introduced.

Data augmentation

Data augmentation enlarges training datasets by adding transitions or perturbations to the existing data to generate new data. In computer vision, the simplest techniques of data augmentation are translation, flipping, clipping, scaling, rotation, and adding Gaussian noise. However, the effect is limited because the new data are generated

based on the existing data, which means totally new features won't be introduced since it doesn't exist in the original dataset.

There are also other methods of data augmentation. For example, conditional GANs (generative adversarial networks) can transform an image from one domain to another domain, but it is computationally intensive [10]. The graph below [11] shows how conditional GANs change images.

Figure 2-13. Using conditional GANs for data augmentation

L1 and L2 regularization

L1 and L2 regularization adds penalties on weights in loss functions in order to reduce the absolute sum of the parameters. The expressions are shown below.

$$\text{Loss}=\text{Error}(y,\hat{y}) \tag{2.11}$$

$$\text{Loss}=\text{Error}(y,\hat{y})+\lambda\sum_{i=1}^{N}|w_i| \tag{2.12}$$

$$\text{Loss}=\text{Error}(y,\hat{y})+\lambda\sum_{i=1}^{N}w_i^2 \tag{2.13}$$

L1 regularization has a sparse solution, which means there are many zeros in the parameters, so it actually does feature selection. L2 regularization's solution is non-sparse. Researchers prefer L2 regularization in their projects.

Dropout

Dropout reduces interdependent learnings between the neurons. It was proposed by Nitish et al [12]. In training phase, the algorithm ignores a random fraction of neuros and corresponding activations for each hidden layer, and each training sample, in each iteration, which can be seen in the example of Figure 2-14. In this way, it forces the neural network to learn more robust features that are useful in many different random subsets of neurons.

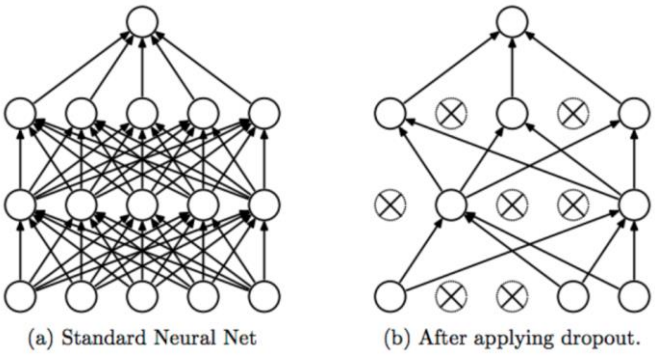


Figure 2-14. Dropout

According to Amar [13], Dropout roughly doubles the number of iterations required to converge. However, training time for each epoch is less than without it.

Early stopping

A method when to stop the training we discussed above is setting the number of epochs of training. It is extremely inefficient and unnecessary. The training can be stopped whenever we want manually. But we need to stop it at the best time and stop it automatically. Too little training will make the model underfitting, while too much training may cause the model overfitting. Early stopping provides a method to stop the training at the point when the performance on validation datasets starts to degrade. The basic procedure to implement early stopping will be described below.

- Split the training data into training set and validation set;
- Train only on training set and evaluate on the validation set;

- Stop training as soon as the loss on the validation set is higher than last time it was evaluated;

- Use parameters in the previous step as the result of the training.

However, the early stopping point on the validation set is not always the point that the training begins to perform overfitting. When we choose to stop the training, the optimization is stopped as well, but it is also possible that we have not found the minimum point yet. We can choose other conditions to trigger the stopping such as using the similar idea in *ReduceLROnPlateau* (a learning rate schedule we discussed above).

2.2.3.4 Normalization

The best dataset for training in Deep Learning should be independent and identically distributed. One of important reasons to make training a deep neural network so difficult is that it involves the superposition of many layers. And the parameters updating in each layer will lead to changes of distribution of input data in the last connected layer in backpropagation. These changes will become larger with depth, which needs the connected layers to constantly adapt to this change. In order to train the model, we need to set the learning rate, initialize the weights, and update the strategy as carefully as possible. This phenomenon was summarized as Internal Covariate Shift (ICS).

Whitening is an important data preprocessing step before feeding data to the neural network. It generally has two purposes. The first one is to remove correlation between features in order to get independence. The other is to make all features have the same mean and variance, which means to make sure the data in the same distribution. Principal Components Analysis (PCA) is one of the most typical methods of whitening.

To make the data in an independent distribution, theoretically we need to whiten the data of each layer. However, computational cost of standard whitening operations such as PCA is high. In addition, we also want whitening operations to be differentiable to make sure that whitening operations can update gradients through

backpropagation. Batch Normalization (BN) [14] is one of the normalization methods which are proposed as a simplified whitening operation to address the problem of ICS.

However, a recent study [15] shows such distributional stability of layer inputs has little to do with the success of Batch Normalization. The real reason is that it makes the optimization landscape significantly smoother. This smoothness induces a more predictive and stable behavior of the gradients, which leads to faster training.

Whatever the real reason is, batch normalization is an efficient method to speed up the training. Batch normalization is to apply normalization in a mini-batch of data during training. The general procedure of such normalization has two steps. The first step is to apply translation and scaling according

$$x_{norm}^{(i)} = \frac{x^{(i)} - \mu}{\sqrt{\sigma^2 + \varepsilon}} \tag{2.14}$$

where $x^{(i)}$ is the i-th input in the input tensor, μ is the mean, σ^2 is the variance, ε is to avoid the denominator to be zero. Then the inputs are in a standard normal distribution with mean 0 and variance 1. In order to ensure that the representation capacity of the neural network does not decline because of normalization, another transformation is applied according

$$\tilde{x}^{(i)} = \alpha * x_{norm}^{(i)} + \beta \tag{2.15}$$

where α and β are learnable parameters same as other learnable parameters such as weights in the neural network.

There are other types of normalization such as layer normalization which is applied data in a layer. It calculates the mean and variance of the input data of each layer, and uses the same procedure as batch normalization.

2.3 MRI image

A medical image is the representation of the internal structure or function of an anatomic region in the form of an array of picture elements called pixels or voxels. It is a discrete representation resulting from a sampling or reconstruction process that

maps numerical values to positions of the space. Medical image file formats can be divided in two categories. The first is formats intended to standardize the images generated by diagnostic modalities, e.g., *Dicom*. The second is formats born with the aim to facilitate and strengthen postprocessing analysis, e.g., *Nifti*, which is a file format created at the beginning of 2000s by a committee based at the National Institutes of Health with the intent to create a format for neuroimaging maintaining the advantages of the analyze format, but solving the weaknesses [16].

The dataset used in this thesis is in the *Nifti* format. The main feature of this format is that it contains affine coordinates which can associate the index (i, j, k) of each voxel with its spatial position (x, y, z). The Python library which can read *Nifti* files is *nibble*.

Nuclear magnetic resonance imaging is also called Magnetic Resonance Imaging (MRI), which is an imaging technique that reconstructs images by collecting signals generated by magnetic resonance phenomena.

A particular setting such as pulse sequences and pulsed field gradients will result in a particular image appearance in MRI. The particular setting is called MRI sequence. The MRI sequences used on our dataset are T1, PD and FS, which can be seen in the Figure 2-15. The sampling of human (knee) data was approved by the Regional Committee for Medical and Health Research Ethics (REK nr. 61225).

As mentioned in Chapter 1.2, we will not discuss more details about knee joints or MRI technique in medical domain unless it is necessary for explaining the deep learning techniques used. So, more details about T1, PD and FS will not be discussed.

(a) T1 (b) PD (c) FS

Figure 2-15. T1, PD and FS MRI images

3. Related work

Semantic segmentation is one of the tasks in computer vision. To optimize the segmentation neural networks, we can also learn from other tasks in computer vision such as image classification and object detection. This chapter will introduce five tasks including image classification, object detection, object tracking, semantic segmentation, and instance segmentation in computer vison. We will discuss more details about image classification because it is the base of other tasks. And then we will focus on semantic segmentation which is the category of this thesis. We will discuss the commonly used neural networks and loss functions in this area.

3.1 Computer vison

Computer Vision (CV) is the most popular research area in deep learning. It has developed a series of techniques which enable computers to understand and analyze the content of digital images such as photographs and videos. In Computer vision, there are mainly five different tasks including image classification, object detection, object tracking, semantic segmentation, and instance segmentation. The details of these tasks will be introduced below.

3.1.1 Image classification

Image classification is aimed to classify an image into a specific category defined by the task. It is the most well-known and simplest computer vision task. Other computer vision tasks such as object detection, semantic segmentation are based on image classification problem. Figure 3-1 is an example of image classification using AlexNet [17]. When an image is input into the neural network, it will output the class of this image (in this case, the image is classified as cat).

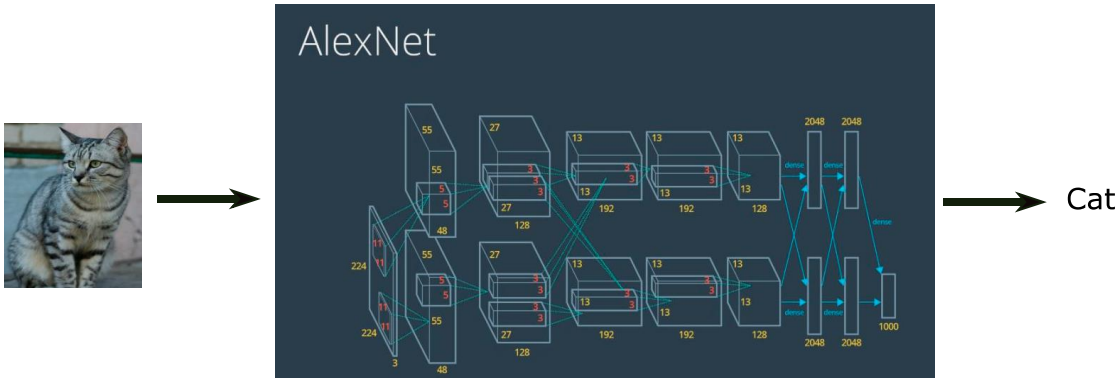


Figure 3-1. An example of image classification

The neural network extracts features of different levels from the input images through different combinations of operations such as convolution and pooling. More details of these operations can be found in Chapter 2.2.2. Using these features extracted from different levels the neural network can recognize the inner patterns of how to classify the images.

Table 3-1 shows a rough development history of neural networks in deep learning. Most of them are responsible for setting the new state of the art for classification and detection in the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC), which is the most known competition in computer vision and evaluates algorithms for object detection and image classification at large scale. Some of them are still widely used as backbone neural networks on various kinds of tasks.

Table 3-1. A rough development history of neural networks

Date	Neural network	Description
1994	LeNet5	It was one of the earliest convolutional neural networks.
2012	AlexNet	It was a wider and deeper version of LeNet5
2013	OverFeat	It was derived from AlexNet and proposed a new technique of learning bounding box.
2014	VGG	It is the first network to use smaller filters (3×3 filter) in each convolutional layer.
2013	Network-in-network	It proposed 1×1 convolution which can provide more combinatorial features.
2014	GoogleNet	It was the first architecture of Inception which chooses filters adaptively.
2016	ResNet	It proposed residual structure.

LeNet5 [18] was proposed in 1994. It was one of the earliest convolutional neural networks. It adopted the combination of convolution, pooling, and nonlinearity activation function as a sequence to build a neural network, which became the most commonly used sequence for neural networks in computer vision. It was trained on CPU because of the underdevelopment of hardware at that moment.

After LeNet5, there was a long time that the neural network was in the incubation stage. Its capacity was unnoticed, while few improvements happened. With the development of mobile cameras and cheap digital cameras, more and more data became available. And with the growing computing power, CPU became faster and as GPU became widely used, deep learning especially computer vision has been in the high speed period of development since the last decade.

In 2012, AlexNet [17] was proposed. It was a wider and deeper version of LeNet5. It employed the techniques of ReLu and dropout. It was trained on a GPU and the speed of training increased dramatically. OverFeat [19] was derived from AlexNet and proposed a new technique of learning bounding box, which is widely used in object detection.

VGG [20] was the first network to use consecutive smaller filters (3×3) to replace larger filters (5×5, 9×9 and 11×11), which was different from the principle of LeNet5 and its derivatives, where large convolutions were used to obtain similar features in an image. Since then smaller filters has become popular.

The idea of network-in-network [21] was simple but very useful. Use 1×1 convolution to provide more combinational features of convolutional layers. This technique is widely used in neural network such as GoogleNet [22], which was the first architecture of Inception that chooses filters (1×1 or 3×3 or 5×5 or pooling) adaptively, and ResNet [23], which proposed residual structure that will be introduced in Chapter 4.1.1.1.

3.1.2 Object detection

Object detection is a task of identifying the objects through a picture or video. Figure 3-2 [24] is an example of object detection. We can see that it provides not only the classes but also indicate the spatial location of those classes.

Figure 3-2. An example of object detection

One of representational neural networks for object detection is the series of R-CNN, including R-CNN [25], Fast R-CNN [26], Faster R-CNN [27], and Mask R-CNN [28]. They are region based algorithm. Figure 3-3 [25] shows the implementation stages of R-CNN.

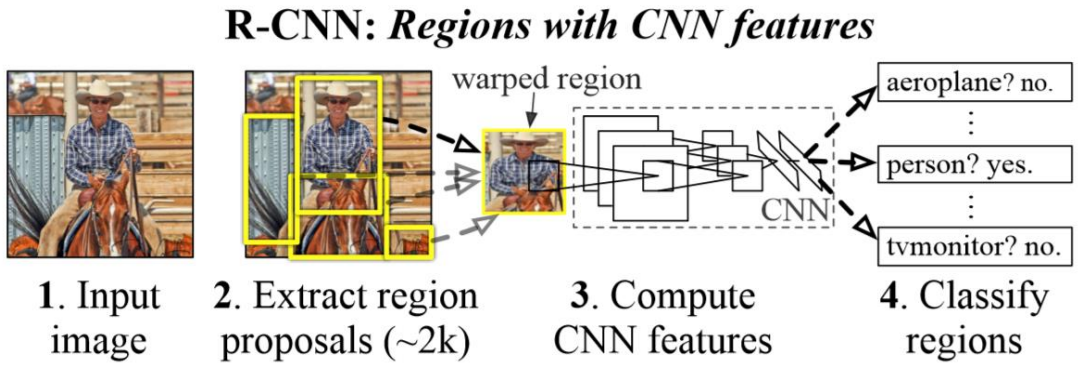


Figure 3-3. The implementation stages of R-CNN

There are four stages to implement R-CNN:

- Region proposals extraction: to generates 1000~2000 region proposals (using the Selective Search algorithm);
- Feature extraction: for each region proposal, using convolutional neural networks (CNN) to extract features;

- Classification: These features are fed into SVM classifiers to determine whether it belongs to this class;

- Bounding box Regression: Use regression to fine-tune the box position, which is not included in the figure above.

Fast-RCNN [26] shared computation in the steps of feature extraction, classification, and bounding box regression, and all of them are implemented using CNN. Because the CNN was implemented on the whole image once rather than on each region proposal thousands of times, it was faster than R-CNN. But it also brought a problem that the number of features for each region proposal is different since they are extracted from the whole image. To solve this problem, Fast-RCNN added ROI layer. After passing through this layer, the number of features will be the same.

Faster-RCNN [27] introduced RPN networks (region proposal network) to replace the Selective Search algorithm, which made possible for the whole network to be trained end to end. Mask-RCNN [28] extended R-CNN to the area of semantic segmentation.

Another important algorithm in object detection is YOLO (You Only Look Once) [29]. Different from R-CNNs, it used a single CNN to predict the bounding boxes and the class probabilities for these boxes. It was efficient in term of speed but did not perform very well on small object detection.

3.1.3 Object tracking

Object tracking aims to detect moving objects, which has many practical applications including surveillance, traffic flow analysis, and self-driving cars etc. Object tracking starts with object detection, but there are more challenges compared to static object detection. Figure 3-4 [30] shows an example of object tracking.

Figure 3-4. Object tracking

3.1.4 Semantic segmentation

Semantic segmentation refers to the task of annotating each pixel in 2-dimension images or each voxel in 3-dimension images to a class label. It is essentially a classification problem. The figure below [31] gives an example of semantic segmentation, which shows its connection with image classification. It is the category of the thesis. We will discuss more details about it in Chapter 3.2.

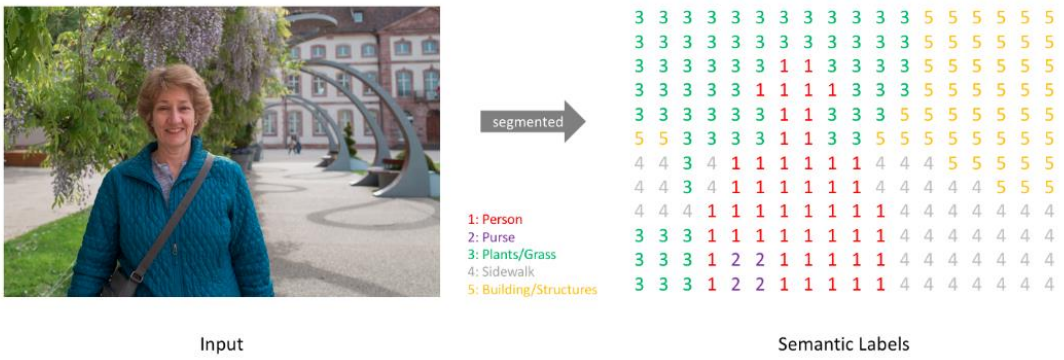
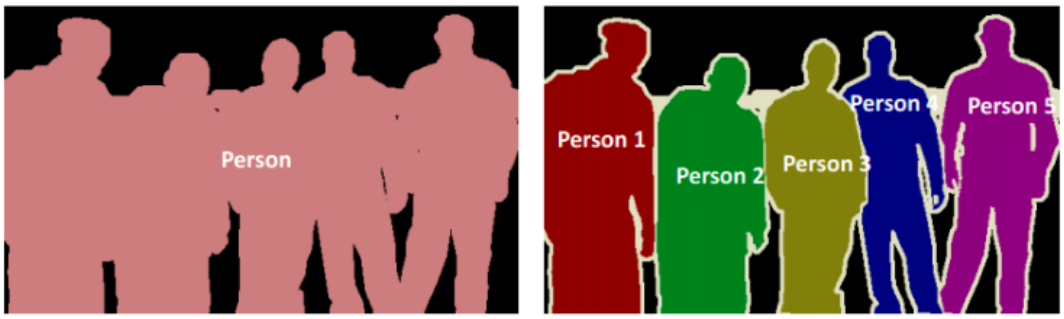


Figure 3-5. An example of Semantic segmentation.

3.1.5 Instance segmentation

Instance segmentation is an extensional task of semantic segmentation. Unlike semantic segmentation that only needs to classify different categories, instance segmentation needs to distinguish different instances even if they are in the same class. Figure 3-6 [32] shows their difference.



a)semantic segmentation b) instance segmentation

Figure 3-6. Instance segmentation and semantic segmentation

3.2 Semantic segmentation

As we discussed above, semantic segmentation is essentially a classification problem on pixel or voxel level. The neural networks used for semantic segmentation and the loss functions used for training these neural networks will be introduced in this sub-chapter. They are the necessary previous knowledge of Chapter 4.

3.2.1 Neural networks

The goal of semantic segmentation is to label each pixel of an image with a corresponding class. Classical convolutional neural networks such as AlexNet, VGG, and GoogleNet perform well on image classification. It is natural to use them for semantic segmentation. However, there are reductions of image resolutions caused by the repeated combinations of convolutions and max-pooling etc. used by these neural networks for increasing the receptive field. The output of semantic segmentation is an image but not a single class. The details lost in this process cannot be used on generating the segmentation image, which would lead to the loss of accuracy.

