THE UNIVERSITY OF
NEWCASTLE
AUSTRALIA

Convolutional Neural Networks for Image Segmentation in Clinical Applications

Indriani Puspitasari Astono

BEng (Hons) (Elec.)

A thesis submitted in fulfilment of the requirements for the degree of

Doctor of Philosophy

in Electrical Engineering

August 2021

This research was supported by an Australian Government Research Training Program (RTP) Scholarship

Statement of Originality

I hereby certify that the work embodied in this thesis is my own work, conducted under normal supervision. The thesis contains no material which has been accepted, or is being examined, for the award of any other degree or diploma in any university or other tertiary institution and, to the best of my knowledge and belief, contains no material previously published or written by another person, except where due reference has been made. I give consent to the final version of my thesis being made available worldwide when deposited in the University's Digital Repository, subject to the provisions of the Copyright Act 1968.

Indriani Puspitasari Astono

BEng (Hons) (Elec.)

August, 2021

Acknowledgements

First of all, I want to express my sincerest gratitude to my principal supervisor, A/Prof. James Welsh, for providing me with this opportunity. Thank you for all the guidance, support and patience throughout this journey. I would also like to thank my first co-supervisor, Dr. Stephan Chalup, for some invaluable advices and guidance in the early stage of my PhD. I would like to thank my second co-supervisor, Dr. Peter Greer, for the support and guidance that made the publication of the second manuscript possible. I would also like to thank Dr. Phillip Jobling and Dr. Christopher Rowe for giving me the opportunity to work with an incredibly interesting problem. Thank you for sharing your enthusiasm and knowledge with me. Thank you for your time and efforts that made the submission of the last manuscript possible.

I would also like to thank my senior, Duc Ngoc Anh Tran, for the invaluable advices and guidance in the first year of my PhD. You inspired me to complete this journey. I would also like to thank my PhD colleagues for the support and encouragement throughout this journey, especially Shashank Gupta, Fah, Siqi Pan, Jeremy Stoddard, Neda Gholizadeh and Samoda Gamage.

A special thank you to all my friends and family for the constant supports and encouragements throughout this journey. Thank you dad, brother and sister for sending me all the way to Newcastle and making sure that I was safe. Thank you Feby, Sharon, Ramizah, Annabelle, Serene and Danny for the endless long distance friendship. To all my badminton friends at Hunter Badminton, thank you for making me feel like home. I really appreciate the care and love that you all have given me. Special thanks to Prema and Geoff for the enormous support at the last bit of this

journey. Also, to my former English teacher, Ms. Juju, thank you for being there to answer my last minute questions.

Lastly, I want to thank Gian E. Triputra for the remarkable digital illustrations in this thesis. Thank you for making the artworks look so presentable. Most importantly, thank you for the tremendous supports and patience throughout this journey. Thank you for being there for me.

Without all of you, this would not have been possible. Thank you.

To my mother,

Suarni Astono

Contents

Abstract

The convolutional neural network (CNN) has been remarkably successful in performing automatic image segmentation in a number of clinical applications. It is able to decrease the time taken for the segmentation process as well as minimise the errors with respect to manual segmentation by a human operator. However, most of the developed CNN architectures have redundant components and trainable parameters. These redundancies cause the implementation of a CNN to be expensive in terms of time and memory usage. In this thesis, we study several CNN architectures in terms of their structures, components and number of trainable parameters to gain a deeper insight into the requirements of a CNN to achieve a state-of-the-art performance on a clinical segmentation task. As a result, we developed a CNN with a novel adjacent upsampling method that achieves a state-of-the-art performance for an organ segmentation task while being much smaller in terms of the number of trainable parameters and computation time. We developed a CNN with an optimised architecture that outperforms other state-of-the-art CNNs with similar components on an organ segmentation task. Furthermore, we developed a novel augmented classification structure to improve the performance of a segmentation network for an object detection task. We also demonstrate the implementation of a CNN on a complex digital pathology segmentation problem with the use of multiple considerations. We show that with the appropriate CNN architecture and implementation, an effective and efficient CNN based approach can be developed to assist medical experts in different segmentation problems.

CHAPTER

1

Introduction

1.1 Background

A convolutional neural network (CNN) can be considered as a type of deep learning algorithm that specialises in, but is not limited to, image processing tasks. Like other deep learning algorithms, a CNN has the ability to recognise patterns and learn from a set of data samples to make sensible predictions for new data samples [1]. The CNN has an architecture consisting of various processing layers and transformations that enables it to represent a highly complex system [2] [3]. It has the ability to automatically learn hierarchical features of an input to perform a certain task [2]. The CNN is used in various image processing applications, including image segmentation.

Image segmentation is a process of partitioning an image into a number of segments to make the image more meaningful and easier to analyse [4]. It is a critical process in the medical imaging field, as accurate organ or lesion segmentation is crucial in, e.g. computer-aided diagnosis (CAD), medical image analysis, image-guided therapy and computer-integrated surgery [4] [5]. Many computer-aided segmentation methods have been developed to improve the accuracy and efficiency of conventional user-guided segmentation methods on different types of medical images, e.g. Magnetic Resonance Imaging (MRI), Computerised Tomography (CT), X-ray and ultrasound [5] [6] [7] [8].

Segmentation methods can be classified into three main categories: manual, semiautomatic and automatic [9] [10]. Manual segmentation relies on an expert, or multiple experts, to perform segmentation based on their expertise and experiences. However, manual segmentation is very labour intensive and is not always reliable, as every segmentation is subject to intra-and inter-expert variability [11]. Semiautomatic segmentation uses algorithms to provide assistance for the expert in order to reduce the time and effort required to perform a segmentation. However, it is still subject to intra-and inter-expert variability as it relies on some amount of manual input from an expert. On the other hand, automatic segmentation uses algorithms to perform segmentation without any manual input. Automatic segmentation overcomes the necessity for manual labour and minimises

the problem with intra-and inter-expert variability, which leads to higher consistency and reproducibility.

Automatic segmentation approaches can be categorised into either traditional or machine learning based approaches. The traditional approaches are based on thresholding, statistical analysis, deformable models and graphs [4]. There are two main machine learning approaches: classical machine learning, based on hand-crafted features, and deep neural networks, based on automatic feature-extraction methods. Machine learning based approaches have been proven to be superior to traditional approaches in numerous automatic image segmentation applications, as machine learning algorithms have the capability to detect patterns and model data automatically [12] [13]. Unlike the classical machine learning algorithms that rely on hand-crafted features as their input, deep neural network algorithms have a feature-learning capability that allows the system to extract features directly from an input image [14]. This minimises the need for prior knowledge when developing a model for a certain task [14]. The CNN has been outperforming traditional computer vision and machine learning algorithms comprehensively in terms of accuracy and precision [15] [16]. It is currently the state-of-the-art deep learning algorithm in image segmentation for many different applications, such as organ and lesion segmentation [12] [13].

There are a number of considerations to be made when developing a CNN model [12]. CNN models incorporate a large number of variables and components in their network architecture, such as structure, number of layers, number of filters, upsampling components, downsampling components, regularisation components, normalisation components and activation function type. The effectiveness of a CNN model is also influenced by a number of implementation considerations, such as the training hyperparameters and algorithms, data pre- and post-processing, as well as the evaluation method.

The main challenge of performing image segmentation using CNNs is in determining the appropriate components for the network to overcome the CNNs' inherent spatial invariance characteristic effectively. The state-of-the-art segmentation networks, such as FCN [17] and U-Net [18], suffer from this

inherent spatial invariance characteristic due to the naïve adaptation of the network architectures that were designed for image classification tasks [19]. This results in these CNNs using a large number of trainable parameters to gain superiority in their performance [17] [18].

For the application of a CNN for image segmentation, the inefficient design of the architecture contributes to the requirement of a large number of training data samples to develop a high-performing model [20] [10] [21] [13]. However, a labelled training dataset in a given clinical application is often limited in quantity as it is expensive and difficult to produce [20] [10] [13]. This limits the application of a CNN based approach in clinical application.

In this research, we evaluate the performance of a number of CNNs to determine the most suitable components for a CNN to overcome the inherent spatial invariance characteristic in an image segmentation task. We establish the importance of the appropriate structure, number of trainable parameters and components used in a CNN architecture for the network to perform efficient image segmentation in clinical applications. We also describe several implementation considerations to improve the effectiveness in the development of a CNN based approach.

Datasets from liver, prostate and nerve segmentation problems are used in this research. Each of these problems has its own challenges. For the liver segmentation problem, the challenges include the variation of the image quality due to it being acquired from multiple medical centres, as well as variation in the size and shape of the liver. For the prostate segmentation problem, some additional challenges are introduced due to the fuzzy boundaries of the prostate and high variance of pixel intensities in the image. For the nerve segmentation problem, the main challenge is due to the enormous size of the images, which are approximately 6400 times larger than typical CT or MRI images. As a result, the data lacks pixel-wise annotations which makes it challenging for the development of a CNN based approach for the segmentation. The nerves also vary largely in size and appearance.

1.2 Research problem and motivation

The convolutional neural network has been remarkably successful in performing medical image segmentation tasks [12] [13]. However, most studies focus on the results and fail to notice inefficiencies in several areas:

1. *Architecture.* To produce accurate segmentation results, some studies developed CNN architectures with very complex structures. These complex structures generally consist of cascading/stacking of several networks and are often developed to achieve a small performance gain (sometimes less than 1%) [22] [23] [24] [25], which could also be achieved by simpler structures. This typically results in the practice of using a CNN architecture with a complex structure instead of a CNN architecture that can be applied more efficiently for a certain task.

2. *Number of trainable parameters.* Generally, in the adoption of a CNN architecture, the number of trainable parameters used is often overlooked. Many adopt the architecture blindly, including the number of trainable parameters, without understanding what is actually required. This results in a CNN with an unnecessarily large number of trainable parameters for a given application that could be solved with a smaller set of trainable parameters to achieve the same performance [26] [17] [27] [28].

3. *Components.* Many studies often suggest the superiority of a new network component based on the performance comparison of their networks to other networks with completely different architectures [29] [17] [18] [30]. As a result, there is limited understanding of the contribution of each component to the overall performance of a given network. This leads to unnecessary integration of new components in many CNN architectures and results in the CNN becoming expensive, in terms of computation time and memory usage.

4. *Implementation considerations.* A clinical dataset often suffers from limited data quantity, high intra-class data variance and/or variation in the

annotations provided by experts. However, a standard data processing method is often adopted without considering its suitability for the problem [12] [31] [32] [33], causing the CNN based approach to be ineffective.

1.3 Thesis outline and contributions

In this thesis, we will address the inefficiencies outlined in Section 1.2. The thesis consists of five main chapters with fundamental knowledge established in the first two chapters, with the next three chapters addressing the four inefficiencies. An outline of the thesis now follows.

Chapter 2 provides some fundamental knowledge about neural networks, including the CNN. The fundamentals of a CNN architecture in terms of structure, number of trainable parameters and components are discussed. We also provide an overview of state-of-the-art CNNs that are specialised in different tasks.

In chapter 3 we discuss a number of fundamental considerations that need to be taken into account in the implementation of a deep learning algorithm, in particular a CNN. We include an overview of various hyperparameters and algorithms, data pre- and post-processing techniques and evaluation methods.

In chapter 4 we discuss limitations of typical segmentation network architectures, in terms of structure, and propose an alternative network that performs comparably well while being much more efficient. The proposed CNN incorporates a novel adjacent upsampling method, which is a trainable upsampling method and only requires a single computation step. We show that a CNN can be structured more efficiently without the use of an excessive number of trainable parameters to solve a particular task.

In chapter 5 we investigate the effects of each individual component of a U-Net architecture with the aim of providing a better understanding of the contribution each component makes to the performance of the network in terms of Dice's score. We present the performance comparisons of several U-Nets with different component choices in their architectures. The results in this chapter provide an efficient U-Net that outperforms traditional methods on a private dataset as well as other CNNs

with similar structure and components on a public dataset.

In chapter 6 we develop an automated object detection approach for a complex digital pathology quantification problem. We provide a performance comparison between a CNN based approach and existing manual and automated quantification methods. A novel augmented classification structure is proposed for a U-Net to perform an object detection task. We also discuss some data pre-processing considerations for the training of the network to address common critical challenges in developing a CNN based approach with clinical data.

Finally, the conclusion of this research is provided in Chapter 7, which includes the summary of the results and suggestions for future directions.

1.4 Publication list

The following publications are a direct result of the research conducted in this thesis.

- I. Astono, J.S. Welsh, and S. Chalup. Adjacent Network for Semantic Segmentation of Liver CT Scans. *2018 IEEE 18th International Conference on Bioinformatics and Bioengineering (BIBE)*, pages 35-40, Taichung, Taiwan, 2018. (Related to the work presented in Chapter 4)

- I.P. Astono, J.S. Welsh, S. Chalup, and P. Greer. Optimisation of 2D U-Net Model Components for Automatic Prostate Segmentation on MRI. *Applied Science*, 10(7): 2601, 2020. (Related to the work presented in Chapter 5)

- I. Astono, C.W. Rowe, J. Welsh, and P. Jobling. MON-535 Deep-Machine Learning for Objective Quantification of Nerves in Immunohistochemistry Specimens of Thyroid Cancer. *Journal of the Endocrine Society*, 4(Supplement_1), 2020. (Related to the work presented in Chapter 6)

- I.P. Astono, J.S. Welsh, C.W. Rowe, and P. Jobling. Objective Quantification of Nerves in Immunohistochemistry Specimens of Thyroid Cancer utilising Deep Learning. *PLOS Computational Biology*, Under Review, 2021.
(Related to the work presented in Chapter 6)

CHAPTER

2

Introduction to Neural Networks

Neural networks are a subset of machine learning algorithms, inspired by neuronal networks in the biological brain. In this chapter, we discuss the fundamentals of the neural network, including its components, structure and learning mechanism. An in-depth overview of the CNN, a type of neural network specialised for high-dimensional data, as well as a brief review of several state-of-the-art CNN architectures is provided. A summary of CNN architectures in terms of their structure, number of trainable parameters and components is also presented.

2.1 Introduction

In this chapter we introduce the fundamental knowledge of neural networks. We discuss the basic elements of a neural network as well as its basic configuration and learning mechanism. We also include a brief introduction to the initial forms of neural network, i.e. the perceptron and multilayer perceptron. Then, we introduce the CNN, which is a type of neural network that is specialised for high-dimensional data, and discuss its structure, trainable parameters and components. We also provide a brief overview of several state-of-the-art CNN architectures for image classification, object detection and image segmentation tasks. Finally, a summary of CNN architectures in terms of their structure, number of trainable parameters and components is provided at the end of this chapter.

2.2 Neural Network Fundamentals

A neural network, sometimes referred to as an artificial neural network, is a system that is made up of artificial neurons that are designed to model the way in which the human brain works, such as learning specific knowledge and storing it in order to perform a certain task, e.g. pattern recognition [34].

The artificial neuron is modelled with three basic elements: a set of synaptic weights, $\mathbf{w}$, an adder, $\sum$, and an activation function, $\varphi(\cdot)$, as shown in Figure 2.1, where n is the dimension of the input, $x_1, \ldots, x_n$ are the input data, $w_{k1}, \ldots, w_{kn}$ are the synaptic weights of neuron k with respect to the input data, z_k is the activation potential of neuron k, b_k is the bias of neuron k, $\varphi(\cdot)$ is the activation function, and a_k is the output response of neuron k.

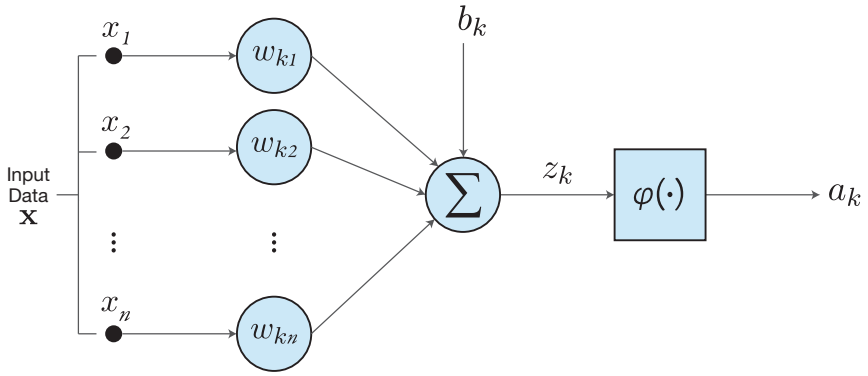


Figure 2.1: Model of an artificial neuron.

In mathematical terms, the output of a neuron can be described by,

$$z_k = \sum_{j=1}^{n} w_{kj} x_j + b_k, \tag{2.1}$$

and

$$a_k = \varphi(z_k). \tag{2.2}$$

This can be written more generally as,

$$h_k(\mathbf{x}) = \varphi\left(\left(\sum_{j=1}^{n} w_{kj} x_j\right) + b_k\right). \tag{2.3}$$

The activation function, $\varphi(\cdot)$, is used to define a neuron output in terms of the activation potential, z [34]. It determines the activation state and output value of the neuron. Several types of common activation functions [34] are

- Threshold function: Produces a value of 0 if the activation potential of the neuron is negative, and 1 if it is positive. It is defined as,

$$\varphi(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0. \end{cases} \tag{2.4}$$

- Piecewise-Linear function: Produces a linear output when the activation potential of the neuron is within a specific region, otherwise a saturation value

of either 0 or 1 will be produced. It is defined as,

$$\varphi(z) = \begin{cases} 1, & z \geq +\frac{1}{2} \\ z, & +\frac{1}{2} > z > -\frac{1}{2} \\ 0, & z \leq -\frac{1}{2}. \end{cases} \tag{2.5}$$

- Sigmoid function: A differentiable s-shaped function that produces a continuous range of values from 0 to 1. It is defined as,

$$\varphi(z) = \frac{1}{1 + e^{-z}}. \tag{2.6}$$

A neural network is made up of a number of artificial neurons that form a structure that can be used as a powerful computing tool to solve complex problems. It has the ability to generalise, i.e. make sensible predictions for new input data of the same category (class) as the learned data that has never been seen by the network.

Training a neural network requires a randomised set of labelled data. It learns by adjusting its synaptic weights in order to produce the desired responses that correspond to each unique input data. The main training objective is to minimise the error between the response of the network and the label associated with each input data. A cost function is formed based on some measure of the error for each output.

The cost function depends on the task, i.e. whether it is for regression, classification or other purposes. For example, a cost function that can be used for regression is the mean squared error function [35] given as,

$$J(\mathbf{w}, \mathbf{b}) = \frac{1}{2m} \sum_{i=1}^{m} \sum_{k=1}^{K} \left(h_k \left(\mathbf{x}^{(i)} \right) - y_k^{(i)} \right)^2, \tag{2.7}$$

where $J(\mathbf{w}, \mathbf{b})$ denotes the cost, $\mathbf{w}$ denotes the synaptic weights and $\mathbf{b}$ denotes the biases, K denotes the number of neurons, m denotes the number of data samples, $\mathbf{x}^{(i)}$ denotes the i^{th} data sample from a dataset, $h_k \left(\mathbf{x}^{(i)} \right)$ denotes the network output response to data $\mathbf{x}^{(i)}$ at neuron k, $\mathbf{y}^{(i)}$ denotes the label of data $\mathbf{x}^{(i)}$.

A cost function typically used for classification tasks is the cross-entropy function [36] given by,

$$J(\mathbf{w}, \mathbf{b}) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log\left(h_k\left(\mathbf{x}^{(i)}\right)\right) + \left(1 - y_k^{(i)}\right) \log\left(1 - h_k\left(\mathbf{x}^{(i)}\right)\right)\right]. \tag{2.8}$$

A network is trained by using an optimisation algorithm to solve $\arg\min_{\mathbf{w},\mathbf{b}} J(\mathbf{w}, \mathbf{b})$.

2.2.1 Perceptron

A perceptron is the simplest form of a neural network that consists of a single artificial neuron with a set of adjustable synaptic weights and bias as shown in Figure 2.1. The perceptron is trained by adjusting the synaptic weights and bias to fit the training data samples so that it can produce the desired responses [34].

According to the perceptron convergence theorem [37], a perceptron is guaranteed to find an optimum solution in the form of a hyperplane that acts as a decision boundary in a two class pattern-classification problem, as illustrated in Figure 2.2.

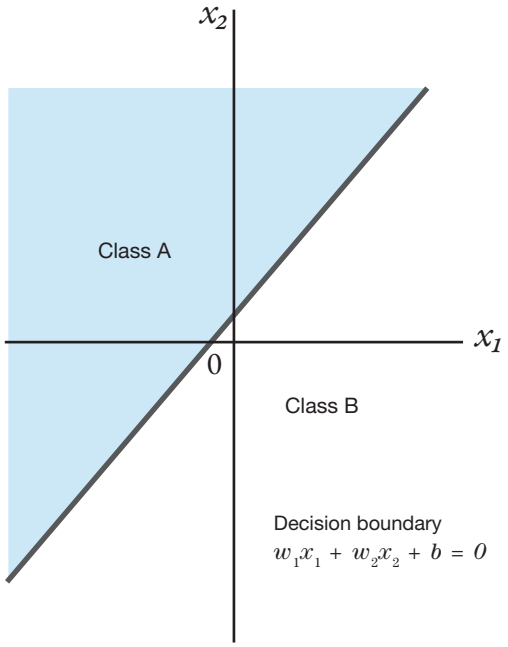


Figure 2.2: A perceptron as a decision boundary for two-dimensional (2D) linearly separable data.

The equation of this hyperplane is given by Eq. 2.1 with $k = 1$. The output response of the perceptron [34] is given by,

$$h(\mathbf{x}) = \text{sgn}(z) = \begin{cases} +1 \text{ if } z > 0 \\ -1 \text{ if } z < 0. \end{cases} \tag{2.9}$$

To expand the network to form a classifier of more than two classes, additional perceptrons are required. Note that sgn$(\cdot)$ is an activation function known as the signum function.

2.2.2 Multilayer Perceptron

A multilayer perceptron (MLP) is a type of neural network that is constructed using multiple perceptrons forming a multi-layer network. It is composed [34] of an input layer, one (or more) hidden layers and an output layer as shown in Figure 2.3.

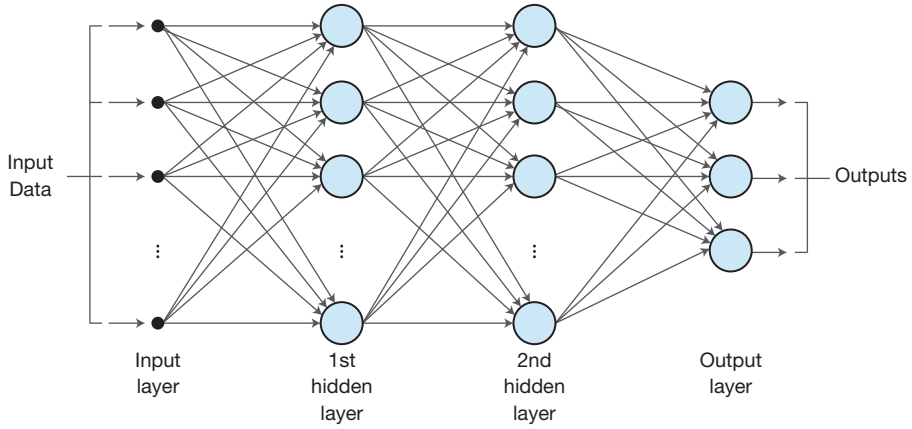


Figure 2.3: Example of a multilayer perceptron with two hidden layers.

The training of an MLP is based on an error correction learning rule known as the error backpropagation algorithm. This algorithm consists of a forward pass and a backward pass.

In the forward pass, the input data is fed into the input layer, followed by a forward propagation through the hidden layers, with fixed synaptic weights, to the output layer to produce a set of outputs [34]. This propagation is illustrated in the two-hidden layer MLP example shown in Figure 2.3 with a single data sample. Let the index of each layer be denoted with a superscript and the index of a neuron in

a given layer be denoted with a subscript. Then,

$$\boldsymbol{a}^{(0)} = a_1^{(0)}, \ldots, a_n^{(0)} = x_1, \ldots, x_n, \tag{2.10}$$

$$\mathbf{z}^{(1)} = z_1^{(1)}, \ldots, z_j^{(1)} = \left(\sum_{i=1}^{n} w_{1i}^{(1)} a_i^{(0)} \right) + b_1^{(1)}, \ldots, \left(\sum_{i=1}^{n} w_{ji}^{(1)} a_i^{(0)} \right) + b_j^{(1)}, \tag{2.11}$$

$$\boldsymbol{a}^{(1)} = a_1^{(1)}, \ldots, a_j^{(1)} = \varphi\left(z_1^{(1)}\right), \ldots, \varphi\left(z_j^{(1)}\right), \tag{2.12}$$

$$\mathbf{z}^{(2)} = z_1^{(2)}, \ldots, z_k^{(2)} = \left(\sum_{i=1}^{j} w_{1i}^{(2)} a_i^{(1)} \right) + b_1^{(2)}, \ldots, \left(\sum_{i=1}^{j} w_{ki}^{(2)} a_i^{(1)} \right) + b_k^{(2)}, \tag{2.13}$$

$$\boldsymbol{a}^{(2)} = a_1^{(2)}, \ldots, a_k^{(2)} = \varphi\left(z_1^{(2)}\right), \ldots, \varphi\left(z_k^{(2)}\right), \tag{2.14}$$

$$\mathbf{z}^{(3)} = z_1^{(3)}, \ldots, z_l^{(3)} = \left(\sum_{i=1}^{k} w_{1i}^{(3)} a_i^{(2)} \right) + b_1^{(3)}, \ldots, \left(\sum_{i=1}^{k} w_{li}^{(3)} a_i^{(2)} \right) + b_l^{(3)}, \tag{2.15}$$

$$\boldsymbol{a}^{(3)} = \mathbf{h}(\mathbf{x}) = \varphi\left(z_1^{(3)}\right), \ldots, \varphi\left(z_l^{(3)}\right), \tag{2.16}$$

where n is the the dimension of the input, j is the number of hidden neurons in the first layer, k is the number of hidden neurons in the second layer, l is the number of output units in the third layer (or the output layer), $x_1, \ldots, x_n$ are the input data from a single data sample, w is the synaptic weight of a neuron, z is the activation potential of a neuron, b is the bias of a neuron, $\varphi(\cdot)$ is the activation function, a is the output of a neuron and $\mathbf{h}(\mathbf{x})$ is the output response of the network to an input data sample, $\mathbf{x}$.

In the backward pass, the learning occurs when the synaptic weights are adjusted to minimise the error. It begins with the computation of error of the actual output

responses of the network, from the forward pass, with the labels of the corresponding input samples. This error is then backward propagated layer by layer through the network until it reaches the input layer. The neural network structure in Figure 2.3 and the corresponding notation will be used to depict this process.

In the instance where the neural network is used for a classification problem with a cross-entropy cost function (Eq. 2.8) and a sigmoid activation function applied to the output layer units, the error of the last layer, $\boldsymbol{\delta}$, can be calculated as follows, for layer three in this particular example (see Figure 2.3),

$$\boldsymbol{\delta}^{(3)} = \frac{\partial J(\mathbf{w}, \mathbf{b})}{\partial \boldsymbol{a}^{(3)}} \frac{\partial \boldsymbol{a}^{(3)}}{\partial \mathbf{z}^{(3)}}, \tag{2.17}$$

where $\boldsymbol{a}^{(3)} = \mathbf{h}(\mathbf{x})$ is the output response of the network to an input data sample $\mathbf{x}$,

$$\begin{aligned} \frac{\partial J(\mathbf{w}, \mathbf{b})}{\partial \mathbf{h}(\mathbf{x})} &= -\left(y_i \frac{\partial \log\left(h_i(\mathbf{x})\right)}{\partial h_i(\mathbf{x})} + (1 - y_i) \frac{\partial \log\left(1 - h_i(\mathbf{x})\right)}{\partial h_i(\mathbf{x})} \right) \\ &= -\left(\frac{y_i}{h_i(\mathbf{x})} + \frac{(1 - y_i)}{(1 - h_i(\mathbf{x}))}(-1) \right) \\ &= -\left(\frac{y_i - h_i(\mathbf{x})}{h_i(\mathbf{x})\left(1 - h_i(\mathbf{x})\right)} \right), \quad \text{for } i = \{1, ..., l\}, \end{aligned} \tag{2.18}$$

$$\frac{\partial \mathbf{h}(\mathbf{x})}{\partial \mathbf{z}^{(3)}} = h_i(\mathbf{x})\left(1 - h_i(\mathbf{x})\right), \quad \text{for } i = \{1, ..., l\}, \tag{2.19}$$

$\mathbf{y}$ is the label of data sample $\mathbf{x}$, $\mathbf{w}$ denotes the synaptic weights, $\mathbf{b}$ denotes the biases, l is the number of output neurons in the third layer (or the output layer) and ℓ indicates the layer.

Thus,

$$\boldsymbol{\delta}^{(3)} = \mathbf{h}(\mathbf{x}) - \mathbf{y}. \tag{2.20}$$

Then, the error of layer three, $\boldsymbol{\delta}^{(3)}$, is backward propagated layer by layer as follows,

$$\boldsymbol{\delta}^{(2)} = \mathbf{w}^{(3)^T} \boldsymbol{\delta}^{(3)} \cdot * \varphi'\left(\mathbf{z}^{(2)}\right), \tag{2.21}$$

$$\boldsymbol{\delta}^{(1)} = \mathbf{w}^{(2)^T}\boldsymbol{\delta}^{(2)} \cdot * \varphi'\left(\mathbf{z}^{(1)}\right), \tag{2.22}$$

where $\cdot*$ denotes the element-wise multiplication operation and $\varphi'(\cdot)$ denotes the derivative of the activation function.

Next, $\boldsymbol{\delta}$ will be used to determine the gradient of the error with respect to the synaptic weights and biases in each layer [37] [34],

$$\begin{aligned}\frac{\partial J(\mathbf{w},\mathbf{b})}{\partial \mathbf{w}^{(\ell)}} &= \frac{\partial J(\mathbf{w},\mathbf{b})}{\partial \boldsymbol{a}^{(\ell)}}\frac{\partial \boldsymbol{a}^{(\ell)}}{\partial \mathbf{z}^{(\ell)}}\frac{\partial \mathbf{z}^{(\ell)}}{\partial \mathbf{w}^{(\ell)}} \\ &= \boldsymbol{\delta}^{(\ell)}\boldsymbol{a}^{(\ell-1)^T},\end{aligned} \tag{2.23}$$

$$\begin{aligned}\frac{\partial J(\mathbf{w},\mathbf{b})}{\partial \mathbf{b}^{(\ell)}} &= \frac{\partial J(\mathbf{w},\mathbf{b})}{\partial \boldsymbol{a}^{(\ell)}}\frac{\partial \boldsymbol{a}^{(\ell)}}{\partial \mathbf{z}^{(\ell)}}\frac{\partial \mathbf{z}^{(\ell)}}{\partial \mathbf{b}^{(\ell)}} \\ &= \boldsymbol{\delta}^{(\ell)},\end{aligned} \tag{2.24}$$

as

$$\frac{\partial \mathbf{z}^{(\ell)}}{\partial \mathbf{w}^{(\ell)}} = \frac{\partial\left(\mathbf{w}^{(\ell)}\boldsymbol{a}^{(\ell-1)} + \mathbf{b}^{(\ell)}\right)}{\partial \mathbf{w}^{(\ell)}} = \boldsymbol{a}^{(\ell-1)}, \tag{2.25}$$

$$\frac{\partial \mathbf{z}^{(\ell)}}{\partial \mathbf{b}^{(\ell)}} = \frac{\partial\left(\mathbf{w}^{(\ell)}\boldsymbol{a}^{(\ell-1)} + \mathbf{b}^{(\ell)}\right)}{\partial \mathbf{b}^{(\ell)}} = 1. \tag{2.26}$$

Finally, the synaptic weights and biases of the networks can be updated iteratively with an optimisation algorithm, such as batch gradient descent, that uses all the data samples in the training set to compute $\arg\min_{\mathbf{w},\mathbf{b}} J(\mathbf{w},\mathbf{b})$, where $J(\mathbf{w},\mathbf{b})$ is given in Eq. 2.8. The amount of updates that the weights receive at each iteration, i.e. step size update, is controlled by a learning rate, η. The incremental updates on the synaptic weights, $\Delta\mathbf{w}$, and biases, $\Delta\mathbf{b}$, at each iteration, t, is as follows [36] [38],

$$\Delta\mathbf{w}^{(\ell)}(t) = -\eta\frac{\partial J(\mathbf{w},\mathbf{b})}{\partial \mathbf{w}^{(\ell)}}(t), \tag{2.27}$$

$$\mathbf{w}^{(\ell)}(t+1) = \mathbf{w}^{(\ell)}(t) + \Delta\mathbf{w}^{(\ell)}(t), \tag{2.28}$$

$$\Delta\mathbf{b}^{(\ell)}(t) = -\eta\frac{\partial J(\mathbf{w},\mathbf{b})}{\partial\mathbf{b}^{(\ell)}}(t), \tag{2.29}$$

$$\mathbf{b}^{(\ell)}(t+1) = \mathbf{b}^{(\ell)}(t) + \Delta\mathbf{b}^{(\ell)}(t). \tag{2.30}$$

2.3 Convolutional Neural Network

A convolutional neural network (CNN) is a type of deep neural network that is specialised in, but not limited to, image processing tasks. The main differences between an MLP and a CNN is that a CNN is usually made up of more than two hidden layers and utilises weight sharing. The utilisation of weight sharing in a CNN results in it having significantly fewer trainable parameters and requiring less training time compared to an MLP of the same depth [12].

The original implementation of a CNN consists of a number of convolutional, pooling (subsampling) and fully connected layers [39]. The CNN essentially employs signal processing techniques for extracting features automatically. The convolutional layers contain convolution filters with different coefficients to produce different translation equivariance features [40]. The pooling layers contain non-linear filters to extract the most significant features in a translation invariant framework [40].

The CNN uses convolutional and pooling layers at the beginning of its architecture in order to extract features of input images. Then fully connected layers, which are the standard one dimensional input MLP, are used to predict the output of the input image [41]. An example of a CNN architecture is shown in Figure 2.4.

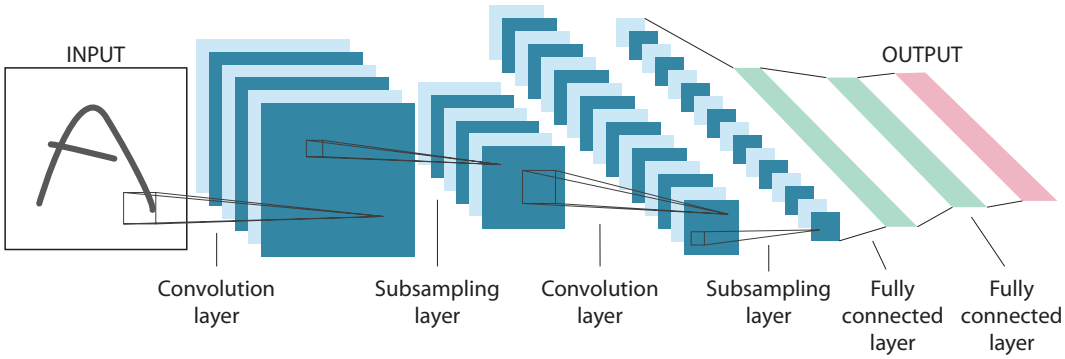


Figure 2.4: Example of a CNN architecture.

2.3.1 CNN Structures and Parameters

In this section, we will introduce the different structures and discuss the number of trainable parameters used in CNNs.

Structure

As discussed in Section 2.3, the original structure of a CNN is more suited for a classification task where it uses convolutional layers as the feature extractors and fully connected layers as the final classifier. A CNN can also be structured to perform other tasks. The most common CNN structures are the contracting, contracting-expanding and multi-stage structures.

A CNN designed for a classification task usually adopts a contracting structure, where the resolution of the feature maps decrease as the network gets deeper [18] [42] [43], where the lowest resolution feature maps are the input to the fully connected layers. A CNN used for classification usually requires the features to be highly invariant in order to make the final decision. The structure incorporates techniques or components that can help integrate spatial invariant properties into high-level features.

A CNN designed for a segmentation task usually adopts a contracting-expanding structure [18] [42] [43]. The contracting part of the structure is used for feature extraction, while the expanding part is used for high-level feature mapping to the original input resolution for pixel-wise predictions. A CNN used for segmentation usually requires the features to be highly equivariant, i.e. preserved spatial

information. The structure usually incorporates techniques or components that can help preserve the spatial information of the features.

A CNN designed for a more complex task, such as object detection task, sometimes adopts multiple algorithms, or networks, to handle different operations of a task. Each algorithm is usually designed and optimised for a specific operation within the task to achieve a higher overall performance when compared to the performance of a single algorithm designed to solve the task completely. For example, a CNN of a region of interest extraction algorithm and a classification network generally results in a better classification performance when compared to a single classification network.

Number of Trainable Parameters

The number of trainable parameters used in a CNN depends on three factors: depth, component arrangement and the number of filters used in each convolutional layer.

A deep CNN allows an effective observation into a larger area of the input image as opposed to a shallow CNN. As the network becomes deeper, a better knowledge of the spatial relationship of a pixel (or a feature) and its surroundings is established [44]. However, an increase in the depth of a network generally leads to an increase in the number of trainable parameters.

A CNN is usually constructed from a set of different components. Depending on the design, a CNN can have significantly fewer trainable parameters than another CNN constructed from the same set of components. For example, a CNN constructed from two $3 \times 3 \times 256$ convolutional layers stacked together, followed by two $1 \times 1 \times 1$ convolutional layers has a significantly higher number of trainable parameters in comparison to a CNN constructed from the same components arranged in an alternate manner, i.e. two pairs of $3 \times 3 \times 256$ convolutional layer and $1 \times 1 \times 1$ convolutional layer stacked together. Thus, the number of trainable parameters used in a CNN not only depends on the components used, but also on the arrangement of the components.

As discussed earlier, a CNN uses convolutional layers for feature extraction. Each convolutional layer consists of a number of filters that determine the type of features

to be extracted from the input. The larger the number of filters in each convolutional layer, the greater the capacity of the network to solve a task [45]. However, the larger the number of filters, the larger the number of trainable parameters in the network.

Therefore, the depth, component arrangement and number of filters used in each convolutional layer should be adjusted carefully by the user according to the task and resources available, as it can lead to an expensive memory usage and larger computation times.

2.3.2 Components of a CNN

In this section, we will introduce the components used to construct a CNN. In particular, the components discussed include the convolutional layer, pooling layer, upsampling layer, skip connections, dropout, normalisation and activation layer.

Convolutional Layer

A convolutional layer uses a set of convolution filters to perform operations to produce feature maps depending on the filter kernel [13]. The shape of the kernel specifies the shape and size of the input region, while the values of the kernel specify the weights on the input. The kernel shape is usually pre-determined in the network architecture, while the kernel values are initialised according to initialisation algorithms that will be discussed in Section 3.2.1 and adjusted during the training process. The resulting feature maps are the features of the input, such as gradients (edges).

In CNNs, the convolution operation [45] of an input image and a convolution filter kernel can be expressed as follows,

$$\mathbf{y}(i,j) = (\mathbf{x} * \mathbf{k})(i,j) = \sum_m \sum_n \mathbf{x}(i+m, j+n)\mathbf{k}(m,n), \tag{2.31}$$

where $*$ is the convolution operator, $\mathbf{y}$ is the matrix of the output image, $\mathbf{x}$ is the matrix of the input image, $\mathbf{k}$ is the matrix of the filter kernel and m and n are the row and column index of the filter kernel, respectively, with the centre of the kernel indicated by (0,0).

In CNNs, a convolutional layer can be implemented with four hyperparameters, i.e. number of filters, kernel size, stride length and padding. The number of filters, which corresponds to the number of feature maps produced in each layer, defines the capacity of a network. The kernel size defines the number of inputs in each direction that are processed at a time. The stride length defines the number of rows and columns that the filter shifts over after each computation. Padding refers to an operation of adding zeros on the perimeter of the input to maintain the output resolution of a layer. The application of padding in a convolutional layer, whether with or without padding, can be determined with a padding hyperparameter. A convolutional layer without predefined stride length and padding usually refers to a convolutional operation with a stride length of 1 and the application of padding, e.g. Eq. 2.31.

An illustration of a 3×3 convolution operation with stride length of 1 and no padding is shown in Figure 2.5, where the input data is convolved with a 3×3 convolution filter to produce output data, i.e. a feature map.

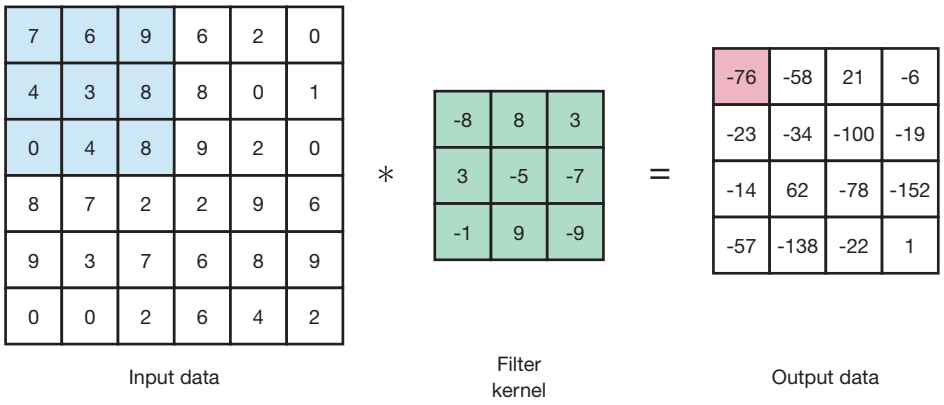


Figure 2.5: Example of a 3×3 convolution operation with stride length of 1 and no padding.