3.2.1.1 FCN

One of the most important breakthroughs, maybe the most important one, in recent years is fully convolutional neural network (FCN) [33], which rewrites the fully connected layers as convolutions to produce a whole image rather than an output for a single pixel. The figure below [33] shows this process. The upper network is for classification, whose output is a series of probabilities. The highest one is tabby cat which is the input image's category. Applying convolutionalization to the final layer, the nether network's output becomes an image as shown in the nether network.

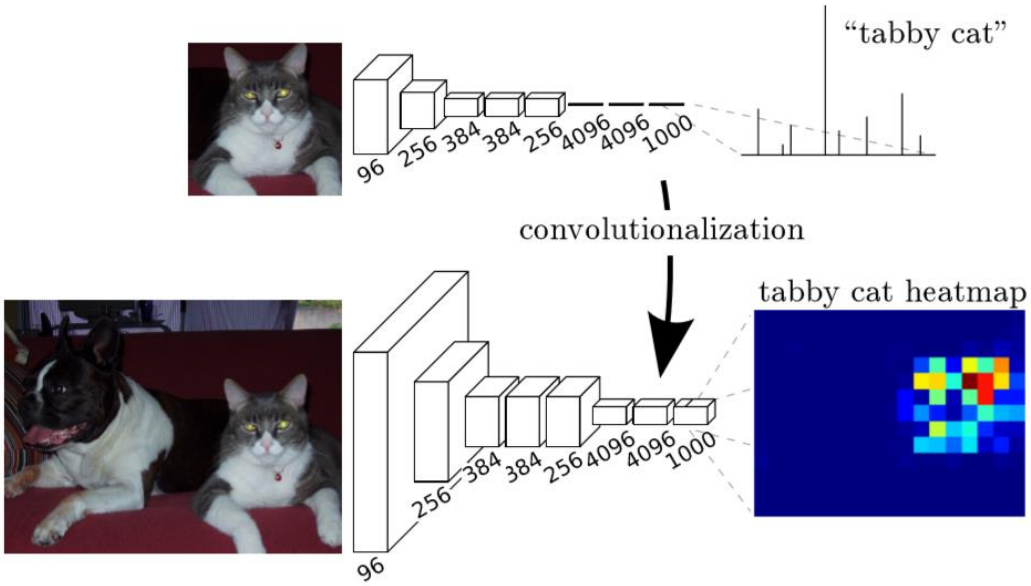


Figure 3-7. Fully Convolutional Network

There are three mainly techniques in FCN. The first one is convolutional, which we discussed above. It uses convolution layers to replace the fully connected layers in classification neural networks such as VGG and ResNet. The second one is using deconvolution to implement upsampling to reconstruct the output image which has the same size as the input image. Deconvolution (or transposed convolution) can be understood as the reverse process of convolution.

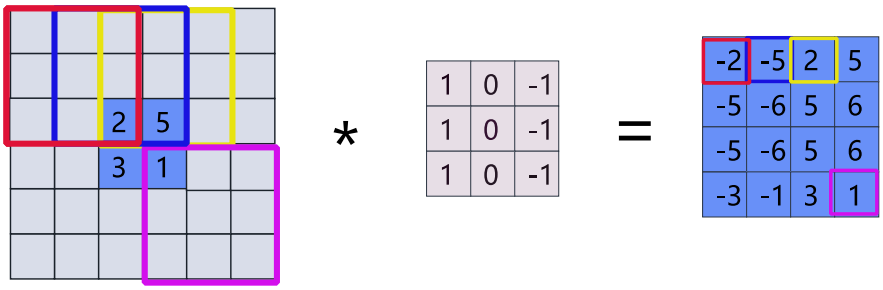


Figure 3-8. Deconvolution

The process of calculating deconvolution is shown in Figure 3-8. Input images are used as the center and full zero padding is implemented to get the different sizes of outputs. In this example, it uses 3×3 deconvolution filter slipping on the input image whose size is 2×2 to get the output image whose size is 4×4.

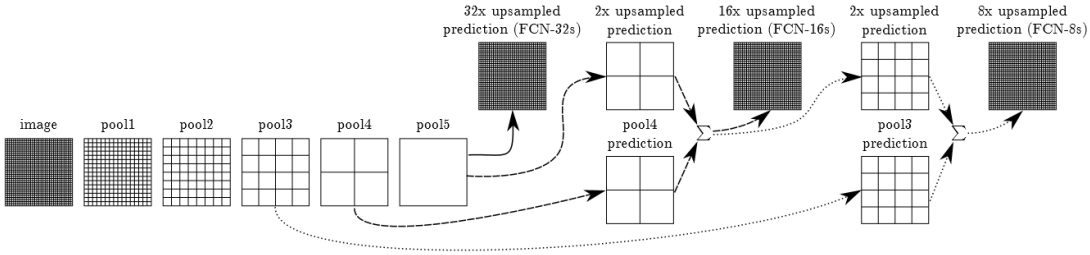


Figure 3-9. Fusing information from layers with different strides

The last notable technique used in the FCN is skip connection, which fuses information from layers with different strides to improve segmentation details. The paper compared three methods of fusing information, which are shown in Figure 3-9 [33]. For FCN-32s, it uses 32 times upsampled pool5 as the prediction; for FCN-16s, it fuses 2 times upsampled pool5 with pool4, and then implements 16 times upsampling as the prediction; for FCN-8s, it fuses 2 times upsampled pool5 with pool4 first, and then upsamples the results, and then fuses it with pool3, and then

implements 8 times upsampling as the prediction. The results are shown in Figure 3-10 [33]. We can see that FCN-8s has the best result, which means that decreasing the stride of pooling layers is the most straightforward way to obtain finer predictions. This method of improving the segmentation accuracy by fusing features from different strides are widely used in other segmentation neural networks such as U-net [34] and SegNet [35].

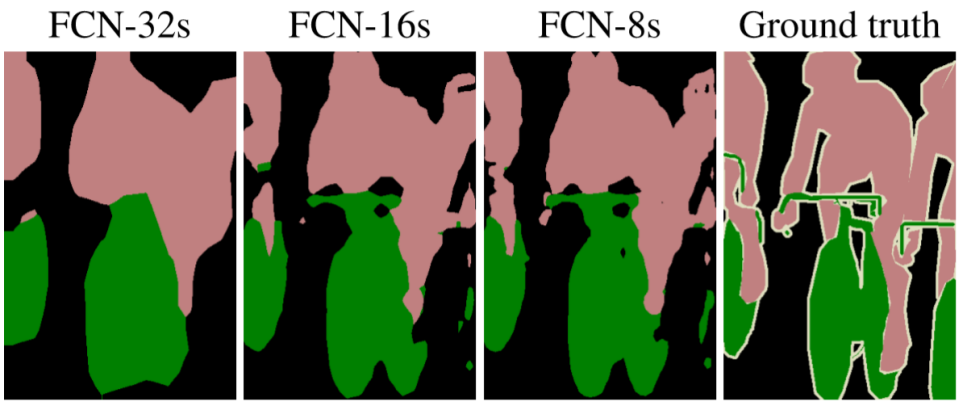


Figure 3-10. Results of refining FCN

3.2.1.2 SegNet

SegNet [35] used elegant encoder-decoder architecture, which is shown below. It used convolution and pooling in the encoder, decovolution and upsampling in the decoder, and Softmax for pixel classification. The image segmentation accuracy was improved by using pooling indices which recorded the position of pooling. This encoder-decoder architecture is one of the most popular architectures for semantic segmentation neural networks.

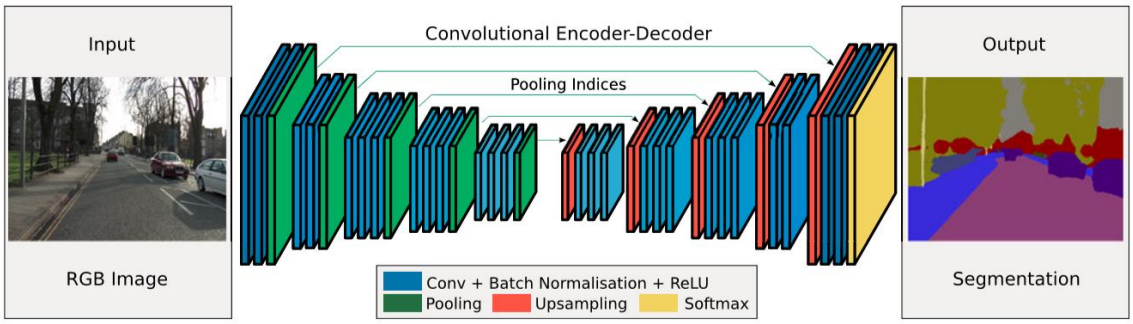


Figure 3-11. SegNet

3.2.1.3 DeepLab

The techniques developed by DeepLab series are another important branch in semantic segmentation. They have used classical convolutional neural networks as backbones and developed other advanced components added in the systems to improve the performance.

DeepLabv1 [36] used VGG as the backbone and introduced atrous convolution and fully-connected Conditional Random Fields (CRF) to solve the problems brought by reduced resolution. DeepLabv2 [37] used Residual-Net and introduced ASPP (Atrous Spatial Pyramid Pooling) which used multiple parallel atrous convolution layers with different sampling rate. DeepLabv3 [38] discussed four types of Fully Convolutional Networks and improved ASPP.

The lasted version is DeepLabv3plus [39], whose architecture is shown in the figure below. It was an encoder-decoder architecture. It used Xception as the basic network. The encoder was comprised of improved Xception and ASPP. The decoder used bilinear upsampling concatenated with the corresponding low-level features from the network backbone. More details will be introduced in Chapter 4.1.2. It is one of the architectures used in this thesis. However, the specific structures such as the basic network have been modified according the feature of the dataset and the requirement of the task, just like the other two architectures in this thesis.

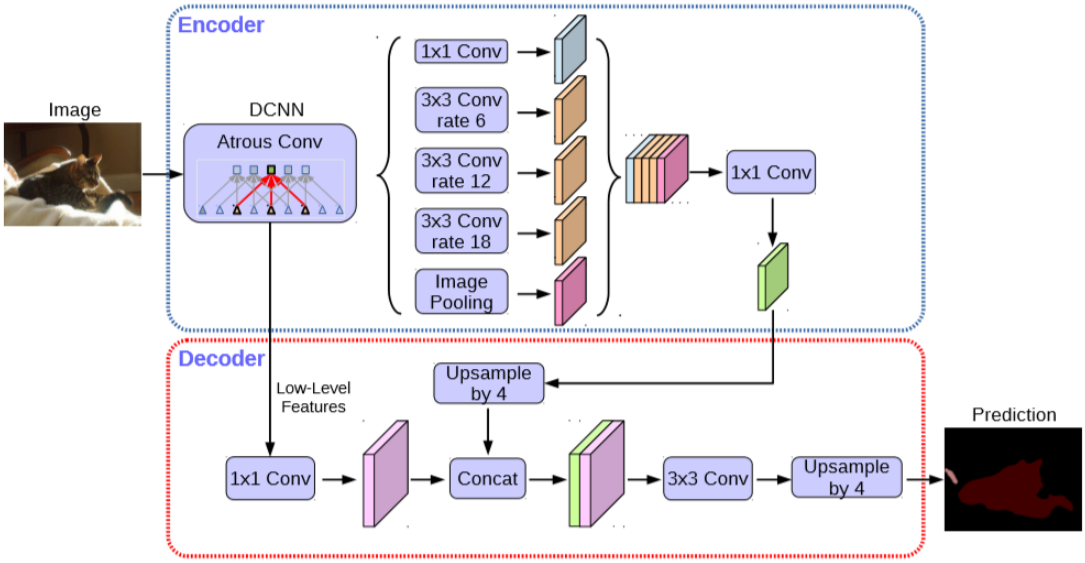


Figure 3-12. DeepLabV3Plus

3.2.1.4 Unet

These neural networks we discussed above perform great on natural image segmentations, but they cannot be easily transformed to be used in medical image segmentations. Medical images segmentation is different from natural images segmentation. First, the training datasets are small because it is laborious to be annotated and there are some regulations making it difficult to access. Second, the features are not that rich compared with natural images. So, too complicated neural networks shouldn't be used. Third, medical images such as MRI or CT are 3D. Expanding 2D neural networks to 3D will increase their complexities and the amount of data needed to be processed will also rise dramatically. It will be consequently that we need higher computational power to train them. The limited computational power has already been a problem on 3D image segmentation even for relatively simple networks. Because of it, most of 3D convolutional neural networks are trained on patch-wise. This is also the method adopted in this work.

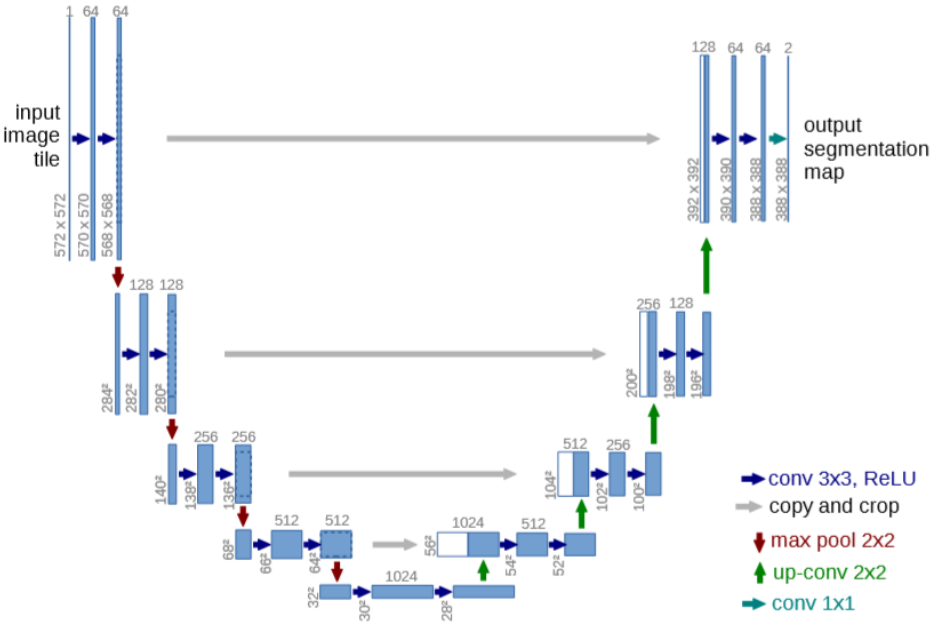


Figure 3-13. U-net

Although these differences hinder neural networks in natural image segmentation to be directly used in medical image segmentation, the advanced techniques developed by natural image segmentation have still been used in various kinds of medical image segmentation neural networks. U-net [34] was inspired by FCN. It is the most

successful neural network used in medical image segmentation. Its architecture is shown in Figure 3-13 [34]. It used skip-connections to merge the features in different levels and symmetrical encoder-decoder architecture same as SegNet, achieving great performances on small datasets.

3.2.1.5 Other networks

As we discussed above, many semantic segmentation neural network in medical domain take advantage of advanced techniques developed in natural image segmentation. According to P. Wang et al. [40], fully convolutional network (FCN), conditional random fields (CRF), and atrous convolution are key components used in the most state-of-the-art semantic segmentation systems. These components have been used in medical image segmentation neural networks too. For example, U-net used the idea of FCN; DeepMedic [41] employed a fully connected CRF as a post-processing step to achieve more structured predictions; FocusNet [42] utilized atrous convolution.

3D U-net [43] is one of the most successful convolutional neural networks used for 3D medical image segmentation. There are other neural networks that used it as the backbone network and added some advanced components to improve the performance. For example, V-net [44] added residual blocks based on it. Dense V-Net [45] used V-net as the backbone, and added dense feature stacks inspired from DenseNet [46].

3.2.2 Loss functions

Loss function is another important factor determining the final performance of the deep learning system. One of the most important problems that is needed to be solved in in medical image segmentation is the imbalanced samples. For example, abnormal organs or lesions usually account for a small portion of the whole medical image. Appropriate loss function needs to be chosen to balance the positive sample and the background.

3.2.2.1 Cross-entropy

For binary classification, cross-entropy loss function is defined by

$$CE=-\frac{1}{N}\sum_{i=1}^{N}(y_i \log p_i+(1-y_i)\log(1-p_i)) \quad (3.1)$$

where y_i is the truth category of input sample x_i , p_i is the predicted possibility of input sample x_i belonging to category 1. This loss function pays equal attention on each category. So it cannot work perfectly on imbalanced classes. For medical image segmentation, the percentage of abnormal organ which needs to be segmented is usually very small. The worst case is the neural network segments nothing if we use cross-entropy loss.

3.2.2.2 Weighted cross-entropy

Weighted cross-entropy can be used to address the imbalance we discussed above. It can be calculated according

$$WCE=-\frac{1}{N}\sum_{i=1}^{N}(w*y_i \log p_i+(1-y_i)\log(1-p_i)) \quad (3.2)$$

where w is the weight which needs to be defined before training.

3.2.2.3 Focal loss

Focal loss was proposed in the task of object detection, in order to solve the problem of imbalance of positive and negative sample ratio. The formula is

$$F=-\frac{1}{N}\sum_{i=1}^{N}(\alpha*y_i*(1-p_i)^{\gamma}*\log p_i+(1-\alpha)(1-y_i)*p_i^{\gamma}*\log(1-p_i)) \quad (3.3)$$

Compared with the formula of cross-entropy, there are two added multipliers $(1-p_i)^{\gamma}$ and p_i^{γ}, which aims to increase the loss value when to predict the class whose p_i is small and decrease it when to predict the class whose p_i is big. So it can increase the attention on positive samples whose percentage in the dataset is small.

3.2.2.4 Dice loss

Dice loss was introduced by V-net [44], and developed based on dice coefficient in order to address the problem of the learning process getting trapped in local minimum

when the predictions are strongly biased towards background. The dice coefficient between two binary volumes is calculated according

$$D(A,B)=\frac{2\times(|A\cap B|)}{|A|+|B|} \tag{3.4}$$

It can also be represented by

$$Dice=\frac{2TP}{2TP+FN+FP} \tag{3.5}$$

where TP is true positive, FP is false positive, FN is false negative. Dice loss can be defined as 1-Dice or -Dice. It performs better on small object segmentation compared with cross-entropy. However, it also has disadvantages. If there are only foreground and background in the dataset, prediction errors of small targets will lead to large changes of dice loss, which will cause sharp gradients and unstable training.

3.2.2.5 Generalized Dice loss

For multi-label segmentation, there is generally one dice coefficient for each class. Generalized Dice loss integrates the dice coefficients of multiple labels together to measure the segmentation results. The formula of generalized dice loss for two labels is

$$GDL=1-2\frac{\sum_{l=1}^{2} w_l \sum_n r_{ln}p_{ln}}{\sum_{l=1}^{2} w_l \sum_n r_{ln}+p_{ln}} \tag{3.6}$$

where r_{ln} is the value of label l in the ground truth, and p_{ln} is its predicted value, w_l is the weight of label l , and is calculated according

$$w_l=\frac{1}{(\sum_{n=1}^{N} r_{ln})^2} \tag{3.7}$$

3.2.2.6 IOU loss

Similar as dice coefficient, IOU (intersection-over-union) loss is also use a metric as a loss function. IOU is actually defined as

$$IOU=\frac{TP}{TP+FN+FP} \tag{3.8}$$

IOU loss as a loss function is defined as

$$IOU=\frac{I(X)}{U(X)}=\frac{X*Y}{X+Y} \tag{3.9}$$

where X is the prediction and Y is the ground truth.

3.2.2.7 Combined loss

To take advantage of multiple loss functions described above or others, we can combine some of them to a new loss function. For example, we can add cross-entropy loss with Dice loss as a new loss function:

$$Loss=CE+(-Dice) \tag{3.10}$$

4. Method

In this chapter, three architectures used in this work including 3D U-net variants, DeepLab variants and a type of combined neural network will be introduced. To explain the 3D U-net architecture, the components used in it will be introduced first. These components including residual blocks, residual SE blocks, dense blocks and dense SE blocks. Four variants of U-net have been developed by using these components. The 3D U-net variants were also used as the backbone network in DeepLab variants and the combined neural network.

These architectures were not developed from scratch. They used the existing architectures as references, and were modified according the features of our dataset and the task. To explain the details how to implement these architectures, the related neural networks will be introduced first, and then the details of how to modify them to the neural network for our dataset will be described. The reason why the introduction of related neural network is presented in this chapter rather than the previous chapter is to make the descriptions of the modified neural network easy to understand. The loss functions and metrics used to train them will also be introduced in the end of this chapter.

4.1 Neural Networks

Besides the problems such as small training datasets and large memory requirement many medical image segmentation tasks have to face, there is one more problem in this work. It involves multiple imbalance classes, which becomes more intricate because multi-label segmentation needs to segment both the small organs and the large ones, which may affect each other in some extent. There are two directions we can consider to improve the accuracy for such a problem: employing innovative neural networks, and using appropriate loss functions.

It is obvious that a better neural network can bring more benefits, so more attention has been paid on improving the neural network. As we discussed above, 3D U-net is one of the most successful neural networks used for medical image segmentation. In MICCAI Kidney-tumor Segmentation Challenge in 2019, the neural networks the first prize winner [47] adopted were based on U-net. It adjusted the networks according their training image features. Another element helping them achieve best score was their efforts at data preprocessing. This is one of the evidences that U-net is the most successful segmentation architecture in medical domain. Based on it, this work initially chooses 3D U-net as the backbone network. Three advanced components including residual block, squeeze-and-excitation (SE) blocks [48], dense blocks are added to form four variants of 3D U-net in order to increase the segmentation accuracy.

DeepLab series [36][37][38][39] also have notable influence on medical image segmentation, especially ASPP (Atrous Spatial Pyramid Pooling. Unlike U-net, which uses symmetrical encoder-decoder architecture, the latest version of DeepLab (DeepLabv3plus) used symmetrical encoder-decoder architecture. This architecture has been also adopted in this work, and it was modified according the feature of our dataset and the requirement of the task.

There are also some researchers attempt to use combined neural networks. Payer C et al. [49] proposed a pipeline of two FCNs, one to localize the center of the bounding box, the other focusing on this region to do segmentation. FocusNet [42] used the similar framework as [49]. AnatomyNet [50] used attention networks which are widely used in natural language processing. A type of neural networks which

combines edge detection neural network with segmentation neural network were tried to improve the accuracy in this thesis as well.

4.1.1 3D U-net variants

In order to address the problem brought by imbalanced multiple labels in the dataset, four variants of 3D U-net have been employed including 3D U-net with residual blocks, with residual blocks adding SE blocks (residual SE blocks), with dense blocks , and with dense blocks adding SE blocks (dense SE blocks).

3D U-net with residual blocks has similar architecture with V-net. Residual blocks enable the networks to improve accuracy from considerably increased depth. SE blocks can adaptively strengthen important features and can be directly applied in both residual networks and non-residual networks with only light weight computational increase. Dense blocks improve the performance through feature reuse by concatenating feature maps learned by different layers instead of using extremely deep or wide architectures.

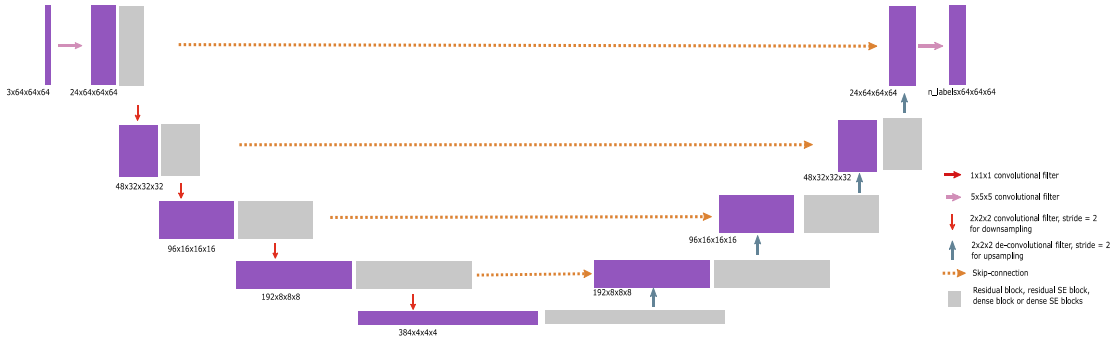


Figure 4-1. The network uses 3D U-net as the backbone

Figure 4-1 shows the architecture of the backbone network, which is a typical encoder-decoder architecture. The downsampling path on the left (encoder) extracts features using convolutions from the input images, and the upsampling path on the right (decoder) uses deconvolutions to reconstruct the details for the final segmentation results. The skip-connections are used to fuse features of different levels obtained in the downsampling path with the features in the upsampling path in order to improve the segmentation accuracy. For the grey blocks in both paths, they have

been filled by using four types of blocks, including residual blocks, residual SE blocks, dense blocks, and dense SE blocks. The details will be explained later.

The numbers of filters used in convolution blocks of all layers are designed to be the multiple of three, because the number of input channels of the datasets is 3 (the images have three weighted volumes, T1, PD and FS for each object). The downsampling is designed to be ended when the size of feature maps are reduced to 4×4×4 if the size of input image is 64×64×64. If more downsampling layers are used, the feature maps in the last layer of encoder will contain too less information. Different sizes of input images or patches were used in the experiments. The figure above only shows one of them (64×64×64). Other descriptions about the size of feature maps in the neural network in Chapter 4.1.1 are based on this figure.

4.1.1.1 Residual Block

Deep residual learning framework [23] was proposed to address the degradation problem when networks become extremely deep. It used residual connection adding stacked layers instead of stacking these layers directly, which makes it easier to optimize compared with the original one. It has already been used in various types of neural networks and achieved great success. In this work, residual blocks are used to fill the grey blocks in Figure 4-1 firstly, which is similar with V-net [44]. For each grey block in different layers, different numbers of convolution blocks are used in residual blocks. The number of convolution blocks and their sizes are shown in Table 4-1.

Table 4-1. The numbers of convolution blocks in grey blocks

Size of Convolution Blocks	Number of Convolution Blocks
48×32×32×32	2
96×16×16×16	3
192×8×8×8	3
384×4×4×4	3

4.1.1.2 Residual SE Block

Squeeze-and-Excitation Networks (SENet) [48] introduced SE block which improves prediction accuracy through modeling the correlations between channels and adaptively strengthening important features. This structure performed great in ILSVR competition in 2017. AnatomyNet [51] took advantage of SE residual blocks to build a 3D SE Res U-net, which performed best among the networks they tested.

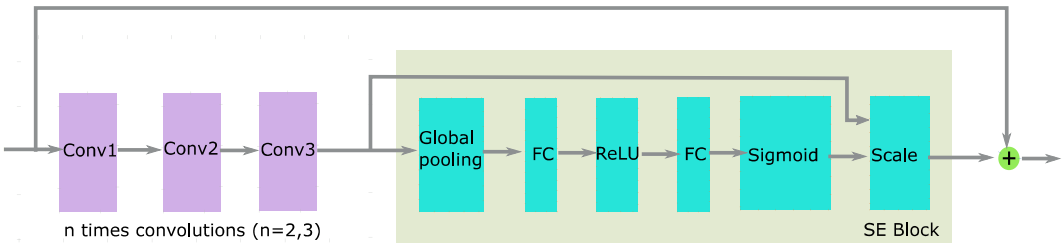


Figure 4-2. Residual SE block

In this work, residual SE blocks are also used to fill the grey blocks. As shown in Figure 4-2, 2 or 3 convolution blocks are used first same as residual blocks described above. The numbers of blocks stacked are same as Table 4-1. Then SE blocks are used to strengthen important features. In SE blocks, the channel-wise features are calculated by global average pooling firstly. For the c-th channel, the feature is squeezed according

$$z_c = \frac{1}{H \times W \times D} \sum_{i}^{H} \sum_{j}^{W} \sum_{k}^{D} u_c(i, j, k) \tag{4.1}$$

where $u_c(i, j, k)$ is an element in the feature map u_c (whose size is H×W×D, and generated by stacked convolution blocks) of the c-th channel. After this operation, the spatial dimension of the feature map of all channels is changed from H×W×D×C to 1×C. The feature of each channel is squeezed to be represented in a number. After squeezing, two fully connected layers are used as a simple gating mechanism to capture channel-wise dependencies. They can be written as:

$$s_c = F(z_c) \tag{4.2}$$

where F refers to the mapping relations of these two fully connected layers. The final output of the SE block is obtained by channel-wise multiplication between the feature map generated by stacked convolution blocks and the scalar output by the two fully

connected layers. The feature map of the c-th channel is recalibrated by the factor s_c according

$$u_c' = s_c * u_c \tag{4.3}$$

In this way SE blocks can be used to adaptively strengthen the important features. SE blocks perform well on improving the segmentation accuracies of small organs in this work.

4.1.1.3 Dense Block

Dense convolutional network (DenseNet) [46] considered that the residual connections combining the input with the output of stacked convolutions by summation impedes the information flow in the network, and then proposed a different connectivity pattern. In traditional convolutional networks, the output of current layer is the input of the next layer, which means L layers have L connections. DenseNet introduced a framework that connects the output of each previous layer to the next layer, which means L layers have $\frac{L(L+1)}{2}$ connections.

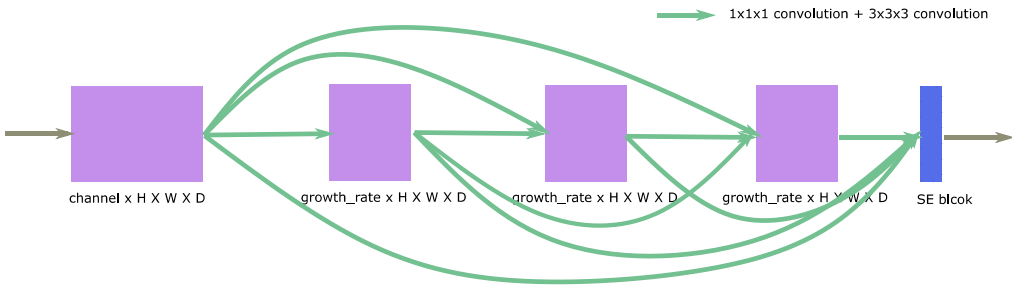


Figure 4-3. Dense SE block

As shown in Figure 4-3, before the SE block it is a 4-layer dense block. Each layer takes all preceding feature-maps as the input to reuse these features in order to take advantage of features gained in different layers to exploit the potential of the network. The input of i-th layer can be represented by:

$$u_i = \text{Concat}([u_0, u_1, \ldots, u_{i-1}]) \tag{4.4}$$

where u_i is the feature map obtained in i-th layer. The growth rate in the Figure 4-3 is the number of convolution filters in each layer except the input layer. The number of input feature maps for l-th layer is c0 + growth_rate × (l − 1). Here c0 is the number of channels of the input layer in the block. This means the input feature maps increase by a factor of growth rate. In addition, 1×1×1 convolutions are added as bottleneck layers before each 3×3×3 convolutions in order to reduce the number of input feature maps for each layer.

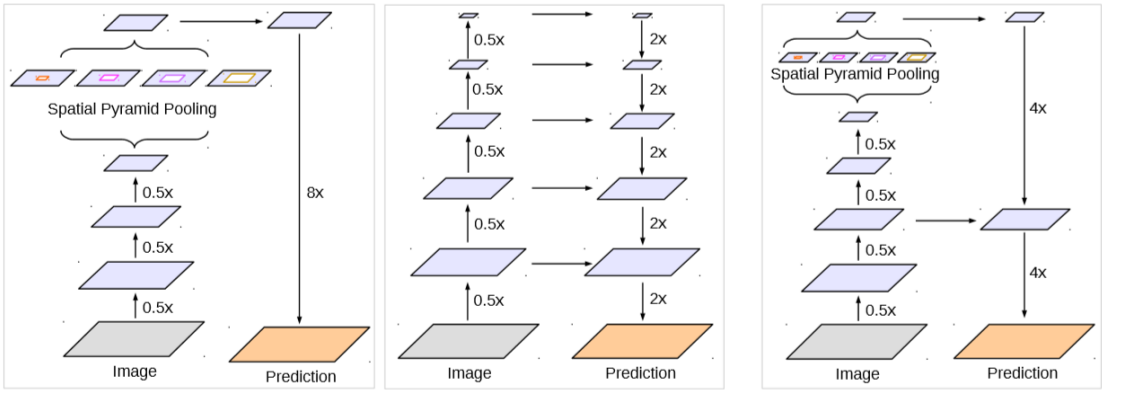
For dense blocks, a relatively small growth rate is sufficient to obtain state-of-the-art results, which enable us to try a relatively larger batch size because of the high memory requirement of the model during training. Three dense block structures whose growth rates decreased gradually are designed in the work. The experiment shows the network with smaller growth rates and trained in a bigger batch size achieved better results.

4.1.1.4 Dense SE Block

To take advantage of both dense blocks and SE blocks, the blocks described in Figure 4-3 are used to fill the grey blocks in Figure 4-1. Three structures have been tested whose dense blocks have the identical hyper parameters as Chapter 4.1.1.3. SE blocks can be easily added in a residual structure and a non-residual structure. Residual SE block described in Chapter 4.1.1.2 is obviously an example that it is used in a residual structure. Dense SE block in this chapter is an example that it is used in a non-residual structure.

4.1.2 DeepLab variants

The fusion of features in different scales is beneficial in improving accuracy of semantic segmentation. In Deeplabv3 [38], it discussed four architectures to capture multi-scale context. The first one is usually applied during the inference stage. For the other three, encoder-decoder architecture, which was used in U-net, is one of them. Atrous convolution and Spatial Pyramid Pooling are another two. Deeplabv3plus [39] took advantage of these three architecture and proposed the architecture of encoder-decoder with atrous convolution as shown in the figure below.



(a) Spatial Pyramid Pooling (b) Encoder-Decoder (c) Encoder-Decoder with Atrous Conv

Figure 4-4. The architecture of encoder-decoder with atrous convolution

Unlike encoder-decoder architecture used in Chapter 4.1.1, encoder-decoder with atrous convolution is asymmetrical. It used ASPP (Atrous Spatial Pyramid Pooling) at the end of encoder and bilinear upsampling in the decoder. This architecture has also employed with modification in this thesis. To explain it clearly, atrous convolution will be introduced first.

4.1.2.1 Atrous convolution

In convolutional neural network, we usually use pooling and convolution with stride to downsample images in order to increase the receptive field and then obtain the features of images in high level. However, there is resolution loss in pooling, and for convolution with stride, there are many parameters involved. Atrous convolution (or dilated convolution [52]) was designed to increase receptive field with fewer parameters. Compared with standard convolution, atrous convolution has one more hyperparameter, dilation rate, which defines intervals between pixels in the kernel. For the same kernel size, using a larger dilation rate can obtain a larger receptive field with the same number of parameters. Atrous convolution can also be used to get the features from different scales.

Figure 4-5 shows an example of how to calculate two-dilated convolution with stride 1. For two-dilated convolution whose kernel size is 3×3, the receptive field is actually 5×5. It has the same number of trainable parameters as the standard convolution we discussed in Chapter 2 whose receptive field is 3×3. So, we can see that it can increase the receptive field with fewer parameters.

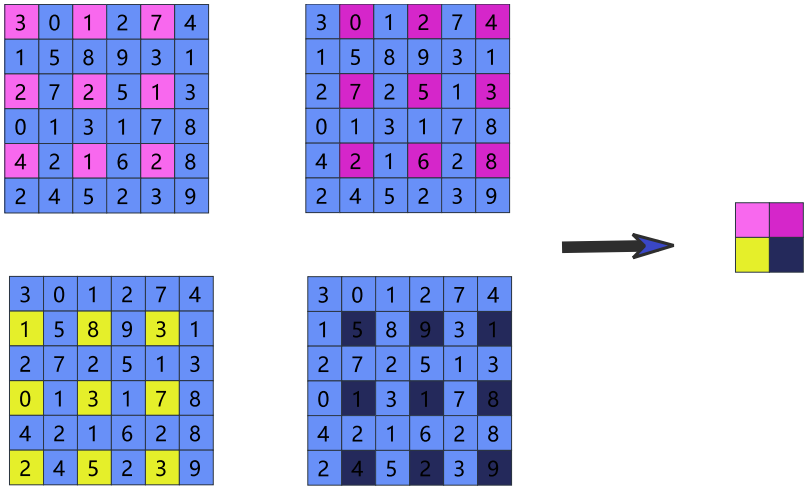
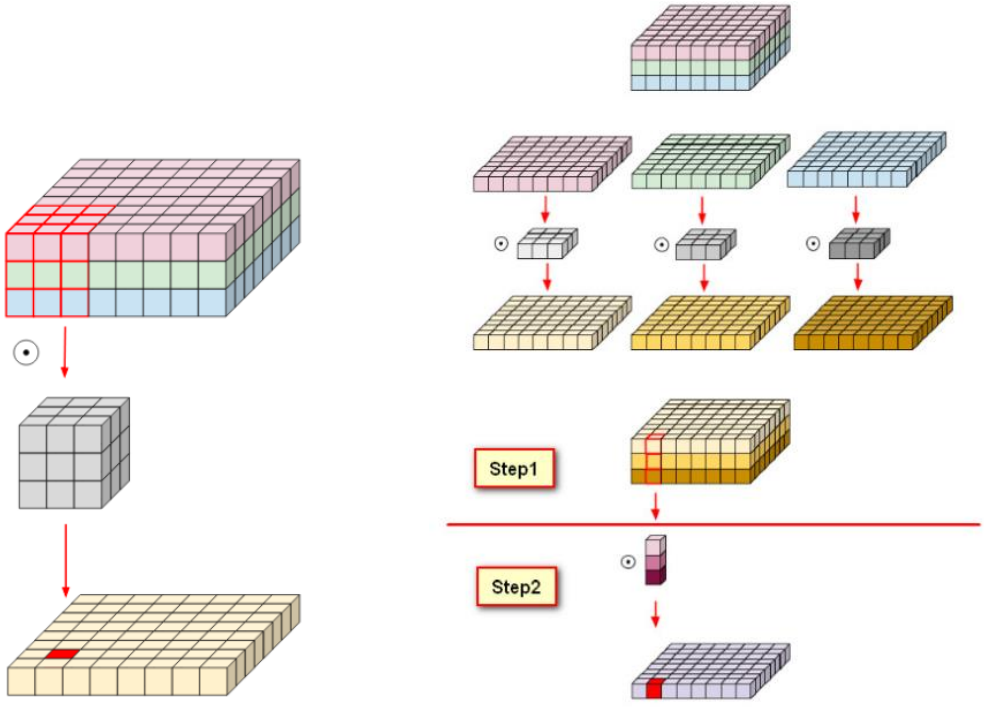


Figure 4-5. Atrous convolution

4.1.2.2 Depthwise separable convolution

To understand ASPP (Atrous Spatial Pyramid Pooling), we should understand depthwise separable convolution firstly. Figure 4-6 [53] compares the calculation of standard convolution and depthwise separable convolution.



a) Standard convolution b) Depthwise separable convolution

Figure 4-6. Depthwise separable convolution

For standard convolution, as explaining in Chapter 2, we use the convolution filter whose channel number is same as the input image (3 in this case) to do element-wise multiplication, and then add the products together to get a scalar as the final result. So, the output is an image whose channel number is one as shown in a).