Pooling Layer

Pooling layers are used to reduce the dimension of the input and introduce translational invariances in the network through the process of downsampling [13]. The most common type of pooling layer used in CNNs is the max pooling layer [46] [26] [29] [17] [18]. It is a type of nonlinear filter that is used to extract significant

features from the previous layer by selecting the largest valued feature from each region of the sliding filter kernel. Linear filter based pooling layers, such as the average pooling layer or strided convolutional layer (convolutional layer with a stride length of more than 1) can also be used in place of, or along with, max pooling layers in certain applications [39] [47] [48] [30] [49].

In CNNs, a pooling layer is implemented with three hyperparameters, i.e. kernel size, stride length and padding. Typically, a 2×2 pooling layer, with stride length of 2 and no padding, is employed in CNNs to perform downsampling by a factor of 2. The illustration of 2×2 max, average and min pooling operations are shown in Figure 2.6, where the maximum, average and minimum feature value from each region of the sliding filter kernel are extracted accordingly.

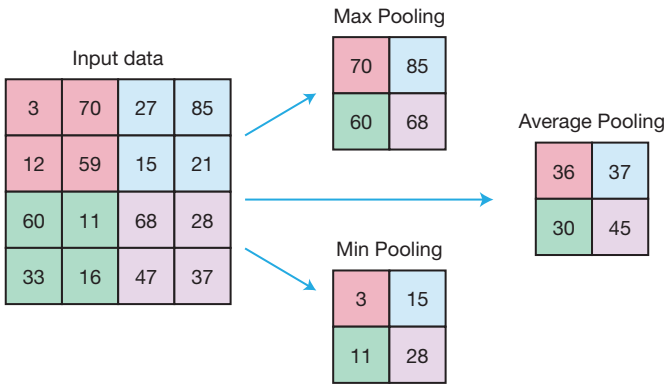


Figure 2.6: Examples of 2×2 max, average and min pooling operations.

Upsampling Layer

Upsampling layers are used to increase the dimension of the input. In digital image processing, upsampling is traditionally performed with interpolation methods. Interpolation is an estimation method used to obtain the value of a data point given the values of the data points around it.

The most basic upsampling method is the nearest neighbour interpolation. The nearest neighbour method chooses the nearest data point to the data point to be estimated and then uses its value as the estimated value without considering the values of other points [50]. However, nearest neighbour interpolation usually has the effect of producing sharp boundaries.

Another common interpolation method is the bilinear interpolation [50]. It involves performing linear interpolation in two directions (e.g. horizontal and vertical directions). It takes a weighted average of a number of predetermined neighbouring data points to make the estimation of the required data point. The resulting image has smoother edges as compared to the nearest neighbour interpolation method.

In CNNs, an interpolation-based upsampling layer is implemented with one hyperparameter, i.e. size. The size determines the upsampling factor for the interpolation to be performed in each direction. Examples of nearest neighbour and bilinear interpolation operations with the size of 2×2 are shown in Figure 2.7.

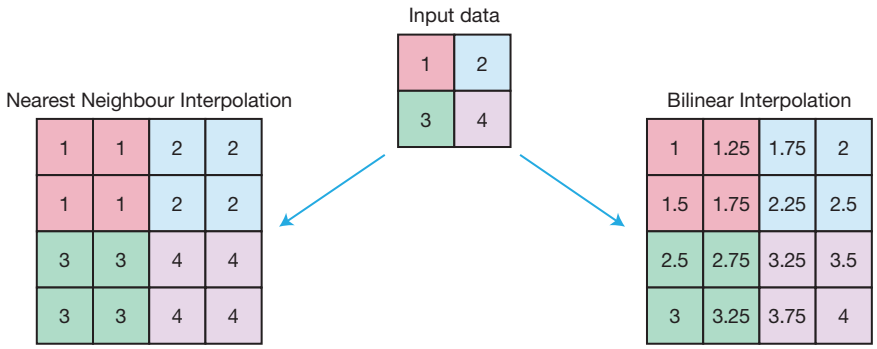


Figure 2.7: Example of 2×2 nearest neighbour and bilinear interpolation operations.

Transposed convolution is a trainable upsampling method. Unlike interpolation-based upsampling methods (e.g. nearest neighbour, bilinear interpolation), transposed convolution allows its parameter to be trained along with the other network parameters by backpropagation. The transposed convolution performs an upsampling with a factor, f, by performing convolution with a fractional input stride of $\frac{1}{f}$, where f is an integer [17]. A combination of a transposed convolutional layer and a non-linear activation function can be used to perform a nonlinear upsampling.

In CNNs, the hyperparameters required for a transposed convolutional layer is similar to a convolutional layer. A transposed convolutional layer without a predefined stride length and padding usually refers to a transposed convolutional layer with a stride length of 1 and no padding.

Another popular trainable upsampling method is called resize convolution [51].

It uses a combination of interpolation and convolutional layers in the form of a stack of $f \times f$ nearest neighbour interpolation and $f \times f$ convolutional layers, where f denotes the upsampling factor.

Generally, either a 2×2 resize convolution [18] or transposed convolution [47] is preferred to perform upsampling by a factor of 2.

Skip Connection

Deep CNNs have proven to be very successful in extracting important features through the use of convolutional layers [26] [29]. However, deep networks tend to experience a degradation problem [52], where network training accuracy increases, becomes saturated, and then reduces rapidly after the network starts converging.

Skip connections were originally used to incorporate a residual learning framework to address the degradation problem in deep networks [52]. The learning framework uses a skip connection from a layer ℓ to a later layer $\ell + n$ to replace the original function $F(\boldsymbol{a}^{(\ell)})$ with a function $H(\boldsymbol{a}^{(\ell)}) = F(\boldsymbol{a}^{(\ell)}) + \boldsymbol{a}^{(\ell)}$ [52], as shown in Figure 2.8.

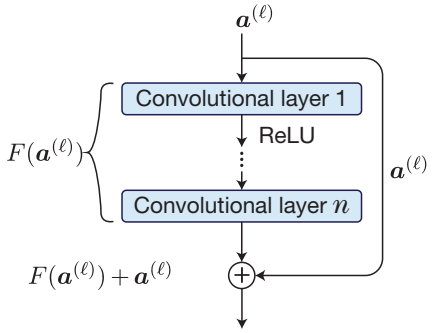


Figure 2.8: Example of a skip connection implementation to incorporate a residual learning framework.

The reason for the use of a residual learning framework is to prevent a deep network from having larger training errors than a corresponding shallow network by simplifying the training of any excess layers in the deep network, e.g. n convolutional layers in Figure 2.8, for an identity mapping. By using residual learning, the solvers can simplify the approximation of identity mappings by forcing the weights of the residual function, $F(\boldsymbol{a}^{(\ell)})$, toward zero, if identity mappings are optimal. This will

help to minimise the additional complexity introduced by any excess nonlinear layers in the deep network when an optimal solution for a task can be achieved with a corresponding shallow network [52]. An element-wise summation layer is used at the end of the skip connection, hence keeping the dimension of the output layer fixed, which adds neither additional parameters nor computational complexity [52].

Skip connections can also be used to merge features of one layer to another layer with a concatenation operation. The idea is to use preceding layer information in the later layer to achieve better performance. It can also be used in combination with upsampling to combine the upsampled predictions to form a fine pixel-wise segmentation result [17].

Dropout

Dropout is a regularisation technique used in the training phase to prevent overfitting. The idea is to train different models and use the average of the predictions to improve the generalisation of a network. The operation is performed by randomly removing neurons at a defined dropout rate, p_d, during training so the weights of the network are tuned based on different connectivity of the neurons in the network [53]. Note that the dropout rate is a value between 0 and 1. During testing, the weights of each neuron are multiplied by the probability of retention, $p_r = 1 - p_d$, to approximate average predictions of the different trained models. This technique can improve the learning outcome of neural networks [53].

An example of an application of dropout in a neural network is shown in Figure 2.9.

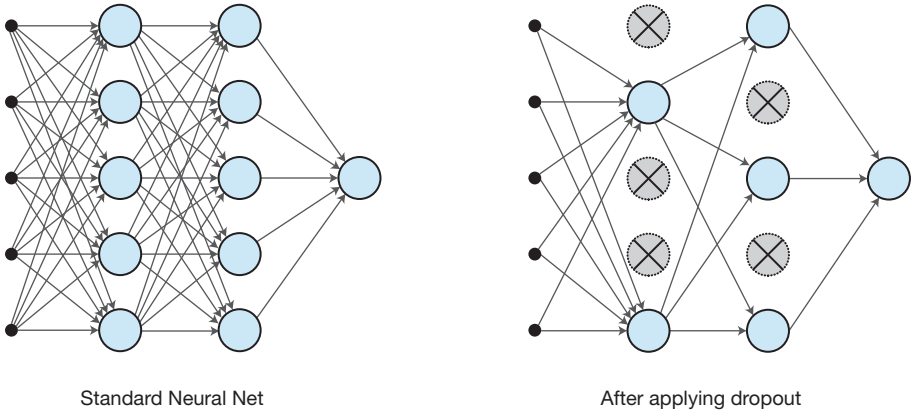


Figure 2.9: Example of a neural network without dropout (left) and with dropout (right).

Normalisation

Normalisation in CNNs is a technique used to maintain the features in each layer to be at the same scale. The two most common types of normalisation layers in CNNs are known as local response and batch normalisation layers.

A local response normalisation layer normalises the input over local regions, e.g. 3×3. The purpose of it is to aid with the generalisation of the network. The basic idea of local response normalisation was inspired by lateral inhibition found in real neurons [46], i.e. to inhibit the activation of the neighbouring neurons caused by the excited neurons. However, it is a fixed algorithm, hence its hyperparameters are unable to be tuned throughout the training process.

A batch normalisation layer normalises the input of the layer by subtracting the mean and dividing by the standard deviation of each mini-batch of inputs (i.e. a subset of training dataset). This mini-batch normalisation introduces some noise to the data and results in a form of regularisation. This operation also enables the use of higher learning rates to speed up the training process [54]. However, applying a normalisation method to the input of each layer may change the representation of the original input, e.g. normalising the input of a sigmoid function may constrain the input to be within the linear region of the sigmoid function, hence not utilising its nonlinear characteristic. Therefore, extra parameters to control the scale and shift of the normalised value are implemented in the batch normalisation layer, which need

to be learned along with the other network parameters. These extra parameters enable the restoration of the original value if it produces better results than the normalised value [54].

Activation Layer

Activation layers are usually applied after convolutional layers in order to decide whether a particular neuron should be activated or not. The operation of a convolutional layer followed by an activation layer in a CNN is expressed as follows [55],

$$\boldsymbol{a}_j^{(\ell)} = \varphi\left(\left(\sum_{i=1}^{n} \mathbf{k}_{j,i}^{(\ell)} * \boldsymbol{a}_i^{(\ell-1)}\right) + b_j^{(\ell)}\right), \tag{2.32}$$

where n is the number of input from the previous layer, b is the bias of each convolution filter, $\mathbf{k}_{i,j}^{(\ell)}$ is the kernel of each convolution filter in the convolutional layer that connects i^{th} input feature map from the previous layer, $\boldsymbol{a}_i^{(\ell-1)}$, with the j^{th} output feature map in layer ℓ, $\boldsymbol{a}_j^{(\ell)}$ and $\varphi(\cdot)$ is the activation function used in the activation layer.

Nonlinear activation functions, e.g. sigmoid, Tanh, ReLU, leaky ReLU (LReLU) or parametric ReLU (PReLU), are usually applied throughout the network to enable the network to approximate most nonlinear functions [56].

As discussed in Section 2.2, the sigmoid function [56] maps the input to an output range of 0 to 1. It is based on a logistic function and expressed as,

$$f(x) = \frac{1}{(1 + e^{-x})}. \tag{2.33}$$

The Tanh function [57] (i.e. hyperbolic tangent function) maps the input to an output range of -1 to 1. It is also based on a logistic function and defined by,

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}. \tag{2.34}$$

The logistic function is a classic activation function that was initially used because of its similarity to a biological neuron's activation rate [56]. It is

generally utilised due to its differentiable property, which makes it suitable for the backpropagation training algorithm. However, its saturation property, often used for decision boundaries, also causes what is known as vanishing gradients during training [56].

The rectified linear unit [56], ReLU, is an unsaturated activation function that overcomes the vanishing gradients problem by eliminating exponential terms. It is defined by,

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \leq 0. \end{cases} \tag{2.35}$$

Unlike the sigmoid and Tanh activation functions, where every result of the weighted sum function in Eq. 2.32 results in an activated neuron, ReLU restricts the activation of the neuron when the weighted sum is less than or equal to zero. This simplifies the network and decreases the computational time during the training process, which makes it the most preferred activation function for hidden neurons. However, the ReLU comes with an issue. Since the gradient of the ReLU function in the negative region of x is zero, once the neuron is inactive, the neuron will not be activated again throughout the gradient descent based training process, creating what is known as the dying ReLU problem [58].

A Leaky ReLU [56] (LReLU) offers a solution to the dying ReLU problem by setting the gradient of the negative region to a small constant value, c, i.e.

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ cx & \text{if } x \leq 0. \end{cases} \tag{2.36}$$

The Parametric ReLU (PReLU) turns the small constant value, c, into a trainable parameter, α, so that the gradient of the negative region can be trained accordingly [59].

Although the LReLU and other modified activation functions, e.g. PReLU, somewhat solve the dying ReLU problem and are shown to be superior to ReLU [56], many state-of-the-art segmentation networks [17] [18] [31] still use ReLU as their

activation functions throughout the network since it produces satisfactory results and is simpler to implement.

For the final layer of the neural network, the choice of activation function depends on the task. To perform linear regression, a linear activation function is often used to produce an output that is proportional to the input without any transformation. To perform classification, a sigmoid function is often used in a classification task with up to two classes, while a softmax activation function is generally used for a multiclass classification task, i.e. more than two classes. The softmax activation function, also known as a normalised exponential, is used to normalise the network output to a probability distribution over the number of output classes [60], and is given by,

$$f(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}, \tag{2.37}$$

where i refers to an element of the input vector $\mathbf{x}$ and K refers to the number of output classes.

2.4 CNN Architectures

CNN architectures can be designed for various computer vision tasks. These tasks can be mainly categorised into; image classification, object detection and image segmentation.

2.4.1 CNN Architectures for Image Classification

The basic idea of image classification is to categorise an image into one of a set of classes according to the most significant features in the image. It is the basis of object detection and image segmentation tasks. The main challenges of the classification task are the object variability due to viewpoint and intra-class variances [61]. An example of an image classification task is shown in Figure 2.10.

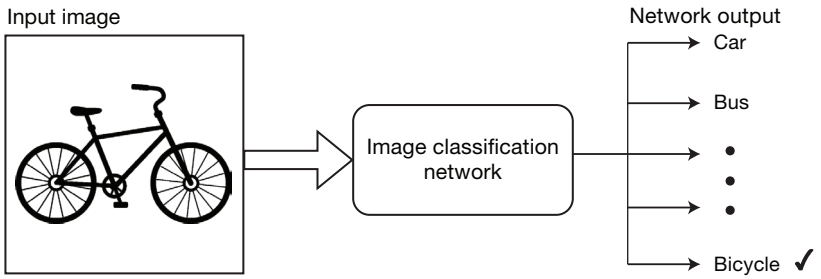


Figure 2.10: An example of a classification task.

The original implementation of a CNN that was discussed in Section 2.3 is known as LeNet-5 [39]. It was developed for image classification, in particular, a handwritten digit recognition task. It consists of convolution, pooling and fully connected layers. The LeNet-5 is composed of seven layers and has a total of 60k trainable parameters. It takes an input with a dimension of 32×32, then applies a Tanh activation function on each convolutional layer and utilises average pooling to perform downsampling.

Another early CNN designed for an image classification task is AlexNet [46]. It won the 2012 ILSVRC competition on classifying the main objects present in images into approximately 10,000 object classes [46]. The AlexNet architecture is composed of eight layers and has a total of 60M trainable parameters [46]. The main attribute of AlexNet is that it adopts the ReLU activation, local response normalisation and dropout regularisation in its architecture. AlexNet [46] requires a fixed input image dimension of 224×224. For training, several data augmentation techniques are used, such as image translation, mirroring and intensity alteration of the RGB channels, to achieve a low error rate. For testing, the network makes a prediction of ten cropped and augmented images from the original 256×256 image and averages the predictions to obtain the final prediction.

The next breakthrough in CNN is the VGG network [26]. The VGG network uses small filter kernels to build deeper networks. Its best performing network, VGG-19 [26], architecture consists of 19 layers and has a total of 144M trainable parameters. The rest of its architecture and implementation techniques were adopted from AlexNet [46].

Another variation of a CNN is GoogLeNet [29]. This architecture won the 2014

ILSVRC image classification challenge [29]. GoogLeNet architecture consists of 27 layers and uses a total of 6.8M trainable parameters. Its main difference with the above-mentioned CNN architectures is that it uses a combination of multiple size convolution filters in parallel, known as an inception module. Each inception module processes the input of the module with the multiple size convolution filters and produces a concatenated output of these filters. Each of these modules allows the combination of different feature levels at the same time [29]. This inception module also adopted 1×1 convolutional layers in its architecture for feature map dimensionality reduction to reduce the overall number of trainable parameters. GoogLeNet also replaced the fully connected layers with an average pooling layer to reduce the number of trainable parameters.

Despite GoogLeNet's success with inception modules, ReSNet [48] won the 2015 ILSVRC competition. It regressed back to the original CNN architecture, which consists of convolution, pooling and fully connected layers. However, it introduced skip connections to facilitate the idea of having deeper networks. The skip connections give options to the model to make full use of the deeper architecture or just a shallow counterpart with a residual learning framework. It also adopts 1×1 convolutional layers in its architecture for feature map dimensionality reduction that allows a 152-layer network to be constructed with a total of 60M trainable parameters. A Batch Normalisation (BN) layer [54] is used to optimise the training of a deeper network.

Following the ResNet success with skip connections, DenseNet [30] was developed by modifying the ResNet learning framework. Instead of using addition on the skip connections from early to later layers, concatenation was used. DenseNet is composed of 250 layers and has a total of 15.3M trainable parameters. Motivated by ResNet, it uses a 1×1 convolution before each 3×3 convolutional layer for dimensionality reduction to minimise the number of trainable parameters in its architecture.

A summary of the main attributes of CNNs for image classification are given in Table 2.1.

Table 2.1: Main attributes of CNNs for image classification

Network	Main Attributes
LeNet-5	Combination of convolution, pooling and fully connected layers
AlexNet	ReLu activation, local response normalisation, dropout regularisation
VGG	Deeper network structure
GoogLeNet	1x1 convolution, global average pooling, inception module
ReSNet	Skip connections to form residual learning framework
DenseNet	Dense block (skip connections from all subsequent layers)

2.4.2 CNN Architectures for Object Detection

Object detection is a more challenging task, as it combines the complexity of two tasks, i.e. image classification and object localisation. As discussed in Section 2.4.1, image classification requires a model to categorise the input image into one of the predefined set of classes. Object localisation requires a model to produce bounding boxes as an output to indicate the object's location and size in an image.

An example of an object detection task is shown in Figure 2.11, where an object detection network produces a bounding box and a class label for each object detected $(0, ..., 4)$ in the input image. The bounding box output is usually in the form of the initial point of the box in x and y coordinates, followed by the width, w, and height, h of the box. Each bounding box is denoted as (x, y, w, h).

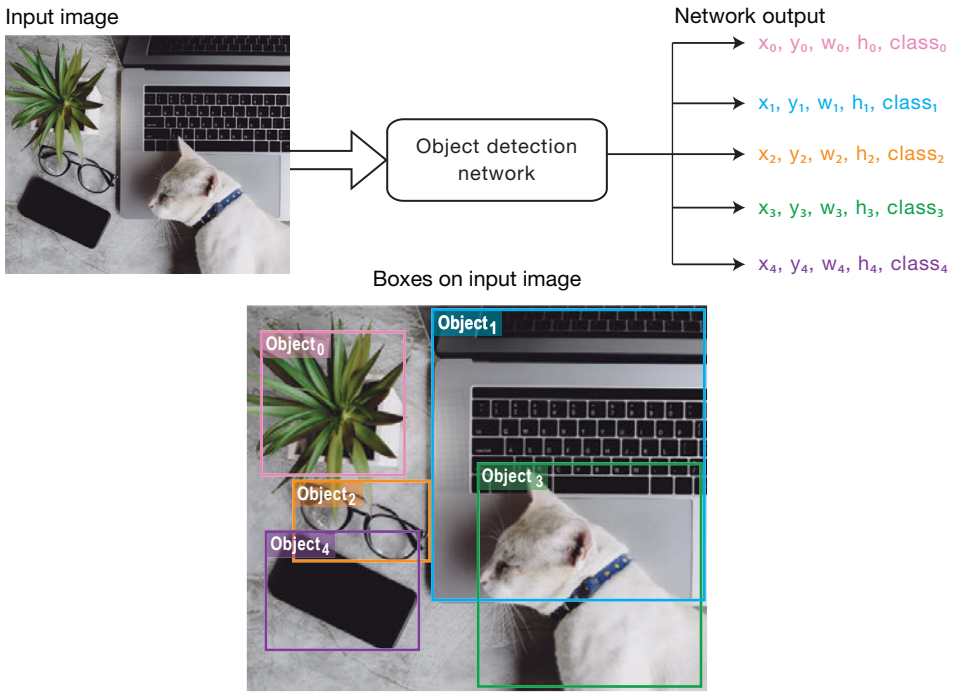


Figure 2.11: An example of an object detection task.

One of the earliest algorithms for object detection is R-CNN [62], which evolved to become Fast R-CNN [27] and then Faster R-CNN [28]. The acronym R-CNN stands for Regions with CNN features. As the name infers, it is a method based on a CNN for object detection. The two methods, R-CNN and Fast R-CNN, rely on a selective search algorithm to provide region proposals and a pre-trained CNN model (AlexNet [46] was used in R-CNN and VGG-16 [26] was used in Fast R-CNN) for feature extraction. In the R-CNN, the features from a CNN are used by another machine learning algorithm, called a support vector machine (SVM), and a linear regression model, to perform classification and bounding box prediction respectively [63]. The SVM is a classical machine learning algorithm that looks for the closest samples from different classes i.e. support vectors, and uses them to form the best separating hyperplane to predict or classify new data samples. However, in the Fast R-CNN, the SVM and the linear regression model are replaced by fully connected layers with softmax and linear activation functions for classification and bounding box prediction, respectively. The speed and detection performances of Fast R-CNN are further improved in Faster R-CNN where the selective search algorithm is replaced with a simple convolutional network called a Region Proposal Network

(RPN) and the Fast R-CNN is kept as the detection network. The RPN is made of an $n \times n$ convolutional layer followed by two 1×1 convolutional layers with softmax and linear activation functions, respectively. Both the RPN and the detection network in Faster R-CNN can be trained to utilise shared convolutional features extracted from a pre-trained CNN, e.g. VGG-16.

Motivated by the complex pipelines of R-CNN and the lack of speed of Fast and Faster R-CNN in real-time object detection, YOLO [49] is created as the first network that can run in real time with high accuracy. Unlike the family of R-CNN that are composed of two stages, i.e. region proposal box generation and classification, YOLO is a single-stage network that predicts bounding boxes and class probabilities of objects in an input image in one evaluation, hence the name "You Only Look Once" (YOLO). The YOLO architecture is inspired by GoogLeNet, where the inception modules are replaced by blocks containing a 1×1 convolutional layer followed by a 3×3 convolutional layer. The whole architecture consists of 30 layers and has a total of 271.7M parameters, where most of the trainable parameters are used in the fully connected layers. YOLO [49] requires a fixed input image dimension of 448×448 for detection. It adopts dropout as its regularisation technique and uses Leaky ReLU instead of ReLU as its activation function, which follows the convolutional layers directly throughout its architecture. A linear activation function is used for the final prediction layer.

A summary of the main attributes of CNNs for object detection are given in Table 2.2.

Table 2.2: Main attributes of CNNs for object detection

Network	Main Attributes
R-CNN	Idea of bounding box prediction for object localisation
Fast R-CNN	Use of a neural network as a linear regression model for bounding box prediction
Faster R-CNN	Cascade use of CNNs for object detection task
YOLO	A new CNN architecture to predict bounding boxes and class probabilities simultaneously

2.4.3 CNN Architectures for Image Segmentation

Image segmentation is a process of separating an image into multiple segments to facilitate analysis. In CNNs, image segmentation is usually referred to as the process of assigning a class label to every pixel in an image from a pre-defined set of classes, hence producing a pixel-wise mask of the image as the output. Thus, the output of a CNN used for image segmentation is more precise in terms of size and location in comparison to the output of a CNN used for object detection [64].

An example of an image segmentation task is shown in Figure 2.12, where every pixel in the input image is assigned with one class from a pre-defined set of classes (e.g. in this case, a person, a horse, a dog, the land and sky). The network output is a pixel-wise mask of the image composed of different class labels, in this case it is visualised with different colours: a person is orange, a horse is purple, a dog is red, the land is green and the sky is blue.

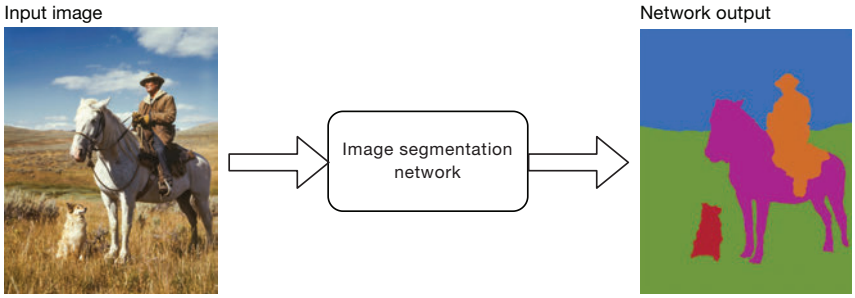


Figure 2.12: An example of an image segmentation task.

One of the most successful CNNs for image segmentation is the Fully Convolutional Network (FCN) [17]. This network is constructed from a classification network architecture, e.g. AlexNet [46], VGG-16 [26] or GoogLeNet [29], by removing the final layer of the network and converting the fully connected layers to convolutional layers. A 1×1 convolutional layer with a fixed number of convolution filters, which represents the number of pre-defined classes, is used to perform classification of each feature at each feature location. This 1×1 classification layer can be used to perform predictions at different feature map resolutions by appending it at multiple output locations throughout the network. Each output map produced from a set of downsampled feature maps has an output resolution lower

than the original input resolution, i.e. coarse output. A transposed convolutional layer with a stride length equal to the upsampling factor is then used to upsample the output maps. The best performing FCN architecture [17] is FCN-8s which has a stride length of 8 and a 1×1 classification layer with a transposed convolutional layer appended at three different locations. The upsampled output maps are then combined to form pixel-dense segmentation outputs with fine details. Its main attribute is the new image segmentation architecture obtained by omitting the fully connected layers of a classification network and then combining the predictions from multiple resolution feature maps with transposed convolutional layers that can be trained to perform upsampling. The best performing FCN-8s [17] is based on VGG-16 and has a total of 134M trainable parameters. An illustration of an FCN architecture is shown in Figure 2.13.

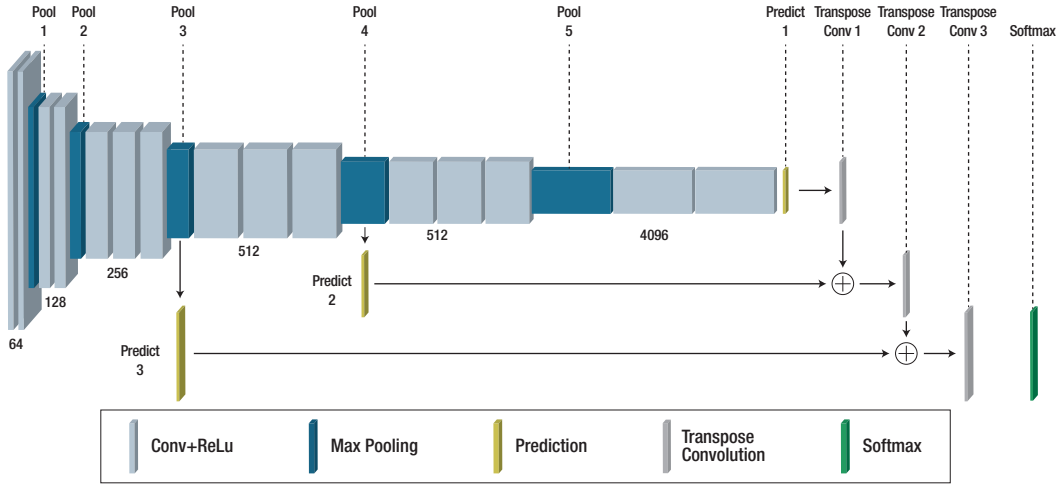


Figure 2.13: An illustration of an FCN architecture.

Another CNN used for image segmentation is known as U-Net [18]. This network is a modification and extension of the FCN. U-Net consists of two CNN classification structures forming a u-shaped, contracting-expanding, structure. The contracting part is used for feature extraction, while the expanding part is used to propagate context information to a higher resolution layer (feature mapping). U-Net also adopts a skip connection to concatenate the features from the contracting to the expanding part to help with feature mapping. The components of U-Net are similar to the FCN except for the upsampling method. It adopts a 2×2 resize convolution

instead of a transposed convolution. The U-Net architecture described in [18] consists of 23 convolutional layers and has a total of 31M trainable parameters. An illustration of a U-Net architecture is shown in Figure 2.14.

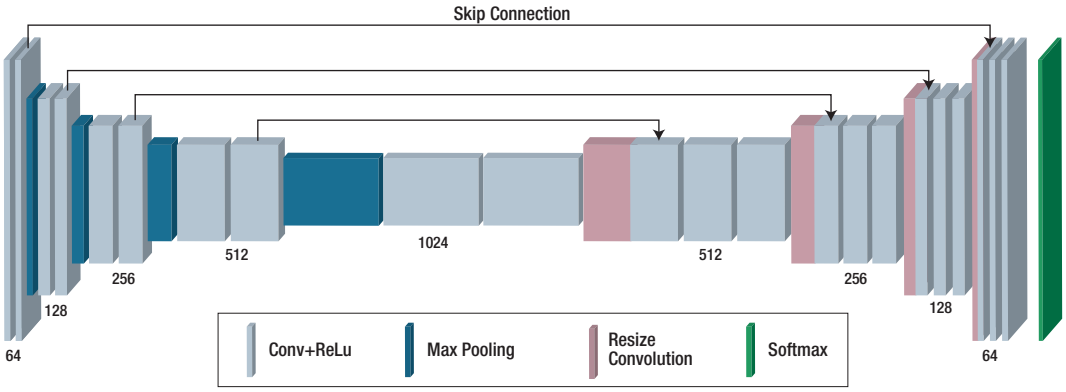


Figure 2.14: An illustration of a U-Net architecture.

The next development is V-Net which is a 3D CNN [47], it is created by replacing the U-Net 2D convolutional layers with 3D convolutional layers. It is similar in structure to U-Net, however it accepts volumetric data. V-Net uses a similar upsampling method to that of the FCN, implements residual learning in its architecture like ResNet, excludes dropout regularisation, and uses PReLU instead of ReLU. It was developed to perform volumetric medical image segmentation, as medical images are often 3D in nature.

A summary of the main attributes of CNNs for image segmentation are provided in Table 2.3.

Table 2.3: Main attributes of CNNs for image segmentation

Network	Main Attribute
FCN	A new CNN architecture for image segmentation
U-Net	U-Shaped network structure for image segmentation
V-Net	A three-dimensional version of U-Net to perform volumetric segmentation

2.4.4 CNN Structure and Parameter Summary

A summary of the structures and number of trainable parameters used in the state-of-the-art CNNs is presented in Table 2.4.

Table 2.4: CNN Structure and Parameter Summary

Network	Task	Structure	Property	Number of Trainable Parameters
LeNet-5 [39]	Classification	Contracting	Invariant features	60k
AlexNet [46]	Classification	Contracting	Invariant features	61M
VGG [26]	Classification	Contracting	Invariant features	144 M
GoogLeNet [29]	Classification	Contracting	Invariant features	6.8M
ReSNet [48]	Classification	Contracting	Invariant features	60M
DenseNet [30]	Classification	Contracting	Invariant features	15.3M
R-CNN [62]	Object detection	Multi-stage	Discrete task oriented	61M
Fast R-CNN [27]	Object detection	Multi-stage	Discrete task oriented	144 M
Faster R-CNN [28]	Object detection	Multi-stage	Discrete task oriented	144 M
YOLO [49]	Regression	Contracting	Invariant features	271.7M
FCN [17]	Segmentation	Contracting-expanding	Equivariant features	134M
U-Net [18]	Segmentation	Contracting-expanding	Equivariant features	31M
V-Net [47]	Segmentation	Contracting-expanding	Equivariant features	297M

As can be seen in Table 2.4, the structure used in the state-of-the-art CNNs depends primarily on the task-whether it is for classification, segmentation, or a more complex task, such as object detection. A CNN designed for a classification or regression task usually adopts a structure to integrate spatial invariant properties into high-level features for making the prediction. A CNN designed for segmentation usually adopts a structure that is suitable to map high-level features back to the original input resolution for segmentation. A CNN designed for a more complex task, such as object detection, usually adopts multiple algorithms/networks to carry out different operations of a task.

The structures of LeNet-5 [39], AlexNet [46], VGG [26], GoogLeNet [29], ReSNet [48], and DenseNet [30] have been designed and optimised for classification tasks. Whereas, FCN [17], U-Net [18], and V-Net [47] adopt the structures of a network that was not originally designed and optimised according to the needs of a segmentation task. The FCN compensates for the unoptimised design by using a combination of multiple predictions of different feature resolutions, while U-Net [18] and V-Net [47] use a stack of two classification networks to form a contracting-expanding structure for a progressively higher resolution mapping from the low resolution feature maps to the original input resolution. For an object detection task, R-CNN [62], Fast

R-CNN [27] and Faster R-CNN [28] use a multi-stage structure to adopt multiple traditional algorithms and/or networks to handle different operations of the task. However, most of the networks designed with multi-stage structures are usually slower and more tedious to utilise compared to networks designed with a single-stage structure, e.g. YOLO [49].

From Table 2.4, we can see that the number of trainable parameters varies widely from 60k to 297M regardless of the structure of the network. The number varies due to the network depth, network architecture and the number of convolution filters used in each layer of the network.

2.4.5 CNN Component Summary

The components used in the state-of-the-art CNNs are presented in Table 2.5. Although the number of different components in the state-of-the-art networks is quite limited, outstanding performance in various applications can still be achieved with the appropriate combinations of these different components and their hyperparameter variations.

Table 2.5: CNN Component Summary

Network	Components					
	Downsampling	**Upsampling**	**Skip Connection**	**Dropout**	**Normalisation**	**Activation**
LeNet-5 [39]	Average pooling	-	-	-	-	Tanh
AlexNet [46]	Max pooling	-	-	Dropout	Local response normalisation	ReLU
VGG [26]	Max pooling	-	-	Dropout	-	ReLU
GoogLeNet [29]	Max pooling	-	-	Dropout	-	ReLU
ReSNet [48]	Max pooling, Strided convolution	-	Residual learning	-	Batch normalisation	ReLU
DenseNet [30]	Max pooling Average pooling	-	Features forwarding	Dropout	Batch normalisation	ReLU
R-CNN [62]	Max pooling	-	-	Dropout	Local response normalisation	ReLU
Fast R-CNN [27]	Max pooling	-	-	Dropout	-	ReLU
Faster R-CNN [28]	Max pooling	-	-	Dropout	-	ReLU
YOLO [49]	Max pooling Strided convolution	-	-	Dropout	-	Leaky ReLU
FCN [17]	Max pooling	Transposed convolution	Features forwarding	Dropout	-	ReLU
U-Net [18]	Max Pooling	Resize convolution	Features forwarding	Dropout	-	ReLU
V-Net [47]	Strided convolution	Transposed convolution	Residual learning, features forwarding	-	-	PReLU

2.5 Conclusion

In this chapter, we introduced some fundamental concepts of neural networks. We discussed the CNN in terms of its typical structures, number of trainable parameters and components. We also provided some discussion on the state-of-the-art CNNs that are utilised in different image-oriented tasks, as well as the summary of their structures, number of trainable parameters and components. Note that, some of the state-of-the-art CNNs are relying on the use of either multiple networks (e.g. U-Net, V-Net), multiple algorithms (e.g. R-CNN family) or a large number of trainable parameters (e.g. VGG, YOLO) to obtain outstanding performance. As a result, a CNN architecture often has redundant components and parameters.

CHAPTER

3

Considerations in the Implementation of a Neural Network

There are a number of considerations to take into account, other than the network architecture, to ensure an effective implementation of a neural network. These considerations include determining various hyperparameters and algorithms, data pre- and post-processing, as well as the evaluation method. In this chapter, we discuss each of these considerations to provide insight into an effective neural network implementation.

3.1 Introduction

This chapter discusses a number of considerations that need to be taken into account to ensure an effective implementation of a neural network, specifically a CNN. It includes various hyperparameters and algorithms, data pre- and post-processing, and the evaluation method. The hyperparameters and algorithms include; weight initialisation scheme, learning rate, momentum algorithm, adaptive learning rate algorithm and regularisation algorithm. The data pre-processing includes normalisation, enhancement, augmentation and balancing, while the data post-processing includes mathematical morphology operations and connected component labelling. Discussion on the evaluation method covers the most common methods, i.e. cross-validation and hold out methods.

3.2 Hyperparameters and Algorithms

As discussed in Chapter 2, an optimisation algorithm is used in the training of a neural network to minimise a cost function with the aim to converge to the global minimum. However, as the problem is generally non-convex, it does not necessarily converge to the global minimum. Instead, it tends to converge to other stationary points, such as saddlepoints or local minima [37]. To optimise the training of a neural network, there are a number of hyperparameters and algorithms that can be considered. The algorithms include weight initialisation, momentum, adaptive learning rate and regularisation. The hyperparameters include learning rate, momentum constant, decay rate and regularisation constant. Each of these hyperparameters affect the convergence and the ability of the network to generalise.

3.2.1 Initialisation

In a neural network, initialisation of the weights is important for a gradient descent optimisation algorithm to achieve global convergence, as the algorithm can become trapped in a local minimum. A good choice of initial weights may allow the optimisation algorithm to converge to the global minimum [37].

One of the most common initialisation scheme is a uniform random initialisation [38] [65]. In this case the bias is set to 0 and the weights, $\mathbf{w}$, at each layer are set as,

$$\mathbf{w}^{(\ell)} \sim U\left[-\frac{1}{\sqrt{n^{(\ell-1)}}}, \frac{1}{\sqrt{n^{(\ell-1)}}}\right], \tag{3.1}$$

where $\sim U\left[-\alpha, \alpha\right]$ is a uniform distribution in the interval $(-\alpha,\ \alpha)$, and $n^{(\ell-1)}$ is the number of hidden neurons in the previous layer.

However, it was observed that the use of the uniform random initialisation causes the variance of the gradient of the cost function on the input biases at each layer to become smaller (i.e. vanishing) as it is propagated deeper into the network [65]. Thus, a normalised initialisation scheme, called Xavier initialisation [65], was proposed to maintain the variances of these gradients across all the layers. The Xavier initialisation [65] has the bias set to 0 and the weights, $\mathbf{w}$, at each layer set as,

$$\mathbf{w}^{(\ell)} \sim U\left[-\frac{\sqrt{6}}{\sqrt{n^{(\ell-1)}+n^{(\ell+1)}}}, \frac{\sqrt{6}}{\sqrt{n^{(\ell-1)}+n^{(\ell+1)}}}\right], \tag{3.2}$$

where $\sim U\left[-\alpha, \alpha\right]$ is a uniform distribution in the interval $(-\alpha,\ \alpha)$, $n^{(\ell-1)}$ is the number of hidden neurons in the previous layer and $n^{(\ell+1)}$ is the number of hidden neurons in the next layer.

The Xavier initialisation scheme results in a significantly faster convergence compared to the random initialisation scheme [65]. However, it was only proven [65] to be significantly helpful in a network that uses activation functions with symmetry around 0 and an output between -1 and 1 (e.g. Tanh) in its hidden neurons.

For networks that use ReLU (or its variations) activation functions in its hidden neurons, the best initialisation scheme is known as He initialisation [59], where it specifically considers the asymmetry of the ReLU activation function. He initialisation [59] has the bias set to 0 and weights, $\mathbf{w}$, at each layer set as,

$$\mathbf{w}^{(\ell)} \sim N\left(0, \frac{2}{n^{(\ell-1)}}\right), \tag{3.3}$$

where $\sim N\left(\mu, \sigma^2\right)$ is a normal distribution with a mean of μ, a standard deviation of σ, and $n^{(\ell-1)}$ is the number of hidden neurons in the previous layer.

He initialisation has been proven to be superior to Xavier initialisation in the training of a very deep CNNs (up to 30 layers) with ReLU activation functions [59].

3.2.2 Learning rate

Learning rate, η, is a parameter in the gradient descent optimisation algorithm that controls the step size at every iteration towards the minimum point of the cost function [45]. The learning rate determines the speed of which model convergence occurs during training [45]. A small learning rate generally leads to small updates on the synaptic weights and slower convergence to a minimum, while a large learning rate leads to large updates on the synaptic weights and hence faster convergence to a minimum. The learning rate also affects the convergence of a model [45]. If the chosen learning rate is too large, it may cause the learning progress to be oscillatory or unstable, in which it may never reach a minimum. On the other hand, if the learning rate is too small, it may get stuck in a local minima or other suboptimal stationary point.