For depthwise separable convolution, it is changed after obtaining the products of element-wise multiplication. Instead of adding them together, we use addition in channel-wise. The calculation process is same as using three convolution filters whose channel numbers are one to operate the corresponding channel of the input image. The result is a feature map which has the same channel number as the input image, which means the convolution is only implemented in the height and width directions. It is the first step as shown in b). Then 1×1 filters whose channel number is same as the input image are adopted, which means the convolution in the depth direction is implemented. It is the second step as shown in b).

Depthwise separable convolution uses 1×1 filters to choose features from the results of element-wise multiplication instead of adding them directly in standard convolution, which obviously keeps more spatial information. In addition, it uses fewer parameters in some cases. For example, in the case that the channel number of input image is 16 to get an output image whose channel number is 32. For standard convolution, we use 32 filters whose sizes are 3×3×16, so the number of parameters here is $3\times3\times16\times32 = 4608$. For depthwise separable convolution, we use a filter whose size is 3×3×16, then we use 32 filters whose size are 1×1×16, so the number of parameters is $3\times3\times16 + (1\times1\times16)\times32 = 656$.

4.1.2.3 ASPP

Spatial Pyramid Pooling was proposed by [54] in order to capture context of images in different strides. ASPP (Atrous Spatial Pyramid Pooling) was introduced by DeepLabv2 [37], where parallel atrous convolution layers with different dilation rate capture multi-scale features, which is shown in the figure below [38].

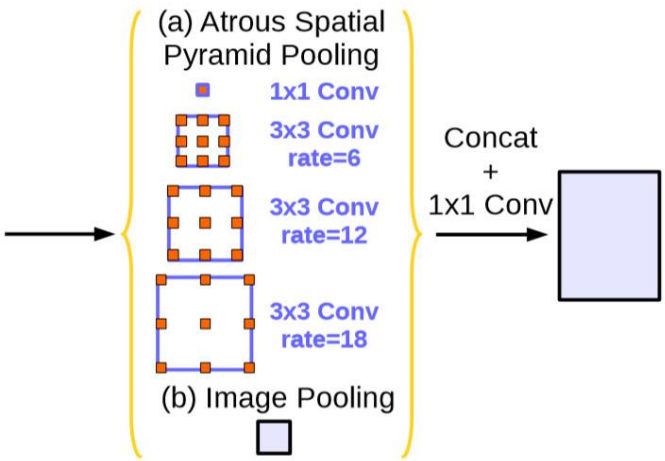


Figure 4-7. ASPP (Atrous Spatial Pyramid Pooling)

It used four atrous convolution layers whose dilation rates are different to capture multi-scale information of the images, and a global average pooling layer to capture the image-level feature. The sizes of output images of these five layers are same as the input image. And then concatenated these five outputs and used 1×1 convolution to reduce the number of channels and choose features.

DeepLabv3plus [39] applied atrous separable convolution to replace the atrous convolution in the four convolution layers of ASPP. Atrous convolution was adopted in the depthwise separable convolution as shown in the figure below [39]. It means to use atrous convolution instead of standard convolution in the first step of depthwise separable convolution.

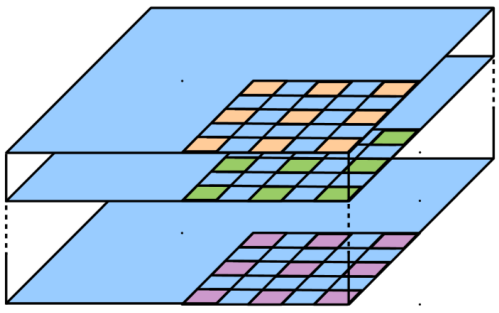


Figure 4-8. Atrous separable convolution

4.1.2.4 Network

The architecture of DeepLabv3plus was shown in Chapter 3.2.1.3. It was an asymmetrical encoder-decoder architecture. The encoder was comprised of improved Xception and ASPP. The decoder used bilinear upsampling concatenated with the corresponding low-level features from the network backbone. A type of DeepLab variant for our dataset was developed based on it. The architecture of the neural network is shown in the figure below.

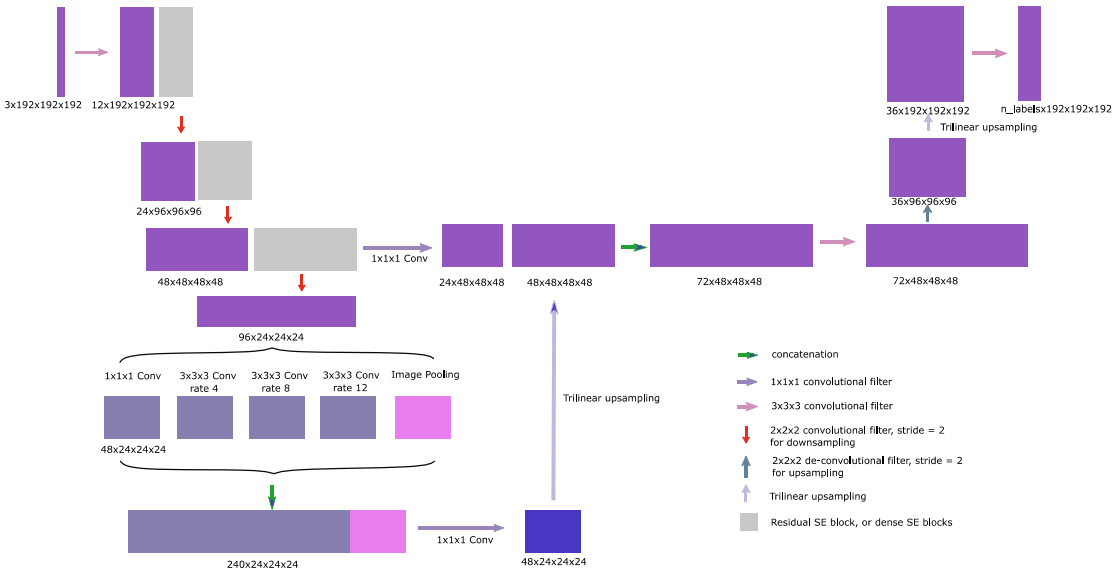


Figure 4-9. The architecture of DeepLab variants

DeepLabv3plus used Xception as the basic network. However, Xception is too complicated, which is not necessary for our dataset. So it is replaced with the encoder part of 3D U-net variants, and the numbers of channels in each layer is reduced by a half. The last layer of encoder is removed and ASPP is added at the bottom of encoder. The dilation rates used in ASPP are [1, 4, 8, 12]. 1×1×1 convolution is used to reduce the number of channels after ASPP, which is same as it is used in dense blocks.

For decoder, trilinear upsampling (similar with bilinear upsampling in 2D) is used to upsample the image, and then concatenate with the low-level feature got in the encoder network. 1×1×1 convolution was applied to reduce the number of channels of the low-level feature before concatenation. After concatenation, 3×3×3 convolution block was added before de-convolution upsampling. Then another trilinear

upsampling was employed to recover the images to the original size. And then use a convolution block to get the final segmentation results.

The overall architecture is similar with DeepLabv3plus except the basic network used in the encoder. De-convolution upsampling block is added between the two trilinear upsampling operations in order to improve the accuracy because medical image segmentation has higher requirement than natural image segmentation. The factor of upsampling in DeepLabv3plus is 4, but in this neural network, it is 2.

As we discussed above, because this architecture use atrous separable convolution, the numbers of channels in encoder was reduced by a half, and trilinear upsampling instead of de-convolution was used in decoder, fewer memory resources are required to train this network. As shown in the figure above, the input patch size is 192×192×192.

4.1.3 Combined neural networks

Combined neural networks can take advantage of multiple neural networks to improve the accuracy. According to Geirhos et al. [55], common CNN architectures are biased towards recognizing image textures, not object shape representations. In medical image analysis, however, expert manual segmentation usually relies on boundary and organ shape identification. Hatamizadeh et al. [56] proposed a type of boundary aware CNN for medical image segmentation. It increased the dice coefficient about 0.05 on BraTS 2018, which provides multimodal 3D brain MRIs and ground truth brain tumor segmentations annotated by physicians [57]. ET-Net [58] used the similar method as [56] and achieved better results compared with other neural networks such as U-net on the datasets they used. So it is possible that this architecture can bring benefits for this task as well.

A type of combined neural network which is consisted of a segmentation network and an edge detection network has been built in this work. 3D U-net variants described in Chapter 4.1.1 are used as the segmentation network. For the edge detection network, CASENet [59] is used as a reference.

4.1.3.1 Edge detection networks

The architecture of CASENet [59] is shown in the figure below [59] where it can be seen that the left part is similar as the encoder in 3D U-net. So, the right part can be integrated into the 3D U-net variants easily.

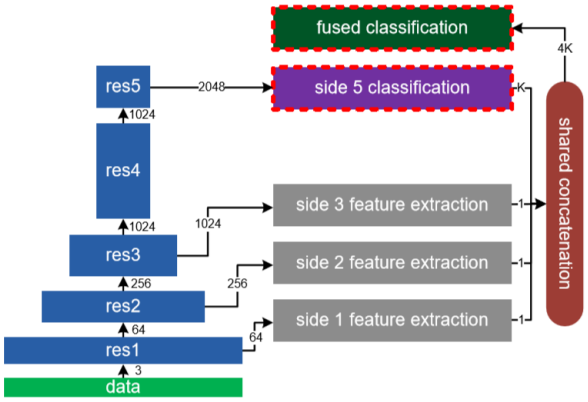
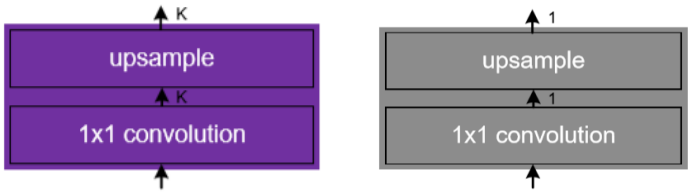


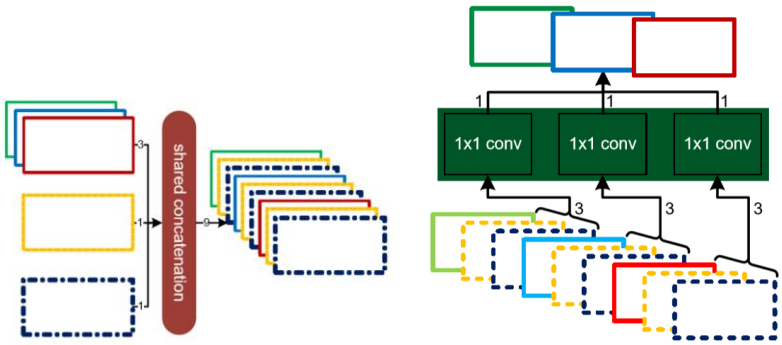
Figure 4-10. CASENet

The left part of CASENet is actually a stack of residual blocks. In the right part, there are four basic components including side classification, side feature extraction, shared concatenation, and fused classification as shown in Figure 4-11 [59].

Side classification used 1×1 convolution to reduce the number of feature maps to K (K is the number of classes in the dataset), and then upsample the feature maps to the original size. Side feature extraction is similar with side classification but to reduce the number of feature maps to one instead of K. Shared concatenation concatenated the feature maps output by two side feature extractions with the K feature maps output by side classification in the order shown in c) (where K=3) to get 3K feature maps. In Figure 4-10, there are three side feature extractions, so the number of feature maps is 4K after shared concatenation. In the end, fused classification uses 1×1 convolution to reduce the number of feature maps to K.



a) Side classification b) Side feature extraction



c) Shared concatenation d) Fused classification

Figure 4-11. Components of CASENet

4.1.3.2 Network

To combine the edge detection network with the segmentation network, 3D U-net variants described in Chapter 4.1.1 are used as the basic network, and the components of CASENet described above are added to construct the combined neural network, whose architecture is shown in Figure 4-12. It is similar with CASENet except the basic network part where 3D U-net variants were used. 3D U-net is the segmentation neural network. The encoder part of it is connected with the components of edge detection. They comprise the edge detection network, whose output should be the edge image as shown in the figure.

Significantly, the number of labels of edge detection neural network is 2 rather than 13 (there are actually 13 labels in the dataset). The reason is that the purpose is to improve the segmentation results rather than edge detection results. Only two labels are needed, edges and non-edges.

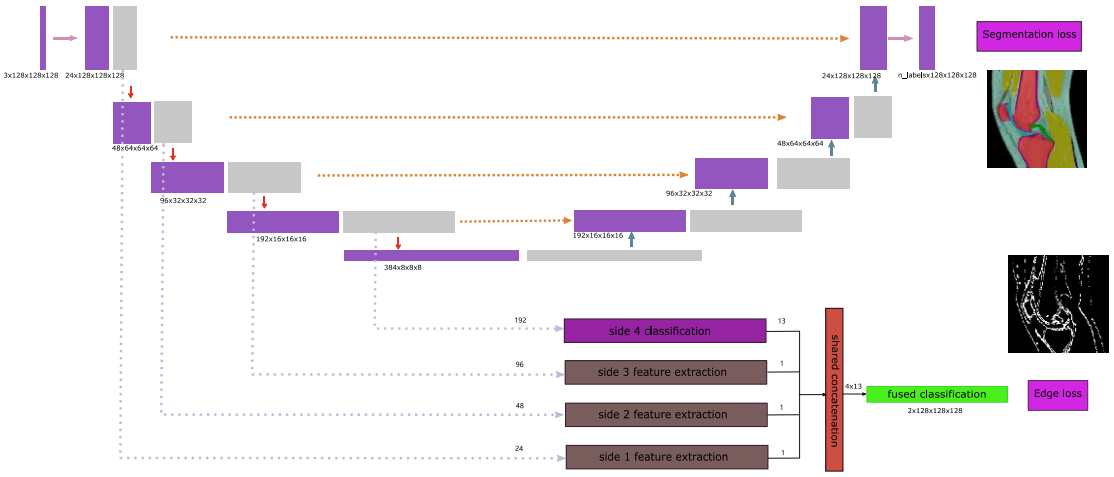


Figure 4-12. Combined neural network

4.2 Loss functions

As we discussed at the beginning of Chapter 4, there are two directions to improve the accuracy when the task involves multiple extremely imbalanced classes: employing innovative neural networks, and using appropriate loss functions. In terms of loss functions, one of the simplest methods is to use weighted loss functions.

4.2.1 For 3D U-net variants and DeepLab variants

Small object segmentation is always a challenge in semantic segmentation. As we discussed in Chapter 3.2.2.4, dice loss was developed based on dice coefficient in order to address the problem that the learning process getting trapped in local minimum when the predictions are strongly biased towards background. The dice coefficient between two binary volumes is calculated according

$$D(A,B)=\frac{2\times(|A\cap B|)+\text{smooth}}{|A|+|B|+\text{smooth}} \tag{4.5}$$

The difference between this equation and equation 3.4 is that it adds smooth (> 0) to both numerator and denominator. The purpose is to avoid the denominator becomes 0 when $|A|+|B|=0$.

In this task, there are several extremely imbalanced classes (which will be discussed in Chapter 5.1). To balance different frequencies of multiple classes in the dataset, weighted dice loss was used. It is calculated according

$$\text{Loss}=1-\frac{1}{n}\times\sum_{i=1}^{n}(w_i\times\frac{2\times\sum_{j=1}^{m}t_{ij}p_{ij}+\text{smooth}}{\sum_{j=1}^{m}(t_{ij}+p_{ij})+\text{smooth}}) \tag{4.6}$$

where n is the number of classes, w_i is the weight of class i, m is the number of voxels of class i, t_{ij} is the j-th voxel of class i using one-hot encoding in the truth, p_{ij} is the corresponding voxel in the prediction.

When weighted dice loss has been chosen, how to set the weights becomes another problem. Two methods have been tried to set the weights. The first one is to set them according the frequencies of classes in the dataset:

$$w_i=\frac{\max([f_0, f_1,\ldots,f_9])}{f_i} \tag{4.7}$$

where f_i is the frequency of *i*-th class, w_i is the weight of *i*-th class. So the weight of the class which has the largest frequency is one. The second method is to set them according the final dice coefficients of these classes when the network with unweighted dice loss is converged.

For 3D U-net variants, because the final dice coefficients of the classes when the network using unweighted dice loss is converged are close to their frequencies in the dataset, the first method is used to set the weights. For DeepLab variants, both of these two methods have been tried.

Some weighted loss functions such as generalized dice loss function as discussed in Chapter 3.2.2.5, which introduced a method to calculate the weights automatically, but it makes the optimization unstable in the extremely unbalanced segmentation [60].

4.2.2 For combined network

For the combined network described in Chapter 4.1.3.2, a combined loss function is needed as well. The gradients of segmentation ground truth are calculated to generate the ground truth of edges as shown in Figure 4-13. For the segmentation, the loss

function is described by equation 4.6. For the edge detection network the loss function is calculated according this equation as well, but there are only two classes, edges and non-edges. The total loss function used for training the whole neural network is

$$\text{Loss}=\text{Seg\_loss}+w*\text{Edge\_loss} \tag{4.8}$$

where w is used for increasing the weight of edge detection loss, Seg_loss is the loss for segmentation.

4.3 Metrics

The mean of total dice coefficient and the dice coefficient of each class on validation dataset is used as the metrics during training. However, if the dice coefficient of each class is calculated by equation (4.5), there is a problem which may cause confusion. When the batch contains very few voxels of a class, which is possible for small organs and small batch size, its dice coefficient could become relatively big even if the overlap between the prediction and the truth is small. The worst case is the batch doesn't contain the class; the dice coefficient will become one. So equation (4.5) was modified to

$$D(A,B)=\frac{2\times(|A\cap B|)}{|A|+|B|+\text{smooth}} \tag{4.9}$$

It reflects the actual overlap even in worst situation. But equation (4.6) is still used to calculate the loss function. It performs better in training because it has smoother gradients compared with (4.9).

To compare the predicted segmentation and the ground truth for each class, the percentage of class *y* predicted as class *x* is calculated according

$$P_{(x,y)}=\frac{n_{(x,y)}}{n_y} \tag{4.10}$$

where $n_{(x,y)}$ is the number of voxels predicted as class *x* but annotated as class *y* in the ground truth, n_y is the number of voxels annotated as class *y* in the ground truth. It is actually the recall rate for class *x* when *x* = *y*. Significantly, it is actually not confusion matrix, it will be named performance matrix in this thesis. The data in the

diagonal line is the recall rate for each class. They should be 1 if all voxels are segmented correctly. The data in the diagonal line would be precision if replace the denominator of equation (4.10) as the number of voxels predicted as class y.

Some of tables in the next chapter will show the performance of various neural networks, which will be evaluated on dice coefficients without adding smooth to neither numerator nor denominator including the total one and the one for each class.

5. Experiments and results

In this chapter, we will discuss the two datasets used to train the neural networks first. Preprocessing is a very important step to deal with the data before feeding them into a neural network, so it will be introduced too. Then the details of the experiments will be introduced. After presenting the results of the experiments, we will discuss the performance of the neural networks and the reason behind it.

5.1 Datasets

The experiments are based on two datasets, which are available in different stages of the thesis. Both of them are provided by Sunnmøre MR-Klinikk. The first dataset contains 15 annotated knee MRI images whose size is 400×400×275. Each knee image has 10 labels in ground truth. The second dataset contains 20 knee MRI images whose size is 400×400×400. Each knee image has 13 labels in ground truth. For both datasets, there are three weighted volumes, T1, PD and FS, provided for each knee.