In the most basic implementation of the gradient descent optimisation algorithm, a fixed learning rate is used [45]. However, in practice, a gradual decrease of the learning rate may result in a better generalisation and a faster convergence [45]. The gradual decrease of learning rate can be achieved by the use of either a decaying or adaptive learning rate.

The most basic type of decaying learning rate decreases linearly over a fixed number of iterations and then stays constant for the remaining iterations. It is formulated as,

$$\eta(t) = (1 - \beta)\eta_0 + \beta\eta_\tau, \tag{3.4}$$

where $\eta(t)$ denotes the current learning rate, t denotes the current iteration number, $\beta = \frac{t}{\tau}$, τ denotes the total number of iterations over which the learning rate decays, η_0 denotes the initial learning rate and η_τ denotes the constant learning rate. A

general rule of thumb is to set τ to be the number of iterations required to pass through the entire training set by a few hundred passes, i.e. epochs, while η_τ is usually set to be approximately 1% of η_0. However, the same trade-offs exists as in the fixed learning rate scheme when determining η_0. Unfortunately, η_0 can only be determined empirically by monitoring the cost function for the first several iterations and adjusting it according to the observed speed of convergence and learning progress.

Unlike the decaying learning rate that adjusts the learning rate over time within a predetermined number of iterations, the adaptive learning rate adjusts the learning rate through the entire learning process. The adaptive learning rate will be discussed in Section 3.2.4.

3.2.3 Momentum

A learning algorithm, known as momentum, is designed to accelerate learning and avoid instability by including previous weight updates in the weight update equation. It uses a momentum constant, α, defined in the range of 0 to 1 to control the proportion of the previous weight update to be included in subsequent updates [45].

The application of a momentum algorithm [45] results in a more stable weight update, as it accumulates an exponentially decaying moving average of all the previous weight updates [66] [45]. It prevents extreme weight update by amplifying the weight update that is in the same direction with the past weight updates and vice versa [66]. The updates on the synaptic weights, $\Delta\mathbf{w}$, and biases, $\Delta\mathbf{b}$, at iteration t become,

$$\Delta\mathbf{w}^{(\ell)}(t) = -\eta\frac{\partial J(\mathbf{w},\mathbf{b})}{\partial\mathbf{w}^{(\ell)}}(t) + \alpha\Delta\mathbf{w}^{(\ell)}(t-1), \tag{3.5}$$

$$\Delta\mathbf{b}^{(\ell)}(t) = -\eta\frac{\partial J(\mathbf{w},\mathbf{b})}{\partial\mathbf{b}^{(\ell)}}(t) + \alpha\Delta\mathbf{b}^{(\ell)}(t-1), \tag{3.6}$$

where $J(\mathbf{w},\mathbf{b})$ is the cost function, $\mathbf{w}$ is the synaptic weights, $\mathbf{b}$ is the biases, η denotes the learning rate and α denotes the momentum constant.

3.2.4 Adaptive Learning Rate

An adaptive learning rate can be implemented through the entire learning process with an adaptive optimisation algorithm, such as adaptive gradients (AdaGrad), root mean squared propagation (RMSProp) and adaptive moments (Adam).

The AdaGrad algorithm [45] implements the adaptive learning rate by scaling the weight update by the inverse of the square root of the accumulated squared gradients. The updates on the synaptic weights, $\Delta\mathbf{w}$, and biases, $\Delta\mathbf{b}$, at iteration t become,

$$\mathbf{v}_{\mathbf{w}}^{(\ell)}(t) = \mathbf{v}_{\mathbf{w}}^{(\ell)}(t-1) + \left[\frac{\partial J(\mathbf{w},\mathbf{b})}{\partial \mathbf{w}^{(\ell)}}(t)\right]^2, \tag{3.7}$$

$$\Delta\mathbf{w}^{(\ell)}(t) = -\frac{\eta}{\epsilon + \sqrt{\mathbf{v}_{\mathbf{w}}^{(\ell)}(t)}}\frac{\partial J(\mathbf{w},\mathbf{b})}{\partial \mathbf{w}^{(\ell)}}(t), \tag{3.8}$$

$$\mathbf{v}_{\mathbf{b}}^{(\ell)}(t) = \mathbf{v}_{\mathbf{b}}^{(\ell)}(t-1) + \left[\frac{\partial J(\mathbf{w},\mathbf{b})}{\partial \mathbf{b}^{(\ell)}}(t)\right]^2, \tag{3.9}$$

$$\Delta\mathbf{b}^{(\ell)}(t) = -\frac{\eta}{\epsilon + \sqrt{\mathbf{v}_{\mathbf{b}}^{(\ell)}(t)}}\frac{\partial J(\mathbf{w},\mathbf{b})}{\partial \mathbf{b}^{(\ell)}}(t), \tag{3.10}$$

where $J(\mathbf{w},\mathbf{b})$ is the cost function, $\mathbf{w}$ denotes the synaptic weights, $\mathbf{b}$ denotes the biases, $\mathbf{v}$ denotes the accumulated squared gradients, ℓ indicates the layer of the weights, η denotes the learning rate, and ϵ denotes a small constant ($\sim 10^{-7}$) applied for numerical stability.

As can be seen in Eq. 3.7 to Eq. 3.10, AdaGrad causes the learning rate to decrease over time in accordance with the accumulated squared gradients that will increase over time. In the case where the accumulated squared gradients become too large, the learning will no longer be effective [45].

The RMSProp algorithm [45] is based largely on the AdaGrad algorithm. However, unlike AdaGrad which includes all of the past gradients, RMSProp uses an exponentially decaying average to prevent the accumulation of the past gradients from becoming too large and reducing the effective learning rate. The updates on

the synaptic weights, $\Delta\mathbf{w}$, and biases, $\Delta\mathbf{b}$, at iteration t become,

$$\mathbf{v}_{\mathbf{w}}^{(\ell)}(t) = \rho\mathbf{v}_{\mathbf{w}}^{(\ell)}(t-1) + (1-\rho)\left[\frac{\partial J(\mathbf{w},\mathbf{b})}{\partial \mathbf{w}^{(\ell)}}(t)\right]^2, \tag{3.11}$$

$$\Delta\mathbf{w}^{(\ell)}(t) = -\frac{\eta}{\sqrt{\epsilon + \mathbf{v}_{\mathbf{w}}^{(\ell)}(t)}}\frac{\partial J(\mathbf{w},\mathbf{b})}{\partial \mathbf{w}^{(\ell)}}(t), \tag{3.12}$$

$$\mathbf{v}_{\mathbf{b}}^{(\ell)}(t) = \rho\mathbf{v}_{\mathbf{b}}^{(\ell)}(t-1) + (1-\rho)\left[\frac{\partial J(\mathbf{w},\mathbf{b})}{\partial \mathbf{b}^{(\ell)}}(t)\right]^2, \tag{3.13}$$

$$\Delta\mathbf{b}^{(\ell)}(t) = -\frac{\eta}{\sqrt{\epsilon + \mathbf{v}_{\mathbf{b}}^{(\ell)}(t)}}\frac{\partial J(\mathbf{w},\mathbf{b})}{\partial \mathbf{b}^{(\ell)}}(t), \tag{3.14}$$

where $J(\mathbf{w},\mathbf{b})$ is the cost function, $\mathbf{w}$ denotes the synaptic weights, $\mathbf{b}$ denotes the biases, $\mathbf{v}$ denotes the moving average of the squared gradients, ℓ indicates the layer of the weights, η denotes the learning rate, ϵ denotes a small constant ($\sim 10^{-6}$) applied for numerical stability and $\rho \in [0,1)$ denotes the decay rate for the moving average.

As can be seen in Eq. 3.11 to Eq. 3.14, RMSProp does not always cause the learning rate to decrease as in AdaGrad, as it adapts more quickly to the gradients of recent iterations. It has been empirically proven that RMSProp is more effective and practical than AdaGrad for deep neural networks [45].

The Adam algorithm [45] is designed to combine the benefits of the AdaGrad and RMSProp algorithms [67]. Adam uses the exponentially decaying average of the gradients and the squared gradients to scale the learning rate. The updates on the synaptic weights, $\Delta\mathbf{w}$, and biases, $\Delta\mathbf{b}$, at iteration t become,

$$\mathbf{m}_{\mathbf{w}}^{(\ell)}(t) = \rho_1\mathbf{m}_{\mathbf{w}}^{(\ell)}(t-1) + (1-\rho_1)\frac{\partial J(\mathbf{w},\mathbf{b})}{\partial \mathbf{w}^{(\ell)}}(t), \tag{3.15}$$

$$\mathbf{v}_{\mathbf{w}}^{(\ell)}(t) = \rho_2\mathbf{v}_{\mathbf{w}}^{(\ell)}(t-1) + (1-\rho_2)\left[\frac{\partial J(\mathbf{w},\mathbf{b})}{\partial \mathbf{w}^{(\ell)}}(t)\right]^2, \tag{3.16}$$

$$\hat{\mathbf{m}}_{\mathbf{w}}^{(\ell)}(t) = \frac{\mathbf{m}_{\mathbf{w}}^{(\ell)}(t)}{1 - \rho_1(t)}, \tag{3.17}$$

$$\hat{\mathbf{v}}_{\mathbf{w}}^{(\ell)}(t) = \frac{\mathbf{v}_{\mathbf{w}}^{(\ell)}(t)}{1 - \rho_2(t)}, \tag{3.18}$$

$$\Delta \mathbf{w}^{(\ell)}(t) = -\eta \frac{\hat{\mathbf{m}}_{\mathbf{w}}^{(\ell)}(t)}{\epsilon + \sqrt{\hat{\mathbf{v}}_{\mathbf{w}}^{(\ell)}(t)}}, \tag{3.19}$$

where $J(\mathbf{w}, \mathbf{b})$ is the cost function, $\mathbf{w}$ denotes the synaptic weights, $\mathbf{b}$ denotes the biases, $\mathbf{m}$ denotes the moving average of the gradients, $\mathbf{v}$ denotes the moving average of the squared gradients, ℓ indicates the layer of the weights, $\rho_1 \in [0, 1)$ denotes the decay rate for the moving average of the gradients (usually set to 0.9) [45], $\rho_2 \in [0, 1)$ denotes the decay rate for the moving average of the squared gradients (usually set to 0.999) [45], $\hat{\mathbf{m}}$ denotes the bias correction for $\mathbf{m}$, $\hat{\mathbf{v}}$ denotes the bias correction for $\mathbf{v}$, η denotes the learning rate and ϵ denotes a small constant ($\sim 10^{-8}$) applied for numerical stability.

This set of equations also applies to the biases, $\mathbf{b}$. Adam is known as a robust optimisation algorithm that implements adaptive learning rate for general applications. However, it is sometimes still necessary to adjust the default learning rate, η, empirically [45].

3.2.5 Regularisation

Regularisation is a technique used to prevent overfitting and improve the generalisation capability of a model. Overfitting occurs when a model becomes too complex that it fits the noise of the training data and fails to generalise. It is the opposite of underfitting, which occurs when a model becomes too simple such that it fails to capture the overall representation of the data [36]. Both the underfitting and ovefitting problems can be avoided by controlling the complexity of a model [45]. One way to control the complexity of a model is to use regularisation [45].

Regularisation can be implemented by adding an additional term to the cost function. The regularisation term includes a constant, λ, multiplied by either an

L1 or L2 measure of the weights [45]. The L1 regularisation term is the sum of the absolute value of the weights of each neuron, while the L2 regularisation term is the sum of the squared weights of each neuron. L1 regularisation often results in a sparse model as some weights tend to be shrunk to zero and removed. L2 regularisation minimises the weights without resulting in a sparse model as the weights are minimised according to the quadratic term. Hence, L1 regularisation is typically used for feature selection, while L2 regularisation is generally used for other purposes [36].

The cost function with L1 regularisation [36] is given by,

$$J(\mathbf{w},\mathbf{b}) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log\left(h_k\left(\mathbf{x}^{(i)}\right)\right) + \left(1-y_k^{(i)}\right)\log\left(1-h_k\left(\mathbf{x}^{(i)}\right)\right)\right] + \lambda\sum_{\ell=1}^{L}\sum_{i=1}^{s_\ell}\left|w_i^{(\ell)}\right|, \tag{3.20}$$

where $J(\mathbf{w},\mathbf{b})$ is the cost function, $\mathbf{w}$ is the synaptic weights, $\mathbf{b}$ is the biases, K is the number of output units, m is the number of data samples, $\mathbf{x}^{(i)}$ denotes the i^{th} data sample from a dataset, $h_k(\mathbf{x}^{(i)})$ is the output response of data $\mathbf{x}^{(i)}$ at neuron k, $\mathbf{y}^{(i)}$ is the target label of data $\mathbf{x}^{(i)}$, λ is the regularisation constant, L is the total number of layers in the network and s_ℓ is the number of units in layer ℓ without the bias unit.

The cost function with L2 regularisation [36] is given by,

$$J(\mathbf{w},\mathbf{b}) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log\left(h_k\left(\mathbf{x}^{(i)}\right)\right) + \left(1-y_k^{(i)}\right)\log\left(1-h_k\left(\mathbf{x}^{(i)}\right)\right)\right] + \lambda\sum_{\ell=1}^{L}\sum_{i=1}^{s_\ell}\left(w_i^{(\ell)}\right)^2. \tag{3.21}$$

If λ is too small, the regularisation might not affect the training and the trained model may still suffer from overfitting. On the other hand, if λ is too large, the model

might be overly simplified so that the trained model may suffer from underfitting instead. Thus, λ has to be tuned properly for the best fit of the model with regularisation. The appropriate value of λ can be found by using a cross-validation method, which is a model evaluation method that will be discussed in Section 3.5.

3.3 Data Pre-processing

Data pre-processing is a process that transforms data into a format that suits the input of a network. It can also be used to transform the input data into a different representation to facilitate the learning process, and to achieve a better generalisation capability of a network. In most practical applications, data pre-processing is an important element of a solution that defines the effectiveness of the learning [37].

The most common data pre-processing for neural network applications includes normalisation, enhancement, augmentation and balancing.

3.3.1 Normalisation

Normalisation is a process that rescales data into values of a similar range, usually between 0 to 1. This is particularly important when processing data with different SI units, where actual values may not directly reflect the importance of the data in achieving the desired output [37].

The most basic normalisation method is the min-max normalisation [68] which is expressed as,

$$x' = \frac{x - \min(X)}{\max(X) - \min(X)} \big(\max(X') - \min(X')\big) + \min(X'), \tag{3.22}$$

where x is an element of initial dataset X, x' is the normalised value and X' denotes the normalised set.

The main disadvantage of min-max normalisation is that it may encounter an error if any new data does not fall within the initial input range, i.e. $[\min(X), \max(X)]$. Thus, this method is only suitable for input data with a known and fixed data range [68].

Another type of normalisation is the z-score normalisation [68], known also as standardisation. It centres the data to 0 and divides it by the standard deviation of the data, i.e.,

$$x' = \frac{x - X_\mu}{X_\sigma}, \tag{3.23}$$

where x is an element of set X, x' is the normalised value, X_μ is the mean of set X and X_σ denotes the standard deviation of set X.

As can be expected from Eq. 3.23, the z-score normalisation solves the problem of the "out-of-bounds" error in min-max normalisation. In image data, X_μ and X_σ can be calculated globally across all channels (e.g. red, green and blue channels in an RGB image) or locally per channel across the data in a mini-batch or the entire training dataset [64].

3.3.2 Enhancement

Data enhancement is a technique used to improve the representation of the data for a particular task. It can be applied to any type of data, including image data. Image enhancement can be performed by modifying attributes, such as the contrast, brightness or intensity distributions, and can be applied in both the spatial and frequency domains to achieve the desired results. In the case of its application in the spatial domain, the enhancement operation is applied directly on the pixel values of an image. For its application in the frequency domain, the enhancement operation is applied on the Fourier transform of an image [69].

Point Operations

A point operation is a type of an image enhancement operation where the output at each pixel location is generated solely based on the pixel at the corresponding location in the input image, i.e. the operation is independent to its neighbouring pixels [70]. Contrast stretching [70] is a basic point operation that uses a piecewise-linear transformation function to stretch the pixel range of an image so that it occupies the full pixel range, e.g. [0, 255] for an 8-bit image. Contrast stretching

increases the dynamic range of the intensity levels of a low-contrast image [71]. The operation is,

$$\mathbf{y}(i,j) = \frac{\mathbf{x}(i,j) - \min(\mathbf{x})}{\max(\mathbf{x}) - \min(\mathbf{x})} (\max(\mathbf{y}) - \min(\mathbf{y})) + \min(\mathbf{y}), \tag{3.24}$$

where $\mathbf{y}$ is the output image, $\mathbf{x}$ is the input image, i, j are the row and column index respectively that represent the location of a pixel in the image. An example of the contrast stretching operation is shown in Figure 3.1. The resulting image has a higher contrast than the input image, hence more details can be observed in the image.

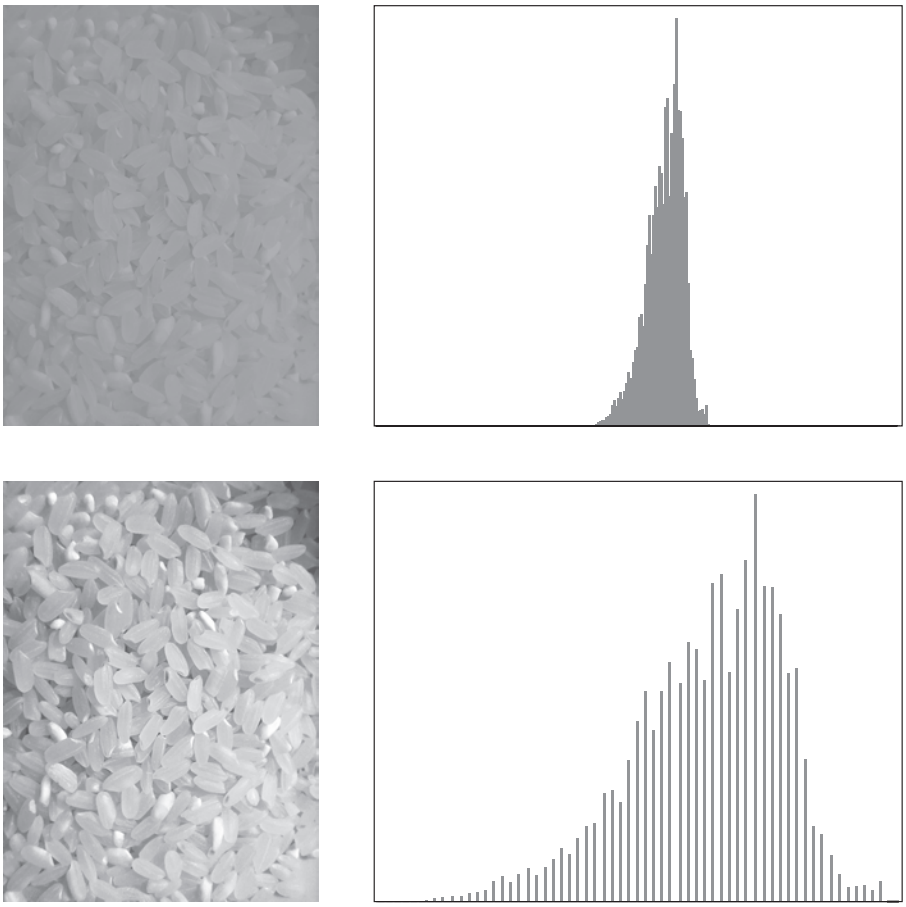


Figure 3.1: Example of a contrast stretching operation. The image and the histogram of the pixel values in the image of the input (top) and output (bottom).

Contrast stretching can also be performed with thresholding so that the output image contains only two intensity levels with extremely high contrast [70]. This operation is usually performed on a greyscale image and expressed as,

$$\mathbf{y}(i,j) = \begin{cases} 255 & \text{if } \mathbf{x}(i,j) \geq T \\ 0 & \text{if } \mathbf{x}(i,j) < T, \end{cases} \tag{3.25}$$

where $\mathbf{y}$ is the output image, $\mathbf{x}$ is the input image, i, j are the row and column index, respectively that represent the pixel location in the image and T is the threshold value.

The threshold value can be determined by various threshold selection techniques, including grey-level histogram analysis and grey-level averaging [70]. An example of a contrast stretching operation with a threshold that is determined based on a grey-level histogram analysis is shown in Figure 3.2.

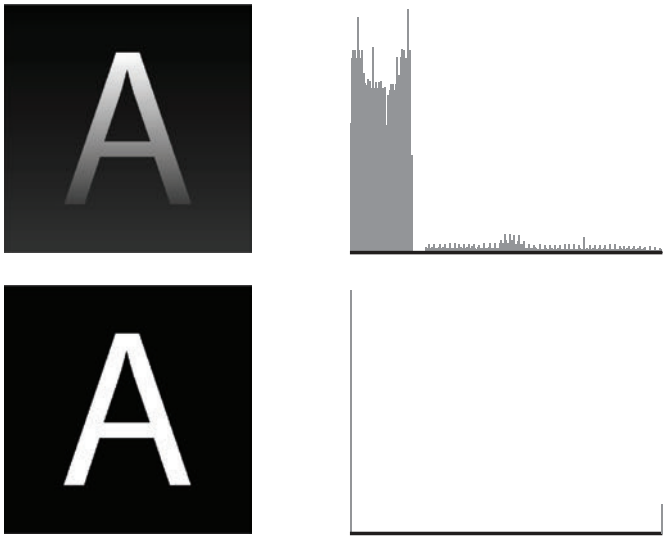


Figure 3.2: Example of a contrast stretching operation with thresholding. The image and the histogram of the pixel values in the image of the input (top) and output (bottom).

Another contrast enhancement operation is called histogram equalisation [71]. The objective of histogram equalisation is to modify the image so that the histogram of the output image is approximately uniform [72]. The histogram equalisation operation is,

$$\mathbf{y}(i,j) = \left\lfloor (K-1) \sum_{k=0}^{\mathbf{x}(i,j)} \frac{c_{\mathbf{x}}(k)}{MN} \right\rfloor, \tag{3.26}$$

where $\mathbf{y}$ is the output image, $\mathbf{x}$ is the input image, M and N are the height and

width of the image in pixels respectively, K is the number of intensity levels used for the pixel values and $c_{\mathbf{x}}$ is the total number of a specific pixel value in the input image.

As can be seen from the example of a histogram equalisation operation in Figure 3.3, the equalised image has a higher contrast than the input image.

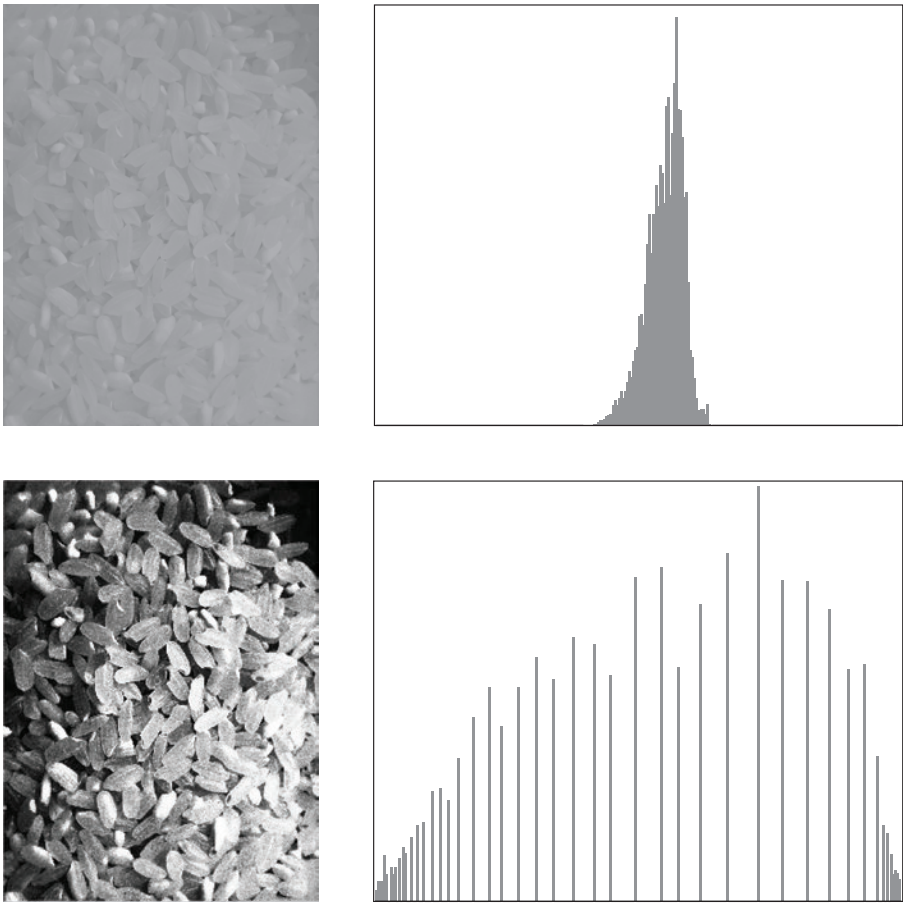


Figure 3.3: Example of a histogram equalisation operation. The image and the histogram of the pixel values in the image of the input (top) and output (bottom).

Neighbourhood Operations

A neighbourhood operation, generally known as a filtering operation, is a type of image enhancement operation where the output at each pixel location is generated based on the pixel at the corresponding location and the neighbouring pixels in the input image [70]. Filters are usually categorised into two types, linear and non-linear.

Linear Filter A linear filter combines the pixel values in a linear fashion using convolution. The output at each pixel location is generated by convolving a filter mask or kernel with the pixel at the corresponding location and its neighbouring pixels in the input image [70]. The filter mask or kernel specifies the size and shape of the input image region to be used for each computation, along with the individual pixel weights. The convolution operation [45] is,

$$\mathbf{y}(i,j) = (\mathbf{k} * \mathbf{x})(i,j) = \sum_m \sum_n \mathbf{x}(i-m, j-n)\mathbf{k}(m,n), \tag{3.27}$$

where $*$ is the convolution operator, $\mathbf{y}$ is the output, $\mathbf{x}$ is input image, $\mathbf{k}$ is the matrix of the filter kernel and m and n are the row and column index of the filter kernel, respectively, with the centre of the kernel indicated by (0,0). Note that the input image is padded with zeros to maintain the dimension of the output.

Unlike a convolution operation in CNNs that actually is a cross-correlation operation [45], the filter kernel of the convolution operation is rotated by 180° about its centre element, hence the negative sign in $\mathbf{x}(i-m, j-n)$, before performing the multiplication and summation. However, if the filter kernel is symmetric, the rotation can be omitted.

There are various types of linear filters. Two of the most commonly used linear filters in data pre-processing for neural networks are the smoothing and difference filters [73].

A smoothing filter computes a weighted average of the input image pixels in the region of the filter kernel and produces an image with smoothed edges as shown in Figure 3.4. This filter comprises of only positive weight coefficients [73].

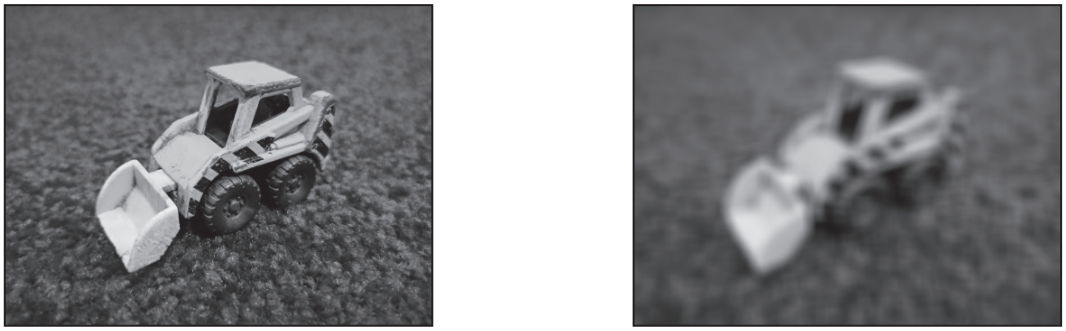


Figure 3.4: Example of a smoothing filter application. The input image (left) and the output of the smoothing filter (right).

The simplest smoothing filter is called a box filter. It is called a box filter as its shape is similar to a box [73]. The filter has positive weights coefficients with equal values. An example of a box filter is shown in Figure 3.5.

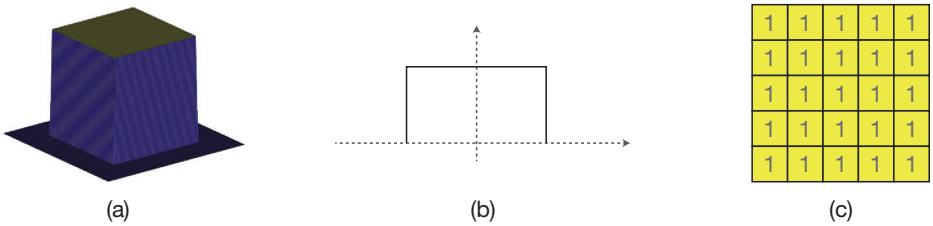


Figure 3.5: Example of a box filter. (a) 3D illustration; (b) Profile; (c) Discrete filter matrices approximations of the continuous function in (a).

One of the applications of a box filter is as an averaging filter. By setting all the filter coefficients to be 1 and dividing the convolution result by the sum of the filter coefficients, the output will be equal to the mean of the pixel values within the filter region [71].

Since the box filter has sharp cut-offs around its edges, it produces transients or strong "ringing" effects in the frequency domain [73]. To reduce this unwanted effect, a Gaussian filter is preferred as a smoothing filter as it is "well-behaved" in the frequency domain, since the filter places more emphasis on the pixels in the centre and less emphasis on the distant pixels, as shown in Figure 3.6.

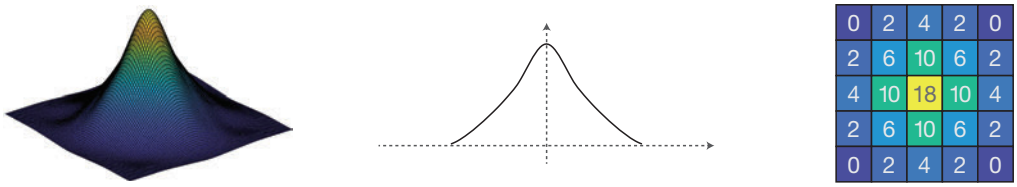


Figure 3.6: Example of a Gaussian filter. (a) 3D illustration; (b) Profile; (c) Discrete filter matrices approximations of the continuous function in (a).

The coefficients of a two-dimensional Gaussian filter [73] can be determined by,

$$\mathbf{k}(m,n) = e^{-\frac{m^2+n^2}{2\sigma^2}}, \tag{3.28}$$

where σ is the standard deviation of the bell-shaped function, m and n are the number of rows and columns from the centre of the filter respectively.

A difference filter, e.g. the "Laplace" or "Mexican hat" filter, is another type of linear filter that computes the weighted difference between the centre and the surrounding pixels. The weighted difference is performed by having positive coefficients at the centre of the filter and negative coefficients surrounding the centre, or vice versa, as shown in Figure 3.7 [73].

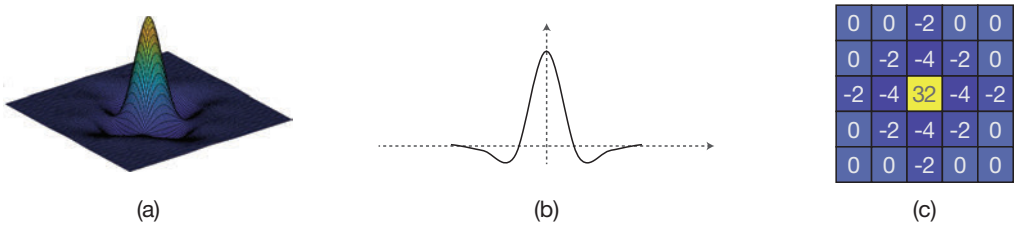


Figure 3.7: Example of a Laplacian filter. (a) 3D illustration; (b) Profile; (c) Discrete filter matrices approximations of the continuous function in (a).

This type of filter enhances the local intensity discontinuities and diminishes the regions with slowly varying intensity levels, which results in light edge lines and dark background. Therefore, it is usually used for edge detection as shown in Figure 3.8.

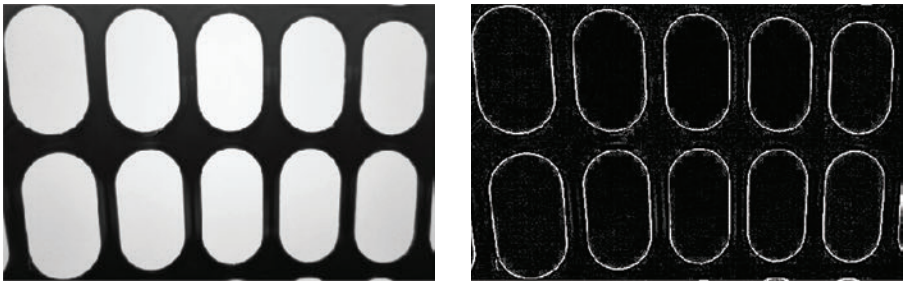


Figure 3.8: Example of a Laplacian filter application for edge detection. The input image (left) and the output of the Laplacian filter (right).

Non-linear Filter Similar to the linear filters, non-linear filters generate each output pixel from the input pixels within the filter kernel region. However, instead of using a linear function, non-linear functions are used to generate the output. Non-linear filters are usually applied to remove impulse noise, also known as salt and pepper noise [71], that appears as black and white dots scattered on an image.

The simplest non-linear filters are the minimum and maximum filter [73]. The minimum filter assigns each output pixel with the minimum pixel value of the input pixels within the filter kernel region [73], as defined by,

$$\mathbf{y}(i,j) = \min\left\{\mathbf{x}(i+m, j+n) \mid (m,n) \in R\right\}, \tag{3.29}$$

where $\mathbf{y}$ denotes the output, $\mathbf{x}$ denotes the input image and R denotes the filter kernel region. Note that the input image is padded with constant values to maintain the dimension of the output.

The maximum filter assigns each output pixel with the maximum pixel value of the input within the filter kernel region [73], as defined by,

$$\mathbf{y}(i,j) = \max\left\{\mathbf{x}(i+m, j+n) \mid (m,n) \in R\right\}. \tag{3.30}$$

Another type of non-linear filter is the median filter. The median filter assigns each output pixel with the median pixel value of the input within the filter kernel region [73], as defined by,

$$\mathbf{y}(i,j) = \text{median}\left\{\mathbf{x}(i+m, j+n) \mid (m,n) \in R\right\}. \tag{3.31}$$

The median pixel is computed by sorting the pixel values within the filter kernel region in ascending sequence and then picking the middle value (for an odd number of elements) or the mean of the two middle values (for an even number of elements).

3.3.3 Augmentation

Data augmentation is a technique used for creating new training data from an initial training dataset to improve the generalisation capability of a network [64]. Its main objective is to expand the initial training set with new plausible examples, i.e. to increase the representation of various data samples that are likely to be seen in the future. Data augmentation can be applied to any type of data, including image data. Image augmentation includes various operations, such as shifting, flipping, rotating, zoom, and it is performed while keeping the dimension of the image fixed [64]. These operations can be implemented in many different ways depending on the desired learning outcome.

The shifting operation, also known as a translation operation, moves all pixels of an image in either a horizontal or vertical direction [64]. This is a useful operation as objects can be located at various locations in an image. The objective is to integrate a translation invariance property into the model, so that it can recognise an object regardless of its location in an image.

Flipping and rotating operations vary the viewpoint of an object. The flipping operation, also known as a mirroring operation, reverses the pixel arrangement in either a horizontal or vertical direction [64]. While the rotating operation randomly rotates an image by a specific angle, between 0 to 360 degrees [64]. The objective is to integrate a viewpoint invariance property into the model, so that it can recognise an object regardless of the angle from which an object is viewed from.

A zoom operation randomly zooms an image by either adding new pixel values around the image, i.e. zoom-out, with an extrapolation technique or adding new pixel values within the image, i.e. zoom in, with an interpolation technique [64]. This operation is useful for most objects, as size varies depending on the viewing distance or the type of object itself. The objective is to integrate a size invariance property into the model so that it can recognise an object regardless of the size of

the object in an image.

There are many more image augmentation techniques, including those that may be designed specifically for a certain task. The use of different image augmentation techniques should be considered on a case by case basis, depending on the type of images and the desired learning outcome. For example, it is unnecessary to apply random flipping and rotating operations on medical images as they have a specific coordinate system that can be used to calibrate all the images to a certain orientation.

3.3.4 Balancing

A class imbalanced dataset (i.e. skewed data distributions) occurs frequently in many real-world applications. However, in training a neural network, it is very important to ensure that both the minority and majority classes are well represented. Data balancing is a method of balancing the proportion of data samples in a training set. Data balancing is performed so that each class, whether it is within (intra-class) or between classes (inter-class), has enough representation in the training set [74] [75]. It is one of the most critical steps in training a neural network. There are two basic techniques that can be performed automatically, the data-level method and the cost sensitive learning method.

The data-level method is a technique that modifies the training distribution to balance out the data samples from different classes [74]. It includes two basic techniques, random under-sampling (RUS) and random over-sampling (ROS) [74]. The RUS removes random data samples from the majority class, while ROS duplicates random data samples in the minority class. The main disadvantage of this method is that under-sampling reduces the amount of information for the model to learn from, while over-sampling may increase the training time due to the size increase of the training set.

Cost sensitive learning is a method that assigns different cost values for the misclassification of different class samples, e.g. the cost of misclassifying an infected person as a healthy person is higher than the contrary [76]. Although it can significantly increase the classification performance, it can only be applied if the

misclassification cost values are known [76]. The misclassification cost value can be estimated with various methods, including the use of the inverse ratio of the class representation [77], e.g. if the ratio of infected persons to healthy persons is equal to 1:10, the cost of misclassifying an infected person as a healthy person should be ten times higher than the contrary. The implementation of cost sensitive learning varies on a case by case basis.

3.4 Data Post-processing

Data post-processing is a process that transforms output data into the required format. It may also be used to enhance or clean the output data to achieve a better overall performance. The process often incorporates prior knowledge of the desired output, such as the shape and minimum size of a target class in a segmentation task [37].

The most fundamental data post-processing processes for image segmentation includes mathematical morphology and connected component labelling operations.

3.4.1 Mathematical Morphology

Mathematical morphology is a set of operations used to analyse and process morphological structures in an image that is based on set theory and topology [70]. The most common set of operations include the hit or miss transformation as well as erosion and dilation. These operations use a set of points called the structuring element, denoted by B_x, that is centred at x to extract structure in a set, X.

Hit or miss transformation is usually defined as a point by point transformation, which means that the structuring element, B_x, has to match exactly, i.e. hits, the set X, as illustrated in Figure 3.9. The hit or miss transformation [70] of a set X with B is denoted by $X \otimes B$ and its operation is defined as,

$$X \otimes B = \left\{ x \mid B_x^1 \subset X; B_x^2 \subset X^{\complement} \right\}, \tag{3.32}$$

where $\subset$ denotes set inclusion, $\complement$ denotes complement, B_x^1 denotes the subset of B_x whose element belongs to the foreground and B_x^2 denotes the subset of B_x whose

element belongs to the background.

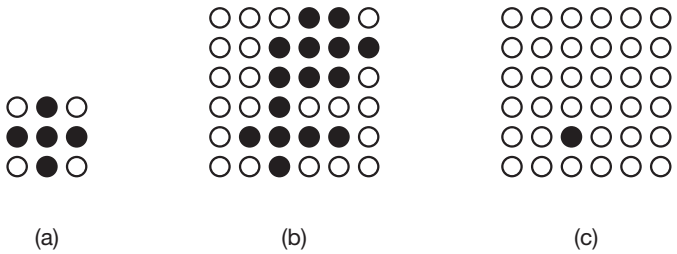


Figure 3.9: Example of a hit or miss operation. (a) Structuring Element B; (b) Image set X; (c) Result of hit or miss operation $X \otimes B$.

Erosion has a similar concept to the hit or miss transformation except that it does not include the background elements in the equation. The erosion of a set X with B is denoted as $X \ominus B$. Its operation [70] [78] is illustrated in Figure 3.11 and is defined by,

$$X \ominus B = \{x \mid B_x \subset X\}. \tag{3.33}$$

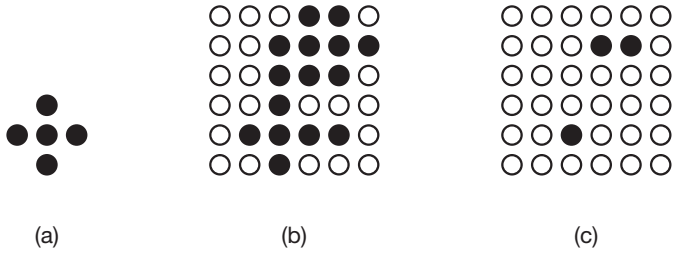


Figure 3.10: Example of an erosion operation. (a) Structuring Element B; (b) Image set X; (c) Result of the erosion operation $X \ominus B$.

Dilation is the dual of the erosion operation, which means that the dilation of the foreground is equal to the erosion of the background. The dilation of a set X with B is denoted by $X \oplus B$. Its operation [70] [78] is illustrated in Figure 3.11 and is defined by,

$$X \oplus B = \left(X^{\complement} \ominus B\right)^{\complement}. \tag{3.34}$$

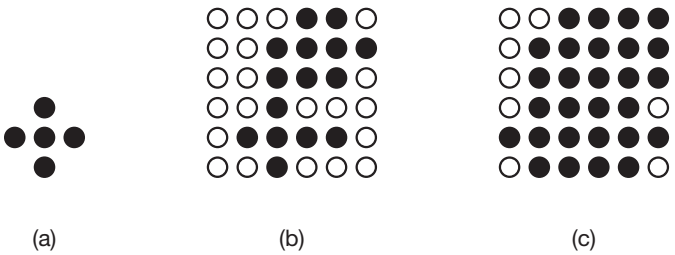


Figure 3.11: Example of a dilation operation. (a) Structuring Element B; (b) Image set X; (c) Result of the dilation operation $X \oplus B$.