5.1.1 Dataset1 (with 10 labels)

There are 10 classes on dataset1. Table 5-1 shows these classes, and their abbreviations and values, which will be useful when analyzing the segmentation results. As declared in Chapter 1.2, the details of these organs in medical domain won't be discussed unless it is helpful for the segmentation task.

Table 5-1. The abbreviations and values of classes (organs) on dataset1

Classes	**Abbreviations**	**value**
Background	BG	0
Bone	BO	1
Posterior cruciate ligament	PCL	2
Anterior cruciate ligament	ACL	3
Muscle	MU	4
Cortical bone	CB	5
Blood vessel (popliteal artery/vein ++)	BV	6
Adipose tissue (fat)	AD	7
Tendons	TE	8
Menisci	ME	9

Figure 5-1 is an example of segmentation on dataset1, which shows the locations of these 9 classes (excluding background). The image is 3D, so it may be impossible to

show all of them in one 2D image. Here they are shown in two images, which are in the different depths of the 3D image.

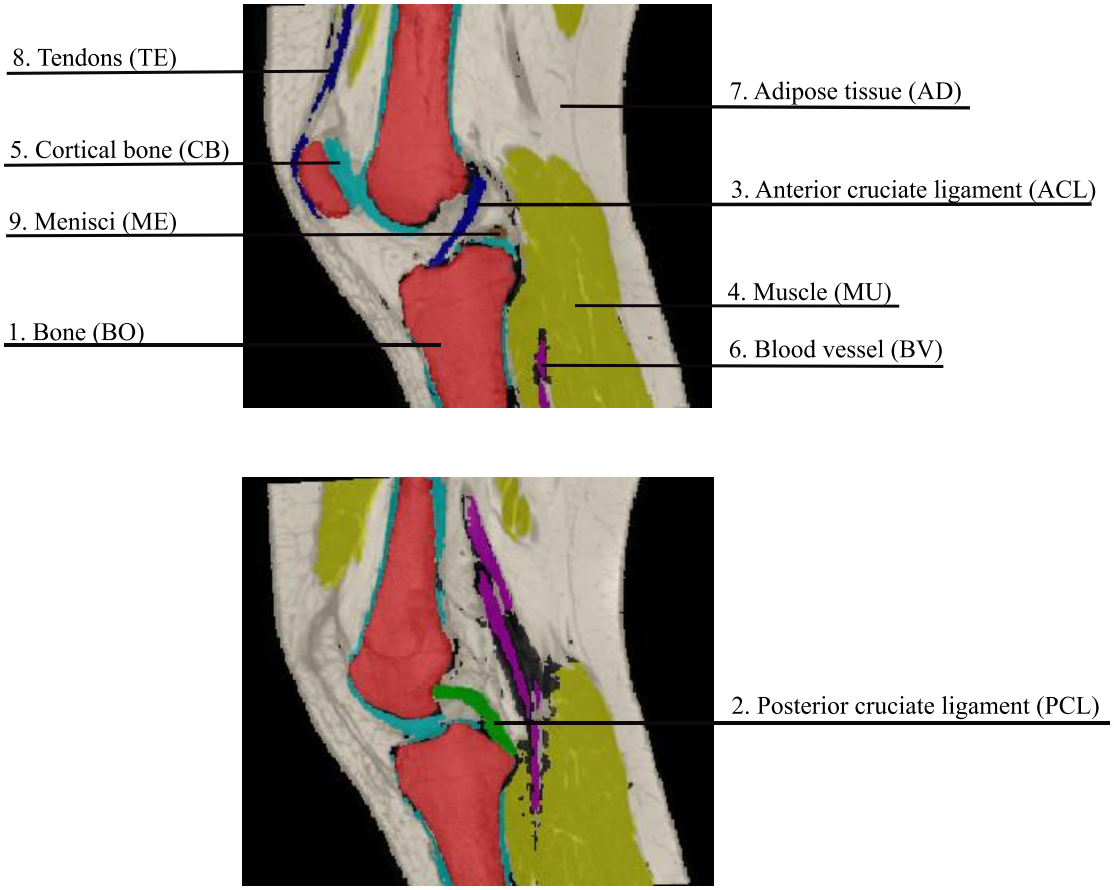


Figure 5-1. An example of segmentation on dataset1

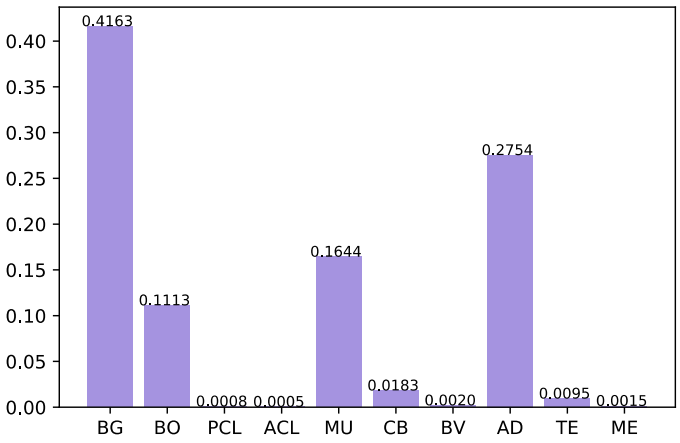


Figure 5-2. The frequency of voxels for each class on dataset1

The frequencies of voxels of these classes are extraordinary different, which can be seen in Figure 5-1. Figure 5-2 shows the details of the frequencies, where we can see that background accounts for 41.63%. The largest label is AD (adipose tissue) accounting for 27.54% while the smallest one is ACL (Anterior cruciate ligament) accounting for 0.05%.

5.1.2 Dataset2 (with 13 labels)

For the second dataset, there are three more classes in the ground truth including artery, collateral ligament and veins. The figure below is an example of segmentation on dataset2. The second image is the intersecting surface of the first image. The abbreviations and values of these 13 classes can be also found in Figure 5-3.

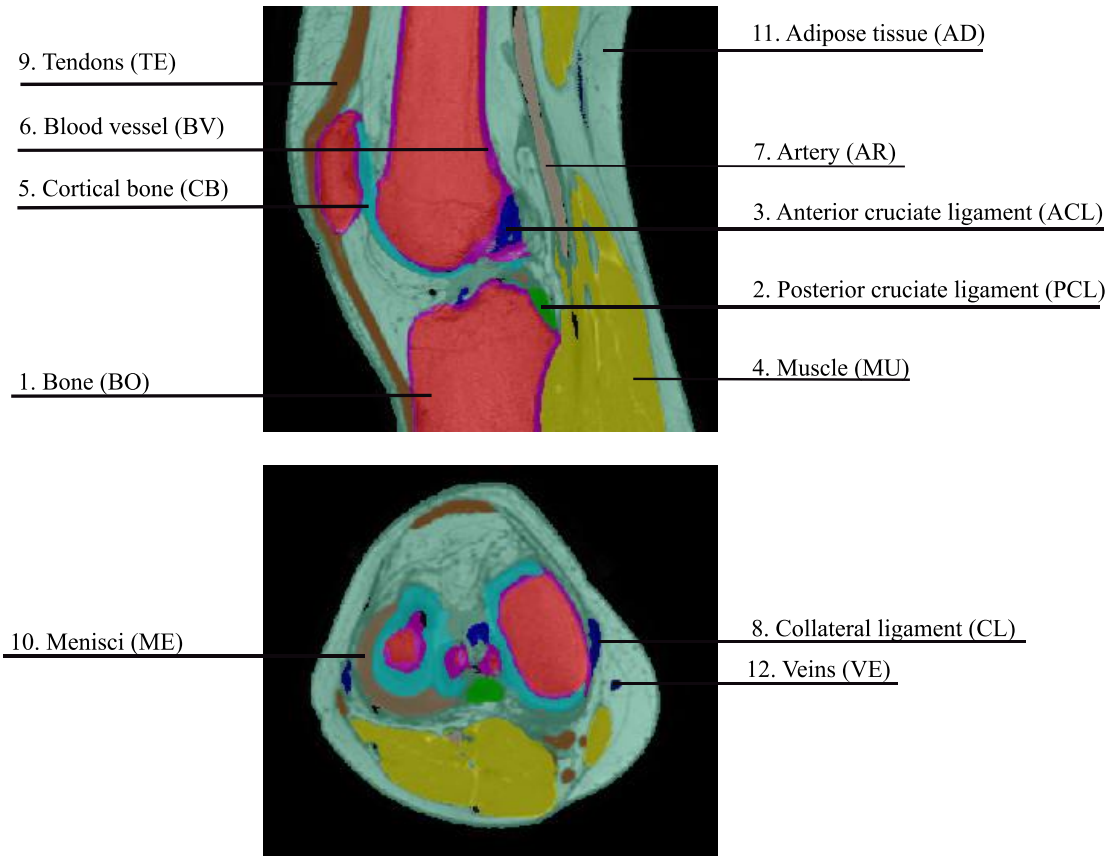


Figure 5-3. An example of segmentation on dataset2

The frequencies of voxels of these classes on dataset2 are even more imbalanced compared with those on dataset1, which can be seen in Figure 5-4. The background accounts for 60.43%. The largest percentage of label and the smallest percentage of

label are same as on dataset1. The largest label is AD (adipose tissue) accounting for 19.17% while the smallest one is ACL (Anterior cruciate ligament) only accounting for 0.03%.

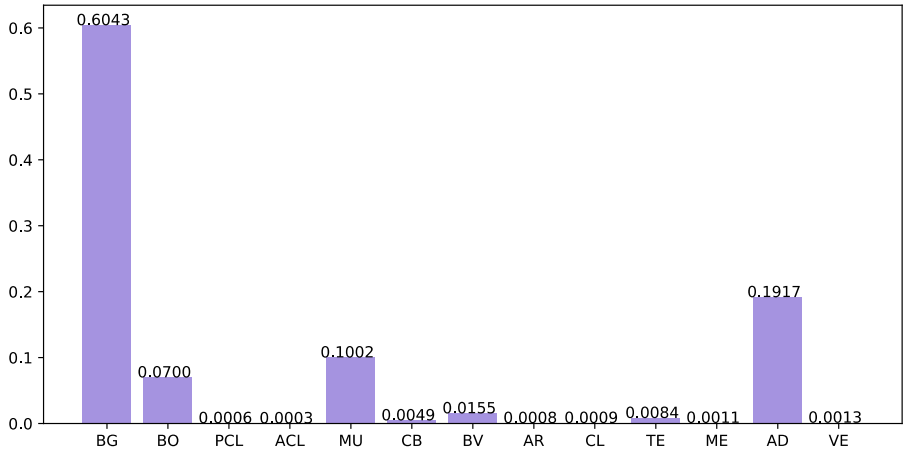


Figure 5-4. The frequency of voxels for each class on dataset2

5.2 Training and Results

The experiments on dataset2 are based on the results of experiments on dataset1. So experiments on dataset1 will be introduced first. For the experiments on dataset1, *Keras* based on *Tensorflow* was used as the deep learning framework because it is friendly to new users. For the experiments on dataset2, *Pytorch* was chosen instead because it has been becoming more popular, and lower level programming (*Keras* is a high-level API) is more flexible for customization. In terms of hardware, all the neural works are trained on GPU GeForce RTX 2080 Ti, whose memory is 11 GB GDDR6.

The codes of the neural networks are relatively simple because the deep learning frameworks, *Keras* or *Pytorch*, provide functions and interfaces which enable users to implement them easily. The implementation details of data preprocessing will be introduced since it is different for diverse types of input data, which have various formats or sizes etc.

5.2.1 Experiments on dataset1

The first dataset used in the experiments contains 15 annotated knee MRI images with 10 labels. 12 images were used for training and 3 of them were used for validation. The networks described in Chapter 4.1.1 (3D U-net variants) were trained on this dataset first. The mean of total dice coefficient and the dice coefficient of each class on validation dataset is used as the metrics during training.

5.2.1.1 Preprocessing

Before starting the training, we need to preprocess the images in order to turn them into the appropriate format, which can be fed into the neural network. Augmentation is also another important stage in preprocessing. However, more data means a longer training time, and the benefit is not that obvious in the experiments. So data augmentation has not been implemented in the experiments.

Cutting image into patches

As we discussed above, 3D medical images are very large, which need large memory size for training them. An image even can't be trained by using only one GPU if the neural network is complicated. In this case, they should be cut into patches, so they can be trained in the unit of patch.

For dataset1, the original image's size is 400×400×275. They were resampled to 274×274×274 in order to decrease the training time. Because the limitation of GPU memory, they were divided into 64×64×64 patches to feed the neural networks as shown in Figure 5-5.

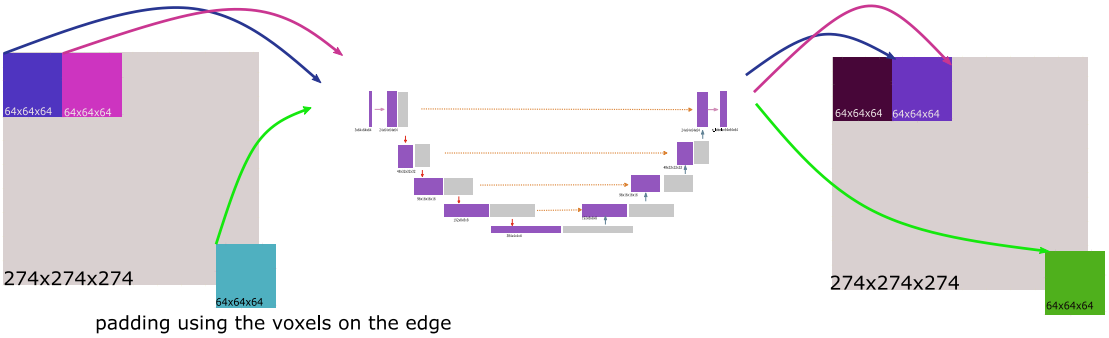


Figure 5-5. Patch-wise training due to the limitation of GPU memory

The 274×274×274 images were cut into 64×64×64 patches. If the remaining part is smaller than patch size, the voxels on the edge will be used for padding. Every time a batch of patches instead of images were fed into the network. The final segmentations were predicted in patches. Then the predicted patches were integrated into an image.

Data format

HDF5 (Hierarchical Data Format) is used to store the data of input images. *HDF5* is a file format which is designed to store and organize huge amounts of numerical data. A *HDF5* file contains comprehensive information of data, which allows the application program to interpret the structure and content without any external information. It allows users to combine relevant data objects together, put them into a hierarchical structure, and add descriptions and labels to these data objects. Many data types can be embedded in a *HDF5* file. It is a platform-independent file format, which can be used on different platforms without any conversion.

The input MRI knee images' format is *nifti* having the extension *.nii*. *Nibabel*, which is a python package developed for medical images processing, has been used to read the data of *.nii* file to *.h5* file (*HDF5* file). *PyTables* is a package for managing hierarchical datasets and designed to efficiently and easily cope with extremely large amounts of data. It is built on top of the *HDF5* library, using *Python* and the *NumPy* package. It was used to process *.h5* file in the experiments on dataset1. Related pseudo- codes are shown below.

```
# load the image using nibabel
images = nib.load(nii_file_path)
# other kinds of preprocessing
images = resample(images)
# get data from the corresponding image
data = images.get_fdata()…
truth = images.get_fdata()…
affine = images.get_fdata()…
# return a hdf5 file handle using PyTables
hdf5_file = tables. open_file(h5_file_name, mode='w')
# copy the data into hdf5 file
data_storage = hdf5_file.create_earray(hdf5_file.root, 'data', …)
truth_storage = hdf5_file.create_earray(hdf5_file.root, 'truth', …)
```

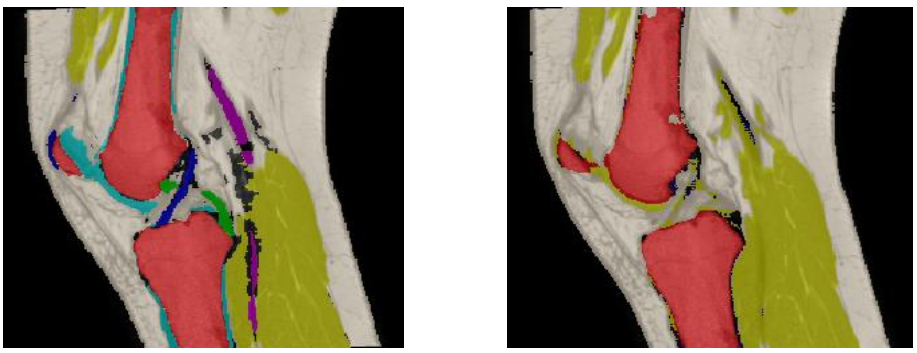
```
affine_storage = hdf5_file.create_earray(hdf5_file.root, 'affine', ...)
data_storage.append(np.asarray(data))
truth_storage.append(np.asarray(truth))
affine_storage.append(np.asarray(affine))
...
hdf5_file .close()
```

5.2.1.2 Training of 3D U-net and its variants

For dataset1, all networks were trained using Nadam optimizer. The batch sizes used in the implementation were adjusted according the networks. For training the networks with dice loss, the learning rate was set to 0.001 initially. For training the networks with weighted dice loss, the initial learning rate was set to 0.01. The learning rates were set to be reduced by a factor of 0.5 after 5 epochs if the validation loss is not decreasing.

Preliminary models

3D U-net and V-net (U-net with residual blocks) with dice loss were tested firstly. The batch size was set to 4. The results show the large organs bone (BO), muscle (MU) and adipose tissue (AD) were segmented but the small ones such as posterior cruciate ligament (PCL) are missing, which can be seen in Figure 5-6.



(a)Ground truth (b) V-net with dice loss

Figure 5-6. Segmentation results of V-net with dice loss

This result is caused by the extremely imbalanced classes. As shown in Figure 5-1, except background, the percentages of BO, MU and AD in the images are much

bigger than others. If the networks segment them correctly, the total dice coefficient has already been at least 0.94 (see Table 5-4). So it is reasonable for them to almost ignore the small organs.

The performance matrix is shown in Figure 5-7. The X-axis is the values of predicted classes, and the Y-axis is the values of ground truth classes. So the column whose X-value is 0, for example, shows the percentages of the classes are predicted as class 0 respectively. We can see almost all small organs are predicted as background (0), bone (1), muscle (4), and adipose tissue (7).

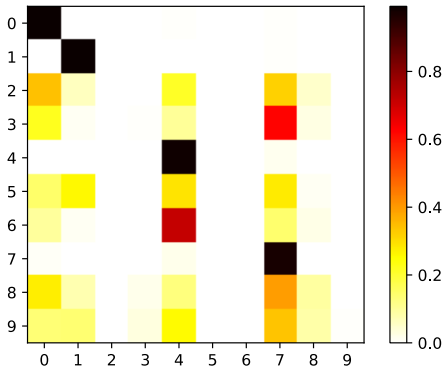
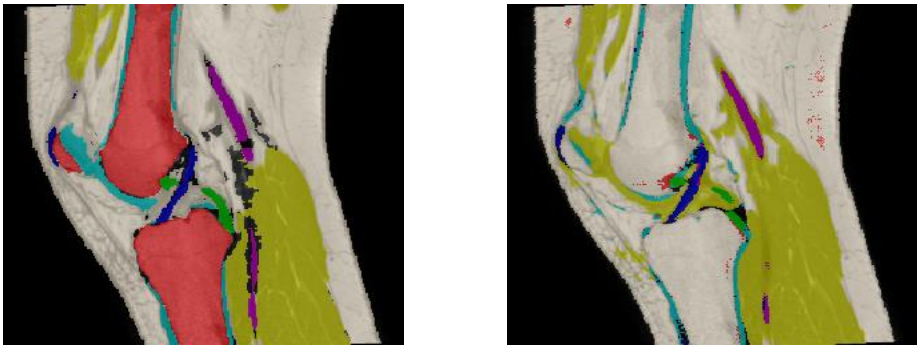


Figure 5-7. Performance matrix for V-net

To address this problem, weighted dice loss was adopted. V-net (U-net with residual blocks) with weighted dice loss and the same batch size was tested. The dice coefficients of small organs are improved but for the large ones they are decreased, especially for bones. Its dice coefficient drops dramatically to 0.06 (See Table 5-4).

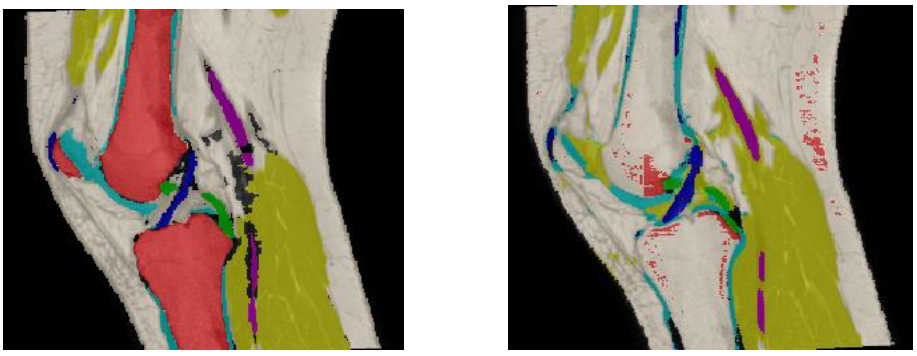


(a) Ground truth (b) V-net with Weighted dice loss

Figure 5-8. Segmentation results of V-net with Weighted dice loss

Model with residual SE blocks

To improve the accuracy, residual SE blocks were used instead. Same batch size and learning rate as V-net were set. From Table 5-4, we can see that the results of small organs are improved compared with that using only residual blocks, which is same as our expectation. However, the accuracy of BO is still low.



(a) Ground truth (b) U-net with residual SE blocks

Figure 5-9. Segmentation results of U-net with residual SE

From Figure 5-10, we can see most of bones (1) are annotated as adipose tissue (7) incorrectly. Most small organs are around bones, so one explanation is that the network sacrificed its accuracy to improve the accuracies of small organs because they have bigger weights. And some small organs especially blood vessels (6) are annotated as muscles (4) incorrectly because they are close.

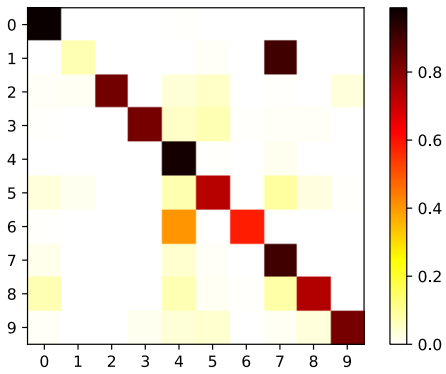


Figure 5-10. Performance matrix for V-net with residual SE blocks

Models with Dense, Dense SE blocks

Then models with dense and dense SE blocks were tested. To choose a better neural network for the dataset, three structures were tested. The hyperparameters of dense blocks in these three networks are shown in Table 5-2. The structure with relatively smaller growth rates enables us to use a bigger batch size. For dense network 1, 2, and 3 in Table 3, batch size was set to 4, 6, and 8 respectively.

Table 5-2. The hyperparameters of dense blocks

Dense Network1		
Size of feature maps	**Number of layers**	**Growth rate**
32x32x32	2	24
16x16x16	3	48
8x8x8	3	96
4x4x4	3	192
Dense Network2		
Size of feature maps	**Number of layers**	**Growth rate**
32x32x32	2	12
16x16x16	3	24
8x8x8	3	48
4x4x4	3	96
Dense Network3		
Size of feature maps	**Number of layers**	**Growth rate**
32x32x32	2	6
16x16x16	3	12
8x8x8	3	24
4x4x4	3	48

For these three structures, dense blocks and dense with SE blocks were tested using identical training procedures. The results are shown in Table 5-3. For large organs, their performances are quite close. The dice coefficients of bones are still low. But for small organs, the networks with smaller growth rates and a bigger batch size perform best for both dense blocks and dense SE blocks.

Table 5-3. Performances of 3 dense structures with SE and without SE

class	Network1		Network2		Network3	
	Dense	**Dense SE**	**Dense**	**Dense SE**	**Dense**	**Dense SE**
BG	0.99	0.98	0.99	0.99	0.99	0.99
BO	0.003	0.005	0.016	0.008	0.06	0.01
PCL	0.79	0.27	0.65	0.76	0.76	**0.81**
ACL	0.56	0.08	0.62	0.49	**0.72**	**0.72**
MU	0.90	0.89	0.89	0.89	0.90	0.90
CB	0.47	0.39	0.36	0.45	**0.56**	0.51
BV	0.66	0.52	0.65	0.68	**0.79**	0.71
AD	0.84	0.83	0.84	0.84	0.84	0.83
TE	0.63	0.51	0.68	0.64	0.69	**0.71**
ME	0.81	0.74	0.81	0.79	0.81	**0.83**
Total	0.98	0.98	0.98	0.98	0.98	0.98

SE blocks' advantage is not that obvious compared with when it was used in residual blocks. The reason may be that dense blocks use 1×1×1 convolutions, which has already reduced the number of channels, and then limits SE blocks' advantage. Based on these results, the structure with smallest growth rate was chosen as the finial structure for both 3D U-net with dense blocks and 3D U-net with dense SE blocks to compare with 3D U-net with other components.

Final result

The Table below shows the performances of U-net and its variants. We can see that 3D U-net with residual SE blocks, dense blocks and dense SE blocks using weighted dice loss perform well on small organs segmentation. They obtained highest dice coefficients on different small organs but all lost accuracies on large organs especially on bones. V-net using dice loss achieved higher accuracies on large organs segmentation.

Table 5-4. Performances of U-net and its variants

Class	Dice Loss		Weighted Dice Loss				
	U-net	V-net	V-net	Residual SE	Dense	Dense SE	Merged results
BG	0.94	0.99	0.98	0.98	0.99	0.99	0.99
BO	**0.95**	**0.95**	0.06	0.14	0.06	0.01	0.97
PCL	0	0	0.63	0.77	0.76	**0.81**	0.81
ACL	0	0.01	0.4	0.65	0.72	**0.72**	0.7
MU	**0.95**	0.93	0.88	0.9	0.9	0.9	0.94
CB	0	0	0.41	**0.67**	0.56	0.51	0.51
BV	0	0	0.5	0.69	**0.79**	0.71	0.71
AD	0.92	**0.96**	0.82	0.85	0.84	0.83	0.96
TE	0	0.16	0.58	**0.73**	0.69	0.71	0.72
ME	0	0.01	0.78	0.79	0.81	**0.83**	0.83
Total	0.94	0.98	0.98	0.98	0.98	0.98	0.99

To make all organs have relatively high accuracies, the large organs (BO, MU, AD) segmented by V-net using dice loss and the small organs (PCL, ACL, CB, BV, TE, ME) segmented by U-net with dense SE blocks were eventually merged as the final result. The process is shown in Figure 5-11. As we discussed in Chapter 4.1.2, in Deeplabv3 [38], it discussed four architectures to capture multi-scale context. The method used in Figure 5-11 is actually similar with the first method described in DeepLabv3. The performance matrix of final result is shown in Figure 5-12. The values on the diagonal line show it achieves relatively high accuracies on all organs.

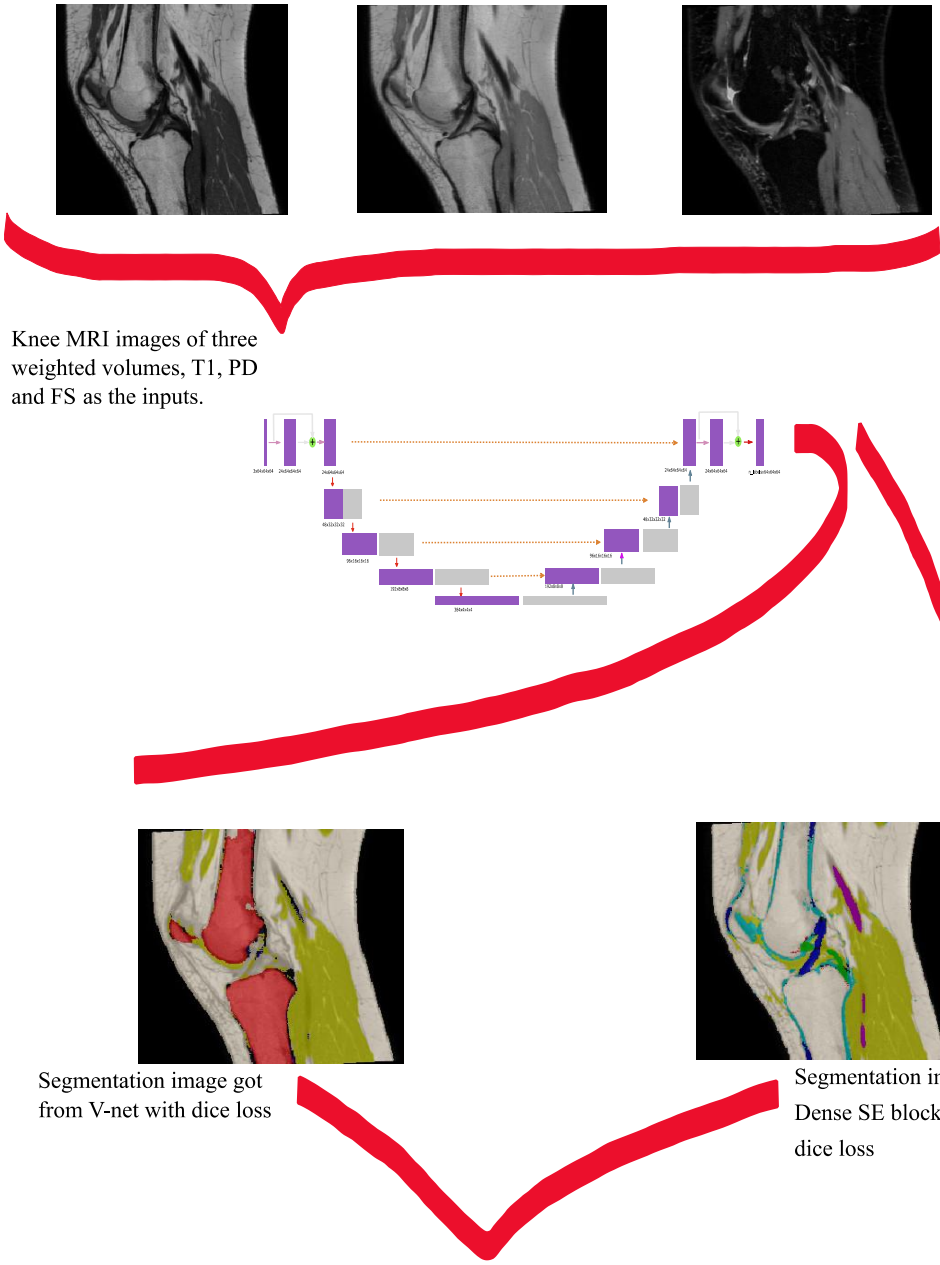


Figure 5-11. The process to get the final result

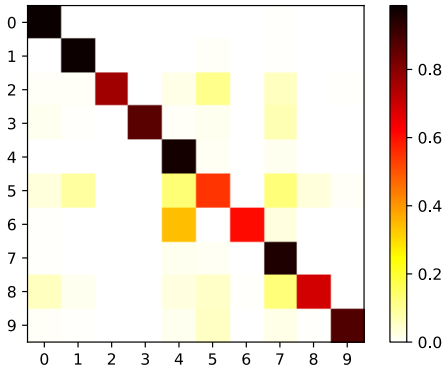


Figure 5-12. Performance matrix for the final result

5.2.2 Experiments on dataset2

There are 20 annotated knee MRI images whose size is 400×400×400 in dataset2. 19 images were used for training and one of them was used for validation. There are also three more labels in the dataset, so it is difficult to achieve same accuracy compared with the experiments on dataset1. The three architectures including 3D U-net variants, DeepLab variants and combined neural networks described in Chapter 4.1 were trained on this dataset. Different from experiments on dataset1, *Pytorch* was used as the deep learning framework in the experiments on dataset2.

For experiments on dataset1, the mean of total dice coefficient and the dice coefficient of each class is used as the metrics during training. For experiments on dataset2, total dice coefficient is deleted, and the mean of dice coefficient of each class is used as the metrics during training. It is also the evaluation score which will be shown in some of the figures in this chapter. Total dice coefficient will still be shown in the table of performance of neural networks.

5.2.2.1 Preprocessing

Downsampling

Because it takes a long time to finish the training on images whose size is 400×400×400. The images were resampled to a smaller size which is 256×256×256 in order to reduce the training time, and then to reduce the time spent on debugging. The

hyperparameters and neural networks were chosen based on experiments on the downsampling images and the finial segmentation results were got on 400×400×400 images. The table below shows the times of finishing one epoch of training using V-net on these two resolutions, where we can see that the time on the downsampling dataset was reduced by more than half.

Table 5-5. Training time on different resolutions

Resolution	Number of iteration	Time (hours)
400×400×400	1216	7
256×256×256	513	3

Cutting image into patches

Same as experiments on dataset1, these images are needed to be cut into patches. For most of neural networks, different patch size which is 128×128×128 was used on this dataset because the implementation in *Pytorch* saves memory in some extents, which enable us to use larger patch size. For DeepLab variants, different patch size that is larger was tried since the components used in the neural network have fewer parameters relatively.

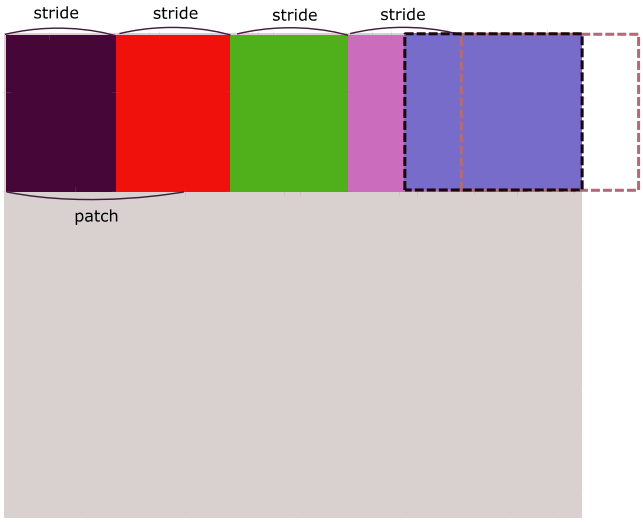


Figure 5-13. The method of cutting patches

Unlike the method of cutting patches in experiment1, stride was used to determine the interval between two patches. So there will be an overlapping between patches if the

stride is smaller than the patch size as shown in Figure 5-13. If the remaining part is not enough for a patch, the boundary of the patch will be moved into the image, which is shown as the patch at the top right corner.

Data format

HDF5 files were still used to store the data of input images. But another python package *h5py* instead of *PyTables* was used here. *H5py* is a python package which provides interface to the *HDF5* data format. The related pseudo-codes are shown below. It is easy to see that this implementation is simpler than using *PyTables*.

```
# load the image using nibabel
images = nib.load(nii_file_path)
# other kinds of preprocessing
images = resample(images)
# get data from the corresponding image
data = image.get_fdata()…
truth = image.get_fdata()…
affine = image.get_fdata()…
# return a hdf5 file handle using H5py
h5f = h5py.File(os.path.join(hdf5_path, folder+".h5"), 'w')
# copy the data into hdf5 file
data = image.get_fdata()
h5f['raw'] = data
h5f['label'] = truth
…
hdf5_file .close()
```

Generate ground truth of edges

Besides the preprocessing operations for dataset1, the edge ground truth for the combined neural network described in Chapter 4.1.3 was needed to be generated. It was used to calculate edge loss function. *Numpy.gradient* was used to calculate the gradients of voxels in the ground truth of segmentation to generate the edge ground truth, which is shown in Figure 5-14.

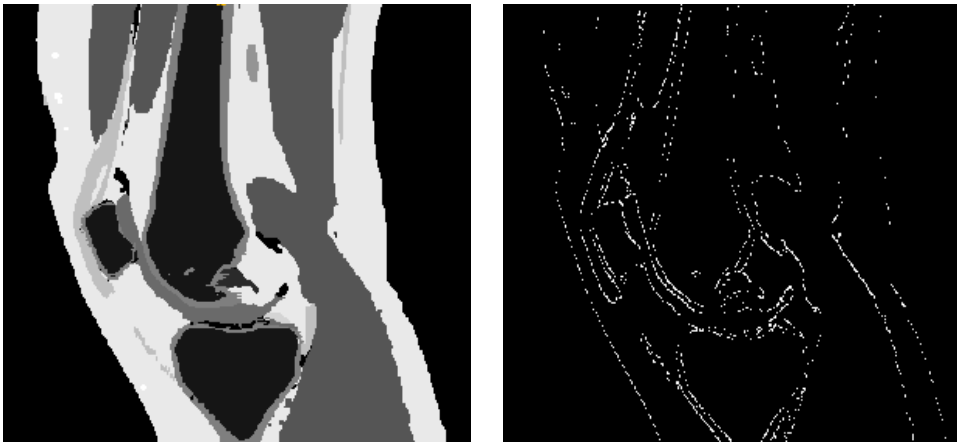


Figure 5-14. Generating edge ground truth

5.2.2.2 Training with downsampling images

For dataset2, all networks were trained using Adam optimizer. The initial learning rate was set to 0.01 for training with dice loss and was set to 0.001 for training with weighted dice loss, which is same as that on dataset1. The learning rates were set to be reduced by a factor of 0.5 after 2 epochs if the validation loss is not decreasing. For different neural networks, the numbers of epochs of training are different. The training was stopped when the loss on the validation dataset has not decreased for at least three epochs.

Larger Patch size vs. larger batch size

For dataset2, *Pytorch* was used as the deep learning framework. It enables us using a larger patch size compared with in *Keras*. To choose between a larger patch size with a smaller batch size and a smaller patch size with a larger batch size, 3D U-net with dense blocks in two situations were trained: with patch size 128×128×128 and batch size 1, and with patch size 64×64×64 and batch size 12. The convergence lines of these two trainings are shown in Figure 5-15, which we can see that the one with larger patch size and smaller batch size achieved higher score in less iteration. So, larger patch size has been chosen in the experiments.