The combination of the erosion and dilation operations results in applications such as opening and closing.

A binary opening is erosion followed by a dilation operation. The binary opening of a set X with B is denoted by $X \circ B$. Its operation [70] [71] [78] is defined by,

$$X \circ B = (X \ominus B) \oplus B. \tag{3.35}$$

The erosion operation eliminates foreground structures that are smaller than the structuring element, while the dilation operation restores the remaining structures to their original size.

A binary closing is dilation followed by an erosion operation. The binary closing of a set X with B is denoted by $X \bullet B$. Its operation [70] [71] [78] is defined by,

$$X \bullet B = (X \oplus B) \ominus B \tag{3.36}$$

The dilation operation closes holes and fissures that are smaller than the structuring element in the foreground structures, while the erosion operation reduces the dilated structures to their original size.

3.4.2 Connected Component Labelling

The connected component labelling operation scans an image and assigns labels to each pixel depending on its connectivity with the neighbouring pixels. It is usually applied to detect the number of connected components in an image, e.g. a binary image. It outputs a symbolic image that contains a label assigned to each pixel of the

input image. The operation [79] [80] usually involves examining n-neighbourhood pixels with an n-connected components operator, where n denotes the number of pixels used to determine the label of a pixel.

The most basic connected component operators are the 4-connected and 8-connected component operators, shown in Figure 3.12 (a) and Figure 3.12 (b) respectively. Examples of the application of a 4-connected and 8-connected components operator on a binary image are shown in Figure 3.13 (b) and Figure 3.13 (c) respectively. As can be seen from the examples, a new label is only assigned to the pixel in the centre, •, of the n-connected components operator when the neighbouring pixels, ○, have not been assigned with any label, otherwise it will be assigned with the existing label of the n-neighbourhood pixels.

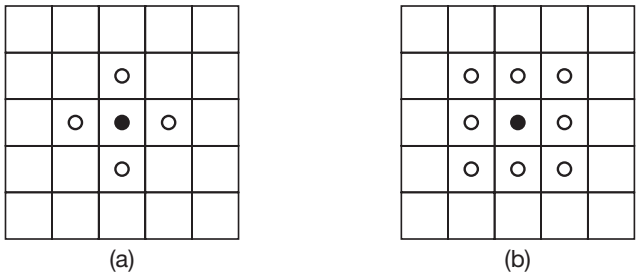


Figure 3.12: Example of n-connected components operators, where • denotes the centre pixel and ○ denotes the neighbouring pixels used in the n-connected component operator. (a) 4-connected component operator; (b) 8-connected component operator.

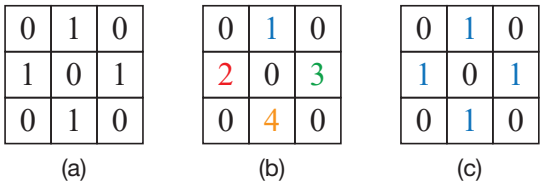


Figure 3.13: Example of connected component labelling. (a) The binary input image; (b) Connected component labelling results produced by the 4-connected component operator on (a); (c) Connected component labelling results produced by the 8-connected component operator on (a).

The connected component labelling operation can be used to remove output artefacts that do not belong to the desired output structure.

3.5 Evaluation Method

In the implementation of a neural network, once a model is developed, it is important to evaluate the model to ensure that it is able to generalise. Cross-validation and holdout methods are model evaluation methods that are generally adopted to evaluate the generalisation capability with respect to a performance measure, such as the cross-entropy cost function (Eq. 2.8).

In cross-validation, a dataset is often partitioned into two subsets, i.e. a training and a test set [36]. Generally, a ratio of 80% to 20% is adopted for partitioning the data into training and test sets, respectively. The training set is used to train the model while the test set is used to evaluate the performance of the model on an unseen set of data. If the amount of data is sufficient, the training set can be further partitioned into training and validation sets with a ratio of 75% to 25%. The validation set can be used to evaluate the model on an unseen dataset during training for hyperparameter tuning and model selection.

The data partitioning in the cross-validation method is generally performed repeatedly with the validation results averaged over the repetitions [36]. Based on the way the data is partitioned for the validation, there are two main types of cross-validation, i.e. exhaustive and non-exhaustive [36].

In the exhaustive cross-validation method, a fixed number of data samples is used as the test set and the remaining data samples are used as the training set for each validation round. It is repeated $\frac{n!}{p!(n-p)!}$ times, i.e. for every possible partitioning of the dataset, where n denotes the total number of data samples in the dataset and p denotes the number of data samples to be used as the test set [81]. This method is named according to the number of data samples used as the test set, e.g. leave-p-out cross-validation for $p > 1$ and leave-one-out cross-validation for $p = 1$. The main disadvantage of this method is that it can be very slow [36].

In the non-exhaustive cross-validation method, the dataset is partitioned into certain partitions, i.e. folds, and not all possible partitions are considered. The non-exhaustive cross-validation method includes k-fold cross-validation. In k-fold cross-validation, the dataset is partitioned into k equal-sized folds ($k > 1$), where

one fold of data is used as the test set and $k-1$ folds of data are used as the training set. The validation method is repeated k times, with each fold of the data being the test set once only [36]. The common values [36] for k are 3, 5 and 10.

In the hold out method, the data is partitioned like in a k-fold cross-validation method, but the data validation is only performed once instead of k times to estimate the performance of a model. The main disadvantage of this method is that not all data samples will be in the test set [36].

The k-fold cross-validation method is the most applicable as it takes a reasonable time to run and gives a better insight into the model generalisation performance than the hold out validation method [36].

3.6 Conclusion

In this chapter, we discussed some fundamental considerations which need to be made to ensure an effective implementation of a neural network. We introduced the most common hyperparameters and algorithms that can be utilised in a neural network. We also discussed several data pre- and post-processing techniques that can be employed to enhance the input and output to improve the overall performance of the network. Finally, we presented several evaluation methods that can be used to demonstrate the generalisation capability of the developed model.

CHAPTER

4

CNN Structure and Parameter Study for a Liver Segmentation Problem

In this chapter, we propose a two-dimensional CNN for an organ segmentation task that incorporates a novel adjacent upsampling method, which is trainable and only requires a single computation step to perform upsampling. We first identify some limitations of typical segmentation networks and then propose an alternative that performs well in comparison. We show that with the appropriate structure, a CNN doesn't require an excessive number of trainable parameters in order to solve a particular problem. We also show that a CNN can be deployed and trained efficiently with limited resources.

4.1 Introduction

As discussed in Section 1.1, the CNN is the state-of-the-art deep learning algorithm in image segmentation for many different applications. In Section 2.4.3, it was noted that a major achievement in image segmentation [17] was made by the modification of a very deep classification CNN, the VGG network, to an FCN for image segmentation. Although the FCN is slow in real-time execution at high resolution, it is a very successful CNN for image segmentation [82]. Since the development of the FCN, many other networks have been developed using the structure and components of a classification architecture to perform image segmentation irrespective of the classification CNNs' inherent spatial invariance characteristic [19].

When CNNs are used for image classification, pooling layers are typically employed to decrease the computational cost in the deeper layers and to introduce spatial invariance in the network by extracting the most significant features from the previous layer. This means that spatial information is lost in CNNs designed for image classification [83]. However, when designing a CNN for an image segmentation task the spatial information needs to be retained during the feature extraction phase to predict the class of each pixel. Most networks (e.g. FCN [17], U-Net [18], Cascaded FCNs [84], FCN-Based [85], U-Net and DenseNet-Based [86]) still adopt the combination of convolutional layers and max-pooling layers without addressing this problem. On the other hand, V-Net [47] and networks based on V-Net [87] avoid the use of max-pooling and use 2-strided convolutional layers instead. However, their architecture fails to take full advantage of the retained spatial information because they still use a contracting-expanding structure to learn the features at the original input resolution to perform image segmentation.

To address the spatial invariance and inefficient network structure problems in CNNs, we propose a simple network that uses an efficient upsampling method [88] to optimise the use of the extracted high-level features to perform image segmentation. Data from a well established liver segmentation task [89] is used as a benchmark to evaluate the performance of the proposed network and other state-of-the-art networks in performing liver segmentation.

4.2 Related Work

In this section, we will briefly discuss several techniques and components used in our proposed network. We will also discuss the receptive field calculation and sub-pixel convolution method that motivate the proposed upsampling method.

Strided Convolution

As discussed in Section 2.3.2, pooling layers in the form of either linear (e.g. average pooling or strided convolution) or non-linear (e.g. max pooling) filters can be adopted in CNNs to incorporate downsampling in a network. However, pooling layers based on non-linear filters may cause the network to lose spatial information [83]. Pooling layers based on a linear filter, such as strided convolution, have been used in several high performing segmentation network architectures [47] [87]. It has also been shown that a stack of standard 1-strided and a 2-strided convolutional layers can be used in place of a 2×2 max pooling layer to improve the performance of a CNN [90].

Large Kernel

In performing image segmentation, a well-designed CNN architecture should be able to perform classification and localisation at the same time [91]. Each pixel should be classified correctly as well as aligned in accordance to the pixel coordinate in the input image. However, to perform a classification task, the network output has to be transformation invariant. On the other hand, to perform a localisation task, the network output has to be transformation sensitive [91].

It is shown that the use of large kernels can help to mitigate the contradiction between classification and localization [91]. For example, the use of a 5×5 kernel results in better image segmentation performance as compared to the use of a 3×3 convolution kernel [91].

Dropout

As discussed in Section 2.3.2, dropout is a regularisation technique used in CNNs to prevent overfitting. A dropout rate between 0.2 - 0.5 is usually adopted in a segmentation network [86] [92]. However, a high dropout rate may cause underfitting in the network. Thus, a typical dropout rate of 0.2 is preferred for a starting point [93].

Deep Networks with Residual Learning

As discussed in Section 2.3.2, a skip connection with an element-wise summation layer can be implemented to incorporate a residual learning framework in a CNN. The skip connection with an element-wise summation layer can be implemented in every convolution block [52] [47]. Here, a convolution block refers to a block that consists of two or more convolutional layers with the same feature map resolution.

Receptive Field

Receptive field (RF) refers to the filter size of a layer, which specifies the region that a neuron is connected to in its previous layer [94]. Effective receptive field (ERF) refers to the area of the input data that can influence the activation of a neuron in the network. Thus, both RF and ERF of the first convolutional layer are the same, whereas the ERF of each layer grows larger as the network becomes deeper, as briefly discussed in Section 2.3.1.

The ERF of each layer relative to the network input [38] is a function of the receptive fields of all previous layers and it can be expressed as,

$$\mathrm{ERF}^{(\ell)} = \mathrm{ERF}^{(\ell-1)} + \left(f^{(\ell)} - 1\right) \times \prod_{i=1}^{\ell-1} s^{(i)}, \tag{4.1}$$

where ℓ denotes the index of the layer, $f^{(\ell)}$ denotes the $f \times f$ filter size in a convolutional layer used to produce feature maps in layer ℓ, $s^{(\ell)}$ denotes the stride length in a convolutional layer used to produce feature maps in layer ℓ and the effective receptive field of the previous layer is denoted as $\mathrm{ERF}^{(\ell-1)}$.

Other than filter size and stride length, as indicated in Eq. 4.1, padding also influences the ERF of a feature in a layer. As discussed in Section 2.3.2, padding is usually applied in a CNN to maintain the output resolution of a layer. Padding also produces extra features at the corner and edges of the output of a convolutional layer, i.e. the feature maps. However, the ERFs at the corner and edges are smaller than the ERFs at any other place. Thus, to find the minimum ERF of a layer with padding, the following equation can be used,

$$\mathrm{ERF}_{min}^{(\ell)} = \left\lfloor \frac{\left(\mathrm{ERF}^{(\ell-1)} + \left(f^{(\ell)} - 1\right) \times \prod_{i=1}^{\ell-1} s^{(i)}\right)}{2} \right\rfloor + 1. \tag{4.2}$$

The RF and ERF in a 2-layer CNN with a 3×3 filter size, stride length of 1, with and without the use of padding are shown in Figure 4.1 (top) and (bottom) respectively. As can be seen from the figure the receptive fields of both convolutional layers, $\mathrm{RF}^{(1)}$ and $\mathrm{RF}^{(2)}$, are 3×3 as both layers use the same filter size. The effective receptive field of layer one, $\mathrm{ERF}^{(1)}$, is the same with its receptive field, $\mathrm{RF}^{(1)}$. The effective receptive field of layer two, $\mathrm{ERF}^{(2)}$, is increased to 5×5. As can be seen in Figure 4.1 (top), the outer values of the feature map in layer one, $\boldsymbol{a}^{(1)}$, and two outer values of the feature map in layer two, $\boldsymbol{a}^{(2)}$, the padding produces extra information at the corner and edges of the output of the convolutional layers as compared to Figure 4.1 (bottom).

However, the ERF of the features resulting from the padded input is smaller than the ERF calculated from Eq. 4.1. As can be seen from Figure 4.1 (top), the ERF of the very first feature in layer two, $\boldsymbol{a}^{(2)}$, (with value of -87) is only of size 3×3, which is the $\mathrm{ERF}_{min}^{(\ell)}$ from Eq. 4.2, instead of the calculated ERF of 5×5, while the ERF of the second feature in layer two, $\boldsymbol{a}^{(2)}$, (with value of 152) is only of size 3×4. Therefore, in the application where it is important to know the ERF of every feature in a feature map, minimum ERF should be considered when calculating the ERF in a network which uses padding.

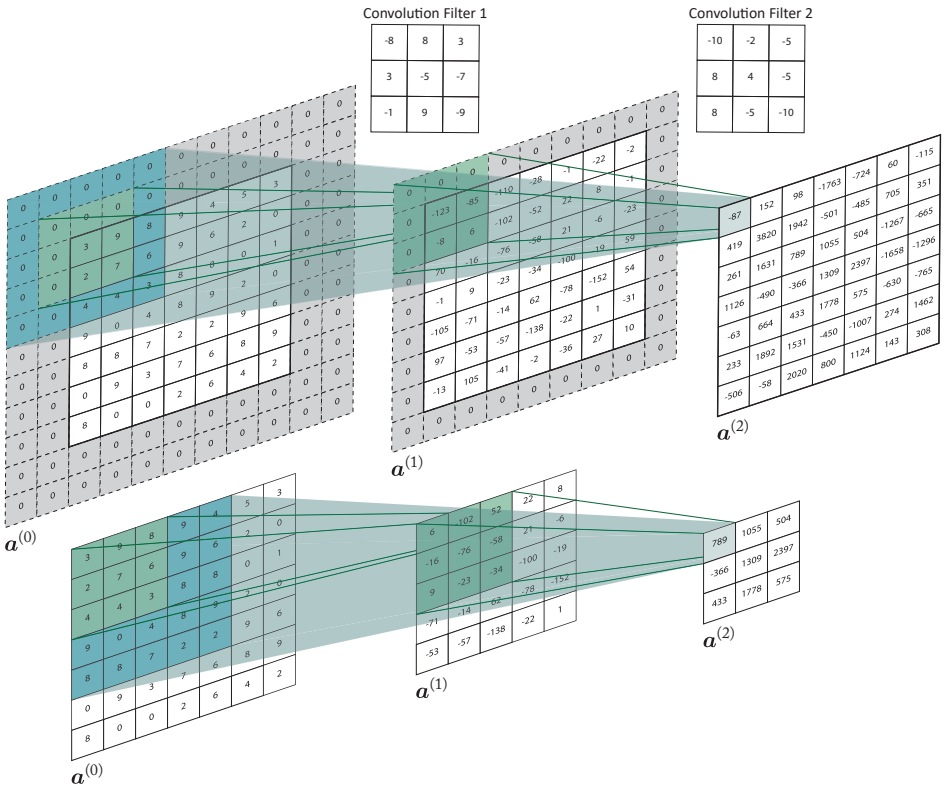


Figure 4.1: Receptive field in a 2-layer CNN with padding (top) and without padding (bottom), where the input layer is denoted as $\boldsymbol{a}^{(0)}$, the output due to convolution filters 1 and 2 are denoted as $\boldsymbol{a}^{(1)}$ and $\boldsymbol{a}^{(2)}$ respectively. The green area indicates the receptive field region of each convolutional layer ($\text{RF}^{(1)}$ and $\text{RF}^{(2)}$), while the blue area indicates the effective receptive field region of the second convolutional layer ($\text{ERF}^{(2)}$) on the input layer.

Sub-pixel Convolution

Sub-pixel convolution is an upsampling method that can be used to increase the resolution of the input. Similar to the transposed convolution method, discussed in Section 2.3.2, sub-pixel convolution parameters can be adjusted through training via the backpropagation.

Sub-pixel convolution can be naively interpreted as a convolution followed by a periodic shuffling operation [88] to form a higher resolution output [51]. The advantage of a sub-pixel convolution is that it is more efficient than a transposed convolution or resize convolution [51].

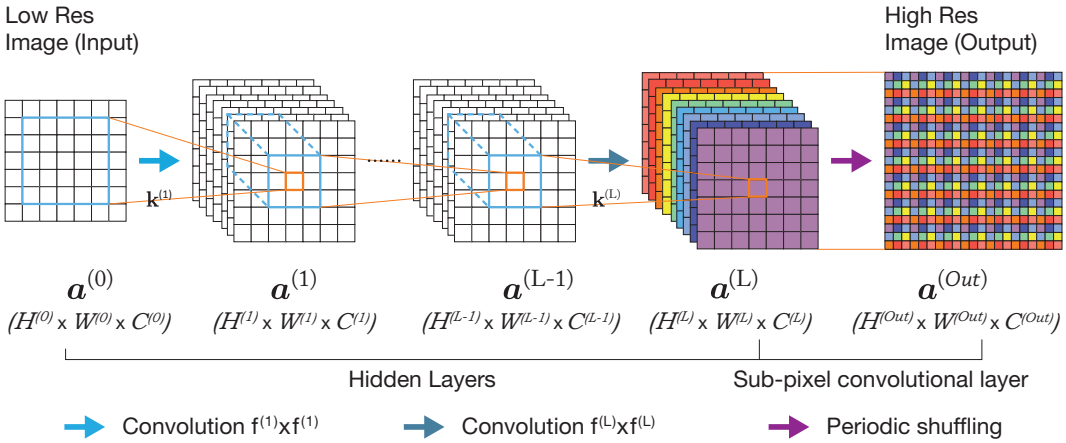


Figure 4.2: Example of a sub-pixel convolutional layer in a sub-pixel convolutional neural network used to upsample low resolution feature maps. The sub-pixel convolutional layer produces a number of feature maps in low resolution space and rearranges it to build a higher resolution image in a single step. The $H^{(\ell)}$, $W^{(\ell)}$ and $C^{(\ell)}$ indicate the height, width and channels of the feature maps in each layer, respectively, while L indicates the last layer.

The sub-pixel convolution operation used on low resolution feature maps extracted from a low resolution input image, $\boldsymbol{a}^{(0)}$, to produce a higher resolution output image, $\boldsymbol{a}^{(Out)}$, as in Figure 4.2 can be expressed as follows,

$$\boldsymbol{a}^{(Out)} = \mathrm{PS}\left(\boldsymbol{a}^{(L)}\right) = \mathrm{PS}\left(\mathbf{k}^{(L)} * \boldsymbol{a}^{(L-1)} + b^{(L)}\right) \tag{4.3}$$

where PS denotes the periodic shuffling operation [88] that rearranges the elements in a tensor of $H^{(L)} \times W^{(L)} \times C^{(Out)} \cdot r^2$ to a tensor of $r \cdot H^{(L)} \times r \cdot W^{(L)} \times C^{(Out)}$, i.e. $C^{(L)} = C^{(Out)} \cdot r^2$, $H^{(Out)} = r \cdot H^{(L)}$ and $W^{(Out)} = r \cdot W^{(L)}$, r is the upsampling factor, $\mathbf{k}^{(L)}$ is the convolution kernel used to produce the last layer, $b^{(L)}$ is the bias of the convolution kernel in the last layer and $\boldsymbol{a}^{(L-1)}$ are the feature maps in the layer before the last layer.

4.3 Proposed Adjacent Upsampling

In this section, we explain our proposed adjacent upsampling method.

Similar to sub-pixel convolution, adjacent upsampling adopts trainable upsampling parameters and requires a single computation step to perform upsampling. However, unlike sub-pixel convolution where it uses an arbitrary size

of convolution kernel prior to the periodic shuffling operation, adjacent upsampling requires the use of a one dimensional convolution operation to perform the prediction of the corresponding pixel and its adjacent pixels. The idea is that it uses the information from the effective receptive field covered by the highest level features (i.e. lowest resolution feature maps) of the network to make single step predictions of the corresponding pixel and its adjacent pixels without the use of any overlapping features. Hence, the information in each feature location of the lowest resolution feature maps will be responsible for the final predictions of the pixel and its adjacent pixels at the corresponding feature location.

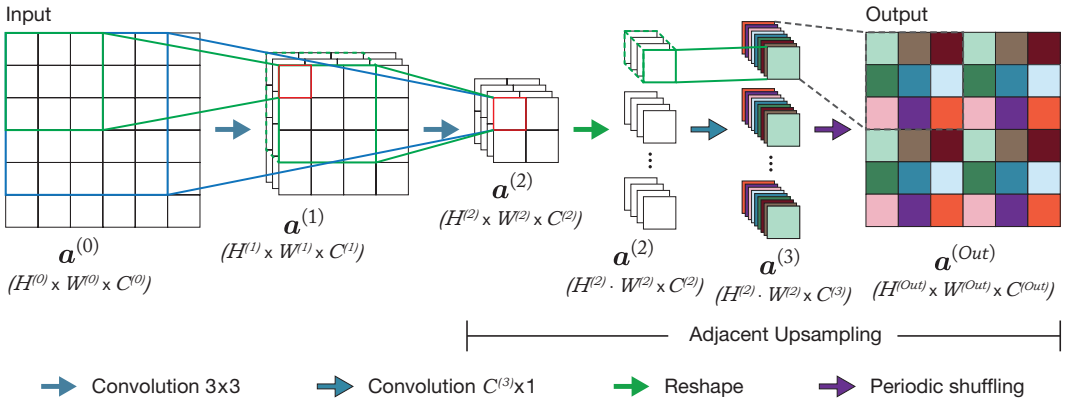


Figure 4.3: Example of an adjacent upsampling operation.

In Figure 4.3, the input layer is denoted as $\boldsymbol{a}^{(0)}$, the output of the first and second 3×3 convolutional layers are denoted as $\boldsymbol{a}^{(1)}$ and $\boldsymbol{a}^{(2)}$ respectively, while the output of the one dimensional convolution used for the adjacent upsampling operation is denoted as $\boldsymbol{a}^{(3)}$ and the final output is denoted as $\boldsymbol{a}^{(Out)}$. The green border indicates the receptive field region of each convolutional layer ($\text{RF}^{(1)}$ and $\text{RF}^{(2)}$), while the blue border indicates the effective receptive field region of the second convolutional layer ($\text{ERF}^{(2)}$) on the input layer.

Adjacent upsampling begins with an effective receptive field calculation to ensure that the lowest resolution feature maps, i.e. $\boldsymbol{a}^{(2)}$ in Figure 4.3, cover the effective receptive field area of the higher resolution output to be predicted, the dashed box in $\boldsymbol{a}^{(Out)}$ of Figure 4.3. The lowest resolution feature maps at each pixel location should then be flattened to one dimensional tensors before the one dimensional convolution operations take place (similar operations can also be performed using

1×1 convolution filters with a stride length of 1). The number of convolution filters required corresponds to the square of the upsampling factor, r, i.e. $C^{(3)} = r^2$. Thus, each filter is used to predict a certain pixel in each upsampling area where the weights of the convolution filters are shared among the inputs of the adjacent upsampling layer, i.e. $\boldsymbol{a}^{(2)}$ in Figure 4.3. Finally, the outputs of the filters are reshaped to two-dimensional tensors and arranged with a periodic shuffling operation [88].

4.4 Proposed Network

In this section, we describe the network architecture and show how the proposed network achieves comparable performance in terms of both DSC and Jaccard scores, yet is superior in terms of computational time and memory usage. The proposed network is compared to both U-Net [95] and a modified 1-strided FCN [17] (FCN-1s) in order to show these properties. We note that the existing networks use transposed convolution or a resize convolution, as discussed in Section 2.3.2, to perform image segmentation at the original input resolution.

4.4.1 Proposed Network Architecture

The configuration of our proposed network architecture for image segmentation is shown in Figure 4.4.

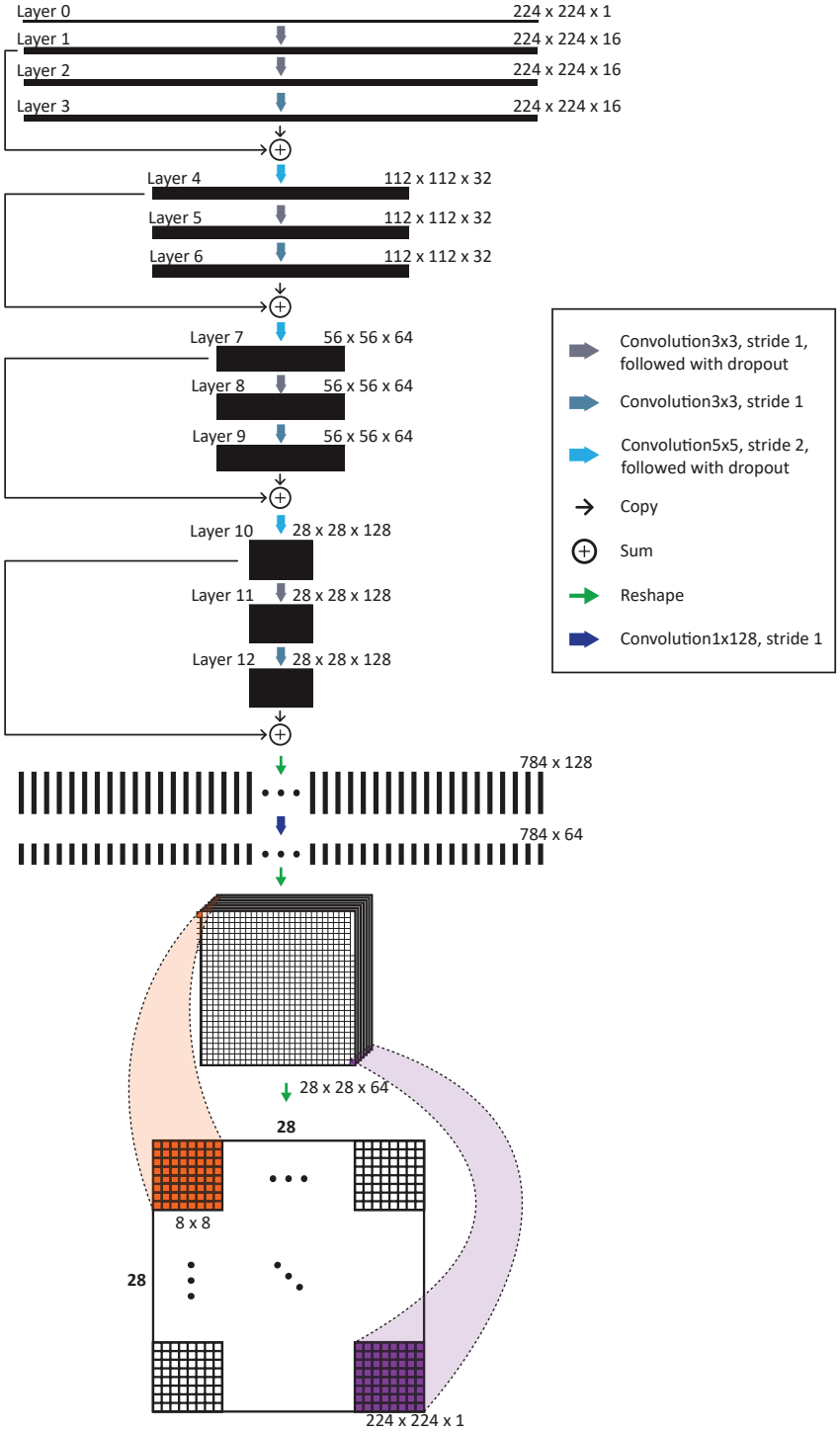


Figure 4.4: Proposed adjacent network architecture.

As can be seen from the network architecture in Figure 4.4, a standard 3×3 convolution network is adopted across the network, as a stack of 3×3 convolutional layers alone is sufficient to achieve state-of-the-art performance [90]. Furthermore, 2-strided convolution is adopted instead of max-pooling to avoid losing the spatial

information of each extracted feature. A 5×5 convolution kernel is used on the 2-strided convolutional layer to capture the large spatial relationship of a feature and its surroundings, without adding too much computational burden, as would be the case if it were applied on every convolutional layer. A skip connection is used in each convolution block to create a residual learning framework to reduce the degradation problem [48]. A dropout rate of 0.2 is used to incorporate regularisation with low risk of underfiitting [93]. Finally, the extracted high-level features at each feature location are used to perform the final classifications of the corresponding pixel and the adjacent pixels of the high-level features, i.e. adjacent upsampling.

To perform adjacent upsampling, the ERF and the minimum ERF of the lowest resolution feature maps are computed to ensure that the area of the high resolution output to be predicted is covered by the network.

The receptive field of the input layer, $f^{(0)}$, is always 1 as it is the reference layer on which we are calculating the ERF. The first convolutional layer uses a filter size of 3×3 (denoted as Conv_3) and the ERF can be calculated using Eq.4.1,

$$\mathrm{ERF}^{(\ell)} = \mathrm{ERF}^{(\ell-1)} + \left(f^{(\ell)} - 1\right) \times \prod_{i=1}^{\ell-1} s^{(i)},$$

where $\ell = 1$, $\mathrm{ERF}^{(\ell-1)} = \mathrm{ERF}^{(0)} = 1$, $f^{(1)} = 3$ and $\prod_{i=1}^{\ell-1} s^{(i)} = s^{(0)} = 1$, which results in $\mathrm{ERF}^{(1)} = 3$.

Since the second and third convolutional layers have the same layer parameters (filter size and stride length), the receptive field calculation will have the same accumulative effect to the previous layer, which is an increment of $\mathrm{ERF}^{(1)} - \mathrm{ERF}^{(0)} = 2$. Thus, the $\mathrm{ERF}^{(2)}$ and $\mathrm{ERF}^{(3)}$ are 5 and 7 respectively.

The fourth convolutional layer uses a filter size of 5×5 (denoted as Conv_5) and the ERF can be calculated using Eq.4.1, where $\ell = 4$, $\mathrm{ERF}^{(\ell-1)} = \mathrm{ERF}^{(3)} = 7$, $f^{(4)} = 5$ and $\prod_{i=1}^{\ell-1} s^{(i)} = s^{(0)} \times s^{(1)} \times s^{(2)} \times s^{(3)} = 1$, which results in $\mathrm{ERF}^{(4)} = 11$.

As the product of the stride length of all the previous layers after the fourth layer becomes 2, the ERF increment of the Conv_3 layer after layer 4 is 4. Thus, the ERFs of the two consecutive Conv_3 layers ($\mathrm{ERF}^{(5)}$ and $\mathrm{ERF}^{(6)}$) are 15 and 19 correspondingly.

Similar to the effect of the Conv_5 layer in the fourth layer, where it multiplies the stride length product of all the previous layers by 4, $ERF^{(7)}$ becomes 27. It also causes the product of the stride length of all the previous layers to be 4 and the ERF of the two consecutive Conv_3 layers ($ERF^{(8)}$ and $ERF^{(9)}$) to increase by 8 from the ERF of their previous layer.

Lastly, the Conv_5 layer in the tenth layer results in an $ERF^{(10)}$ of 59, it also increases the stride length product of all the previous layers to 8. Thus, the ERFs of the remaining Conv_3 layers ($ERF^{(11)}$ and $ERF^{(12)}$) are 75 and 91 respectively.

Similar computations also apply to the minimum ERF calculation. The ERF and the minimum ERF for each layer in the network of Figure 4.4 are given in Table 4.1.

Table 4.1: Adjacent Network ERF

Layer (ℓ)	Layer Type	Output Shape	Filter Size ($f^{(\ell)}$)	Stride Length ($s^{(\ell)}$)	$ERF^{(\ell)}$	$ERF^{(\ell)}_{min}$
0	input	224x224x1	1	1	1	1
1	Conv_3	224x224x16	3	1	3	2
2	Conv_3	224x224x16	3	1	5	3
3	Conv_3	224x224x16	3	1	7	4
4	Conv_5	112x112x32	5	2	11	6
5	Conv_3	112x112x32	3	1	15	8
6	Conv_3	112x112x32	3	1	19	10
7	Conv_5	56x56x64	5	2	27	14
8	Conv_3	56x56x64	3	1	35	18
9	Conv_3	56x56x64	3	1	43	22
10	Conv_5	28x28x128	5	2	59	30
11	Conv_3	28x28x128	3	1	75	38
12	Conv_3	28x28x128	3	1	91	46

Once the ERF and minimum ERF of the last layer has been computed, adjacent upsampling with an upsampling factor less than or equal to the minimum ERF can be applied to make high resolution predictions efficiently. This omits the need of using step-by-step upsampling learning to project the high-level feature back to its original input resolution before making the final class prediction. In this case, as we are adopting an FCN feature extraction architecture, our network is capable of performing upsampling with a factor of 46. However, only an upsampling factor of 8 is required to perform the predictions at the original input resolution.

4.4.2 Configuration Detail

The proposed network, shown in Figure 4.4, takes a $224 \times 224 \times 1$ input. The network starts with a convolution block consisting of a stack of three convolutional layers and dropout layer, with a dropout rate of 0.2, in between each convolutional layer. The skip connection links the first convolutional layer to the third convolutional layer. Next, a 2-strided convolutional layer is used to reduce the dimension by one half, followed by a convolution block consisting of a stack of two standard convolutional layers using twice as many filters as the convolution filters in the previous convolution block. The network configuration continues in this pattern until the feature layer dimension is one-forth of the original input dimension, i.e. 28×28. Next, the $28 \times 28 \times 128$ feature layer is reshaped to a 784×128 feature layer before being convolved with the last 64, 1×128, convolution filters. The final $224 \times 224 \times 1$ output is obtained by rearranging the 64 predictions at each feature location to 8×8, as shown at the bottom of the network in Figure 4.4.

The ReLU activation function is used after each convolutional layer in the network as it is the most successful activation function being used in the state-of-the-art networks [46] [26] [29] [48] [30] [17] [18], while the sigmoid activation function is used in the final prediction layer as there are only two classes involved in this problem, i.e. the background and the liver, as discussed in Section 2.3.2.

4.5 Automatic Liver Segmentation

Liver segmentation in CT scans is a fundamental step in many medical imaging related tasks associated with the abdomen. It can aid automatic liver tumour and vessel segmentation, as well as 3D visualisation and surgery planning [96]. Liver segmentation remains a challenging task due to reasons, such as low liver contrast compared to other surrounding organs, blurry boundaries, adjacent vessels having various appearances and pathologies with heterogeneous densities [96]. In addition, high variations in liver appearances, e.g. shapes and sizes, at different cross-sectional axial positions also contribute to the difficulty of automatic segmentation.

In this section, we describe the data and challenges in the data preparation

processes, as well as the training process. We then present the results obtained using the proposed adjacent network on a public dataset [89].

4.5.1 Data

To evaluate the performance of the proposed adjacent network against two state-of-the-art 2D CNNs U-Net [95] and FCN-1s [17], we use datasets from the MICCAI 2017 LiTS Challenge [89] consisting of 130 professionally labelled CT scans. The MICCAI 2017 LiTS Challenge datasets and segmentations are provided by six medical centres from several clinical sites around the world. The datasets come in two batches with sizes of 28 and 102 CT scans. As the scans come from different medical centres, some of the scan parameters are also different, e.g. scan dimensions, voxel size and scan orientation. Each scan consists of a different number of 2D slices ranging from 42 to 1026 slices. In these datasets a 2D slice is composed of 512×512 pixels. The voxel size of the 2D slices, both width and height, varies from 0.557 to 1mm, while the slice thickness varies from 0.45 to 6mm. The scan orientation comes in 3 different anatomical coordinate systems, which are left-anterior-superior (LAS), right-anterior-superior (RAS) and left-posterior-superior (LPS) [97].

The 2D CNN architecture proposed in this chapter is not affected by the high variance of the slice thickness as each of the processing layers operate on 2D data. For the scan orientation, data augmentation can be used to learn different orientations or anatomical coordinate systems. However, it is unnecessary to make the learning more complex as they can all be reordered and flipped to a certain coordinate systems. Here, we use the LAS coordinate system.

Prior to training and testing, data is prepared by reshaping the given 512×512 two-dimensional slice images to 224×224 to fit the model input layer. Next, the abdominal area is masked using an edge detection filter and connected component labelling operation, to omit the background produced by different scanners. Finally, the images are enhanced with contrast stretching and histogram equalization before being centred by mean subtraction. The data pre-processing results are shown in Figure 4.5.

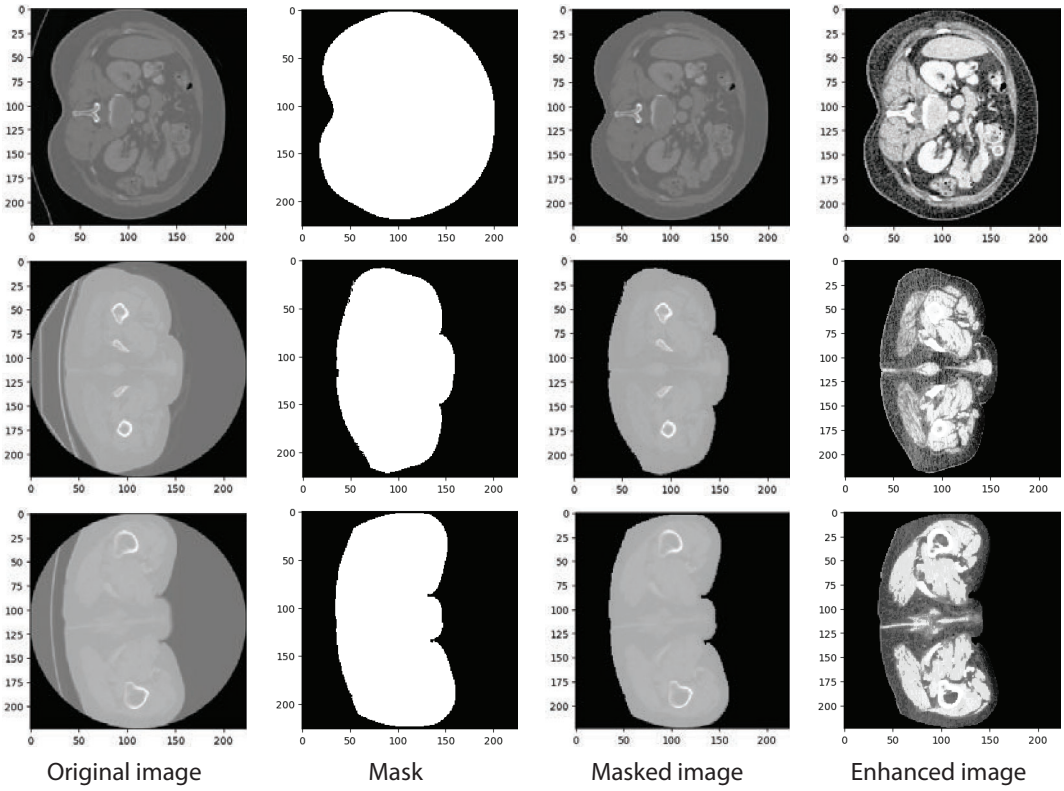


Figure 4.5: Data pre-processing results prior to training and testing.

4.5.2 Training

From the available 130 scans available, 46 scans are used for training, validation and testing. For training, 26 scans are taken from the batch of 28, with 2 randomly selected datasets to be used as the validation set. The test set consists of the 2 scans that were used for validation and 18 scans randomly picked from the batch of 102. The testing is done twice in batches of 10 scans, i.e. 2 split tests (Split 1 and Split 2). This allows the evaluation of the generalization capability of each of the networks on datasets from unseen scanners.

For training, He initialisation is used for the weight initialisation and the Adam optimiser is used with a 10^{-4} learning rate and a binary cross-entropy cost function.

As can be seen from Table 4.2, the proposed adjacent network has the least number of trainable parameters and the fastest training time per epoch when compared to U-Net and FCN-1s. This comparison is made on 4Gb GeForce GTX 960 GPU with Intel(R) Core(TM) i5-3550 CPU @3.30GHz and 16Gb RAM.

Table 4.2: Network Performance Comparison

Network	Trainable Parameters	Training Time per Epoch
U-Net	31M	1931 Seconds
FCN-1s	14.7M	790 Seconds
Adjacent Net	669.6k	211 Seconds

4.5.3 Results

In this segmentation problem we use overlap based scores, i.e. Dice's similarity coefficient (DSC) and Jaccard similarity coefficient (JSC), as the performance metrics to evaluate the three networks.

The DSC metric is twice the cardinality of the intersection, in voxels, between the predicted (P) and the ground truth (GT) regions divided by the sum of the cardinalities of the P and GT regions [98], given by

$$\text{DSC} = \frac{2|P \cap GT|}{|P| + |GT|}, \tag{4.4}$$

where $|P|$ and $|GT|$ are the cardinalities of the P and GT regions respectively.

The JSC metric is the cardinality of the number of voxels in the intersection of the predicted (P) and ground truth (GT) regions divided by the cardinality of the number of voxels in their union [98], given by

$$\text{JSC} = \frac{|P \cap GT|}{|P \cup GT|}. \tag{4.5}$$

As discussed in Section 4.5.2, the networks are evaluated on 2 split tests. Performance results of each split test are shown in the box-and-whiskers plots of Figures 4.6 and 4.7.

We can see from Figures 4.6 and 4.7 that the adjacent network has the highest maximum DSC and JSC in both split tests. This shows the capability of the adjacent network to exceed the performance of its competitors in both evaluation metrics. It also achieves the highest minimum DSC and JSC on the split 2 test when compared to the others. Although the minimum split 1 test score of the adjacent network

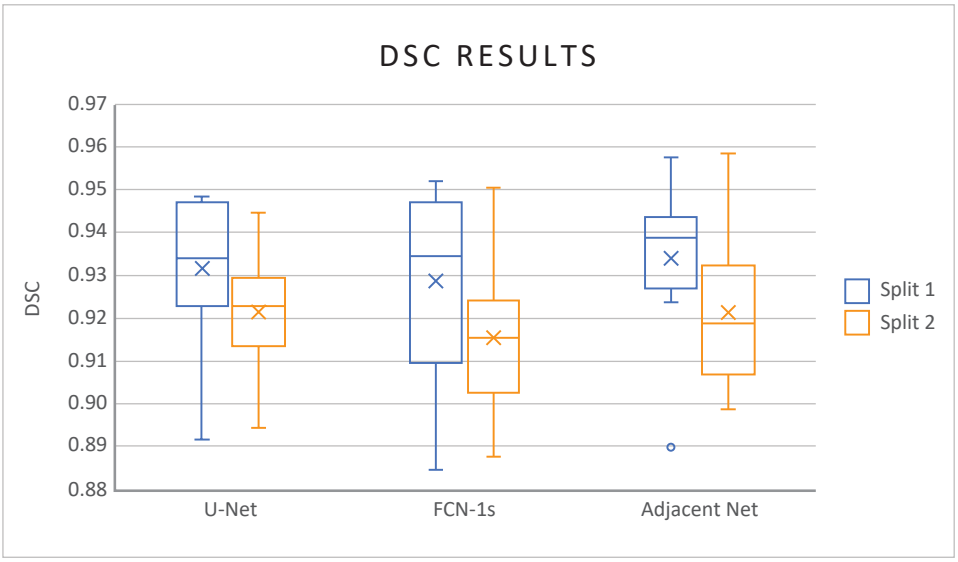


Figure 4.6: Box-and-whiskers plot of U-Net, FCN-1s and Adjacent Network DSC for split 1 and split 2 tests. The × symbol in the box indicates the mean value, while the line in the box indicates the median value. The ∘ represents the outlier, the box represents the interquartile range and the whiskers represent the upper and lower extreme, excluding the outliers.

is not the highest, it is approximately the same as the competitors (0.2% lower DSC than U-Net and 0.5% higher DSC than FCN-1s; 0.3% lower JSC than U-Net and 0.8% higher JSC than FCN-1s). Figures 4.6 and 4.7 show that the adjacent network performs comparably well in terms of both the DSC and JSC, while using considerably less trainable parameters.

Table 4.3: Network Performance Comparison

Network	**Avg Dice Score (in %)**			**Avg Jaccard (in %)**		
	Split 1	Split 2	Avg	Split 1	Split 2	Avg
U-Net	93.20	92.18	92.69	87.30	85.52	86.41
FCN-1s	92.89	91.57	92.23	86.79	84.49	85.64
Adjacent Net	93.42	92.16	92.79	87.70	85.51	86.61

The overall performance results are summarised in Table 4.3, which includes the average (Avg) of both split test scores. All the networks evaluated in the comparison achieve high DSC and JSC. The proposed adjacent network performs comparably well to the other more complex networks, even though the training

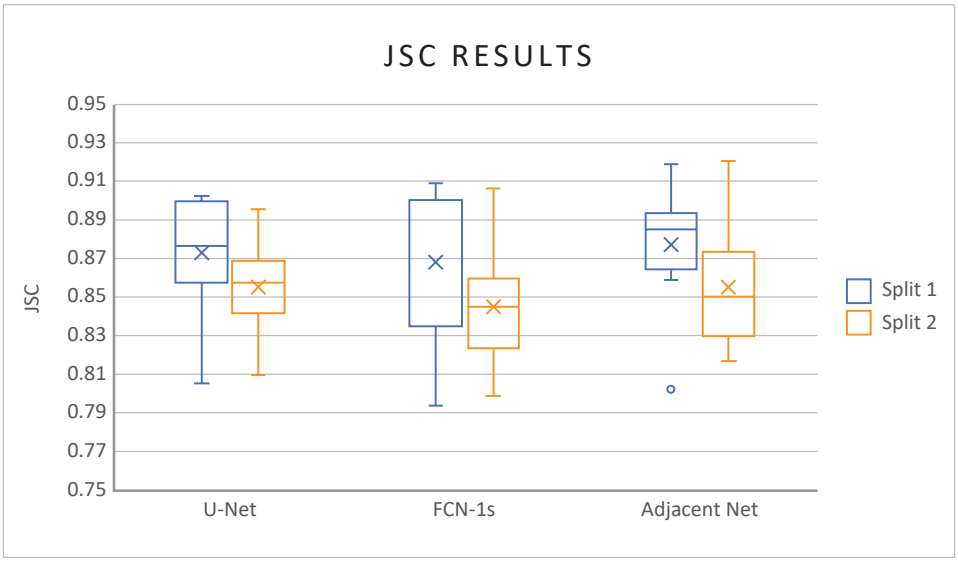


Figure 4.7: Box-and-whiskers plot of U-Net, FCN-1s and Adjacent Network JSC for split 1 and split 2 tests. The × symbol in the box indicates the mean value, while the line in the box indicates the median value. The ∘ represents the outlier, the box represents the interquartile range and the whiskers represent the upper and lower extreme, excluding the outliers.

time is significantly less (see Table 4.2), achieving a slightly higher average in both the DSC (0.1% higher than U-Net and 0.56% higher than FCN-1s) and JSC (0.2% higher than U-Net and 0.39% higher than FCN-1s). In the Split 1 test, the proposed network achieves a 0.22% higher DSC and a 0.4% higher JSC than U-Net. However, in the Split 2 test, U-Net achieves a 0.01% higher DSC and JSC than the proposed network. On the other hand, FCN-1s scores just slightly lower than both U-Net and the proposed adjacent network in both tests.

A selection of labelled and segmented CT images are shown in Figures 4.8, 4.9 and 4.10.

None of the networks achieve perfect scores, and some segmentation errors occur in their predictions, as shown in Figure 4.8.

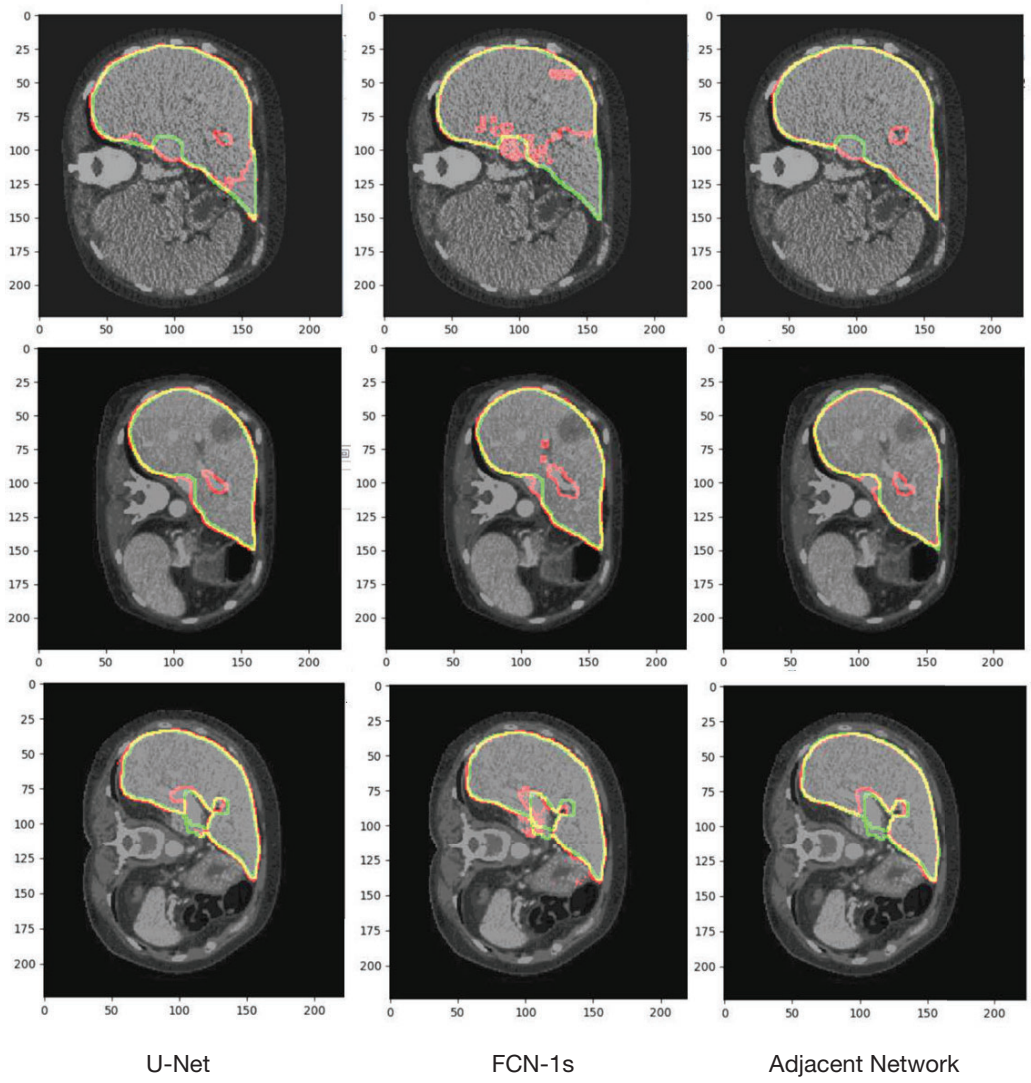


Figure 4.8: Liver segmentation results by U-Net, FCN-1s, Adjacent Network on 3 different CT scan slices (row 1 to 3). The red, green and yellow contours show the network prediction, ground truth and correct prediction respectively.

As can be seen in Figure 4.9, when the scan contains multiple liver parts, all of the networks have similar difficulties in detecting the smaller parts. However, the majority of the predictions are correct as shown in Figure 4.10.

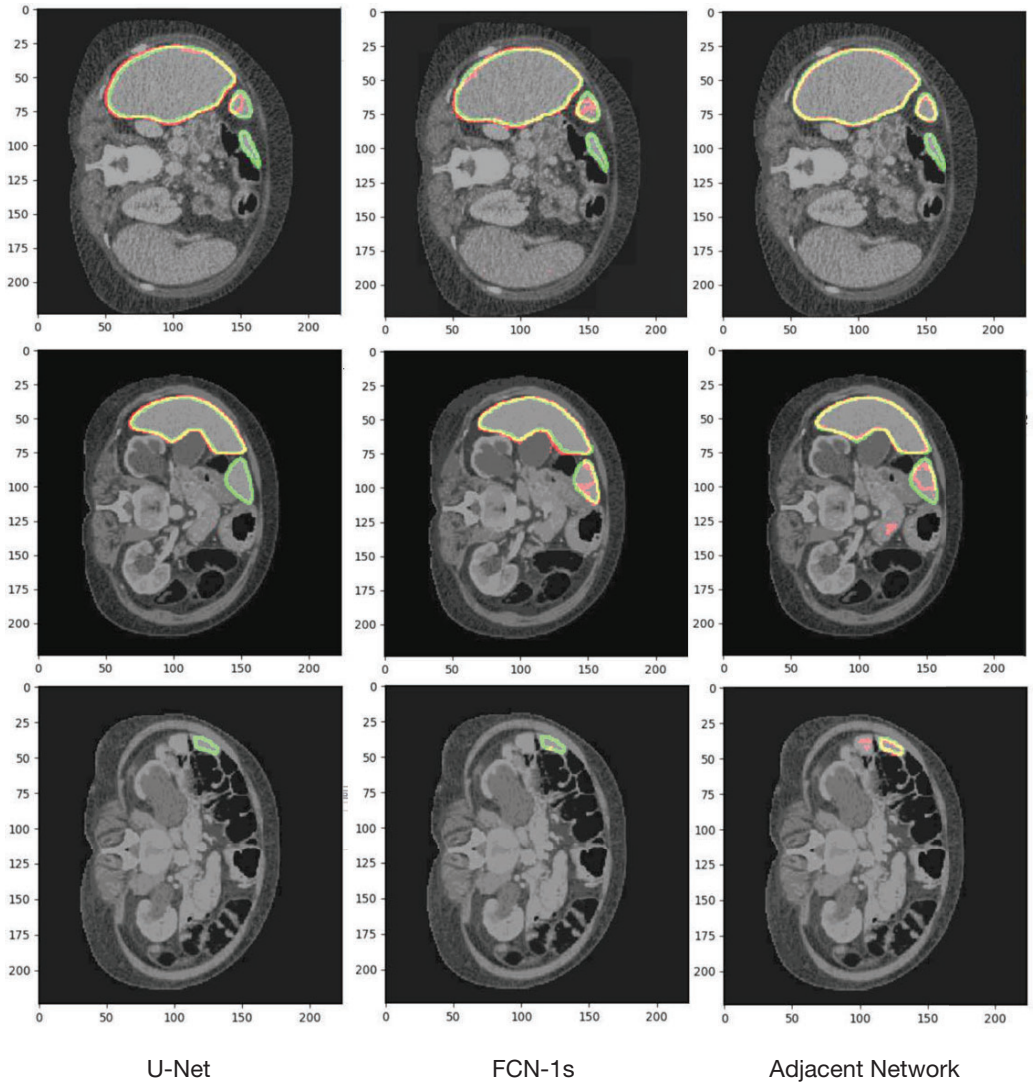


Figure 4.9: Liver segmentation results by U-Net, FCN-1s, Adjacent Network on 3 different CT scan slices (row 1 to 3). The red, green and yellow contours show the network prediction, ground truth and correct prediction respectively.

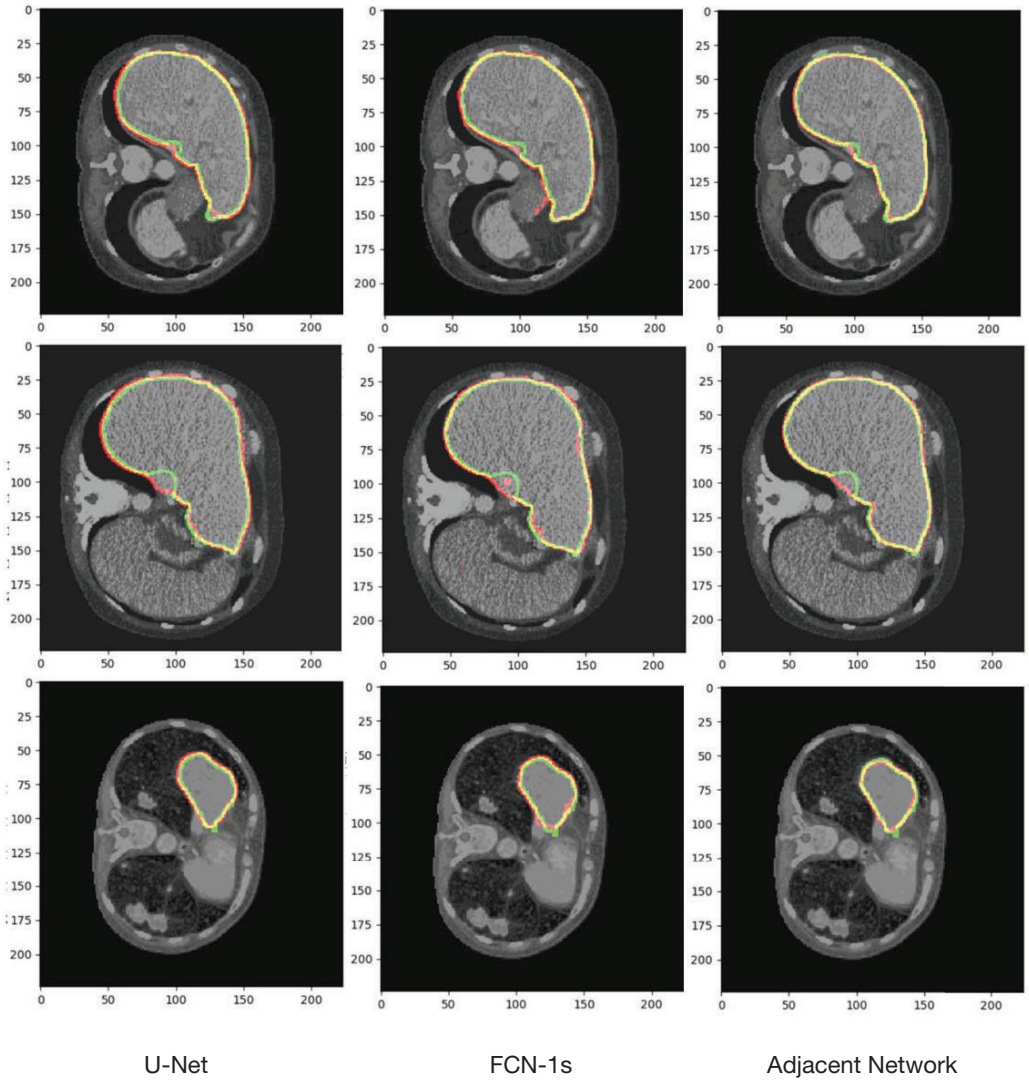


Figure 4.10: Liver segmentation results by U-Net, FCN-1s, Adjacent Network on 3 different CT scan slices (row 1 to 3). The red, green and yellow contours show the network prediction, ground truth and correct prediction respectively.

4.6 Conclusion

In this chapter, we propose a CNN that incorporates a novel adjacent upsampling method to make a more effective pixel-wise prediction from the extracted high-level features. The proposed adjacent network uses a much smaller number of trainable parameters and less memory, whilst it is able to perform comparably well with respect to state-of-the-art networks such as U-Net and FCN-1s. Also, the proposed network is much easier to train due to the smaller number of trainable parameters, and is significantly faster in computation time due to its more efficient architecture.

CHAPTER

5

Optimisation of a U-Net Architecture for Automatic Prostate Segmentation on MRI

In this chapter, we develop an optimised U-Net architecture by considering the effects of each individual component on the overall network performance specifically for a prostate segmentation task. The segmentation network is optimised with respect to the Dice score, while maintaining a small number of trainable parameters. U-Net is chosen for this optimisation as it is a well known state-of-the-art segmentation network and its contracting-expanding structure offers a range of flexibility for its components. The optimised network is shown to outperform traditional methods on a private dataset as well as other comparable state-of-the-art 2D networks reported on a public dataset, PROMISE12. For the model evaluation on the public dataset, the model predictions were submitted to the grand-challenge website and evaluated by the organiser.

5.1 Introduction

In this chapter, a prostate segmentation dataset is used due to its level of complexity. Unlike liver, prostate has fuzzy boundaries and pixel intensities that vary largely inside and outside the prostate [99]. The prostate also has similar range of pixel intensities within the prostate and non-prostate region [99]. These factors cause prostate segmentation to be a very challenging task that is suitable for this study.

Radiation therapy (radiotherapy) is a cancer treatment that uses ionizing radiation to kill cancer cells or control the growth of tumours. It is a very common treatment for all stages of prostate cancer. However, this treatment can damage the normal cells around the cancer cells, putting surrounding organs at risk of post-treatment complications [100]. In the case of prostate cancer, the main objective is to deliver a maximum dose of radiation to the prostate and minimise the dose received by the bladder and rectum [101]. For this reason, accurate prostate segmentation is required.

As discussed briefly in Section 1.1, manual labelling of an organ can be a time consuming and challenging process. It involves one or more experts scanning through the dataset and labelling the organ. As a result, labels produced by experts are subject to intra-and inter-expert variability due to varying expertise levels [102]. Intra-expert variability means that an expert may segment a specific image differently when performed multiple times, while inter-expert variability means that different experts may segment the same image differently [103].

Automatic segmentation can speed up the segmentation process as well as minimise the intra-and inter-expert variability problem [104, 105]. The main two traditional methods employed for automatic prostate segmentation on MRI images are the atlas-based and deformable model-based methods [106]. In the atlas-based method, a set of images and their corresponding labels are combined together after non-rigid registration (NRR) to create a reference atlas and a corresponding labelled structure. The atlas image, in this case, contains the prostate and its surrounding tissue with the corresponding labelled structure representing the probability of a voxel being a part of the prostate. The NRR of the atlas to the new, unseen MRI

scan is used to obtain the segmentation of the prostate of a new patient [106]. In the deformable model-based method, a good initialisation of the model is required. The model can be initialised by atlas-based segmentation [106, 107], where a surface is extracted from a thresholded probabilistic segmentation and the model is deformed to closely match the organ boundary by the use of the grey-level information of the image. The grey-level model is developed offline with one-dimensional grey-level profiles taken along the normals for each vertex of the surface for the images. A distance metric is then used to match the profiles of the model and the profiles extracted from the image [106]. As both methods either rely on the atlas-based method or a good initialisation, they are prone to errors [108] and can be time consuming [106].

Deep learning models based on a CNN have achieved outstanding results in automatic prostate segmentation tasks [12] [109–114]. There are many CNNs that differ by both the components and the way these components are constructed in their architectures, as discussed in section 2.3.2. Many studies often present a performance comparison of completely different model architectures [109–113] or CNN based approaches which include different post-processing methods in their pipeline [12] [115] [116]. As a result, the comparison is unreliable and the effect of an individual component on the overall performance is hard to distinguish and/or is unknown.

In order to develop an efficient and effective deep learning model based on a CNN, it is important to understand which components are most beneficial for a segmentation task. Thus, we focus on the effects of the individual components to the network in performing prostate segmentation.

In this chapter, we use a private dataset [117] of T2 weighted MR images with a fixed 2D resolution and voxel size across the whole dataset to optimise a 2D U-Net architecture [18] in terms of the network DSC score in performing a prostate segmentation. The use of this dataset, with a fixed 2D resolution and voxel size, allows us to mitigate the possibility of artefacts caused by resizing operations in the data preparation process that may ultimately affect the optimisation process. Performance evaluation of networks with different architectures will be presented to

provide an insight into the contribution of each network component in a prostate segmentation application.

Performance of the optimised network is evaluated on both a private dataset [117] and the public dataset, PROMISE12 [118], of T2 weighted MR images. With respect to the private dataset, we show that the performance of the optimised network is a significant improvement over that of the traditional segmentation methods [117, 119]. With respect to the PROMISE12 dataset, the performance of the optimised network is compared to other state-of-the-art 2D CNNs as can be seen on the public leaderboard [118] scored by the organiser. Challenges due to inter-expert variability associated with the dataset are also discussed, as this is a problem that is often overlooked in medical imaging segmentation. We provide suggestions about how the optimised network could be used to address aspects of this problem.

5.2 Dataset and Performance Metric for Optimisation

In this section, we describe the dataset and performance metric used for the optimisation process.

5.2.1 Dataset

For the network architecture optimisation in Section 5.3, we use a private dataset [117] which is collected following ethical approval and informed consents. The dataset is obtained using a Siemens Skyra 3.0 Tesla magnet, without the use of an endorectal coil, located at the Calvary Mater Newcastle Hospital, Australia. The dataset consists of 41 prostate, T2 weighted, MRI scans with three expert (E1, E2 and E3) manual delineations on each scan. Each scan contains $320 \times 320 \times 60$ voxels with a voxel size of $0.625 \times 0.625 \times 2$ mm. The expert E2 labels are used for both training and testing due to having the highest mean Dice's similary coefficient score against the majority voting labels [120]. The majority voting labels are the labels agreed by at least 2 out of the 3 experts. As majority voting tends to remove information, such as uncertain labels [121], it is not used for the evaluation process

to reduce bias in the optimisation process towards smaller volumes of the organ being segmented.

5.2.2 Performance Metric

The Sørensen–Dice coefficient or Dice's similary coefficient [98], DSC, is used to evaluate the performance of each model. The DSC metric is given in Eq. 4.4.

Five-fold cross-validation is used for the evaluation of the model on the entire dataset. First, one scan is extracted to be used as the validation set and the remaining 40 scans are shuffled randomly and then divided into 5 folds. Every fold consists of 32 scans for the training set and 8 scans for the test set. The average DSC score is the mean score of the five-fold cross-validation.

5.3 Network Architecture Optimisation

In this section, we determine the components that can improve the performance of a 2D U-Net for prostate segmentation in terms of Dice's score. The specific components we consider are associated with downsampling, upsampling, skip connections, dropout, normalisation and activation layers.

As discussed in Section 2.4.3, a U-Net has an equal number of downsampling and upsampling layers in the architecture, forming a contracting-expanding structure that can be beneficial for a segmentation task [18]. Thus, we use a U-Net architecture [95] for our base architecture. We begin with what we denote as UNet_S, a simplified U-Net that uses a quarter of the number of filters used in the U-Net of [95], which results in a significantly lower number of trainable parameters ($\sim$ 1.9M) than the original architecture ($\sim$ 31M). The UNet_S architecture is shown in Figure 5.1.

The UNet_S consists of multiple convolution blocks in its architecture. Each convolution block (Conv Block) contains two pairs of 3×3 convolution and ReLU activation layers as shown in Figure 5.2 (a). For downsampling, a 2×2 max pooling layer with stride length of 2 is used. For upsampling, a 2×2 resize convolution with ReLU activation is used. Skip connections concatenate the same resolution feature maps from the contracting part to the expanding part, however, none are within the

convolution block. There are 2 dropout layers with a dropout rate of 0.5 after the fourth and fifth convolution block. Finally, a 1×1 convolutional layer with sigmoid activation is used to produce a probability map.

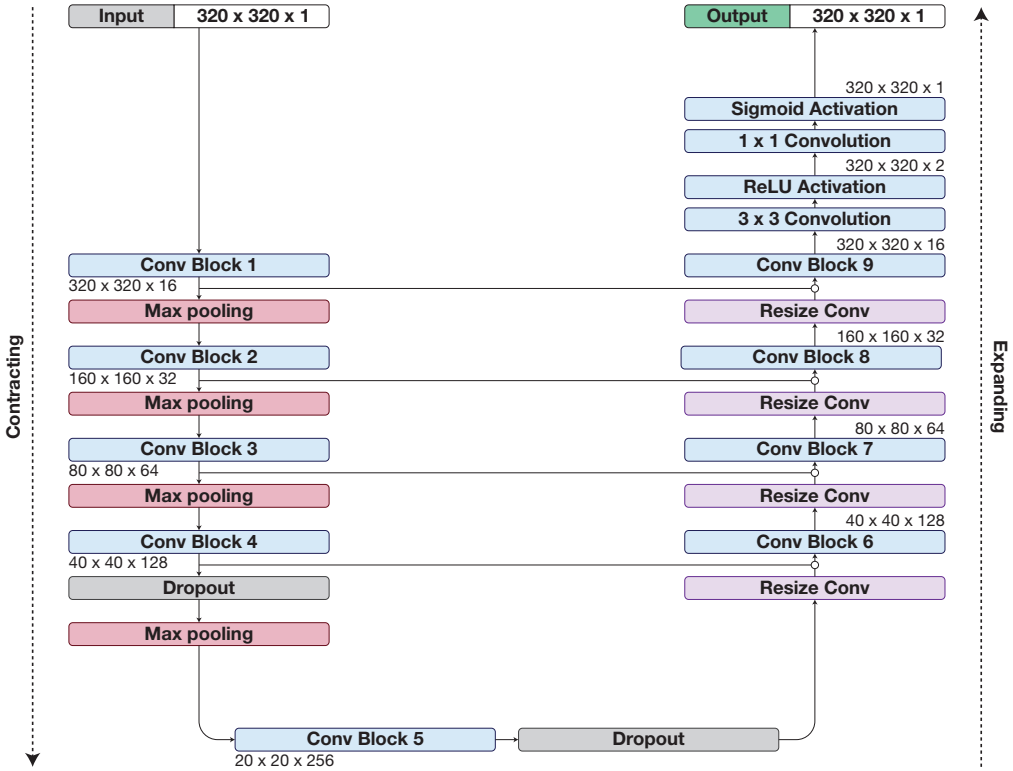


Figure 5.1: UNet_S architecture, the base architecture.

Several components of the network architecture will be investigated to find the most suitable components for prostate segmentation problem. For the downsampling component, we will investigate the use of 2-strided convolution, RMS pooling, L2 pooling and average pooling in the place of max pooling. The 2-strided convolution and average pooling layers are the pooling layer that are being used by some of the state-of-the-art CNNs [39] [48] [30] [49] [47]. While, RMS and L2 pooling are the other possible pooling methods that we think have the potential to replace the max-pooling layer. For the upsampling component, we will investigate the use of transposed convolution, as it was being used successfully by FCN [17] and V-Net [47], in comparison to resize convolution. For the skip connection, we will investigate its use within the convolution block, by comparing the original implementation with the summation operation, as in ResNet [48] and V-Net, [47] and the modified

implementation with concatenation operation, as in DenseNet [30]. Then, we investigate the need of using dropout by comparing the use of the most common dropout value that was adopted in the state-of-the-art CNNs [46] [26] [17] [47], i.e. 0.5, to not using dropout, i.e. 0. We also investigate the need of using a batch normalisation layer, as well as its most suitable placement with respect to the convolution and activation layers. Finally, we investigate the use of LReLU, as LReLU is the most basic modified ReLU that was suggested to overcome the dying ReLU problem as discussed in Section 2.3.2.

In summary, for the optimisation process, we use 6 phases to determine an optimised U-Net architecture:

1. Determine the appropriate downsampling and upsampling component
2. Determine the appropriate operation for a skip connection
3. Determine the requirement of dropout
4. Determine the requirement of batch normalisation
5. Determine the appropriate pooling layer
6. Determine the appropriate activation function and placement of batch normalisation

Table 5.1 provides the components of the network being optimised.

Table 5.1: Network architecture details.

Phase	Network	Components					
		Downsampling	Upsampling	Skip Connection within Conv Block	Dropout	Batch Normalisation	Activation
1	UNet_S	Max Pooling	Resize Conv*	-	0.5	-	Relu
	UNet_S1	Max Pooling	Transposed Conv*	-	0.5	-	Relu
	UNet_S2	Strided Conv†	Resize Conv*	-	0.5	-	Relu
2	UNet_S.1	Max Pooling	Resize Conv*	Summation	0.5	-	Relu
	UNet_S1.1	Max Pooling	Transposed Conv*	Summation	0.5	-	Relu
	UNet_S2.1	Strided Conv†	Resize Conv*	Summation	0.5	-	Relu
	UNet_S.2	Max Pooling	Resize Conv*	Concatenation	0.5	-	Relu
	UNet_S1.2	Max Pooling	Transposed Conv*	Concatenation	0.5	-	Relu
	UNet_S2.2	Strided Conv†	Resize Conv*	Concatenation	0.5	-	Relu
3	UNet_S.2.1	Max Pooling	Resize Conv*	Concatenation	0	-	Relu
4	UNet_S.2.0.1	Max Pooling	Resize Conv*	Concatenation	0.5	Before Activation	Relu
5	UNet_S.2.0.1.1	Avg Pooling‡	Resize Conv*	Concatenation	0.5	Before Activation	Relu
	UNet_S.2.0.1.2	RMS Pooling	Resize Conv*	Concatenation	0.5	Before Activation	Relu
	UNet_S.2.0.1.3	L2 Pooling	Resize Conv*	Concatenation	0.5	Before Activation	Relu
6	UNet_S.2.0.1.1.1	Avg Pooling‡	Resize Conv*	Concatenation	0.5	Before Activation	LRelu
	UNet_S.2.0.1.1.2	Avg Pooling‡	Resize Conv*	Concatenation	0.5	After Activation	Relu

*Resize Conv refers to resize convolutional layer. *Transposed Conv refers to transposed convolutional layer. †Strided Conv refers to strided convolutional layer. ‡Avg Pooling refers to average pooling layer.

In phase 1 of the evaluation, we investigate the performance of the convolutional layers in performing upsampling and downsampling. The UNet_S1 is UNet_S where the resize convolution is replaced with a transposed convolution, and UNet_S2 is UNet_S where max pooling is replaced with 2×2, 2-strided convolution. As can be seen from the DSC results shown in Table 5.2 phase 1, the UNet_S with max pooling and resize convolution components produced the highest DSC score.

For phase 2, we examine the use of a skip connection in each convolution block to improve the performance of the networks in phase 1. Two configurations of the skip connection are considered for the convolution blocks in the network. The UNet_S.1, UNet_S1.1 and UNet_S2.1 have skip connections with element-wise summation (Figure 5.2b), whilst the UNet_S.2, UNet_S1.2 and UNet_S2.2 have skip connections with concatenation (Figure 5.2c).

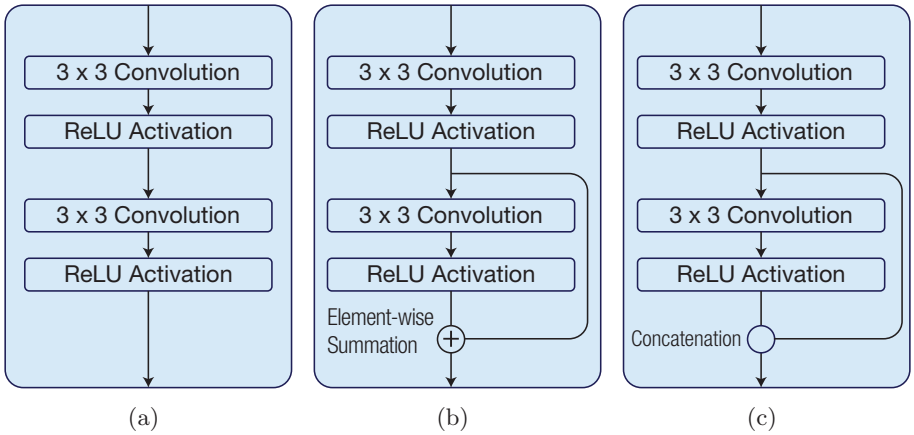


Figure 5.2: Convolution blocks. (**a**) Convolution block in UNet_S, UNet_S1 and UNet_S2; (**b**) Skip connection with element-wise summation in UNet_S.1, UNet_S1.1 and UNet_S2.1; (**c**) Skip connection with concatenation in UNet_S.2, UNet_S1.2 and UNet_S2.2.

As can be seen in Table 5.2, phase 2, the skip connection with element-wise summation decreases the performance of UNet_S.1, UNet_S1.1 and UNet_S2.1 when compared to the UNet_S. On the other hand, the skip connection with concatenation improves the performance of UNet_S.2, UNet_S1.2 and UNet_S2.2 as compared to UNet_S.

From phases 1 and 2, we conclude that the use of max pooling for downsampling and resize convolution for upsampling combined with a concatenation skip connection in each convolution block, i.e. UNet_S.2, performs better than the other configurations (see Table 5.2). In phase 3, we perform the modifications only on the best network from phase 2.

In phase 3, we modify the dropout rate from 0.5 to 0 to investigate the need of dropout. As can be seen from Table 5.2, phases 2 and 3, the network with a dropout rate of 0.5, UNet_S.2, performs better than the modified network, UNet_S.2.1, which has a dropout rate of 0.

In phase 4, we investigate the effect of batch normalisation on the best network achieved so far, i.e. UNet_S.2. In UNet_S.2.0.1, batch normalisation layers are placed in between the convolution and activation layers, as shown in Figure 5.3 (a).

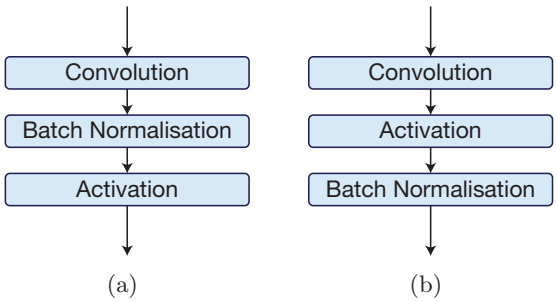


Figure 5.3: Batch normalisation layer implementations. (**a**) Batch normalisation layer after each convolutional layer in UNet_S.2.0.1. (**b**) Batch normalisation layer after each activation layer in UNet_S.2.0.1.1.2.

It can be seen in Table 5.2 phase 4, that the previous best result obtained by UNet_S.2 is improved by including batch normalisation layers, as in UNet_S.2.0.1. This is due to the result of the additional model regularisation introduced by the use of the batch normalisation layer, as discussed in Section 2.3.2.

As discussed in Section 2.3.2, the max pooling layer extracts significant features, such as light edges and lines, of the previous layer. However, the prostate does not have a well-defined boundary, hence a different pooling layer may perform a better segmentation. Therefore, in phase 5, we replace the max pooling layer in the UNet_S.2.0.1 with average pooling (UNet_S.2.0.1.1), RMS pooling (UNet_S.2.0.1.2) and L2 pooling (UNet_S.2.0.1.3) layers.

The network with the average pooling layer, UNet_S.2.0.1.1, is shown to perform better than the networks utilising the other types of pooling layers, as can be observed in Table 5.2 phase 5.

Finally, we investigate an alternative activation function and placement of batch normalisation layers on the best network, UNet_S.2.0.1.1. First, we replace the ReLU with LReLU activation layers in UNet_S.2.0.1.1.1. Then, we investigate the network performance when placing the batch normalisation layer after the activation layer, as shown in Figure 5.3 (b), in UNet_S.2.0.1.1.2. The results are shown in Table 5.2 phase 6. It can be observed that neither the modification of the ReLU to the LReLU nor the change in position of the batch normalisation layer with the ReLU layer improves on the performance of the UNet_S.2.0.1.1.

Table 5.2: Network architecture optimisation results.

Phase	Network	5-Fold Cross-Validation DSC (%)					
		Fold 1	**Fold 2**	**Fold 3**	**Fold 4**	**Fold 5**	**Avg**
1	**UNet_S**	85.28	86.86	83.36	86.16	81.53	**84.64**
	UNet_S1	85.26	83.04	79.66	81.43	79.16	81.70
	UNet_S2	85.07	85.25	80.20	81.18	80.84	82.51
2	UNet_S.1	84.40	86.30	80.29	80.49	82.98	82.89
	UNet_S1.1	83.37	84.22	78.04	82.05	79.35	81.41
	UNet_S2.1	83.70	82.53	79.19	82.68	82.03	82.03
	UNet_S.2	85.46	88.01	82.91	84.76	84.89	**85.21**
	UNet_S1.2	85.43	84.67	79.06	80.60	83.13	82.58
	UNet_S2.2	85.28	81.63	81.08	84.35	80.33	82.53
3	UNet_S.2.1	82.31	84.97	80.11	84.08	81.47	82.59
4	**UNet_S.2.0.1**	84.33	87.26	82.80	88.79	84.73	**85.58**
5	**UNet_S.2.0.1.1**	84.02	88.87	83.97	89.23	85.07	**86.23**
	UNet_S.2.0.1.2	84.55	84.71	81.36	87.38	83.98	84.40
	UNet_S.2.0.1.3	84.12	87.55	82.80	88.92	84.17	85.51
6	UNet_S.2.0.1.1.1	81.68	83.89	79.45	88.06	84.00	83.41
	UNet_S.2.0.1.1.2	84.64	87.95	82.68	87.75	84.68	85.54

Therefore, the UNet_S.2.0.1.1 is selected as the network that is best suited to the prostate segmentation task. It is a simplified U-Net with a concatenation skip connection in each convolution block, 2×2 average pooling layers used for downsampling and a batch normalisation layer placed between each convolution and ReLU activation layer. It achieved an average increase of 1.59% and a maximum increase of 3.54% in the DSC score compared to the initial simplified UNet. It has a minimum DSC score of 83.97% and a maximum DSC score of 89.23%, while the simplified UNet has a minimum DSC score of 83.36% and a maximum DSC score of 86.86%.

5.4 Optimised Network Architecture

The configuration of the optimised network, UNet_S.2.0.1.1, architecture is shown in Figure 5.4. The network takes a $320 \times 320 \times 1$ input and processes it directly with a batch normalisation layer. The network comprises of multiple convolution blocks with each block consisting of two 3×3 convolutional layers, two batch normalisation layers and two ReLU activation layers. A skip connection with concatenation is used to pass the feature maps between the outputs of the activation layers within a convolution block to combine the feature maps using a concatenation. The concatenated feature maps are passed deeper into the contracting part as well as to the expanding part to improve the spatial information in the higher level feature maps. Four 2×2 average pooling layers are used to downsample the feature maps from a resolution of 320×320 to a resolution of 20×20. A dropout layer, with a dropout rate of 0.5, is applied after the fourth and fifth convolution blocks. A 2×2 resize convolution with batch normalisation and ReLU is used to perform upsampling. Finally, a 1×1 convolutional layer with batch normalisation and sigmoid activation layers is used to produce a probability map.