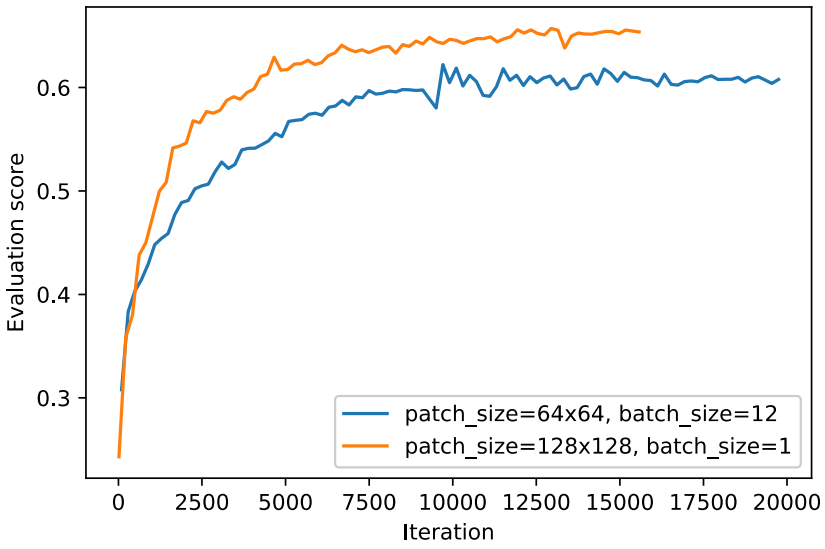


Figure 5-15. Larger Patch size vs. larger batch size

3D U-net variants

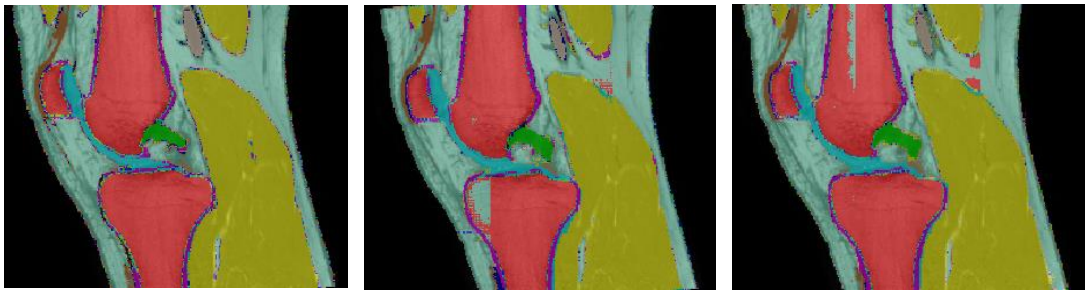
Same as the experiments on dataset1, V-net with dice loss and the four 3D U-net variants with weighted dice loss were tested. And the weight of each class was set according the frequency of the class in the dataset, the first method of setting weights described in Chapter 4.2.1. Instead of using patch size 64×64×64, the patch size was set to 128×128×128 and batch size was set to 1. Table 5-6 shows the results of all networks trained on the downsampling dataset.

From it, we can see that V-net with dice loss still performs poorly in small organs segmentation although more small organs are segmented compared with the same network on dataset1. With weighted dice loss, the accuracies of small organs segmentation are improved. But the accuracy of the class whose frequency is lowest, which is ACL, is still extremely low (the highest one is less than 0.1). Other classes have low accuracies are also those whose frequencies are small such as CL, ME and AR. In general, networks with SE blocks performed better on small organs segmentation compared with that without SE blocks, which is similar with what was observed in the experiments on dataset1. So, only networks with SE blocks were chosen to be trained on original images whose size is 400×400×400.

Table 5-6. The restults of 3D U-net variants on downsampling dataset

class	Dice loss	Weighted dice loss			
	V-net	V-net	Residual SE	Dense	Dense SE
BG	0.987	0.992	0.989	0.990	0.990
BO	0.917	0.831	0.938	0.932	**0.948**
PCL	0.002	0.153	**0.347**	0.338	0.228
ACL	0	0.083	**0.096**	0.084	0.041
MU	0.927	**0.944**	0.956	0.928	0.939
CB	0.567	0.500	0.457	0.495	**0.515**
BV	0.556	0.622	0.603	0.555	**0.648**
AR	0.057	0.198	**0.270**	0.152	0.194
CL	0.041	**0.470**	0.248	0.237	0.370
TE	0.726	0.701	0.723	0.641	**0.726**
ME	0.191	0.133	0.126	**0.139**	0.107
AD	0.919	0.876	0.919	0.904	**0.925**
VE	0	0.396	0.491	0.305	**0.528**
Total	0.979	0.987	0.983	0.984	0.985

Another notable result is the bones are segmented more completely than in the same network on dataset1, which can be seen in Figure 5-16. Although there are still some mistakes, but the improvement is considerably compared with that in Figure 5-9.



a) Ground truth b) Residual SE c) Dense SE

Figure 5-16. the segmentation restults on downsampling dataset

One explanation is that here a larger patch size which enables the networks to see more parts of the images was used. And the bones are inevitably cut into several parts since they are large and their positions are in the middle when patch-wise training is

used. It is difficult for the neural network to distinguish its pattern because of bones' special structure if they are scattered into different batches. When larger patch size is used, more parts of bones are in the same patch. So it can be segmented more correctly.

Combined network

For the combined neural network described in Chapter 4.1.3, patch size 128×128×128 was used. The initial learning rate was set to 0.01 and would be reduced by a factor of 0.5 if the evaluation score on validation dataset isn't improved in 2 epochs. The combined neural networks using 3D U-net with residual SE blocks as the segmentation network is named ResidualXUnet. The one using 3D U-net with dense SE blocks as the segmentation network is named DenseXUnet. The results of these two neural networks are shown in the table below, where we can see that the accuracies haven't been improved too much, even decreased, compared with the corresponding neural network without edge detection network.

Table 5-7. Training results of two combined network

class	Residual SE	ResidualXUnet	Dense SE	DenseXUnet
BG	0.989	0.991	0.990	0.991
BO	**0.938**	0.899	0.948	**0.957**
PCL	**0.347**	0.230	**0.228**	0.219
ACL	**0.096**	0.081	0.041	**0.106**
MU	**0.956**	0.942	0.939	0.937
CB	0.457	**0.518**	**0.515**	0.500
BV	0.603	**0.636**	0.648	**0.657**
AR	**0.270**	0.207	0.194	**0.215**
CL	0.248	**0.414**	**0.370**	0.307
TE	**0.723**	0.716	**0.726**	0.695
ME	0.126	**0.148**	0.107	**0.135**
AD	**0.919**	0.903	**0.925**	0.914
VE	0.491	**0.539**	**0.528**	0.503
Total	0.983	0.985	0.985	0.986

The reason can be found from Figure 5-17, which is the convergence line of the combined neural network using Residual SE blocks. Although the weight of dice loss and the weight of edge loss are set to 1 and 1000 respectively, we can still see that the optimization of dice loss was dominant during the training. The possible main reason is the limitation of the capacity of the edge detection network. However, if more attention has been paid on edge detection network, the efficiency of segmentation network may be affected since the computing resource is limited.

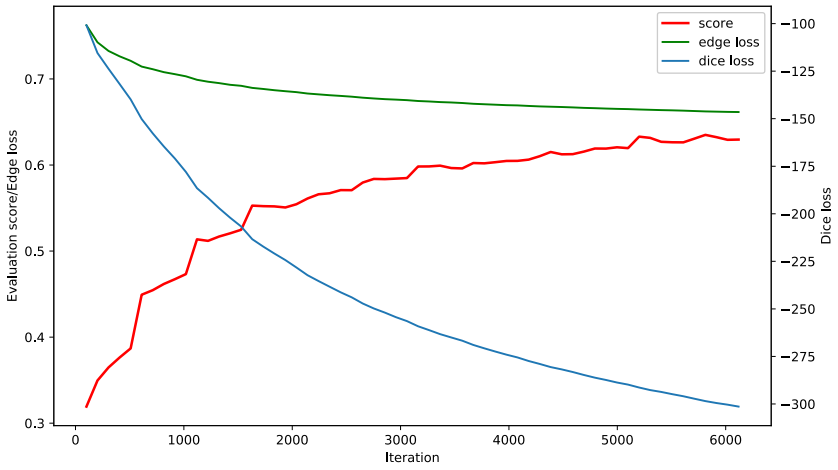


Figure 5-17. Convergence line of the combined neural network

Traning time: DeepLab variants vs. other networks

The patch size used for training DeepLab variants is 192×192×192, which is larger than other networks in the experiments. The DeepLab variants using the encoder of 3D U-net with dense SE blocks was tested first. The initial learning rate was set to 0.01, and would be reduced by a factor of 0.5 if the evaluation score on validation dataset wasn't improved in 3 epochs.

When training DeepLab variant on downsampling dataset, it achieved a relatively high accuracy in few epochs as shown in the figure below, where we can see that the evaluation score of DeepLab variant achieved 0.63 (the green line) far more quickly than 3D U-net with dense SE blocks, which has a similar encoder part with DeepLab variant here (similar but different, the details can be found in Chapter 4.1.2).

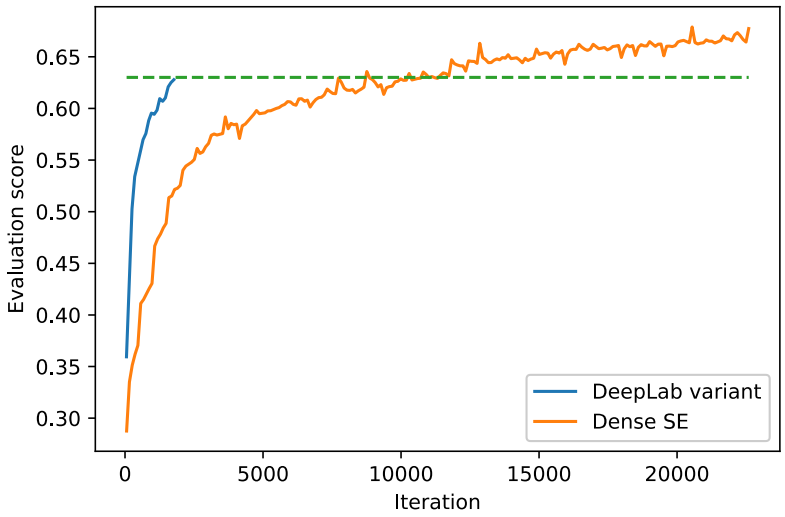


Figure 5-18. Convergence curve of DeepLab no downsampling dataset

In addition, the training speed of DeepLab variant was very fast. The table below shows the training time of an epoch of DeepLab variant (the encoder part used dense SE blocks), 3D U-net with dense SE blocks, 3D U-net with residual SE blocks and the corresponding combined neural networks (DenseXUnet and ResidualXUnet).

Table 5-8. Training time of an epoch on downsampling dataset

Network	Number of iteration	Time (hours)
DeepLab variant	152	0.14
3D U-net with Dense SE	513	0.5
DenseXUnet	513	0.5
3D U-net with Residual SE	513	3
ResidualXUnet	513	3.5

The most important reason is that DeepLab variant used larger patch size. One explanation that dense SE block was faster than residual SE is that residual block uses add to fuse features, but dense block uses concatenation, which is light computing burden compared with adding operation. Because the training speed is fast, most of experiments about DeepLab variant were conducted on the original dataset directly.

5.2.2.3 Training with original images

On this part, we will discuss more about the performance of DeepLab variants and how to improve it on the original images whose size is 400×400×400. Its performance will also be compared with other networks introduced in Chapter 4.1.

DeepLab variants

Most of experiments about DeepLab variants have been directly conducted on the original dataset. Two variants were trained including the one whose basic network uses Residual SE blocks (ResidualDeeplab) and the one whose basic network uses Dense SE blocks (DenseDeeplab). Both of them were trained with dice loss and weighted dice loss.

Table 5-9. Performance of DeepLab variants

class	Dice loss		Weighted dice loss			
			weight1		weight2	
	Dense Deeplab	Residual Deeplab	Dense Deeplab	Residual Deeplab	Dense Deeplab	Residual Deeplab
BG	0.986	0.983	0.984	0.984	0.977	0.984
BO	**0.958**	0.936	**0.926**	0.894	0.809	0.797
PCL	**0.752**	0.522	0.752	**0.807**	0.703	0.800
ACL	0.462	**0.504**	0.249	0.375	0.345	**0.457**
MU	0.969	**0.976**	0.959	**0.964**	0.926	0.950
CB	0.825	**0.861**	0.756	**0.800**	0.639	0.701
BV	**0.813**	0.781	0.748	**0.759**	0.553	0.627
AR	0.622	**0.671**	**0.622**	0.610	0.534	0.422
CL	**0.681**	0.589	0.596	**0.684**	0.449	0.536
TE	0.685	**0.745**	**0.751**	0.750	0.519	0.609
ME	**0.844**	0.819	0.797	**0.806**	0.706	0.794
AD	**0.927**	0.910	**0.894**	0.879	0.798	0.799
VE	**0.290**	0.265	0.359	**0.443**	0.227	0.344
Total	0.987	0.983	0.984	0.984	0.977	0.984

For the training with weighted dice loss, two methods described on Chapter 4.2.1 were used to set the weights. The results are shown in Table 5-9. Weight1 means using the final dice coefficients when the corresponding network with unweighted dice loss is converged. Weight2 means setting them according their frequencies. The initial learning rate was set to 0.001 for the training with dice loss and weighted dice loss using weight1, and 0.01 for the training with weighted dice loss using weight2 because it has relatively big weight for each class.

The segmentation accuracy of small organs is quite acceptable when used dice loss (See Figure 5-20 d) and f)), which is different from the performance of 3D U-net variants. DenseDeeplab with dice loss had higher accuracy on small organs such as PCL, BV and CL than ResidualDeeplab with dice loss. It achieved almost same, even higher accuracy compared with these networks with weighted dice loss. The figure below is the performance matrix of DenseDeepLab with dice loss, where we can see it achieved relatively high accuracies on all organs. ACL (3) has the smallest frequency among these classes, so the segmentation accuracy is low. However it is still higher than VE, a large part of which were predicted as AD. VE is not the smallest organ in the dataset, but it is the most difficult one to be segmented.

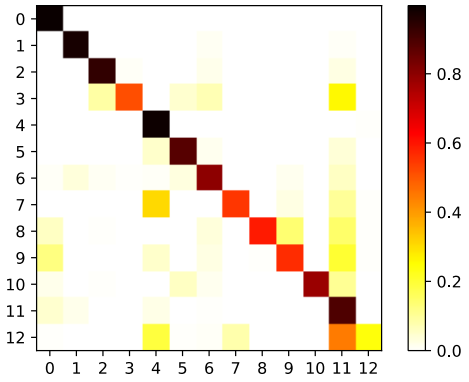


Figure 5-19. Performance matrix for the DenseDeeplab with dice loss

In terms of weighted dice loss, ResidualDeeplab in general performed better than DenseDeeplab, but not too much. The results also show that it is better to set the weight according the dice coefficient when the corresponding network with non-weighted dice loss converges (see Table 5-9). For 3D U-net variants, because the dice coefficients of the classes when the corresponding network using unweighted dice

loss is converged are almost same as their frequencies in the dataset, this situation cannot be observed. The increase of weights of small organs may cause the improvement of their accuracies, but it may also damage the performance on large organs. And the bottleneck of small organ segmentation is normally the neural network rather than the loss function.

From Figure 5-20 e), some of bones (BOs) were segmented incorrectly. From the experiments on downsampling dataset, we can learn that a larger patch size may increase the segmentation accuracy of bones. However, a larger patch size can't be used since the computing resource is limited.

Table 5-10. Performance of ResidualDeeplab with more data

class	Residual Deeplab	ResidualDeeplab (with more data)
BG	0.984	0.981
BO	0.797	**0.807**
PCL	0.800	**0.805**
ACL	0.457	**0.523**
MU	0.950	**0.955**
CB	**0.701**	0.624
BV	0.627	0.627
AR	0.422	**0.680**
CL	0.536	**0.586**
TE	**0.609**	0.591
ME	**0.794**	0.652
AD	0.799	0.799
VE	0.344	**0.347**
Total	0.984	0.982

In order to allow the neural network to have a larger view, the downsampling images whose size was 256×256×256 was added into the training dataset. It was hoped that the neural network could have a larger receptive field and then improve the segmentation accuracy of bones if it can learn from the downsampling images. For each downsampling image, there are 8 patches, while for each original image, there

are 27 patches. To balance the frequency of downsampling images and original images, it was made that the neural network learned the original image once when it learned the downsampling image three times. The segmentation results of ResidualDeeplab with more data is shown in Table 5-10, where we can see that the accuracy of bones was not improved too much, but the accuracies of other organs are improved especially for AR. However, the accuracies of some organs are decreased such as ME. So, it is not an efficient way to address the problem and also it took a longer time to train more data.

Other networks

Table 5-11. Performance of networks on original dataset2

class	Dice loss		Weighted dice loss	
	V-net	**ResidualDeeplab**	**ResidualSE**	**ResidualDeeplab**
BG	0.980	**0.983**	0.984	0.984
BO	0.866	**0.936**	0.694	**0.894**
PCL	0	**0.522**	0.664	**0.807**
ACL	0	**0.504**	**0.505**	0.375
MU	0.964	**0.976**	0.932	**0.964**
CB	0.740	**0.861**	0.761	**0.800**
BV	0.777	**0.781**	0.714	**0.759**
AR	0	**0.671**	**0.638**	0.610
CL	0.014	**0.589**	**0.751**	0.684
TE	0.642	**0.745**	0.712	**0.750**
ME	0.501	**0.819**	**0.909**	0.806
AD	0.854	**0.910**	0.807	**0.879**
VE	0.010	**0.265**	0.426	**0.443**
Total	0.981	0.983	0.985	0.984

Based on the results of experiments on the downsampling dataset, only 3D U-net with residual SE blocks and 3D U-net with dense SE blocks were chosen to train on original dataset for 3D U-net variants. To compare with the preliminary model, V-net (3D U-net variants with Residual blocks) with dice loss was also trained. The

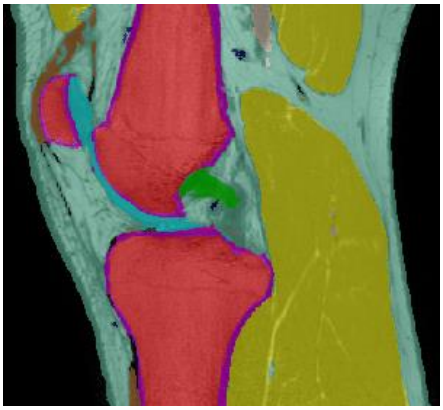
segmentation results of V-net and 3D U-net with residual SE blocks can be seen in Figure 5-20 b) and c).

Table 5-11 shows the performance of 3D U-net with Residual blocks (V-Net) using dice loss, 3D U-net with Residual SE blocks (ResidualSE) using weighted dice loss. The accuracies of small organs are improved compared with those on the downsampling dataset but the accuracies of large organs, especially bones, are decreased. The main reason is that the neural networks actually have a smaller view of the images although the patch sizes used are same. The patch-wise training actually decreases the receptive field of the images at the beginning. For the downsampling dataset, the view was reduced to $(\frac{128}{256})^3$=0.125. However, the view was reduced to $(\frac{128}{400})^3$≈0.033. So, the improvement couldn't be kept on the original images.

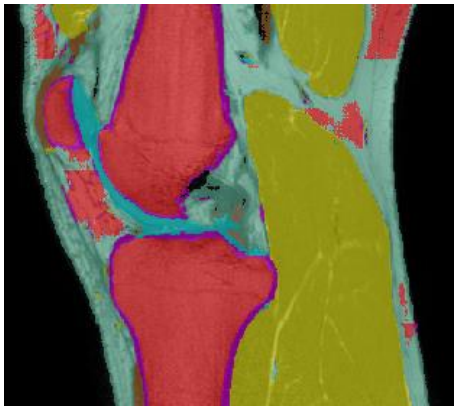
3D U-net with dense SE blocks performed worse than 3D U-net with residual SE blocks even though the patch size was increased to 144×144×144 to try to improve the accuracy. The result is shown neither in Table 5-11 nor in Figure 5-20. The possible reason is that dense SE blocks use fewer filters, so the representation capacity of the neural network may be limited when the resolution of the images is high.

Table 5-11 also shows the performance of ResidualDeeplab. We can see ResidualDeeplab in general performed better with both loss functions including dice loss and weighted dice loss even ResidualSE achieved higher accuracy on some small organs such as ACL and AR.