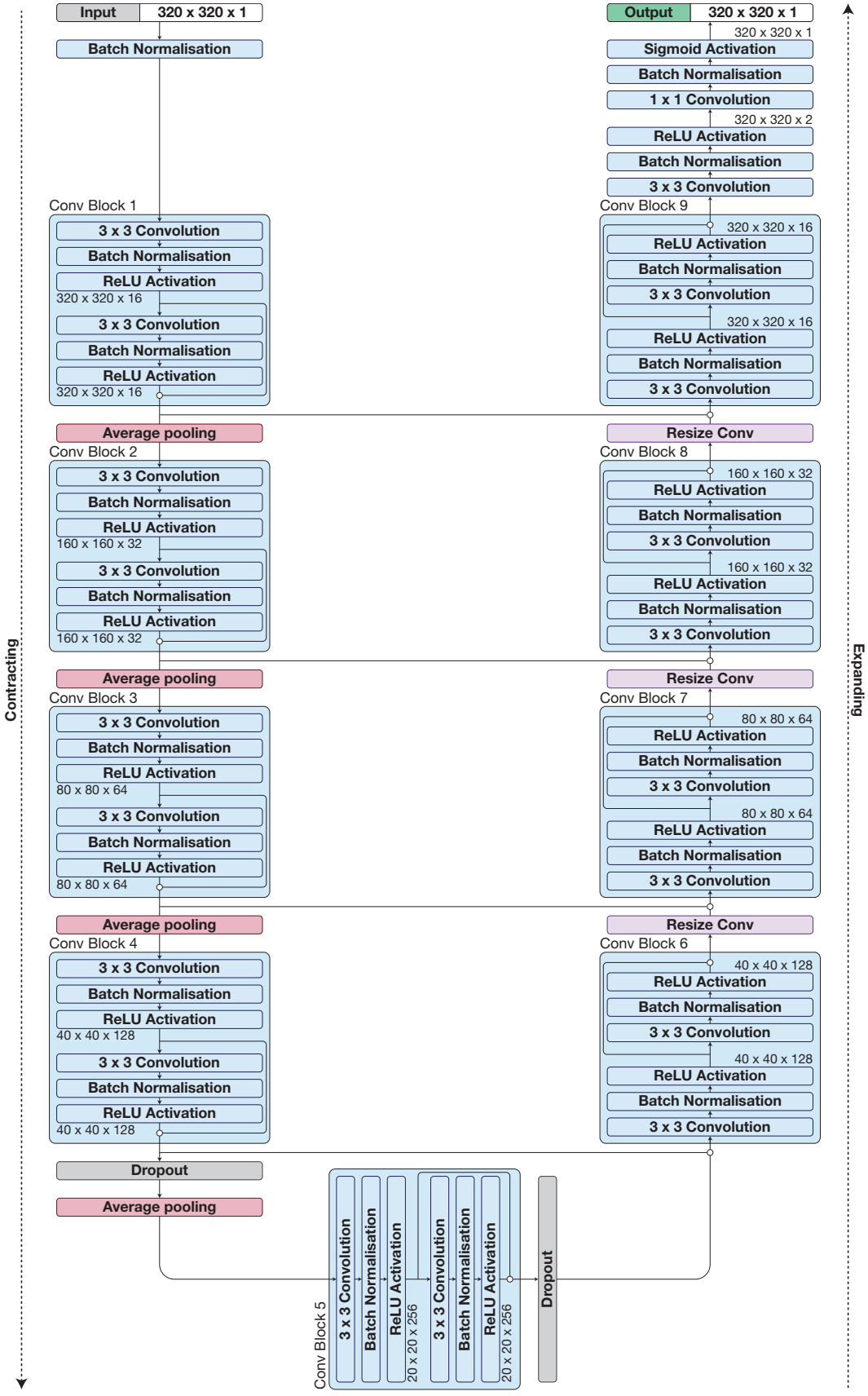


Figure 5.4: Optimised UNet_S.2.0.1.1 architecture .

5.5 Prostate Segmentation using the Optimised U-Net

In this section, we explain the training process and the results of the optimised U-Net on both the private and the PROMISE12 [118] datasets.

5.5.1 Application on the Private Dataset

The same dataset (scan dimension of $320 \times 320 \times 60$ voxels with voxel size of $0.625 \times 0.625 \times 2$ mm) used for the model evaluation, in Section 5.2, is used for both training and testing in this section. However, instead of using labels from one expert as the label of the prostate, we use the majority voting of the labels from three experts, as used in the other studies [117, 119] on the same dataset, to extract the consensus label of the prostate for training.

Performance Metric for the Private Dataset

Five-fold cross-validation mean DSC, median DSC, average symmetric surface distance (ASD) and Hausdorff distance are used for the evaluation of the model performance.

The ASD is the average Euclidean distance from all the points on the predicted region boundary, B_P, to the nearest ground truth region boundary, B_{GT}, and from all the points on the B_{GT} to the nearest B_P [98], given by

$$ASD = \frac{\left(\sum_{x \in B_P} D\left(x, B_{GT}\right) + \sum_{y \in B_{GT}} D\left(y, B_P\right)\right)}{|B_P| + |B_{GT}|}, \tag{5.1}$$

where the $|B_P|$ and $|B_{GT}|$ are the cardinalities of the B_P and B_{GT} regions respectively and the distance from a voxel x to a set of voxels A is given by

$$D(x, A) = \min_{y \in A} d(x, y), \tag{5.2}$$

with $d(x, y)$ denoting the Euclidean distance between 2 voxels, x and y.

The Hausdorff distance measures the maximum distance from a point in the set of surface points of the predicted regions, P, to the nearest point in the set of surface

points of the ground truth regions, GT, [98], given by

$$d_H(P, GT) = \max_{x \in P} \min_{y \in GT} d(x, y). \tag{5.3}$$

Training on the Private Dataset

The training process consists of two pre-processing steps, i.e. data balancing and normalisation. As discussed in Section 3.3.4, to minimise bias on the weight tuning in the training process, a balanced portion of the data from each classes has to be used for training. In this case, we use RUS to extract an equal number of slices with and without prostate labels. We extract all the 2D scan slices in a volume that have a prostate label, then randomly select an equal number of the 2D scan slices that do not have a prostate label. For the normalisation, we use z-score normalisation where we subtract the mean of the training set from each voxel of the training, validation and test sets, then divide the result by one standard deviation of the training set.

The weights are initialised with the He initialisation. An Adam optimiser initialised with a 10^{-4} learning rate and a binary cross-entropy cost function are used for the training.

The model is developed with Keras [122]. Both training and testing are performed on 4 Gb GeForce GTX 960 GPU with Intel(R) Core(TM) i5-3550 CPU @3.30 GHz and 16 Gb RAM. The training time per epoch is approximately 275 s, while testing time is 2.85 s per scan and 0.0474 s per slide.

Results on the Private Dataset

The performance difference between the optimised U-Net and three traditional prostate segmentation methods [117, 119] is presented in this section. The three prostate segmentation methods that were used in previous studies are the multi-atlas [117], multi-object weighted and standard (unweighted) deformable model approaches [119].

Examining the scores over the whole dataset, the optimised U-Net has a mean DSC of 87.38%, median DSC of 88.19%, median ASD of 0.72 mm and median Hausdorff of 4 mm. As shown in Table 5.3, the optimised U-Net outperforms the

traditional methods by at least 7% and 6% in mean and median DSC, respectively, 1.32 mm in median ASD and 5.6 mm in median Hausdorff distance.

Table 5.3: Performance comparison between the optimised U-Net and traditional methods.

Method	Mean DSC	Median DSC	Median ASD (mm)	Median Hausdorff (mm)
Multi-atlas	0.80	0.82	2.04	13.3
Weighted	0.79	0.81	2.08	9.6
Unweighted	-	0.70	3.20	12.9
UNet_S.2.0.1.1	0.87	0.88	0.72	4

As shown in Figure 5.5, the optimised U-Net performed well on all the cross-validation folds by having DSC scores in the range of 0.73 to 0.94, where the third quartile of the five folds are at least 0.9, and only 4 out of 40 predictions have a DSC score below 0.81. The mean DSC scores, across all folds, are at least 0.86 and the median DSC scores are at least 0.87, while the best mean and median DSC scores achieved by the traditional method are 0.80 and 0.82, respectively in this dataset [117, 119]. Without the outliers, the minimum DSC scores of fold 2, 3, 4 and 5 are all 0.84 and above, which is above the mean and the median DSC scores achieved by the traditional methods.

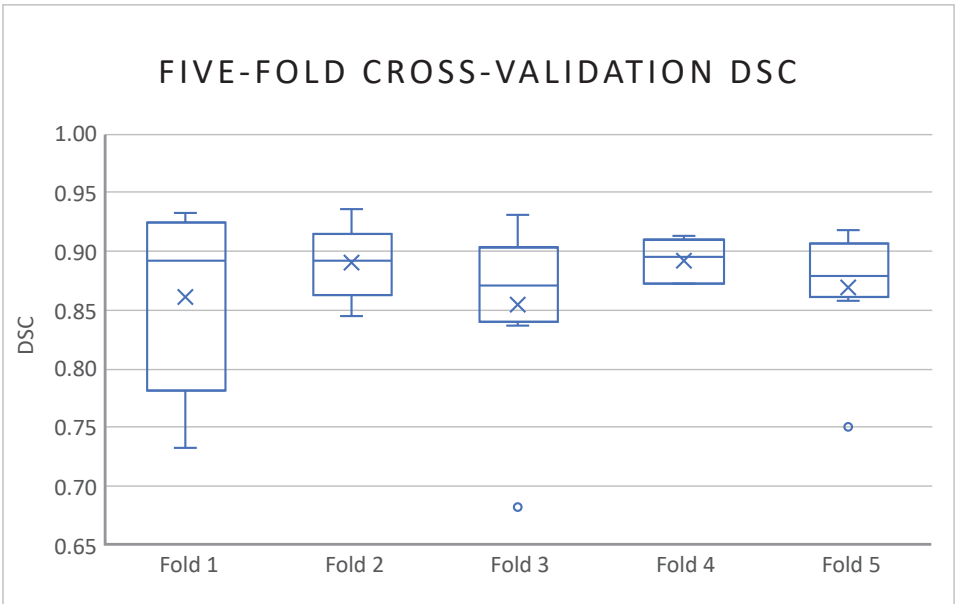
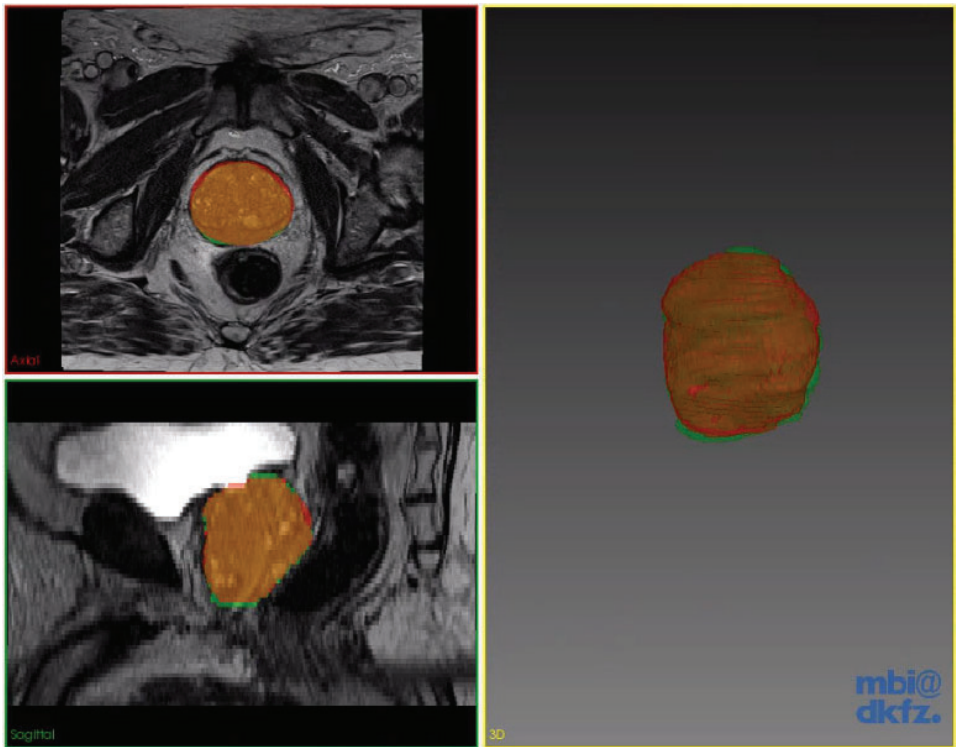
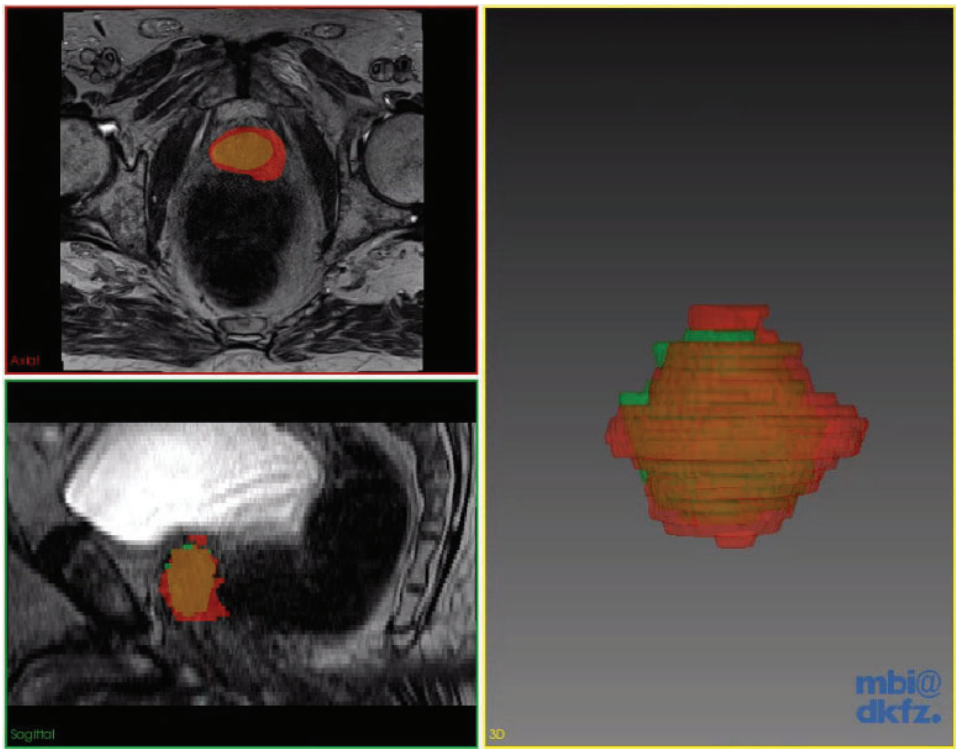


Figure 5.5: Box-and-whisker plot of the optimised U-Net five-fold cross-validation Dice's similary coefficient scores, where the × symbol in the box indicates the mean value and the line in the box indicates the median value. The ∘ represents the outlier, the box represents the interquartile range and the whiskers represent the upper and lower extreme, excluding the outliers.

Excluding the obvious outliers in Figure 5.5, we present the best and worst predictions of the segmentation in Figure 5.6 (a) and (b) respectively. These prediction volumes have DSC scores of 0.94 and 0.73. It is easily observed that the prediction volume with a DSC score of 0.94 is very similar to the ground truth segmented volume. We note that even for the worst case, having a DSC score of 0.73, the majority of the prediction volume still agreed well with the ground truth segmented volume.



(a)



(b)

Figure 5.6: Prostate segmentation results from the optimised U-Net. Model prediction in red, ground truth in green. (**a**) Prediction with DSC score of 0.94; (**b**) Prediction with DSC score of 0.73.

5.5.2 Application on the PROMISE12 Dataset

The PROMISE12 dataset [118] consists of 80 T2-weighted MR images of the prostate. It is collected with different acquisition protocols, e.g. different slice thickness, with/without endorectal coil, from multiple centres and vendors. The training set consists of 50 T2-weighted MRI scans with a single prostate label, i.e. the reference segmentation. The test set consists of 30 T2-weighted MRI scans without any label.

Performance Metric for the PROMISE12 Dataset

The organiser of the PROMISE12 challenge [118] evaluates the models according to the segmentation from a reference standard and second observer. The reference standard segmentations were set by an experienced observer and revised by another experienced observer. The second observer segmentations were performed in a blind fashion and revised by the other experienced observer for completeness and anomalies. The second observer segmentations are used to obtain the average user errors for each measure.

Mean DSC, absolute relative volume difference, average boundary distance and 95% symmetric Haussdorff distance are used to calculate the overall score for each model [118]. The overall score is the average of all the performance metrics. Each measurement score is based on the following,

$$\text{score}(x) = \max(ax + b, 0) \tag{5.4}$$

where a and b are set in such a way that the maximum score obtained when the segmentation is identical to the second observer is equal to 85 points, i.e. a and b are equal to 88.24 and 11.76, respectively. A maximum of 100 points can be achieved when the segmentation excludes the segmentation errors made by the second observer [118].

The DSC and average boundary distance (i.e. ASD) have been discussed in Section 5.2.2 and 5.5.1 respectively.

The absolute relative volume difference of the reference standard segmentation

region, GT, and model segmentation region, P, is given by [118],

$$\text{aRVD}(GT, P) = \left| 100 \times \left(\frac{|GT|}{|P|} - 1 \right) \right|, \tag{5.5}$$

where the $|GT|$ denotes the number of voxels in the reference standard segmentation region and $|P|$ denotes the number of voxels in the model segmentation region.

The symmetric Hausdorff distance is the maximum of the Hausdorff distance from the predicted regions, P, to the ground truth regions, GT, and vice versa. It is given by [118],

$$d_H^{sym}(P, GT) = \max\left(d_H(P, GT), d_H(GT, P)\right), \tag{5.6}$$

but instead of the maximum, the 95^{th} percentile of the Hausdorff distance is used.

Training on the PROMISE12 Dataset

In addition to the two pre-processing steps, i.e. data balancing and normalisation, 2D resizing to a size of 320×320 is required in this dataset, as the MRI scans come in two different sizes, 320×320 and 512×512. The resizing is performed by resampling the pixels based on the pixel area relation, where the weighted sum of pixel values within the scale window will be divided by the area of scale window at each pixel location [123].

The weights are initialised with the He initialisation. An Adam optimiser initialised with a 10^{-4} learning rate and a binary cross-entropy cost function are used for the training.

The model is developed with Keras [122]. Both training and testing are performed on 4 Gb GeForce GTX 960 GPU with Intel(R) Core(TM) i5-3550 CPU @3.30 GHz and 16 Gb RAM. The training time per epoch is approximately 200 s, while the testing time is 2.85 s per scan and 0.0474 s per slide.

Results on the PROMISE12 Test Set

In this section, the performance of the optimised U-Net is evaluated on the PROMISE12 test set and compared to the state-of-the-art CNNs on the

PROMISE12 leaderboard by submitting the network predictions to the MICCAI PROMISE12 grand-challenge website [118] where the scoring is conducted by the organiser. Here, the test set mean DSC and PROMISE12 overall score [118] are used for the evaluation of the networks. The performance results of some networks as well as the architecture type, pre-processing and post-processing details are presented in Table 5.4. Further details of the results can be obtained from the MICCAI PROMISE12 grand-challenge website [118].

In Table 5.4, the optimised U-Net is compared with the top 12 state-of-the-art 2D CNNs on the PROMISE12 leaderboard. All the 3D and combined 2D/3D CNNs have been excluded from the table as they are not directly comparable to the network architecture in this chapter. We also exclude networks that incorporate any data-enhancing pre- or post-processing and those with stacked networks as discussed in the sequel.

Table 5.4: Performance comparison between the optimised U-Net and state-of-the-art 2D CNNs on the PROMISE12 test set [118].

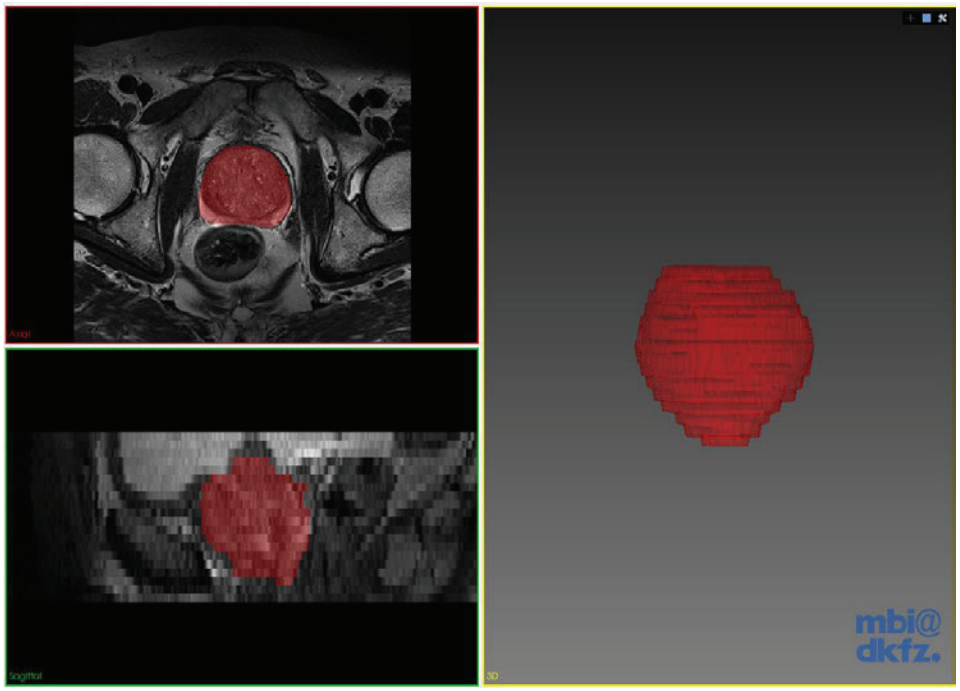
Rank	Team	Network	Type	Pre-processing	Post-processing	Mean DSC (%)	Overall Score
35	u3004443	Z-Net	Single	Yes	Yes	90.50	87.8068
59	hkuandrewzhang (Revised_U-net)	Z-Net	Single	Yes	No	90.24	87.3217
86	wanlichen (WNet)	W-Net [22]	Stacked	No	No	89.96	86.5028
92	sho89512	U-Net w/ Dense Dilated Block	Single	No	No	88.98	86.3676
95	fumin	RUCIMS (U-Net w/ Dense Dilated Block)	Single	Yes	No	88.75	86.2589
122	**Indri92**	**UNet_S.2.0.1.1 (U-Net)**	**Single**	**No**	**No**	**89.00**	**85.4954**
140	ddd52317102008	Adversial Network	Adv. Net.	No	No	87.90	84.5935
163	mirzaevinom	MBIOS (U-Net)	Single	Yes	No	88.06	83.6633
167	pppppppjw	U-Net w/ Dense Block	Single	No	No	86.80	83.5027
168	michaldrozdzal	UdeM 2D (ResNet)	Stacked	No	Yes	87.42	83.4522
179	mariabaldeon	AdaResU-Net [32]	Single	Yes	No	86.51	82.7937
194	wanlichen (WNet)	U-Net w/ skip connection	Single	No	No	86.29	82.1644

As can be seen in Table 5.4, both of the Z-Nets perform better than our optimised U-Net in terms of both mean DSC and overall score. However, both teams that use Z-Net employ either data-enhancing pre-processing, or post-processing, to improve

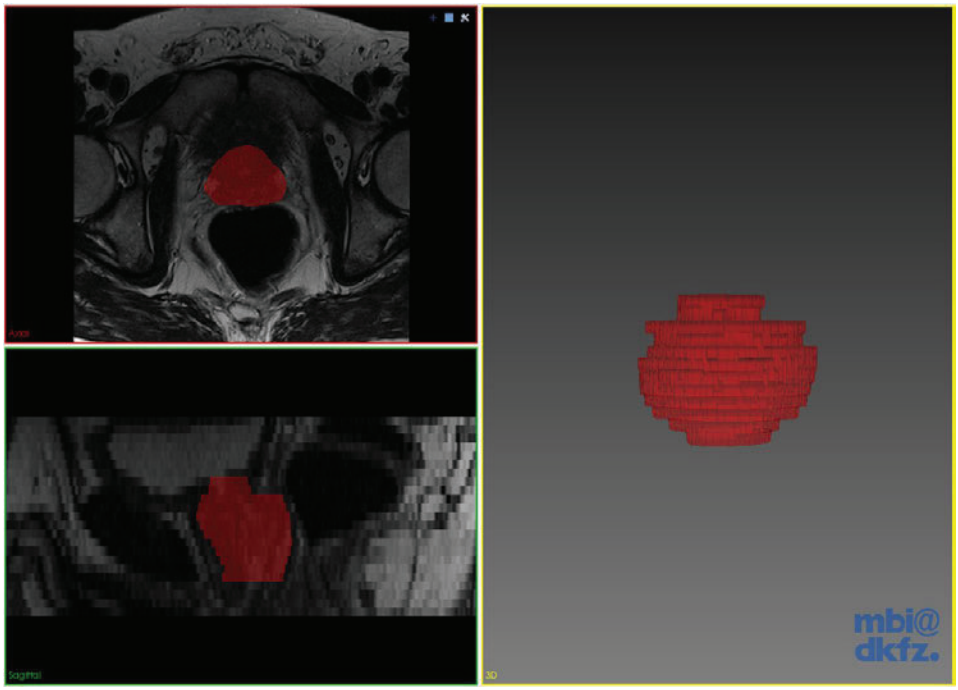
the model performance. Therefore, the Z-Nets results are incomparable with our results as we do not employ any data-enhancing pre-or post-processing. W-Net does not use any data-enhancing pre- or post-processing and it has a mean DSC of 0.8996 and overall score of 86.5028. However, W-Net is a stacked U-Net, i.e. it utilises a double U-Net to perform the segmentation, hence it is not comparable with our optimised U-Net architecture that consists of a single U-Net. Similar to our approach, sho89512 uses a single U-Net architecture and does not employ any data-enhancing pre-or post-processing method. However, although it performs better in the overall score (86.3676 vs. 85.4954), it performs slightly worse than our optimised U-Net in the mean DSC (0.8898 vs. 0.89). As can be seen in Table 5.4, our optimised U-Net performs significantly better compared to the rest of the 2D U-Net-based networks with or without the use of data-enhancing pre-and post-processing. Note that, it also performs better than the adversial network.

Most importantly, the optimised U-Net is seen to be better than U-Net (MBIOS), U-Net with residual connection (Udem 2D) and U-Net with skip connection (by wanlichen (WNet)) as a result of the optimisation. Another key point to note is that the optimised U-Net performs better than the U-Net with dense block (by pppppppjw), and only slightly worse than U-Net with dense dilated block (by sho89512 and fumin).

A couple of examples of the PROMISE12 segmentation results are shown in Figure 5.7 (a) and (b) respectively. These prediction volumes have DSC scores of 0.93 and 0.88, respectively. As the ground truth of both examples are not available, only the prediction volumes are presented in Figure 5.7.



(a)



(b)

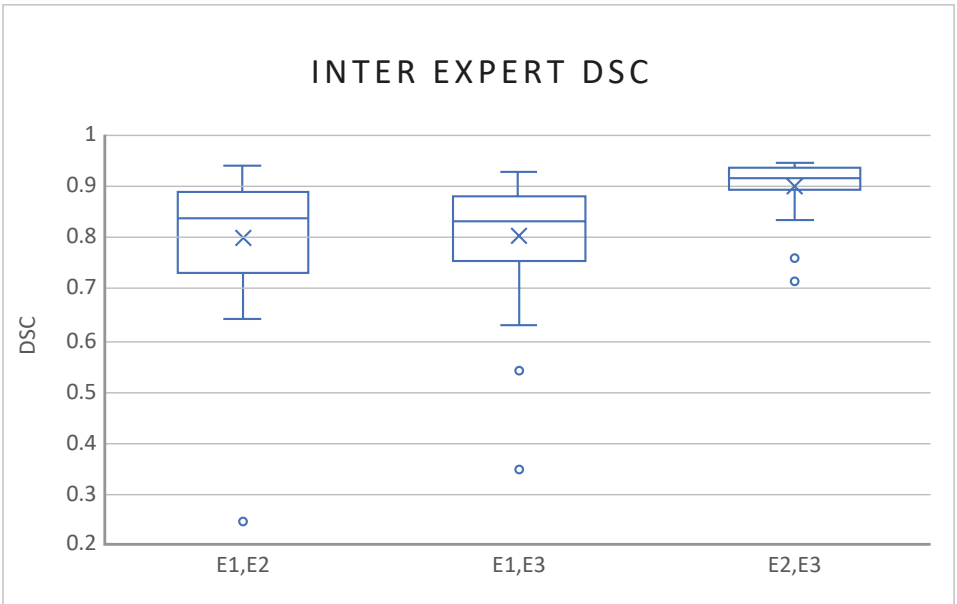
Figure 5.7: PROMISE12 segmentation results from the optimised U-Net. Model prediction in red. (**a**) Prediction with DSC score of 0.93; (**b**) Prediction with DSC score of 0.88.

5.6 Discussion

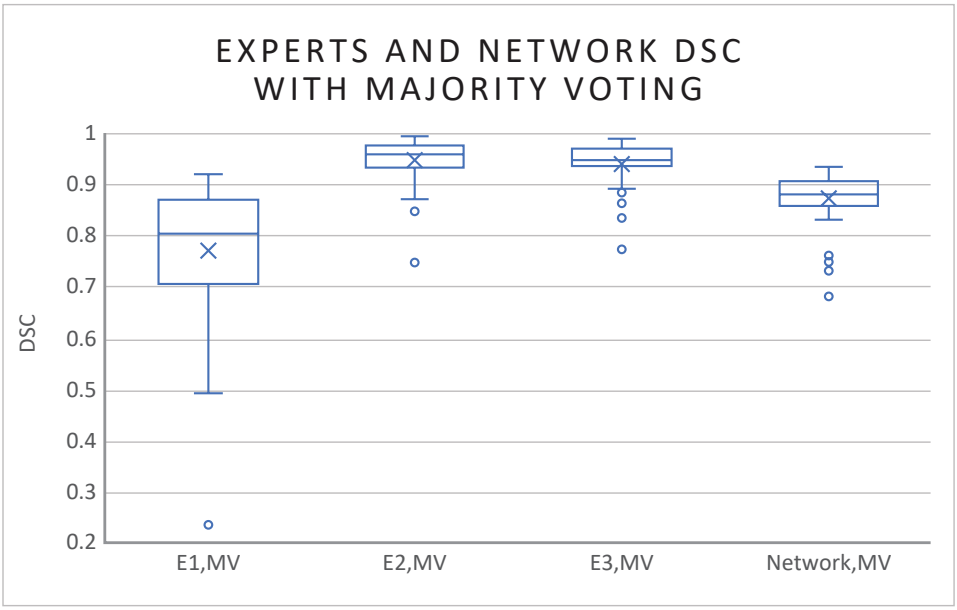
From the results, we can see that our U-Net performs better than other U-Nets as a result of the optimisation. This shows that several components that were added/modified works well for the prostate segmentation problem. For example, the average pooling works better than max pooling as prostate does not have a well-defined boundary, the resize convolution results in better higher-resolution feature mapping than the transposed convolution, the concatenation operation at the end of skip connections helps in retaining the low-level features to be used in deeper layers, while the dropout and batch normalisation layer help in the training of the network to prevent overfitting.

We also found that the inter-expert variability on the labelling of a dataset may cause the development of a CNN model to be difficult. As can be seen in the box-and-whisker plot in Figure 5.8(a), the inter-expert variability (between experts E1 and E2, E1 and E3, and E2 and E3) is very large for the prostate segmentation of the private dataset.

It is known that the labelling of medical images is always subject to the problem of intra-and inter-expert variability. One possible solution is to create larger labelled datasets by using an automatic segmentation model. In this way, the model can be used to provide an initial segmentation as a reference for the expert to save time. For example, our optimised U-Net can be used to provide assistance for the expert as it is shown in the box-and-whisker plot in Figure 5.8(b) that its performance is within the performance range of the experts. It can also be seen that the optimised U-Net performs better than expert one, E1, and almost as well as the other experts with respect to the majority voting label.



(a)



(b)

Figure 5.8: Box-and-whisker plots, where the × symbol in the box indicates the mean value and the line in the box indicates the median value. The ∘ represents the outlier, the box represents the interquartile range and the whiskers represent the upper and lower extreme, excluding the outliers. (a) Inter-expert DSC scores to highlight the inter-expert variability problem; (b) DSC scores of the three experts and the network against the majority voting label (MV).

5.7 Conclusion

In this chapter, we developed an optimised 2D U-Net with respect to the Dice score while maintaining a small number of trainable parameters to perform prostate segmentation, without any data-enhancing pre- or post-processing. We established the importance of the individual components within the U-Net architecture to perform prostate segmentation. We found that resize convolution results in better performance for upsampling than transposed convolution. Within each convolution block, combining feature maps with a skip connection is only beneficial when using a concatenation operation. With respect to pooling, the use of average pooling offers significant improvement over strided convolution, max, RMS or L2 pooling. The implementation of model regularisation by including a dropout layer, with a dropout rate of 0.5, and a batch normalisation layer before the activation layer, also improves the network performance. We show that the optimised U-Net in this chapter outperforms traditional segmentation methods on a private dataset by approximately 6% and 7% in a median and mean DSC scores, respectively. Furthermore, it outperforms (in terms of DSC) other comparable state-of-the-art 2D CNNs on the PROMISE12 public dataset as can be seen on the public leaderboard [118] scored by the organiser. In addition, we discuss intra-and inter-expert label variability and its effect on network performance, as well as provide suggestions to reduce the associated errors.

CHAPTER

6

Objective Quantification of Nerves in Immunohistochemistry Specimens

In this chapter, we develop a novel CNN based approach for a complex digital pathology segmentation problem. We provide a performance comparison between a CNN based approach and existing manual and automated quantification methods. We propose a novel augmented classification structure to improve the performance of a U-Net for an object detection task. We address common critical challenges that are often encountered in developing a CNN based approach with clinical data by closely considering the data pre-processing for the training of the network.

6.1 Introduction

A growing number of studies have shown that nerves are involved in the initiation and progression of a number of cancers [124]. For example, in prostate, breast, pancreatic and bowel cancers, the presence of nerves has been associated with increased tumour aggressiveness and a higher potential for metastatic spread [125] [126] [127]. With respect to thyroid cancer, it has been shown that nerve density is higher in papillary thyroid cancer (PTC) compared to follicular thyroid cancers (FTC) and benign thyroid tissues [128].

The detection of nerves in tissue sections presents challenges. Large nerve trunks that distribute the axons of central neurons to their peripheral targets are relatively easy to identify. However, the terminal fields of neurons consist of individual fine axons with diameters between 500 nm and 1 µm. As such, experts need to use specific neuronal markers and examine specimens at high magnification to identify them. This narrows the field of view and makes manual quantification challenging.

A number of manual counting strategies exist, in which small regions of interest are first identified, followed by counting of the stained nerves (as determined by immunohistochemical labelling of specific neuronal markers) by a trained expert in these regions [129] [125] [128] [130] or in a random selection of these regions [127] [126]. Automatic methods, e.g. computerised nerve planimetry, involve an expert manually selecting a few regions of interest centred around the target observation to define a colour filter range in each image. The image is then converted to a binary format using the colour filter, where the planimetry of the total nerve surface area will be determined [129] [130].

Manual counting has high precision. However, it is labour-intensive and usually has a low sensitivity [131]. It is also susceptible to bias and inconsistency due to intra- and inter-expert variability, as fatigue and other external factors may affect an expert's judgement [131]. On the other hand, computerised nerve planimetry is very fast and has a high sensitivity, but typically has a very low precision. It should be noted that some studies, that use computerised nerve planimetry, use manual region definition prior to planimetry analysis [129] [130].

Nerves vary largely in size and appearance. Figure 6.1 contains several examples of the different ways in which a nerve can be observed in a immunohistochemical sample, while Figure 6.2 contains a number of different nerve appearances.

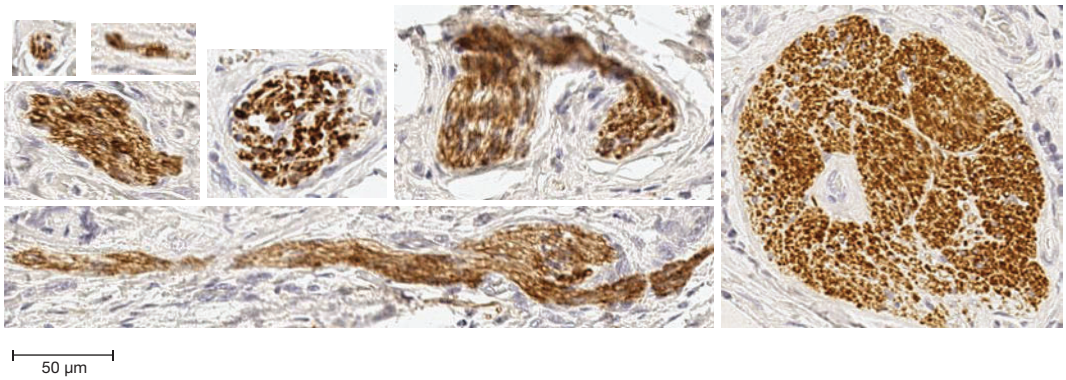


Figure 6.1: Examples of different views of a nerve.

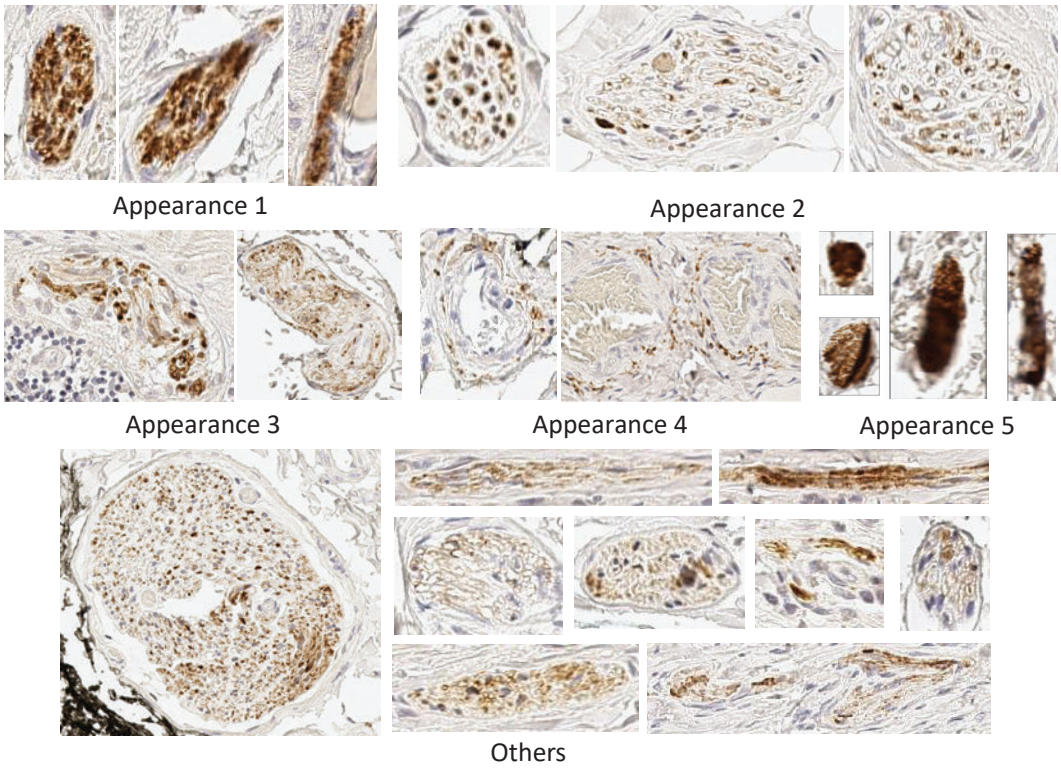


Figure 6.2: Examples of variation in the appearance of nerves.

As can be seen from both figures, the main identifier of the nerves is the immunohistochemical labelling that gives the colour contrast to the nerves, brown in this case. Unfortunately, the colour cannot be solely used as the nerve detection criteria. A key problem is that even the most specific immunohistochemical labelling

of nerve proteins inevitably has some degree of non-specific labelling of background tissue that contributes to false positive nerve detection. There are many different types of non-specific staining, a selection of which are shown in Figure 6.3. As can be seen from Figure 6.1, Figure 6.2 and Figure 6.3, the differences between non-specific staining and the nerves are not obvious in terms of colour, size and appearance. This non-specific staining causes the nerves to be difficult for the experts to distinguish and also leads to an overestimation of the number of nerves by the existing automatic methods.

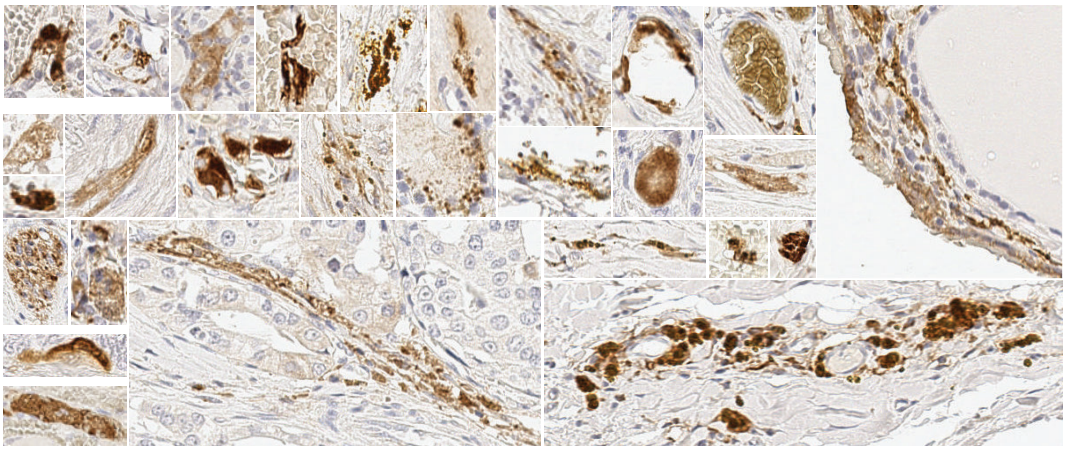


Figure 6.3: Examples of non-specific staining.

The variation of nerves in size and appearance also results in no formal definition of a standardised nerve detection criteria. As a result, it is difficult to compare detection and quantification methods across studies. It can be seen from Figure 6.1 that the size of the nerve can vary from $25\mu m^2$ to $22,500\mu m^2$. While, from Figure 6.2, it can be seen that nerves can have many different appearances, e.g. a nerve can appear like a single formation (appearance 1), separated small formations (appearance 2), separated formations (appearance 3), separated clusters (appearance 4), smudges (appearance 5) or other appearances (appearance 6). Although a nerve trunk can be easily detected due to its characteristic morphology and location features, a cluster of axons is more difficult to objectively quantify. A non-standardised detection criteria makes the development of an objective approach difficult.

Another significant challenge is presented by the very large image size, e.g.

$40,000 \times 40,000$ pixels, of a digital whole-slide section. The large image size causes fatigue in experts and a large computational time in computerised methods. There are also problems due to incomplete expert annotations and coarsely annotated data. The data used in this chapter is from a recent study [128], where expert manual counting occurred at 4x magnification, hence it is inevitable that many smaller nerves are not detected or annotated.

In this chapter, we propose a nerve detection approach that uses a CNN to improve nerve quantification for thyroid cancer biomarker studies. The main contribution is the development of a nerve detection approach based on a segmentation network incorporating a novel augmented classification structure. We evaluate the potential of our proposed CNN based approach in performing the nerve detection and quantification task on data made available from the study in [128].

6.2 Related Work

Many CNN based approaches have been applied to object detection tasks, for example, object detection in photographs [132] [27] [28], organ detection in medical images [84] [133] or mitosis, cytoplasm and nuclei detection in stained whole slide images (WSIs) [134] [135] [136]. However, to our best knowledge, no CNN based approach has been applied specifically to the nerve detection and quantification problem. In this section we will present the rationale for our approach based on some object detection problems that are applicable to this nerve detection problem.

6.2.1 Colour Thresholding

In image processing it is typical to convert an image from the red, green and blue (RGB) colour space to the hue, saturation and value (HSV) colour space for the purpose of colour image segmentation and/or thresholding [137]. This is because the HSV colour space organises colour in a similar way to the perception of the human eye [137], in that luma/intensity information are separated from chroma/colour information in the HSV colour space [138]. This makes colour range definition in the HSV colour space more straightforward in comparison to the RGB colour space.

To perform colour thresholding (i.e. filtering) on a WSI in a typical image processing program, e.g. ImageJ [139], an expert takes a sample of the target instances to initialise the colour filter range in the HSV colour space. Then, the expert will adjust the threshold limit manually until the desired segmentation output is obtained.

6.2.2 Object Detection Approaches in a WSI

In computer vision, an object quantification task is usually formulated as an object detection task [140] [141]. Some of the most successful approaches in object detection evolved from a reliance on either a multi-scale sliding-window (i.e. exhaustive search) [142] [143] [144], a selective-search [145] [132] [27] [146] or cascaded deep learning models [28].

However, a WSI generally has a size of approximately $40,000 \times 40,000$ pixels, or larger, instead of 500×500 pixels as in a typical object detection task using deep learning [142] [143] [144] [145] [132] [27] [146] [28]. Hence it is inefficient to apply a sliding-window or selective search to the entire WSI [82] or input the entire WSI to the deep learning model. Therefore, small image patches are generally generated by an ROI extraction algorithm, e.g. a sampling algorithm [147] [148] [75] or a segmentation algorithm [149] [135], from each WSI. Sampling algorithms extract ROIs from sampled image patches, while segmentation algorithms extract ROIs from every image patch. Sampling algorithms are usually applied to global-level tasks, e.g. tissue-level cancer localisation, where the target observation is at the WSI level [82]. On the other hand, segmentation algorithms, e.g. colour thresholding or its variants [149] [135], are usually applied to local-level tasks, e.g. cell or nuclei detection, where the target observation is at the pixel level [82]. Once the ROIs have been obtained, the features can then be extracted and processed through either a manual [149] [135] or an automatic [135] operation.

For the manual operation, the features are usually extracted according to a specific handcrafted characteristic before going through a process of selection (e.g. principal component analysis [135]) to remove irrelevant and redundant features [150]. The selected features will then be passed to a classical machine learning

classifier, such as a support vector machine (SVM) [82], random forest [135] or decision tree (DT) [150], to be used for the final prediction. The commonly used features are usually based on morphological and statistical characteristics [150]. Examples of the more commonly used morphological features are the measure of area, roundness and elongation, while examples of commonly used statistical features are the mean, median and variance.

For the automatic operation, the features are usually extracted and predicted simultaneously by a CNN. Although both U-Net and FCN have been shown to be successful in various WSI segmentation applications [82], U-Net has proven to be superior [151]. U-Net has also been shown to outperform human experts for lymphocyte detection in immunohistochemically stained tissue sections of breast, colon and prostate cancer [151]. The superiority of U-Net performance against the FCN has also been shown on the application of renal tissue segmentation [152].

6.2.3 Training a Deep Learning Model

To develop a deep learning model for segmentation, including a CNN, a complete pixel-wise annotated dataset is required as the ideal supervision information [153]. However, as a complete pixel-wise annotated dataset is often unavailable in real-world applications, a basic assumption, e.g. a cluster assumption or a manifold assumption, can be adopted to annotate the non-annotated data for training [153]. Besides maximising the use of non-annotated data for training, it is crucial to ensure that the data for each class is balanced.

6.3 Tissue Preparation and Segmentation Label Extraction

In this section, we describe the tissue preparation and digitisation process to create the dataset. Then, we define the nerve detection criteria and describe the process used to obtain the segmentation label from incomplete coarsely annotated data.

6.3.1 Tissue Preparation and Digitisation

The images used in this chapter are from a dataset of histological specimens of benign and malignant thyroid tissue that has previously been described [128]. This study was approved by the Hunter New England Human Research Ethics Committee, who granted a waiver of consent for access to archival pathology material and approved the experimental protocol (2019/ETH13695). All study methodologies were carried out in accordance with relevant guidelines and regulations. This dataset [154] was chosen, firstly, because of the high specificity of the immunohistochemistry for nerve tissue with minimal background staining, and secondly, because a digital library of annotated nerves was already available. Briefly, 112 whole slide sections of 4 µm thickness, from formalin-fixed paraffin-embedded blocks of benign and malignant thyroid tissue, were labelled with immunohistochemistry for the pan-neuronal marker protein gene-product 9.5 (PGP9.5) using the Ventana Discovery automated slide stainer (Roche Medical Systems, Tuscon, Az), then counterstained with haematoxylin. The primary antibody was the anti-rabbit polyclonal PGP9.5 antibody (catalogue number #Ab15503, Abcam, Cambridge, United Kingdom) at 1:600 dilution. Slides were then digitised at 20 × magnification using the Aperio AT2 scanner (Leica Biosystems, Victoria, Australia). The dataset had been annotated by manual counting of where the annotated large nerve trunks met the following criteria: immunoreactivity (positive PGP9.5 staining), typical anatomical appearance, and three or more axons visualised. Smaller nerves and individual axons were not manually annotated. Manual review was performed using a grid overlay and manual scanning of the specimens by two human operators at 4 × magnification using QuPath (Queens University, Belfast) [155].

6.3.2 Criteria for Nerve Detection

As discussed in Section 6.1, nerve detection is considered to be a difficult task, as nerves vary in size, appearance and immunoreactivity to the PGP9.5 staining colour range. However, it is critical to define comprehensive criteria to reduce bias in the study. Thus, we develop the criteria based on four parameters: size, morphology,

anatomical location and immunoreactivity to PGP9.5 as detailed in Table 6.1.

Table 6.1: Criteria for Nerve Detection

Criterion	Definition	Justification
Anatomical location	Plausible anatomical location for neural tissue (e.g. vasa nevorum, interstitial spaces)	Structures outside of a plausible anatomical location were excluded.
Size	Greater than 25µm^2 (∼100 pixels)	Corresponds to a minimum size of 3 axons in cross-section. Smaller structures are difficult to confidently distinguish from non-specific staining
Morphology	Typical neural structures	Axon: linear structure, discrete edge
		Nerve: cluster of axons surrounded by perineurium
Immunoreactivity	Focal and discrete PGP9.5 staining	Diffuse and non-specific uptake of Dab was excluded.

As can be seen from Table 6.1, a nerve should be in a plausible anatomical location and have a minimum size of 25 µm^2 (∼100 pixels). In terms of appearance and colour, a nerve should show immunoreactivity to PGP9.5 staining (i.e. be brown in colour) and show typical neural structure (e.g. edges) clearly.

However, the minimum size of the manually annotated nerve in the dataset from [128] is approximately 100 µm^2 (400 pixels), which is about four times the minimum nerve size that we would like to detect. Hence, we use a 20×20 pixel (400 pixels) morphological closing operation, as discussed in Section 3.4.1, to combine predicted positive instances located close to each other and consider it as a single predicted positive instance. Predicted positives instances that are far from each other (e.g. axons of a nerve cluster) and that cannot be morphologically closed, will be counted as discrete predicted positive instances.

6.3.3 Segmentation Label Extraction

Here we consider the case of incomplete coarsely annotated data. Thus, we use assumptions, based on colour and location, to determine the pixel-wise segmentation labels of the coarsely annotated data to maximise the use of non-annotated data for learning, as discussed in Section 6.2.3. If a pixel is brown in colour and intersects with the annotations, it is labelled positive, otherwise it is labelled negative.

The main challenge is in determining the colour filter range for the pixel-wise segmentation labels that results in a minimum number of label artifacts. It is difficult to determine a colour filter range that can detect the immunoreactivity to PGP9.5 staining with high sensitivity and high precision (i.e. high true positives and low false positives), as a colour filter range with high sensitivity usually results in low precision. Moreover, all the true positives and false positives cannot be determined from incomplete annotations. Thus, the range is defined empirically by observing the number of true positives (i.e. detected annotations) and estimated false positives (i.e. label artifacts from detected non-annotated brown stains). A colour filter predicted positive instance is considered to be a true positive if the predicted positive instance intersects with any of the annotations. Here we use a colour filter range that results in approximately 98% sensitivity to minimise the label artifacts.

Another challenge is to extract true negative training data samples. With incomplete coarsely annotated data of only true positives, the negative data samples cannot be extracted reliably. There are many nerves that were falsely annotated negative by exclusion in the expert-annotated data. Therefore the colour filter predicted positive instances located outside the annotations cannot be used directly as negative training data samples. To solve this problem, we define an area within each training WSI where negative training data samples are to be extracted. The regions are chosen such that the number of unspecified brown stains is maximised and the number of false negatives is minimised.

6.4 Nerve Detection Approach and Proposed Architecture

In this section, a detailed description of the proposed nerve detection approach is provided. We describe a novel classification structure to augment a U-Net to improve the performance of the object detection task.

6.4.1 Nerve Detection Approach

The nerve detection approach consists of three main stages: pre-processing, network prediction and post-processing. The approach starts with the pre-processing for ROI extraction using a colour filter, followed by image patch preparation for the CNN input and ends with the combination of the network prediction results for nerve quantification. The end-to-end process can be seen in Figure 6.4.

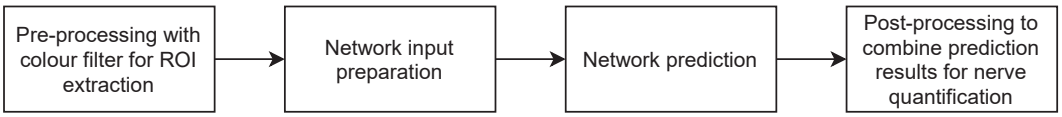


Figure 6.4: End-to-end process flowchart.

Pre-processing includes a filtering process based on colour for the extraction of ROIs and network input preparation. For the extraction of the ROIs, we downsampled the WSI by 4 to give an image of approximately $10,000 \times 10,000$ pixels. The WSI is then divided into 256 non-overlapping image blocks, where each image block consists of approximately 625×625 pixels. We then apply the colour filter as described in Section 6.3.3. Note that the binary output of the colour filter is quite noisy, as it catches many small artifacts of brown stains that do not belong to a nerve. A 20×20 pixel morphological closing is then performed to combine brown stains that potentially belong to a nerve, i.e. separated axons in a nerve fibre or nerve trunks, and extract the resulting combined structures that exceed 100 pixels as ROIs. Finally, we combine ROIs that overlap each other to form a larger ROI. The ROI extraction flowchart is given in Figure 6.5.

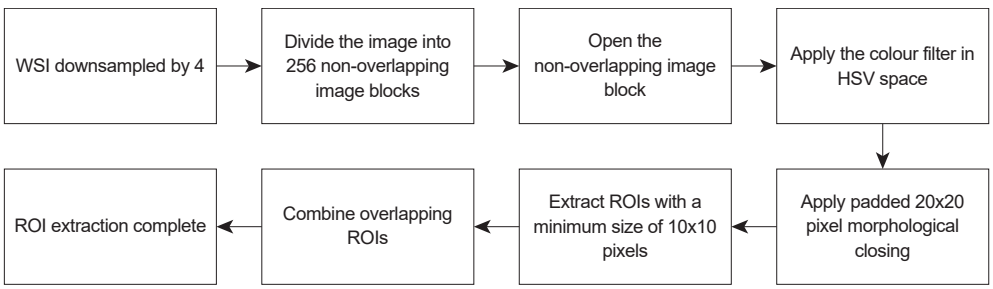


Figure 6.5: ROI extraction flowchart.

The CNN we use accepts a fixed 160×160 pixel, 3 channel (RGB) image. Due

to the variation in the size of the nerves, we convert each of the arbitrary size ROIs to 160×160 pixel image patches. To accomplish this, we use the process shown in Figure 6.6. Here, we divide the ROI into 50% overlapping, 160×160 pixel image patches without the use of any reshaping function. Basically, we check if the size of the ROI is greater than 160×160 pixels; if it is true, we divide the ROI height and width with the image patch size and obtain the number of image patches that will be extracted from the ROI. The number of image patches extracted from each ROI can be calculated as follows,

$$N_d = 2 \left\lceil \frac{ROI_d}{P_d} \right\rceil - 1, \tag{6.1}$$

where d denotes the dimension in pixels (either height or width), N denotes the number of 50% overlapping patches that are required to cover the size of either the ROI height or width, and P denotes either the patch height or width.

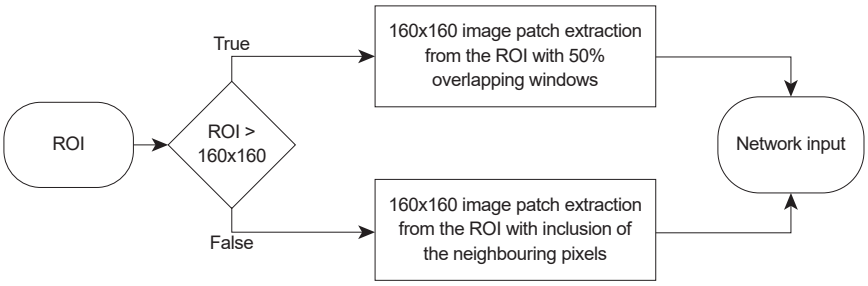


Figure 6.6: Network input preparation flowchart.

After the pre-processing process is complete, a CNN is used to perform pixel-wise segmentation of the nerve in each image patch. The output of the network is the same size as the input and is a binary array of pixel-wise predictions. However, the output may still contain predicted positive instances that are too small, or a cluster of predicted positive instances that are separated from each other. Hence, a post-processing process is required to omit the small predicted positive instances below the minimum size threshold, or to combine the predicted positive instances into a larger instance for better nerve quantification.

The post-processing begins with an assembly process to combine the prediction patches to form a complete ROI. Then, the complete ROI is processed by a 20×20

pixel morphological closing with a 100 pixel minimum size thresholding, as performed in the pre-processing step. However, for the combination process, only the prediction boxes that overlap by at least 50% are combined to ensure the final predictions are as precise as possible. Hence if there are two prediction boxes that slightly overlap each other, we count them as two individual nerves. The post-processing flowchart is shown in Figure 6.7.

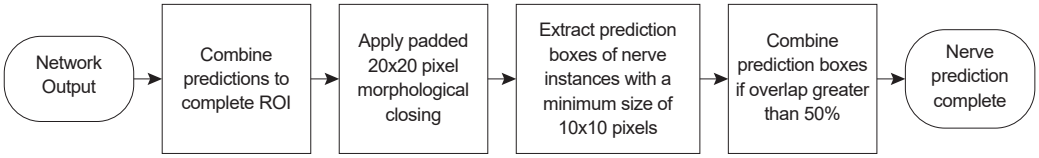


Figure 6.7: Post-processing flowchart.

6.4.2 Proposed Architecture

In this section, we propose the addition of a classification structure to the U-Net architecture from Chapter 5, as the nerve detection task involves an image classification process. Although a multi-stage classification and segmentation approach can be used, the training can be computationally expensive, as separate training processes are required. Thus, we propose to augment the U-Net architecture from Chapter 5 with an image classification structure that can be trained in an end-to-end manner to improve the segmentation results.

As discussed in Chapter 5, the U-Net uses a contracting-expanding structure to perform pixel-wise classifications. The contracting part of the network is used for automatic feature extraction of the input image, while the expanding part of the network is used to map the extracted high-level features back to the original input resolution and use them to perform pixel-wise classification for the segmentation output. The augmented classification structure is designed to use the extracted high-level features for image classification and then have the classification output assist the final segmentation results. The aim of the augmented classification structure is to reduce the false positive predictions of the segmentation network. The proposed network architecture with the augmented classification structure is shown in Figure 6.8.

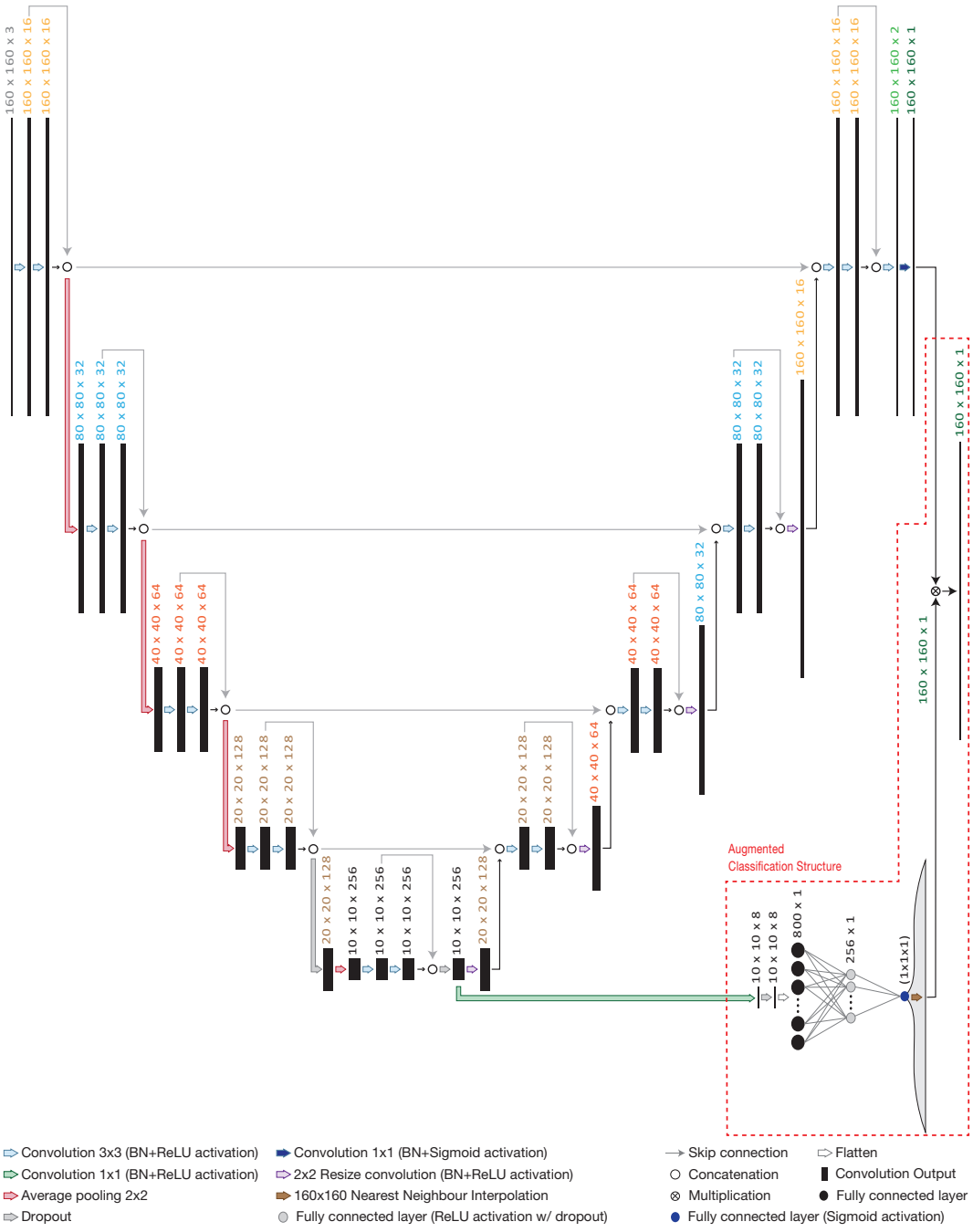


Figure 6.8: Proposed network architecture with augmented classification structure.

The augmented classification structure consists of a 1×1 convolutional layer for a feature layer dimensional reduction, a 2-layer fully connected network for classification and a reshaping layer for the classification output shape adjustment. The reshaping layer takes the 1×1 classification output (with probability range of 0 to 1) and reshapes it to the input image size of 160×160 by duplication. A dropout layer, with a dropout rate of 0.5, is applied after the 1×1 convolutional layer

and before the last fully connected layer for regularisation. Finally, the reshaped output is multiplied with the output of the segmentation network to obtain the final network output, which will be thresholded by 0.5. This results in a segmentation network that only produces a positive segmentation output when the results of the classification and segmentation output are higher than 0.5. The block diagram of the proposed network with the augmented classification structure is shown in Figure 6.9.

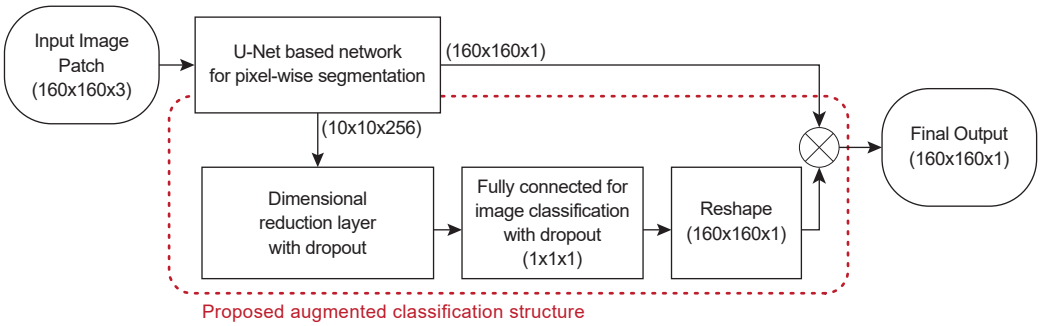


Figure 6.9: Block diagram of the proposed network architecture with augmented classification structure.

6.5 Training and Results

In this section, we describe the training process, including the way in which the dataset is used and divided for the development of the model, as well as the test results at a WSI level. The results of the proposed approach are compared with the results from the manual counting and other automatic approaches.

6.5.1 Dataset

The dataset consists of 112 stained thyroid tissue WSIs annotated by two experts (see Section 6.3.1). The dataset is split into training, validation and test sets. The training set consists of 80 WSIs, which includes 52 WSIs containing the largest number of annotated nerves and 28 randomly selected WSIs. The validation set consists of 10 randomly selected WSIs, while the test set consists of the remaining 22 WSIs.

6.5.2 Training

To ensure the training is effective, a balanced dataset is required. Here training is performed with a training generator that performs data augmentation (e.g. flip and rotate) and generates balanced training data samples, where an equal number of positive and negative data samples are randomly chosen at every iteration.

The He initialisation is used for the weight initialisation. The Adam optimiser is utilised in the model training using a learning rate of 10^{-4} and a binary cross-entropy cost function. The model is trained over 250 epochs with handpicked positive training data samples (∼2000 data samples). A few examples of both the positive and negative training data samples and their corresponding labels are shown in Figure 6.10.

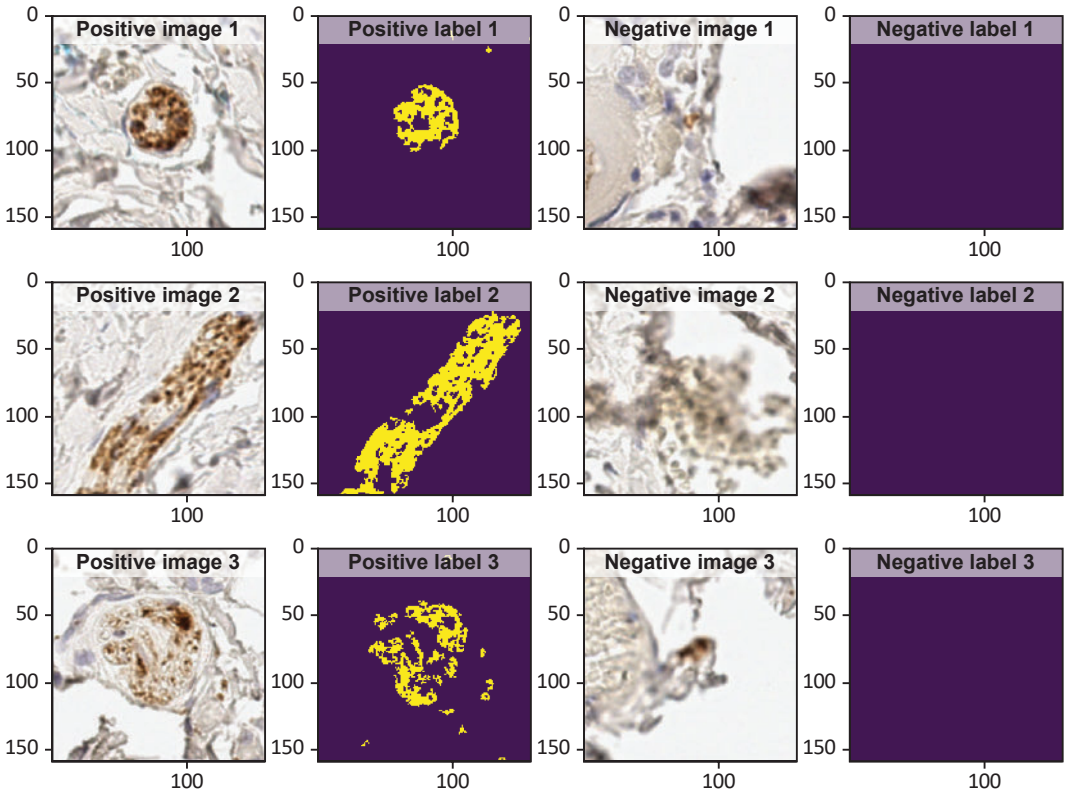


Figure 6.10: Examples of the positive and negative training samples.

We exclude about 500 positive training data samples that contain a significant number of label artifacts to ensure accurate representation of the target class, i.e. nerves. Examples of excluded positive training data samples due to label artifacts

are shown in Image A and B of Figure 6.11, while the pixel-wise segmentation labels of the corresponding image patches are shown in Label A and B of Figure 6.11. As can be seen from example A and B, a significant number of label artifacts, i.e. positive pixel-wise labels located outside the red box, are apparent in the form of a blob and scattered structure respectively.

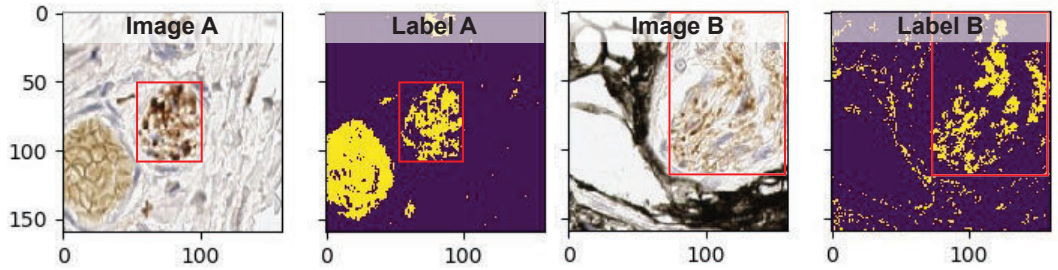


Figure 6.11: Excluded positive training data samples due to label artifacts from the colour filter. The red box surrounds the actual nerve.

6.5.3 Performance Metric

The main objective of this chapter is to provide an automatic approach for the quantification of nerves in a thyroid tissue WSI. From this perspective, it is important to evaluate the proposed approach at a WSI level instead of at a patch level. As there is no precise pixel-wise label available, the performance will be evaluated by a hit or miss method. A hit indicates a true positive, while a miss indicates either a false negative or false positive. A hit is determined when a predicted positive instance intersects with an expert annotation, i.e. true positive from manual annotations, TP_m. When multiple predicted positive instances intersect with an expert annotation, it will be scored as one hit. An expert annotation that has no intersection with any predicted positive instance will be scored as a miss, i.e. false negative, FN. If a predicted positive instance does not intersect any of the expert annotations, a hit is then determined by experts, who evaluate whether the predicted positive contains any nerve, i.e. additional detection true positive, TP_a. If the predicted positive instance contains no nerve, as determined by the experts, it will be scored as a miss, i.e. false positive, FP. The evaluation (scoring) is performed by three experts, i.e. E1, E2 and E3, and the final

number of true positives will be determined from the average across the experts.

Sensitivity, also referred to as true positive rate (TPR), is used to evaluate the number of annotated nerves detected. Sensitivity [156] is given by,

$$TPR = \frac{TP_m}{(TP_m + FN)}, \tag{6.2}$$

where TP_m indicates the number of true positives detected from the manual annotations and FN indicates the number of false negatives.

Precision, also referred to as positive predicted value (PPV), is used to evaluate the ability of an approach to detect nerves. Precision [156] is given by,

$$PPV = \frac{TP}{(TP + FP)}, \tag{6.3}$$

where TP indicates the total number of true positives, i.e.

$$TP = TP_m + TP_a, \tag{6.4}$$

TP_a indicates the number of true positives in the additional detections, i.e. nerves missed by the experts in the manual annotations, and FP indicates the number of false positives.

6.5.4 Results

In this section, we present a performance comparison between manual counting by experts and two automatic approaches, i.e. colour filter and CNN based approaches. The colour filter based approach (CF) relies only on a colour filter, while the CNN based approach uses a CNN to perform a pixel-level nerve detection on the colour filter output, as described in Section 6.4.1. Here, the two CNN based approaches are denoted as APR-A and APR-B. APR-A indicates the use of the U-Net from Chapter 5 (Figure 6.8 without the augmented classification structure) and APR-B indicates the use of the proposed U-Net, which includes the augmented classification structure, as shown in Figure 6.8. The main objective of this section is to evaluate the proposed CNN based approach (APR-B) in terms of performance for automatic

nerve detection.

Sensitivity (TPR) and precision (PPV) are used for performance evaluation of the three approaches on each WSI in the test set. The evaluation scores of the corresponding metrics for each WSI are presented in Table 6.2 and Table 6.3. Due to the large number of predicted positive instances by the CF, we use a random sampling to obtain 100 predictions for each WSI to facilitate the experts in the evaluation of the true positives. The performance evaluation of the CF in Table 6.3 will be based on these 100 samples, where the proportion of correct predictions will be multiplied by the total number of predictions, and hence denoted as estimated performance.

Table 6.2 provides the number of detected nerves for the CF, APR-A and APR-B with respect to the expert manual annotations. The identified manual annotations columns indicate the number of expert annotations that are detected by each approach. The additional detection columns indicate nerves detected by the approaches that were missed by the experts. Note, however, that the additional detections may contain false positives.

Table 6.2: Results with respect to the expert manual annotations

WSI ID	Manual Annotations	Identified manual annotations						Additional detections		
		CF		APR-A		APR-B				
		TP_m	TPR	TP_m	TPR	TP_m	TPR	CF	APR-A	APR-B
10012	10	9	0.9	9	0.9	9	0.9	847	21	14
10023	12	12	1	11	0.92	11	0.92	2658	9	11
10029	11	11	1	8	0.73	8	0.73	3390	7	7
10036	9	8	0.89	7	0.78	8	0.89	4316	239	159
10039	35	35	1	34	0.97	34	0.97	6591	421	274
10049	15	15	1	14	0.93	13	0.87	7185	27	20
10064	28	28	1	27	0.96	28	1	1922	18	21
10067	27	27	1	27	1	25	0.93	815	117	61
10071	21	20	0.95	17	0.81	17	0.81	3113	42	28
10072	15	14	0.93	13	0.87	13	0.87	492	22	12
10073	2	2	1	1	0.5	1	0.5	8348	11	9
10078	15	15	1	13	0.87	12	0.8	9884	208	102
10087	0	0	-	0	-	0	-	877	8	6
10088	8	8	1	8	1	8	1	1689	28	47
10093	11	11	1	10	0.91	10	0.91	2773	3	5
10096	1	1	1	1	1	1	1	1019	24	10
10097	19	18	0.95	17	0.89	18	0.95	6907	356	203
10102	25	25	1	24	0.96	24	0.96	4799	224	167
10113	16	15	0.94	14	0.88	13	0.81	1504	83	38
10114	14	14	1	14	1	14	1	6529	819	507
10116	16	16	1	15	0.94	15	0.94	5005	148	74
10121	8	8	1	8	1	8	1	4278	215	121
Overall	**318**	**312**	**0.98**	**292**	**0.90**	**290**	**0.89**	**84941**	**3050**	**1896**

Abbreviations: WSI: Whole Slide Image; CF: Colour filter; APR-A: CNN based approach A ; APR-B: CNN based approach B; TP_m: True positives detected from the manual annotations; TPR: Sensitivity.

As can be seen from Table 6.2, the CF has the highest rate of annotated nerve detection, with an average sensitivity of 98%. The CNN based approaches have an average sensitivity of 90% for APR-A and 89% for APR-B. However, the CF predicted a total of 84,941 additional positive detections, while the CNN based approaches predicted a total of 1,896 (APR-B) and 3,050 (APR-A), which is more than an order of magnitude less than the CF. However, the quality of the overall performance can only be determined by the evaluation of the true positives of these predictions.

Table 6.3: Precision score of each automatic approach

WSI ID	Additional detections						Total					
	CF		APR-A		APR-B		CF		APR-A		APR-B	
	Est. TP_a	Est. FP	TP_a	FP	TP_a	FP	TP	PPV	TP	PPV	TP	PPV
10012	45	802	17	4	11	3	54	0.06	26	0.87	20	0.87
10023	89	2569	5	4	7	4	101	0.04	16	0.80	18	0.82
10029	0	3390	5	2	5	2	11	0.00	13	0.87	13	0.87
10036	432	3884	169	70	126	33	440	0.10	176	0.72	134	0.80
10039	712	5879	270	151	220	54	747	0.11	304	0.67	254	0.82
10049	48	7137	19	8	16	4	63	0.01	33	0.80	29	0.88
10064	26	1896	8	10	13	8	54	0.03	35	0.78	41	0.84
10067	244	571	93	24	56	5	271	0.32	120	0.83	81	0.94
10071	197	2916	23	19	22	6	217	0.07	40	0.68	39	0.87
10072	77	415	18	4	9	3	91	0.18	31	0.89	22	0.88
10073	167	8181	4	7	7	2	169	0.02	5	0.42	8	0.80
10078	659	9225	92	116	54	48	674	0.07	105	0.48	66	0.58
10087	9	868	6	2	5	1	9	0.01	6	0.75	5	0.83
10088	146	1543	18	10	33	14	154	0.09	26	0.72	41	0.75
10093	55	2718	2	1	3	2	66	0.02	12	0.92	13	0.87
10096	27	992	5	19	5	5	28	0.03	6	0.24	6	0.55
10097	1036	5871	268	88	155	48	1054	0.15	285	0.76	173	0.78
10102	720	4079	145	79	125	42	745	0.15	169	0.68	149	0.78
10113	140	1364	70	13	31	7	155	0.10	84	0.87	44	0.86
10114	1372	5157	533	286	314	193	1386	0.21	547	0.66	328	0.63
10116	233	4772	112	36	58	16	249	0.05	127	0.78	73	0.82
10121	399	3879	65	150	29	92	407	0.09	73	0.33	37	0.29
Overall	**6833**	**78108**	**1947**	**1103**	**1304**	**592**	**7145**	**0.09**	**2239**	**0.70**	**1594**	**0.78**

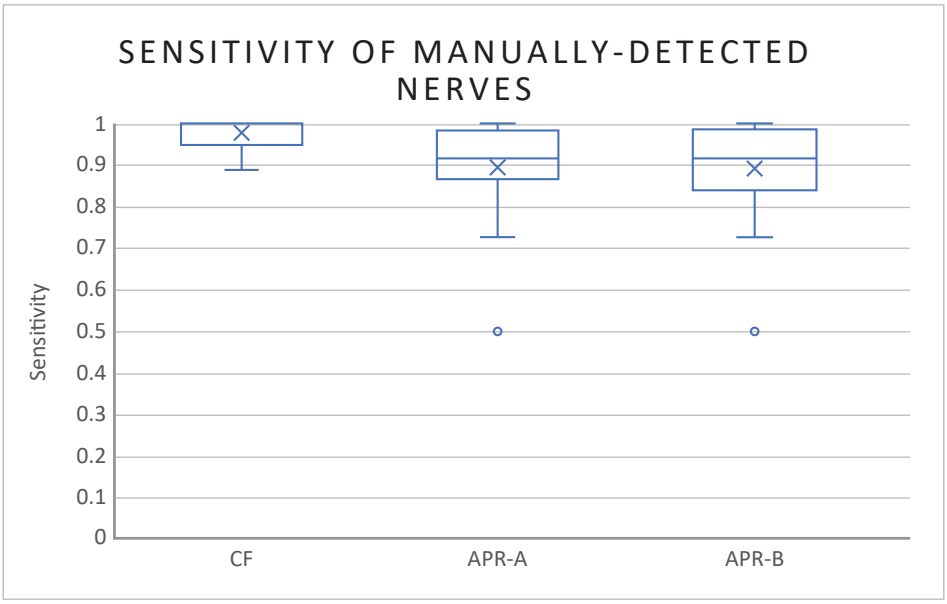
Abbreviations: WSI: Whole Slide Image; CF: Colour filter; APR-A: CNN based approach A; APR-B: CNN based approach B; Est.: Estimated performance based on 100 samples; TP_a: The number of true positives in the additional detections; FP: False Positives; TP: Total true positives; PPV: Precision.

Table 6.3 details the additional detections from the colour filter and CNN based approaches in terms of true and false positives. It provides the number of nerves (i.e. true positives, TP_a) that were not detected by expert manual annotation, and the number of predicted positive instances that were not nerves (i.e. false positives, FP) for each of the automatic approaches. It also shows the precision of the automatic approaches.

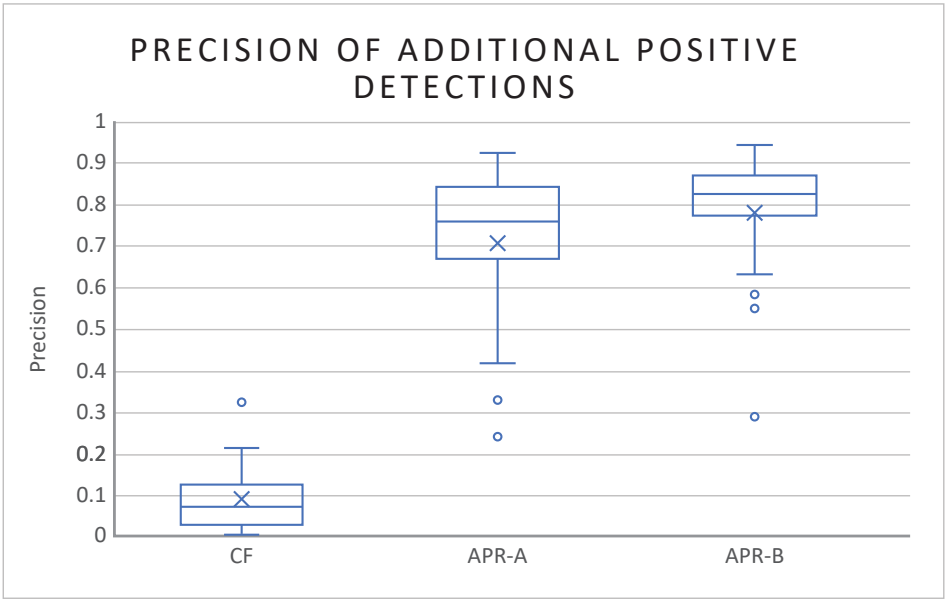
As can be seen from Table 6.3, all of the automatic approaches are capable of detecting a significantly higher number of true positives, at least five times more

than the number of nerves manually detected by the experts shown in Table 6.2. However, compared to the CF, the APR-A and APR-B perform substantially better in correctly identifying nerves (precision). The CF has an average precision of 9%, while APR-A and APR-B have an average precision of 70% and 78%, respectively. Although, the CF has the highest number of additional positive detections, the number of false positives is extremely large, hence the detection is not meaningful. This significant improvement in precision makes the detection substantially more meaningful.

A summary of the sensitivity and precision of each of the approaches, as evaluated on the test set, is presented in the box-and-whiskers plots shown in Figure 6.12 where it can be seen that APR-B outperforms the CF and APR-A.