For the combined neural networks, their performances on the downsampling dataset are not same as expected. They were also trained on the original images. For the one whose segmentation neural network is Dense SE, the training time was acceptable, but the results are not good. For the one whose segmentation neural network is Residual SE, the training speed was too slow and the coverage rate also showed that it was not valuable to continue the training. So, more trials about them have been stopped.

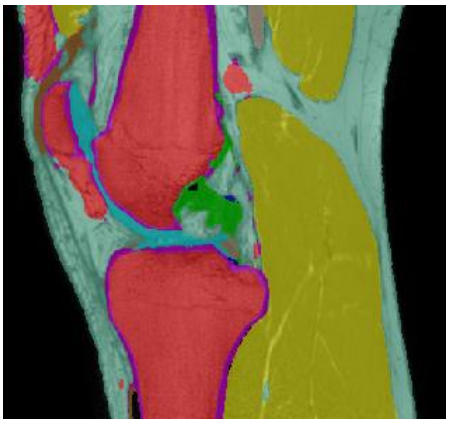


a)Ground truth



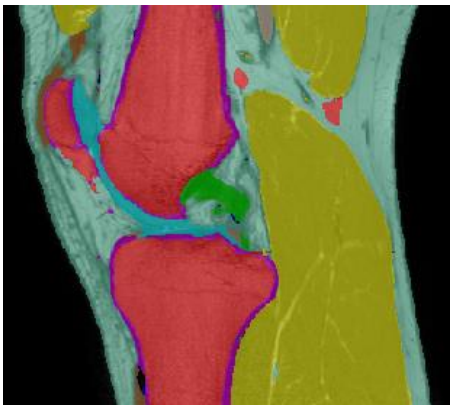
b)Vnet with dice loss

c)Residual SE with weighted dice loss



d)ResidualDeeplab with dice loss

e)ResidualDeeplab with weighted dice loss



f)DenseDeeplab with dice loss

Figure 5-20. Segmentation results of networks on dataset2

6. Discussions

In last chapter, we discussed lots of details about the performance of the three architectures described in Chapter 4, and the reasons why they performed in that way. In this chapter, conclusions will be made for the experiments and discussion in Chapter 5. In the sub-chapter Contribution, these conclusions will be expanded to a general condition. There are plenty of researches which are valuable to do in the domain of medical image segmentation. The future work described in the end is based on the work in this thesis, so it will not go too far.

6.1 Conclusions

Deep convolutional neural networks (CNN) have been used in automated medical image segmentation in recent years. It has decreased time and labor force involved. This work attempted to use annotated knee MRI images provided by Sunnmøre MR-Klinikk to train three types of neural networks including 3D U-net variants, DeepLab variants and combined neural networks in order to choose a system which achieves high accuracy on the segmentation task which has 13 extremely imbalanced classes.

For 3D U-net variants, three advanced components including residual blocks, squeeze-and-excitation (SE) blocks, and dense blocks were used to construct four variants of 3D U-net. The two variants of SE structure were used as the basic network for the other two architectures. For DeepLab variants, the number of filters used in the encoder (the basic network) was reduced by a half because Atrous Spatial Pyramid Pooling (ASPP) was used at the bottom of the encoder, and a small upsampling factor was used and one more deconvolution layers were added in the decoder for high accuracy requirement of medical image segmentation. CASENet was used as the reference of edge detection neural network, whose components were integrated into 3D U-net variants (the basic network) to form a type of combined neural network, which is the third architecture.

There are two datasets used in the experiments. The first dataset contains 10 labels in the ground truth. 3D U-net and its four variants with residual blocks, residual SE blocks, dense blocks and dense SE blocks were tested respectively on it. To reduce the training time, the size of these images was resampled from 400×400×275 to 274×274×274.

The experiments show the preliminary models including 3D U-net and V-net (3D U-net with residual blocks) couldn't segment the small organs correctly. However, with weighted dice loss, this situation has been changed. SE and dense structures have better performance on small organs segmentation. By using weighted dice loss, 3D U-net with residual SE blocks, dense blocks and dense SE blocks obtained highest dice coefficients on different small organs. V-net using dice loss achieved higher accuracies on large organs. The large organs segmented by V-net and the small organs

segmented by U-net with dense SE blocks were merged as the final result, which achieved relatively high accuracies on all organs.

The second dataset contains 13 labels and is more imbalanced in terms of the frequency of voxels of each class. The experiments are based on the results of the experiments on the first dataset. For the first dataset, only 3D U-net variants were tested on it. While all three types of neural networks were tested on the second dataset. Same as the experiments on the first dataset, these images were resampled from 400×400×400 to 256×256×256. The three types of neural networks were initially tested on the downsampling dataset in order to reduce the training time and debugging time. Based on the results, neural networks which have better performance were chosen to be trained on the original dataset.

The performance of 3D U-net variants on downsampling dataset is similar with those on the first dataset. However, when they were tested on the original dataset, the performance of dense structure was decreased. The possible reason is that the representation capacity of dense structure with fewer filters cannot meet the requirement of high resolution images.

Because the training speed and the convergence rate are fast for DeepLab variants on the downsampling dataset, the most of experiments about it were trained on the original images directly. It achieved relatively high accuracies on both small organs and large organs even with dice loss, which is quite different from 3D U-net variants that couldn't segment small organs correctly with dice loss. For the two methods of setting weights when weighted dice loss is used, the neural network performed better with weights set by the finial dice coefficients of the classes when the network with unweighted dice loss converged. The performance of combined network is not good as expectation because the capacity of edge detection network is limited.

6.2 Contributions

Various neural networks based on three architectures have been developed in this thesis including 3D U-net, which is a symmetrical encoder-decoder architecture, DeepLab, which is an asymmetrical encoder-decoder architecture with ASPP, and a

type of combined neural network, which takes advantages of different neural networks to attempt to improve the accuracy.

For 3D U-net architecture, four advanced blocks were used to replace the standard convolution blocks in 3D U-net including residual blocks, residual SE blocks, dense blocks and dense SE blocks The experiments show SE blocks can increase the accuracy of small organs through modeling the correlations between channels and adaptively strengthening important features, and dense blocks can obtain good results with relatively fewer filters, which may enable us to try a relatively larger batch size or patch size when the GPU memory is limited. The 3D U-net variants were used as the backbone network for the other two architectures in this thesis.

DeepLab architecture is the most efficient one among the three architectures. It can achieve a relatively high accuracy at high speed although 3D U-net may achieve higher accuracy on some of small organs. The main reason why it performs best is that it uses techniques such as ASPP, which enables the neural network to obtain a large receptive field and capture multi-scale context with fewer parameters.

The performance of combined network is not good as expectation because the capacity of edge detection network is limited, and it needs to be improved. However, the performance of segmentation network may be affected if more attention is paid on edge detection network because the computing resource is limited.

To design or choose a neural network for multi-label medical image segmentation, a large receptive field is needed. It enables the neural network to learn more features in the high level, and then to classify the organs correctly. In order to get a larger receptive field, downsampling operations such as convolution with stride, pooling are used, which cause resolution loss. Although fully convolutional neural network and other techniques used in DeepLab solve this problem in some extent, the contradiction between receptive field and resolution still exists especially when the computing resource is limited. When patch-wise training is chosen, the receptive field is reduced at the beginning. From the experiments in this thesis, the best strategy is to use an efficient neural network which can obtain a large receptive field and capture multi-scale context with fewer parameters. Because it uses less computing resource, a larger patch size can be tried, which means it increases the receptive field at the beginning

compared with the networks which use a smaller patch size. However, the resolution loss is still a problem. In this thesis, the segmentation accuracy of small organs is still needed to be improved.

6.3 Future work

The three architectures used in this thesis are three of the most commonly used architectures in medical image segmentation. Even the best one which is DeepLab still performed not well enough in small organ segmentation especially on VE. So more advanced components or neural network should be tried to improve the accuracy. For the combined neural network, because fewer efforts were put into the edge detection neural network search, it is possible that the performance of this architecture can be improved if a more advanced edge detection neural network is used.

The neural networks in this thesis were only tested on knee. They may have different performance on other datasets or organs. They should be tested on other public datasets to confirm the performance. Different datasets have different features, which enable us to learn more about the neural networks.

In this thesis, most of time has been spent to search the best combination of the layer and the appropriate hyperparameters. However, this job can also be done by using neural networks. V-NAS [61] formulated the structure learning as differentiable neural architecture search, and let the network itself choose between 2D, 3D or Pseudo-3D (P3D) convolutions at each layer. It can be tried next step.

References

[1] J. Sykes, “Reflections on the current status of commercial automated segmentation systems in clinical practice,” J. Med. Radiation Sci., vol. 61, no. 3, pp. 131–134, 2014.

[2] Geert Litjens, Thijs Kooi, Babak Ehteshami Bejnordi et al., “A survey on deep learning in medical image analysis,” J. Medical Image Analysis, Volume 42, 2017, pp. 89-101.

[3] URL: https://cs231n.github.io/neural-networks-1/

[4] Sumit Saha, “A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way,” 2018, URL: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[5] Sebastian Ruder, “An overview of gradient descent optimization algorithms,” 2016, URL: https://ruder.io/optimizing-gradient-descent/

[6] Andrew Ng, “Improving Deep Neural Networks: Hyperparameter tuning, Regularization and Optimization” URL: https://www.coursera.org/learn/deep-neural-network?specialization=deep-learning

[7] Diederik P. Kingma, Jimmy Ba, “Adam: a Method for Stochastic Optimization,” International Conference on Learning Representations, San Diego, 2015, pp. 1-13.

[8] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, Ping Tak Peter Tang, “On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima” (2016), arXiv:1609.04836 [cs.LG].

[9] Sagar Sharma, “Activation Functions in Neural Networks,” 2017, URL: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

[10] Arun Gandhi, “How to use Deep Learning when you have Limited Data,” 2018, URL: https://nanonets.com/blog/data-augmentation-how-to-use-deep-learning-when-you-have-limited-data-part-2/

[11] Jun-Yan Zhu, Taesung Park, Phillip Isola, Alexei A. Efros, “Unpaared Image-To-Image Translation Using Cycle-Consistent Adversarial Networks,” The IEEE International Conference on Computer Vision (ICCV), 2017, pp. 2223-2232.

[12] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting,” J. Mach. Learn. Res. 15, 1 (January 2014), 1929–1958.

[13] Amar Budhiraja, “Dropout in (Deep) Machine learning,” 2016, URL: https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5

[14] Sergey Ioffe, Christian Szegedy, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift,” (2015) arXiv:1502.03167 [cs.LG].

[15] Santurkar, Shibani and Tsipras, Dimitris and Ilyas, Andrew and Madry, Aleksander, “How Does Batch Normalization Help Optimization,” Advances in Neural Information Processing Systems 31, 2018, 2483-2493

[16] Larobina, M., Murino, “Medical Image File Formats,” J Digit Imaging 27, 200–206 (2014).

[17] Alex Krizhevsky and Sutskever, Ilya and Hinton, Geoffrey E, “ImageNet Classification with Deep Convolutional Neural Networks,” Advances in Neural Information Processing Systems 25, 2012, 1097–1105.

[18] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, “Gradient-based learning applied to document recognition,” in Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, Nov. 1998.

[19] Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, et al., “OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks, ” (2013) arXiv:1312.6229 [cs.CV] .

[20] Karen Simonyan, Andrew Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” (2014) arXiv:1409.1556 [cs.CV] , unpublished.

[21] Min Lin, Qiang Chen, Shuicheng Yan, “Network In Network,” (2013) arXiv:1312.4400 [cs.NE] , unpublished.

[22] Christian Szegedy, Ii Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, et al., “Going Deeper with Convolutions,” (2014) arXiv:1409.4842 [cs.CV], unpublished.

[23] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, “Deep Residual Learning for Image Recognition,” The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770-778.

[24] Syrya Remanan, “Beginner’s Guide to Object Detection Algorithms,” 2019. URL: https://medium.com/analytics-vidhya/beginners-guide-to-object-detection-algorithms-6620fb31c375

[25] Ross Girshick, Jeff Donahue, Trevor Darrell, Jitendra Malik, “Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation,” The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2014, pp. 580-587

[26] Ross Girshick, “Fast R-CNN,” The IEEE International Conference on Computer Vision (ICCV), 2015, pp. 1440-1448.

[27] Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks,” 2015, Advances in Neural Information Processing Systems 28, 91-99.

[28] Kaiming He, Georgia Gkioxari, Piotr Dollar, Ross Girshick; “Mask R-CNN,” The IEEE International Conference on Computer Vision (ICCV), 2017, pp. 2961-2969

[29] Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi, “You Only Look Once: Unified, Real-Time Object Detection,” The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 779-788

[30] Priya Dwivedi, “People Tracking using Deep Learning,” 2019, URL: https://towardsdatascience.com/people-tracking-using-deep-learning-5c90d43774be

[31] Jeremy Jordan, “An overview of semantic image segmentation,” 2018, URL: https://www.jeremyjordan.me/semantic-segmentation/

[32] Saurabh Pal, “Semantic Segmentation: Introduction to the Deep Learning Technique Behind Google Pixel’s Camera!” 2019, URL: https://www.analyticsvidhya.com/blog/2019/02/tutorial-semantic-segmentation-google-deeplab/

[33] Long, J., Shelhamer, E., Darrell, T., “Fully convolutional networks for semantic segmentation,” The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 3431-3440.

[34] Ronneberger, O., Fischer, P., Brox, T., “U-Net: convolutional networks for biomedical image segmentation,” In: Navab, N., Hornegger, J., Ills, W.M., Frangi, A.F. (eds.) MICCAI 2015. LNCS, vol. 9351, pp. 234–241. Springer, Heidelberg (2015).

[35] V. Badrinarayanan, A. Kendall and R. Cipolla, “SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 39, no. 12, pp. 2481-2495, 1 Dec. 2017.

[36] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, Alan L. Yuille, “Semantic Image Segmentation with Deep Convolutional Nets and Fully Connected CRFs (2014),” arXiv:1412.7062 [cs.CV], unpublished.

[37] L. Chen, G. Papandreou, I. Kokkinos, K. Murphy and A. L. Yuille, “DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 40, no. 4, pp. 834-848, 1 April 2018.

[38] Liang-Chieh Chen, George Papandreou, Florian Schroff, Hartwig Adam, “Rethinking Atrous Convolution for Semantic Image Segmentation” (2017), arXiv:1706.05587 [cs.CV], unpublished.

[39] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, Hartwig Adam, “Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation,” The European Conference on Computer Vision (ECCV), 2018, pp. 801-818

[40] P. Wang et al., "Understanding Convolution for Semantic Segmentation," 2018 IEEE Winter Conference on Applications of Computer Vision (WACV), Lake Tahoe, NV, 2018, pp. 1451-1460.

[41] Kamnitsas K. et al. “DeepMedic for Brain Tumor Segmentation” (2016), In: Crimi A., Menze B., Maier O., Reyes M., Winzeck S., Handels H. (eds) Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries. BrainLes 2016. Lecture Notes in Computer Science, vol 10154. Springer, Cham.

[42] C. Kaul, S. Manandhar and N. Pears, “Focusnet: An Attention-Based Fully Convolutional Network for Medical Image Segmentation,” 2019 IEEE 16th International Symposium on Biomedical Imaging (ISBI 2019), Venice, Italy, 2019, pp. 455-458.

[43] Ö. Çiçek, A. Abdulkadir, S. S. Lienkamp, T. Brox, and O. Ronneberger, “3D U-net: Learning dense volumetric segmentation from sparse annotation,” in Proc. MICCAI, 2016, pp. 424–432.

[44] F. Milletari, N. Navab and S. Ahmadi, “V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation,” 2016 Fourth International Conference on 3D Vision (3DV), Stanford, CA, 2016, pp. 565-571.

[45] E. Gibson *et al.*, “Automatic Multi-Organ Segmentation on Abdominal CT With Dense V-Networks,” in IEEE Transactions on Medical Imaging, vol. 37, no. 8, pp. 1822-1834, Aug. 2018.

[46] Gao Huang, Zhuang Liu, Laurens van der Maaten, Kilian Q. Iinberger; “Densely Connected Convolutional Networks,” The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 4700-4708.

[47] Fabian Isensee, Klaus H. Maier-Hein, “An attempt at beating the 3D U-Net” (2019), arXiv:1908.02182 [eess.IV], unpublished.

[48] Jie Hu, Li Shen, Gang Sun, “Squeeze-and-Excitation Networks,” The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018, pp. 7132-7141.

[49] Payer C., Štern D., Bischof H., Urschler M. (2018), “Multi-label Whole Heart Segmentation Using CNNs and Anatomical Label Configurations,” In: Pop M. et al. (eds) Statistical Atlases and Computational Models of the Heart. ACDC and MMWHS Challenges. STACOM 2017. Lecture Notes in Computer Science, vol 10663. Springer, Cham

[50] Yan Wang, Yuyin Zhou, Ii Shen, Seyoun Park, Elliot K. Fishman, Alan L. Yuille, “Abdominal multi-organ segmentation with organ-attention networks and statistical fusion,” Medical Image Analysis, vol 55, 2019, pp. 88-102, ISSN 1361-8415

[51] Intao Zhu, Yufang Huang etc., “AnatomyNet: Deep learning for fast and fully automated whole-volume segmentation of head and neck anatomy,” in Medical Physics, vol. 46, Issue 2, pp. 576-589, Feb 2019.

[52] Fisher Yu, Vladlen Koltun, “Multi-Scale Context Aggregation by Dilated Convolutions,” arXiv:1511.07122 [cs.CV], Published as a conference paper at ICLR 2016.

[53] Atul Pandey, “Depth-wise Convolution and Depth-wise Separable Convolution,” 2018, URL: https://medium.com/@zurister/depth-wise-convolution-and-depth-wise-separable-convolution-37346565d4ec

[54] K. He, X. Zhang, S. Ren and J. Sun, “Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition,” in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 37, no. 9, pp. 1904-1916, 1 Sept. 2015.

[55] Geirhos, R., Rubisch, P., Michaelis, C., Bethge, M., Wichmann, F.A., Brendel, W. “ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness,” International Conference on Learning Representations (ICLR) (2019)

[56] Hatamizadeh A., Terzopoulos D., Myronenko A. “End-to-End Boundary Aware Networks for Medical Image Segmentation,” Machine Learning in Medical Imaging. MLMI 2019. Lecture Notes in Computer Science, vol 11861. Springer, Cham.

[57] Bakas, S., et al. “Advancing the cancer genome atlas glioma MRI collections with expert segmentation labels and radiomic features,” Sci. Data 4, 170117 (2017).

[58] Zhang Z., Fu H., Dai H., Shen J., Pang Y., Shao L, “ET-Net: A Generic Edge-aTtention Guidance Network for Medical Image Segmentation,” Medical Image Computing and Computer Assisted Intervention – MICCAI 2019. MICCAI 2019. Lecture Notes in Computer Science, vol 11764. Springer, Cham.

[59] Zhiding Yu, Chen Feng, Ming-Yu Liu, Srikumar Ramalingam, “CASENet: Deep Category-Aware Semantic Edge Detection,” The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 5964-5973.

[60] Sudre C.H., Li W., Vercauteren T., Ourselin S., Jorge Cardoso M., “Generalised Dice Overlap as a Deep Learning Loss Function for Highly Unbalanced Segmentations” (2017), In: Cardoso M. et al. (eds) Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support. DLMIA 2017, ML-CDS 2017. Lecture Notes in Computer Science, vol 10553. Springer, Cham

[61] Z. Zhu, C. Liu, D. Yang, A. Yuille and D. Xu, “V-NAS: Neural Architecture Search for Volumetric Medical Image Segmentation,” 2019 International Conference on 3D Vision (3DV), Qu ébec City, QC, Canada, 2019, pp. 240-248, doi: 10.1109/3DV.2019.00035.