(a)



(b)

Figure 6.12: (a) Sensitivity of the CF, APR-A and APR-B in the detection of the expert annotated nerves; (b) Precision of CF, APR-A and APR-B in the detection of nerves (bottom). The × symbol in the box indicates the mean value, while the line in the box indicates the median value. The ∘ represents the outlier, the box represents the interquartile range and the whiskers represent the upper and lower extreme, excluding the outliers.

Several examples of true positives detected by the automatic approaches but missed by the experts are shown in Figure 6.13 (A-H). It can be seen that the automatic approaches can detect nerves of various sizes and appearances. Nerves

within the model input size (160 × 160 pixels) can be detected as shown in Figure 6.13 (A-D and G-H), as well as larger nerves as shown in Figure 6.13 (E-F). A nerve in the form of a cluster of axons that are close to each other is considered as a single nerve, as shown in Figure 6.13 (A-F). A nerve in the form of multiple clusters of axons that are far apart from each other is considered as separate nerves and quantified as multiple discrete nerves, as shown in Figure 6.13 (G-H).

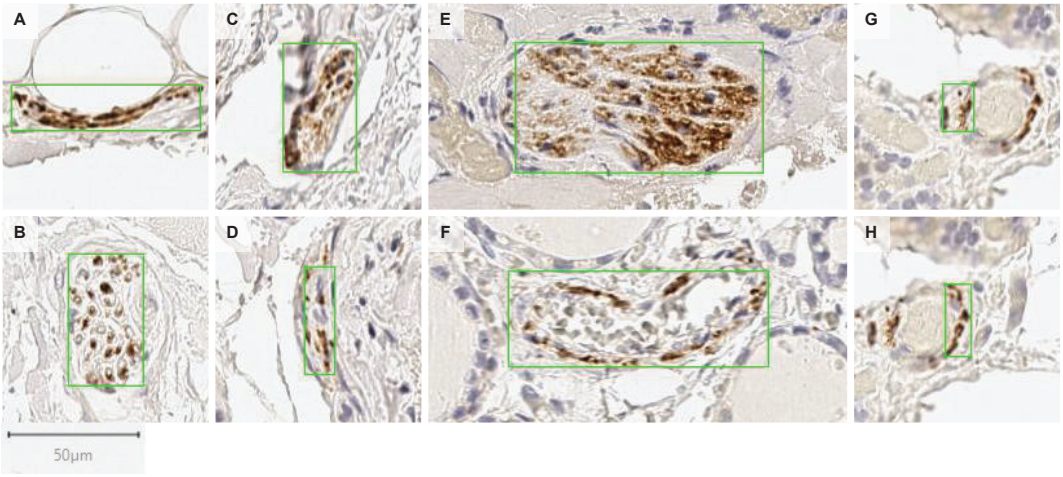


Figure 6.13: Examples of nerves missed by the experts which are detected (i.e. true positives, TP_a) by the automatic approaches. Each image patch contains one prediction instance, indicated by a green box.

The results show that the average precision score of APR-B is 8% higher than APR-A. The use of the augmented classification structure reduced the number of false positives by approximately 46% of the U-Net used in APR-A. The augmented classification structure helps to reduce the false positive predictions, for example, by excluding those shown in the top row of Figure 6.14. However, some false positives, especially those located in the vicinity of nerves, still remain, as shown at the bottom row of Figure 6.14.

Although APR-B has the highest average precision score in the test set, it does not perform as well on some WSIs. We found that the reduction of precision in some of the WSIs can be caused by either fewer detected true positives or the presence of a large number of a specific type of unspecified brown stains in the WSI, as shown in Figure 6.15.

Overall, the experts detect the least number of nerves in the test set, with a

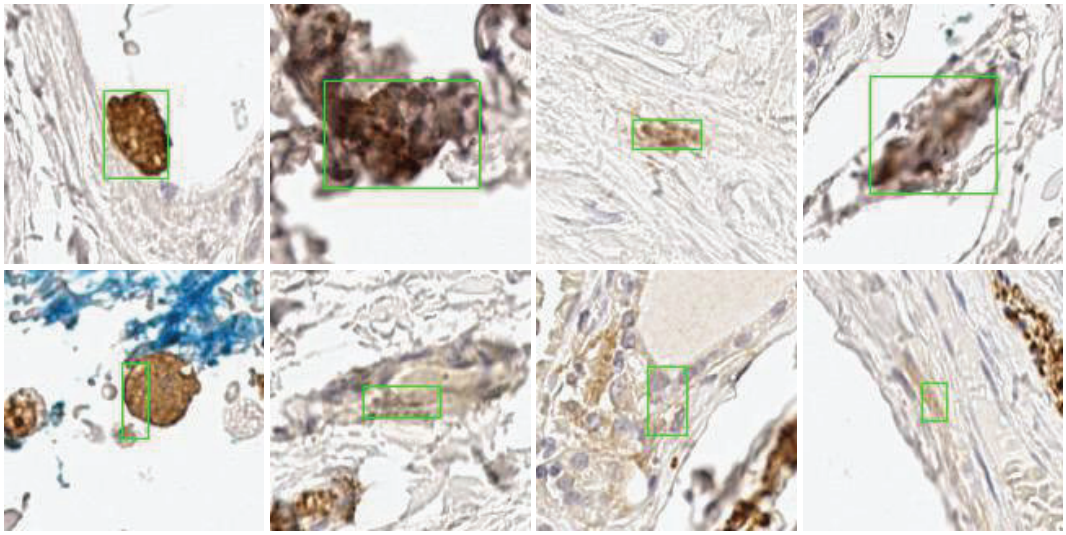


Figure 6.14: Excluded false positives by APR-B (top row). False positives retained by APR-B (bottom row). Green box indicates a false positive instance.

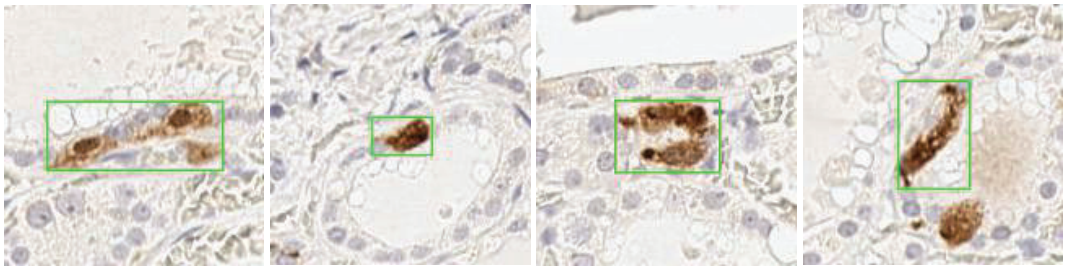


Figure 6.15: Examples of a specific type of unspecified brown stain present in some WSIs, causing a large number of false positives detected by APR-B. Green box indicates a false positive instance.

very high precision and low sensitivity. The CF detects approximately 22 times the nerves detected by the experts, however, with an extremely low precision of 9% due to the large number of false positives. The CNN based approaches have an average precision of 70%, APR-A, and 78%, APR-B. This shows that incorporating a CNN into a traditional workflow will result in a considerable improvement, with more meaningful results obtained.

6.6 Discussion

The proposed CNN based approach has significantly improved the precision of the detections compared to the existing automatic methods, however, there are a small number of slides where the performance is not that good. For example on slide 10078 and 10121, there is a degradation in performance that is caused by a lack of representation of some of the unspecified brown stains in the training and validation set. The proposed CNN based approach results in a low precision score when a large number of poorly represented unspecified brown stains are present in the slide.

High inter-and intra-expert variance causes the training set and the scoring criteria to be inconsistent, which also leads to a degradation in performance. This inconsistency makes the learning as well as the development of a CNN model more difficult and complicated. The inter-expert variance can be seen in Table 6.4, where the percentage of the scoring agreements between the experts (E1 and E2, E1 and E3, and E2 and E3) has an average of 78.73%. This means that there is a margin of error in the scoring of approximately 22%.

Table 6.4: Inter-expert scoring agreement

Experts	Average Agreement (%)
E1,E2	76.49
E1,E3	81.45
E2,E3	78.27
Average	**78.73**

The proposed CNN based approach has a large dependency on the colour filter output, hence the performance of the CNN based approach is limited by the performance of the colour filter.

6.7 Additional Implementation Considerations

In this section, we will address the insufficient class representation problem in the training set, as well as the dependency of the proposed CNN based approach on the performance of the colour filter.

6.7.1 Refinement of Data Representation

In this section, we will reduce the insufficient representation problem in the training set by reselecting the negative training data samples. We will present our approach for selecting the negative training data samples as well as the results of our proposed network, used in APR-B, trained with the reselected negative training data samples.

As discussed in Section 6.2.3, it is important to develop a deep learning model with a balanced dataset. In Section 6.5.2, we used a balanced set of 50% positive and 50% negative training data samples in every iteration of the training to fulfil the requirement. However, even though the data samples are balanced in terms of classes, the random sampling method used to extract the balanced dataset does not take the intra-class variance into account. When taking the intra-class variance into consideration, the classification problem becomes more specific to the morphological criteria, because the non-target class, i.e. the non-specific brown stains, has most of the properties in our target class, i.e. detection criteria in terms of anatomical location, size and immunoreactivity. Hence, the non-target class should be further divided into multiple categories and the negative training data samples should be extracted evenly from each category to ensure the representation of each category is sufficient. However, due to its size, it is not practical to hand pick the negative training data samples individually.

From a manual observation of the types of non-specific brown stains present in our training set, we can organise them into multiple categories depending on their morphological appearance, as shown in Figure 6.16. Then, we define the upper limit on the amount of data samples to be extracted from each WSI according to the varieties of the non-specific brown stains present in the WSI. Finally, the negative data samples are extracted automatically in a random order from the WSIs in the

training set according to the pre-defined upper limit.

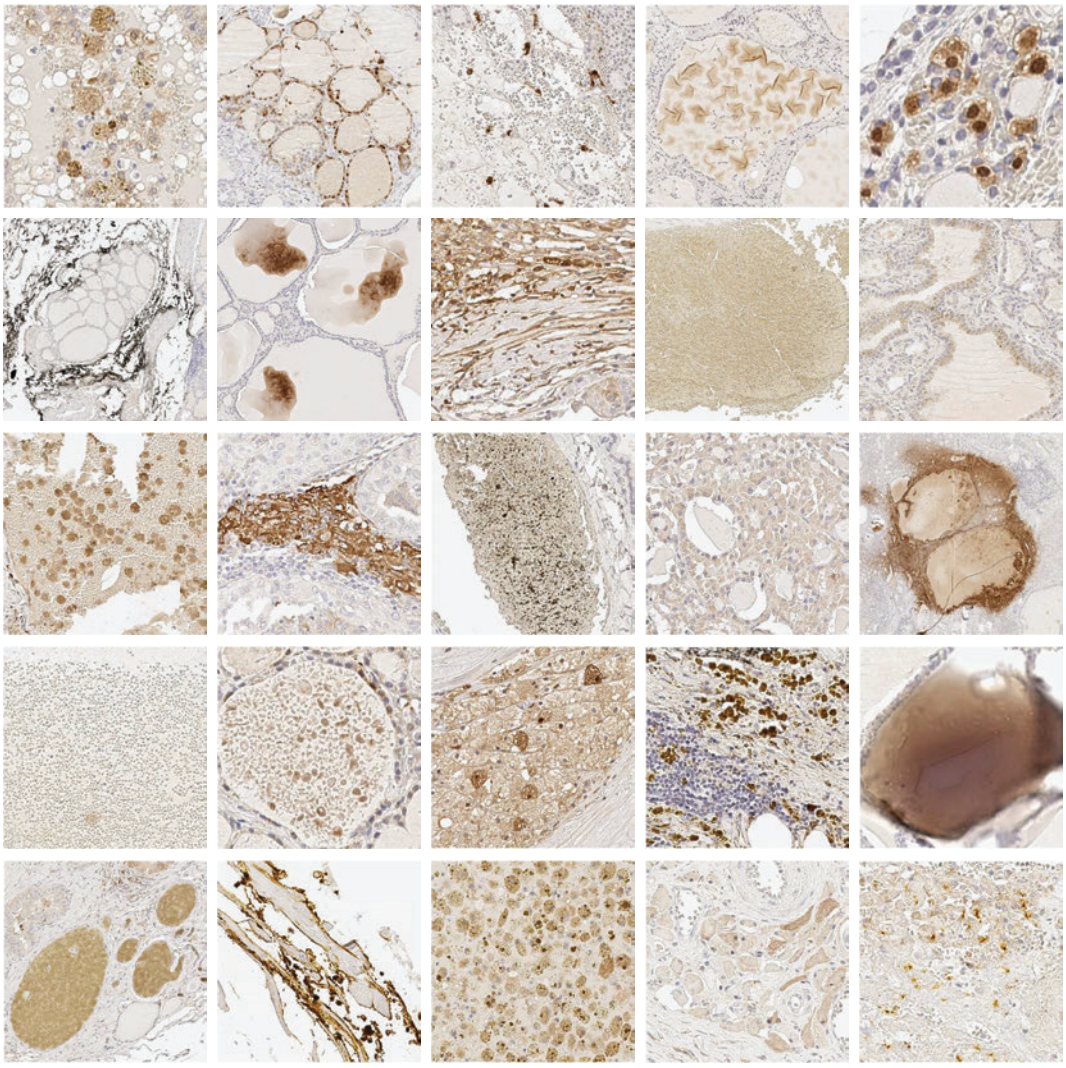


Figure 6.16: Negative training data sample categories according to their morphological appearance.

The results of the proposed network, used in APR-B, trained with the resampled training data, APR-BR, are given in Table 6.5. The average sensitivity decreased by 4% (from 89% to 85%), however, the average precision improved by 9% (from 78% to 87%) for APR-BR with respect to APR-B. The total number of detected nerves (TP) also increased by approximately 55% compared to APR-B (from 1594 to 2484), which is approximately seven times more than the number of nerves manually detected by the experts. Furthermore, the precision of slide 10078 and 10121 significantly improved from 58% and 29% to 78% and 79% respectively for APR-BR with respect to APR-B. This shows that the performance of a deep learning

model can be improved by increasing the representation of various unspecified brown stains that are present within the class, i.e. intra-class representation.

Table 6.5: Additional results of APR-B with resampled training data samples (APR-BR)

WSI	Manual	IMA (APR-BR)		AD (APR-BR)			Total (APR-BR)	
ID	Annotations	TP_m	TPR	Total	TP_a	FP	TP	PPV
10012	10	9	0.90	24	21	3	30	0.91
10023	12	11	0.92	9	7	2	18	0.90
10029	11	7	0.64	6	5	1	12	0.92
10036	9	5	0.56	158	138	20	143	0.88
10039	35	30	0.86	240	199	41	229	0.85
10049	15	13	0.87	13	10	3	23	0.88
10064	28	25	0.89	9	7	2	32	0.94
10067	27	25	0.93	68	62	6	87	0.94
10071	21	16	0.76	17	16	1	32	0.97
10072	15	13	0.87	12	10	2	23	0.92
10073	2	1	0.50	8	7	1	8	0.89
10078	15	11	0.73	97	73	24	84	0.78
10087	0	0	-	5	4	1	4	0.80
10088	8	8	1.00	37	28	9	36	0.80
10093	11	10	0.91	3	3	0	13	1.00
10096	1	1	1.00	7	4	3	5	0.63
10097	19	17	0.89	341	287	54	304	0.85
10102	25	23	0.92	215	182	33	205	0.86
10113	16	14	0.88	98	93	5	107	0.96
10114	14	13	0.93	947	786	161	799	0.83
10116	16	15	0.94	174	144	30	159	0.84
10121	8	8	1.00	157	123	34	131	0.79
Overall	**318**	**275**	**0.85**	**2645**	**2209**	**436**	**2484**	**0.87**

Abbreviations: WSI: Whole Slide Image; IMA: Identified manual annotations; AD: Additional detections; APR-BR: Proposed network in APR-B trained with resampled training data; TP_m: True positives detected from the manual annotations; TPR: sensitivity; TP_a: indicates the number of true positives in the additional detections; FP: False Positives; TP: Total true positives; PPV: Precision.

6.7.2 Direct Implementation

In this section, we will remove the dependency of our proposed CNN based approach on the colour filter, that was used for the purpose of training data selection and implementation efficiency. As discussed in Section 3.3.4, for highly skewed data, the training set should represent the intra-class variance well enough to train the model effectively. Without the colour filter, it will be difficult to select significant negative data samples from the WSI for the training. The use of a colour filter also reduces the time of the testing process as the ROI extraction process, in Section 6.4.1, can be applied on 4× downsampled images without significantly affecting the final results and the extracted ROIs can be utilised repeatedly to evaluate any other model.

Here, we perform a direct implementation of our best model, used in APR-BR, on the WSIs in the test set with a 50% sliding window in both horizontal and vertical directions. We omit the ROI extraction process from the nerve detection approach in Section 6.4.1 and begin the detection with the network input preparation process that divides the WSI into 160×160 image patches with a 50% overlapping window. This direct implementation takes approximately 20 times longer than the nerve detection approach using a colour filter. However, the implementation is more straightforward and its performance does not rely on the output of the colour filter. The results are presented in Table 6.6 in the Direct APR-BR column.

As can be seen from Table 6.6, the number of identified manual annotations decreases from 275 to 266. This leads to an investigation that reveals the network input preparation process results in the ROIs extracted by the colour filter to be mostly located in the middle of the image patches. In this case, as the adopted data augmentation does not include a shifting or translation operation, the network lacks sufficient integration of the translation-invariance property as discussed in Section 3.3.3. Thus, we apply both horizontal and vertical shifting data augmentation on the 160×160 image patches and train the network further. The results are shown in Table 6.6 in the Direct APR-BRS column. As can be seen from the table, the detection of the manual annotations increased from 266 (Direct APR-BR) to 287 (Direct APR-BRS), which means that by performing shifting data augmentation,

we can increase the sensitivity of the network.

Table 6.6: Additional results on direct implementation of APR-BR and APR-BRS

WSI ID	Manual Annotations	Identified manual annotations				Additional detections			Total	
		Direct APR-BR		Direct APR-BRS		Direct APR-BRS			Direct APR-BRS	
		TP_m	TPR	TP_m	TPR	Total	TP_a	FP	TP	PPV
10012	10	10	1.00	9	0.90	50	34	16	43	0.73
10023	12	11	0.92	11	0.92	63	39	24	50	0.68
10029	11	5	0.45	8	0.73	21	11	10	19	0.66
10036	9	5	0.56	9	1.00	385	304	81	313	0.79
10039	35	28	0.80	31	0.89	419	316	103	347	0.77
10049	15	12	0.80	12	0.80	26	15	11	27	0.71
10064	28	23	0.82	25	0.89	33	21	12	46	0.79
10067	27	23	0.85	25	0.93	146	126	20	151	0.88
10071	21	18	0.86	18	0.86	40	29	11	47	0.81
10072	15	13	0.87	14	0.93	45	37	8	51	0.86
10073	2	1	0.50	1	0.50	25	17	8	18	0.69
10078	15	13	0.87	13	0.87	99	70	29	83	0.74
10087	0	0	-	0	-	11	10	1	10	0.91
10088	8	8	1.00	8	1.00	100	70	30	78	0.72
10093	11	9	0.82	11	1.00	17	11	6	22	0.79
10096	1	1	1.00	1	1.00	32	18	14	19	0.58
10097	19	16	0.84	18	0.95	511	395	116	413	0.78
10102	25	22	0.88	24	0.96	405	285	120	309	0.72
10113	16	13	0.81	14	0.88	245	185	60	199	0.77
10114	14	13	0.93	13	0.93	1059	806	253	819	0.76
10116	16	15	0.94	15	0.94	318	222	96	237	0.71
10121	8	7	0.88	7	0.88	296	198	98	205	0.68
Overall	**318**	**266**	**0.83**	**287**	**0.89**	**4346**	**3219**	**1127**	**3506**	**0.75**

Abbreviations: WSI: Whole Slide Image; APR-BR: Proposed network in APR-B trained with resampled training data; APR-BRS: Proposed network in APR-B trained with resampled training data and shifted augmentation; TP_m: True positives detected from the manual annotations; TPR: sensitivity; TP_a: indicates the number of true positives in the additional detections; FP: False Positives; TP: Total true positives; PPV: Precision.

We increased the average sensitivity of Direct APR-BRS, with respect to APR-BR, to 89%, however, the average precision (75%) is lower than the APR-BR (87%) and also APR-B (78%). However, the minimum precision of Direct APR-BRS is 58%, which is much higher than APR-A (24%) and APR-B (29%) while only slightly lower than APR-BR (63%). The precision of slide 10078 (74%) and 10121 (68%) are also within the range of other slides. This shows that, similar to APR-BR, Direct APR-BRS is not longer suffering from insufficient representation. Also, Direct APR-

BRS detects the most number of nerves (3506) compared to APR-A (2239), APR-B (1594) and APR-BR (2484). The overall lower precision score may be caused by higher misidentification of small nerves, either by the model or the expert, as Direct APR-BRS detects significantly more small nerves compared to APR-A, APR-B and APR-BR. However, we show that the proposed network architecture is capable of achieving satisfactory performance without being dependant on other algorithms. We also show that the performance of a network can be improved with an improved implementation.

6.8 Conclusion

In this chapter, we defined a detailed nerve quantification criteria and developed an automatic nerve detection system based on a CNN. We proposed a novel augmented classification structure for a U-Net to reduce the number of false positives in an object detection task. This new CNN based approach, APR-B, resulted in having a much higher precision score (78%) with respect to the colour filter (9%) while also being more consistent than the manual counting. The proposed approach also resulted in an increase of the nerve detection capability of approximately 5 times with respect to manual counting by the experts, while maintaining an 89% sensitivity.

In addition, we presented results that show that some additional considerations with respect to the training data, particularly class representation, can be used to further improve the performance of a CNN.

CHAPTER

7

Conclusion and Further Research

This thesis investigated on the development of effective and efficient CNN based approaches for image segmentation tasks in terms of time and resources. The research focussed on the development of a CNN that is more efficient, as well as an implementation that is more effective. This chapter summarises the main contributions and results of all the studies in this thesis, and provides some directions that can be pursued in further research.

7.1 Conclusion

A CNN is a type of a deep learning algorithm that has been remarkably successful in performing automatic segmentation in various clinical applications. Automatic segmentation is an important field of research in clinical applications as it minimises errors that are due to bias, fatigue and expert variability. It enables the analysis of a larger amount of data with higher speed, accuracy and consistency in various clinical applications. However, the application of CNNs has become increasingly expensive in terms of both time and memory usage in recent years.

In this thesis, we investigated several CNN architectures, in terms of their structure, number of trainable parameters and components, to develop a number of CNNs that are capable in achieving state-of the-art performances efficiently on different clinical segmentation tasks. We also demonstrate the implementation of some considerations that can improve the effectiveness in the development of a CNN based approach for a clinical application.

In Chapter 4, we investigated the state-of-the-art CNNs for image segmentation in terms of structure and number of trainable parameters. We proposed a CNN with a novel adjacent upsampling method that uses smaller number of trainable parameters while still being able to perform comparably well with other networks that use a significantly larger number of trainable parameters. We found that a CNN can be structured according to a given task for a more efficient implementation in terms of time and memory usage.

In Chapter 5, we investigated the main components of a U-Net with respect to the network performance in a medical image segmentation task. We examined the contribution of each component on the overall performance in terms of Dice's score. As a result, we developed an optimised U-Net architecture that uses a significantly lower number of trainable parameters than the original architecture. The optimised U-Net uses average pooling for pooling, resize convolution for upsampling, skip connection with concatenation for combination of feature maps in each convolution block, dropout and batch normalisation for model regularisation and ReLU for activation layers throughout the network. We showed that the optimised U-Net

outperforms other CNNs with similar structures and components on the public dataset, PROMISE12 [118].

In Chapter 6, we implemented the optimised U-Net developed in Chapter 5 on a complex digital pathology segmentation problem for an object detection task. We compare the performances of a CNN based approach and existing manual and automated quantification methods. We also developed a novel augmented classification structure to improve the performance of a U-Net for an object detection task. We discussed some data pre-processing considerations for the training of the network to address critical challenges that are often encountered in the development of a CNN based approach with clinical data.

In conclusion, we demonstrated that, with the right CNN architecture, a computationally efficient network with state-of-the-art performance can be developed. We also showed that the results of a CNN based approach depend not only on the network architecture, but also on data quality, data quantity and implementation. Finally, the results showed that with the right CNN architecture and implementation, an effective and efficient CNN based approach can be developed for various segmentation problems.

7.2 Suggestions for Further Research

Based on the results of this thesis, there are a number of further research directions that can be pursued for the development of a more effective and efficient CNN based approach.

For the optimisation of a CNN architecture in terms of the structure and number of trainable parameters, we found that it is important to identify the required model properties for a specific task. Thus, determining the required properties, e.g. the invariance properties, minimum number of layers/depth, minimum number of filters in each layer, given the type of a task could be a focus for future study. Automatic design algorithm, such as policy gradient algorithm [157] and genetic algorithm [158], could be explored in the future for a more efficient optimisation process.

For the optimisation of a CNN architecture in terms of the component, the

Attention Gates (AGs) [159] and Squeeze-and-Excitation (SE) blocks [160,161] could be the next components to investigate and compare as they have been shown to increase the performance of a U-Net in performing segmentation [159, 161]. In addition, optimisation of the components for 3D CNNs would be a logical and next step to further the work in this thesis, as it processes 3D input and extracts 3D features, which can be beneficial for performing volumetric medical image segmentation [162].

For the development of a higher performing CNN in terms of Dice's score, the adjacent network, developed in Chapter 4, with the integration of the best performing components in Chapter 5 could be the next architecture to investigate for an image segmentation task. While the novel augmented classification structure, developed in Chapter 6, could be further investigated when it is integrated in other segmentation network for an object detection task.

For the development of a CNN model for a specific organ segmentation application, a combination of different types of medical images could be investigated as an input to the neural network model to provide more initial features for the network to use to perform the segmentation, as a combination of Multiparametric MRI (such as T1w, T2w, ADC and PDw) has been shown to be significantly beneficial in prostate segmentation [114,163] and prostate cancer detection [164].

For the data, we found that inconsistency in data labelling due to inter-and intra-expert variance can make the learning as well as the development of a deep learning model more difficult and complicated. However, as the inter-and intra-expert variance in medical image labelling cannot be avoided, a larger quantity of data could be used to minimise the effect of the inter-and intra-expert variances in the data. Thus, in future study, an automatic segmentation model could be used to provide basic guidance for the expert to produce a larger labelled dataset so that a more robust model can be developed.

Finally, other performance metrics should be the next to consider for the optimisation of the CNN architecture in terms of structure, trainable parameters and components in future study.

Bibliography

[1] A. Kunjir and B. Shaikh. A survey on machine learning algorithms for building smart systems. *International Journal of Innovative Research in Computer and Communication Engineering*, 05:1052–1058, Jan 2017.

[2] Z. Yuan, C. Xu, J. Sang, S. Yan, and M.S. Hossain. Learning feature hierarchies: A layer-wise tag-embedded approach. *IEEE Transaction on Multimedia*, 17(6):816–827, June 2015.

[3] M. Fatima and M. Pasha. Survey of machine learning algorithms for disease diagnostic. *Journal of Intelligent Learning Systems and Applications*, 09:1–16, Jan 2017.

[4] A. El-Baz, X. Jiang, and J.S. Suri. *Biomedical Image Segmentation: Advances and Trends*. CRC Press, 2016.

[5] D.T. Lin, C.C. Lei, and S.W. Hung. Computer-aided kidney segmentation on abdominal CT images. *IEEE Transactions on Information Technology in Biomedicine*, 10(1):59–65, 2006.

[6] X. Gong, C. Ma, P. Yang, Y. Chen, C. Du, C. Fu, and J. P. Lu. Computer-aided pancreas segmentation based on 3D GRE Dixon MRI: a feasibility study. *Acta Radiol Open*, 8(3):2058460119834690, 2019.

[7] C. Qin, D. Yao, Y. Shi, and Z. Song. Computer-aided detection in chest radiography based on artificial intelligence: a survey. *BioMedical Engineering OnLine*, 17(1):113, 2018.

[8] T. P. Shiji, S. Remya, and Vinu Thomas. Computer aided segmentation of breast ultrasound images using scale invariant feature transform (SIFT) and bag of features. *Procedia Computer Science*, 115:518–525, 2017.

[9] M.P.A. Starmans, S.R. van der Voort, J.M.C. Tovar, J.F. Veenland, S. Klein, and W.J. Niessen. Chapter 18 - radiomics: Data mining using quantitative medical image features. In S.K. Zhou, D. Rueckert, and G. Fichtinger, editors, *Handbook of Medical Image Computing and Computer Assisted Intervention*, pages 429–456. Academic Press, 2020.

[10] I.R.I. Haque and J. Neubert. Deep learning approaches to biomedical image segmentation. *Informatics in Medicine Unlocked*, 18:100297, 2020.

[11] R. Merjulah and J. Chandra. Chapter 10 - classification of myocardial ischemia in delayed contrast enhancement using machine learning. In D.J. Hemanth, D. Gupta, and V. Emilia Balas, editors, *Intelligent Data Analysis for Biomedical Applications*, pages 209–235. Academic Press, 2019.

[12] B. Sahiner, A. Pezeshk, L.M. Hadjiiski, X. Wang, K. Drukker, K.H. Cha, R.M. Summers, and M.L. Giger. Deep learning in medical imaging and radiation therapy. *Medical Physics*, 46(1):e1–e36, 2019.

[13] A.S. Lundervold and A. Lundervold. An overview of deep learning in medical imaging focusing on MRI. *Zeitschrift für Medizinische Physik*, 29(2):102–127, 2019.

[14] Y. Bengio. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–127, January 2009.

[15] C. Farabet, C. Couprie, L. Najman, and Y. LeCun. Learning hierarchical features for scene labeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1915–1929, Aug 2013.

[16] S. Gupta, R. Girshick, P. Arbeláez, and J. Malik. Learning rich features from rgb-d images for object detection and segmentation. In D. Fleet, T. Pajdla,

B. Schiele, and T. Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 345–360, Cham, 2014. Springer International Publishing.

[17] E. Shelhamer, J. Long, and T. Darrell. Fully convolutional networks for semantic segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(4):640–651, 2017.

[18] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In N. Navab, J. Hornegger, W.M. Wells, and A.F. Frangi, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 234–241, Cham, 2015. Springer International Publishing.

[19] A. Garcia-Garcia, S. Orts-Escolano, S. Oprea, V. Villena-Martinez, P. Martinez-Gonzalez, and J. Garcia-Rodriguez. A survey on deep learning techniques for image and video semantic segmentation. *Applied Soft Computing*, 70:41–65, 2018.

[20] Mohammad Hesam Hesamian, Wenjing Jia, Xiangjian He, and Paul Kennedy. Deep learning techniques for medical image segmentation: Achievements and challenges. *Journal of Digital Imaging*, 32(4):582–596, 2019.

[21] S. Minaee, Y.Y. Boykov, F. Porikli, A.J Plaza, N. Kehtarnavaz, and D. Terzopoulos. Image segmentation using deep learning: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Early Access, Feb 2021, doi: 10.1109/TPAMI.2021.3059968.

[22] W. Chen, Y. Zhang, J. He, Y. Qiao, Y. Chen, H. Shi, and X. Tang. W-net: Bridged U-net for 2D medical image segmentation. *Computing Research Repository*, abs/1807.04459, 2018.

[23] D. Lachinov, E. Vasiliev, and V. Turlapov. Glioma segmentation with cascaded unet. In A. Crimi, S. Bakas, H. Kuijf, F. Keyvan, M. Reyes, and T. van Walsum, editors, *Brainlesion: Glioma, Multiple Sclerosis, Stroke and*

Traumatic Brain Injuries, pages 189–198. Springer International Publishing, 2019.

[24] L. Wu, Y. Xin, S. Li, T. Wang, P. Heng, and D. Ni. Cascaded fully convolutional networks for automatic prenatal ultrasound image segmentation. In *2017 IEEE 14th International Symposium on Biomedical Imaging (ISBI 2017)*, pages 663–666, 2017.

[25] Y. Liu, N. Qi, Q. Zhu, and W. Li. Cr-u-net: Cascaded u-net with residual mapping for liver segmentation in ct images*. In *2019 IEEE Visual Communications and Image Processing (VCIP)*, pages 1–4, 2019.

[26] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In Y. Bengio and Y. LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[27] R. Girshick. Fast R-CNN. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV '15, page 1440–1448, USA, 2015. IEEE Computer Society.

[28] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1137–1149, 2017.

[29] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, June 2015.

[30] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269, 2017.

[31] X. Li, H. Chen, X. Qi, Q. Dou, C. W. Fu, and P. A. Heng. H-denseunet: Hybrid densely connected UNet for liver and tumor segmentation from CT volumes. *IEEE Transactions on Medical Imaging*, 37(12):2663–2674, 2018.

[32] M. Baldeon-Calisto and S.K. Lai-Yuen. AdaResU-Net: Multiobjective adaptive convolutional neural network for medical image segmentation. *Neurocomputing*, 2019.

[33] S. Elguindi, M.J. Zelefsky, J. Jiang, H. Veeraraghavan, J.O. Deasy, M.A. Hunt, and N. Tyagi. Deep learning-based auto-segmentation of targets and organs-at-risk for magnetic resonance imaging only planning of prostate radiotherapy. *Physics and Imaging in Radiation Oncology*, 12:80–86, 2019.

[34] S.S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, 1999.

[35] J.D. Kelleher, B.M. Namee, and A. D'Arcy. *Fundamentals of Machine Learning for Predictive Data Analytics: Algorithms, Worked Examples, and Case Studies*. The MIT Press, 2015.

[36] Y. Liu. *Python Machine Learning By Example*. Packt Publishing, 2017.

[37] C.M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Inc., USA, 1995.

[38] K. Salman, R. Hossein, S.S.A. Ali, B. Mohammed, M. Gerard, and D. Sven. *A Guide to Convolutional Neural Networks for Computer Vision*. Morgan & Claypool, 2018.

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

[40] S. Pattanayak. *Pro Deep Learning with TensorFlow: A Mathematical Approach to Advanced Artificial Intelligence in Python*. Apress, USA, 1st edition, 2017.

[41] D. Graupe. *Deep Learning Neural Networks*. World Scientific, 2016.

[42] R. Yamashita, M. Nishio, R.K.G. Do, and K. Togashi. Convolutional neural networks: an overview and application in radiology. *Insights into Imaging*, 9(4):611–629, Aug 2018.

[43] H. Zhu, F. Shi, L. Wang, S. Hung, M. Chen, S. Wang, W. Lin, and D. Shen. Dilated dense u-net for infant hippocampus subfield segmentation. *Frontiers in neuroinformatics*, 13:30–30, 2019.

[44] N. Mboga, C. Persello, J.R. Bergado, and A. Stein. Detection of informal settlements from VHR images using convolutional neural networks. *Remote Sensing*, 9:1106, Oct 2017.

[45] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. The MIT Press, 2016.

[46] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. *Neural Information Processing Systems*, 25, 2012.

[47] F. Milletari, N. Navab, and S.A. Ahmadi. V-net: Fully convolutional neural networks for volumetric medical image segmentation. In *2016 Fourth International Conference on 3D Vision (3DV)*, pages 565–571, Oct 2016.

[48] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[49] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, June 2016.

[50] J. May. *Multivariate Analysis*. EDTECH, 2018.

[51] A.P. Aitken, C. Ledig, L. Theis, J. Caballero, Z. Wang, and W. Shi. Checkerboard artifact free sub-pixel convolution: A note on sub-pixel

convolution, resize convolution and convolution resize. *Computing Research Repository*, abs/1707.02937, 2017.

[52] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, June 2016.

[53] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[54] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, 07–09 Jul 2015. PMLR.

[55] Y. Sun, L. Zhu, G. Wang, and F. Zhao. Multi-input convolutional neural network for flower grading. *Journal of Electrical and Computer Engineering*, 2017:9240407, 2017.

[56] M. M. Lau and K. Hann Lim. Review of adaptive activation function in deep neural network. In *2018 IEEE-EMBS Conference on Biomedical Engineering and Sciences (IECBES)*, pages 686–690, 2018.

[57] G.S.S. Gomes, T.B. Ludermir, and L.M.M.R. Lima. Comparison of new activation functions in neural network for forecasting financial time series. *Neural Computing and Applications*, 20(3):417–439, 2011.

[58] L. Lu, Y. Shin, Y. Su, and G.E. Karniadakis. Dying ReLU and initialization: Theory and numerical examples. *Communications in Computational Physics*, 28(5):1671–1706, 2020.

[59] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the 2015*

IEEE International Conference on Computer Vision (ICCV), ICCV '15, page 1026–1034, USA, 2015. IEEE Computer Society.

[60] C.M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[61] W. Rawat and Z. Wang. Deep convolutional neural networks for image classification: A comprehensive review. *Neural Computation*, 29:1–98, 2017.

[62] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587, 2014.

[63] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[64] J. Brownlee. *Deep Learning for Computer Vision: Image Classification, Object Detection, and Face Recognition in Python*. Machine Learning Mastery, 2019.

[65] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. JMLR Workshop and Conference Proceedings.

[66] S. Samarasinghe. *Neural Networks for Applied Sciences and Engineering*. Auerbach Publications, USA, 2006.

[67] D.P. Kingma and J. Ba. Adam: A method for stochastic optimization. In Y. Bengio and Y. LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[68] J. Han, J. Pei, and M. Kamber. *Data Mining, Southeast Asia Edition*. The Morgan Kaufmann Series in Data Management Systems. Elsevier Science, 2006.

[69] R. Maini and H. Aggarwal. A comprehensive review of image enhancement techniques. *Journal of Computing*, 2(3), 2010.

[70] D. Vernon. *Machine Vision: Automated Visual Inspection and Robot Vision*. Prentice Hall, 1991.

[71] R.C. Gonzalez and R.E. Woods. *Digital Image Processing*. Prentice Hall, 2002.

[72] S. Qureshi. *Embedded image processing on the TMS320C6000™ DSP: Examples in code composer studio™ and MATLAB*. Springer US, 2005.

[73] W. Burger and M.J. Burge. *Principles of Digital Image Processing: Fundamental Techniques*. Springer London, 2009.

[74] J.M. Johnson and T.M. Khoshgoftaar. Survey on deep learning with class imbalance. *Journal of Big Data*, 6(1):27, 2019.

[75] S. Wang, Y. Zhu, L. Yu, H. Chen, H. Lin, X. Wan, X. Fan, and P. A. Heng. RMDL: Recalibrated multi-instance deep learning for whole slide gastric image classification. *Medical Image Analysis*, 58:101549, 2019.

[76] S. Wang, W. Liu, J. Wu, L. Cao, Q. Meng, and P. J. Kennedy. Training deep neural networks on imbalanced data sets. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 4368–4374, 2016.

[77] A. Fernández, S. García, M. Galar, R. Prati, B. Krawczyk, and F. Herrera. *Learning from Imbalanced Data Sets*. Springer International Publishing, 2018.

[78] C. Solomon and T. Breckon. *Fundamentals of Digital Image Processing: A Practical Approach with Examples in MATLAB*. Wiley Publishing, 1st edition, 2011.

[79] L.G. Shapiro. Connected component labeling and adjacency graph construction. In T.Y. Kong and A. Rosenfeld, editors, *Topological Algorithms for Digital Image Processing*, volume 19 of *Machine Intelligence and Pattern Recognition*, pages 1–30. North-Holland, 1996.

[80] T. Acharya and A.K. Ray. *Image processing: principles and applications*. John Wiley, Hoboken, N.J., 2005.

[81] A.C. Faul. *A Concise Introduction to Machine Learning*. CRC Press, 2019.

[82] C.L. Srinidhi, O. Ciga, and A.L. Martel. Deep neural network models for computational histopathology: A survey. *Medical Image Analysis*, 67:101813, 2021.

[83] S. Sabour, N. Frosst, and G.E. Hinton. Dynamic routing between capsules. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 3859–3869, Red Hook, NY, USA, 2017. Curran Associates Inc.

[84] P.F. Christ, M.E.A. Elshaer, F. Ettlinger, S. Tatavarty, M. Bickel, P. Bilic, M. Rempfler, M. Armbruster, F. Hofmann, M. D'Anastasi, W.H. Sommer, S.A. Ahmadi, and B.H. Menze. Automatic liver and lesion segmentation in CT using cascaded fully convolutional neural networks and 3D conditional random fields. In S. Ourselin, L. Joskowicz, M.R. Sabuncu, G. Unal, and W. Wells, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2016*, pages 415–423, Cham, 2016. Springer International Publishing.

[85] Y. Zhang, Z. He, C. Zhong, Y. Zhang, and Z. Shi. Fully convolutional neural network with post-processing methods for automatic liver segmentation from CT. In *2017 Chinese Automation Congress (CAC)*, pages 3864–3869.

[86] K.C. Kaluva, M. Khened, A. Kori, and G. Krishnamurthi. 2D-densely connected convolution neural networks for automatic liver and tumor segmentation. *Computing Research Repository*, abs/1802.02182, 2018.

[87] E. Gibson, F. Giganti, Y. Hu, E. Bonmati, S. Bandula, K. Gurusamy, B. Davidson, S. P. Pereira, M. J. Clarkson, and D. C. Barratt. Automatic multi-organ segmentation on abdominal CT with dense v-networks. *IEEE Transactions on Medical Imaging*, 37(8):1822–1834, 2018.

[88] W. Shi, J. Caballero, F. Huszár, J. Totz, A. P. Aitken, R. Bishop, D. Rueckert, and Z. Wang. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1874–1883, June 2016.

[89] Patrick Christ. LiTS - liver tumor segmentation challenge, 2017. Available on https://competitions.codalab.org/competitions/17094.

[90] J.T. Springenberg, A. Dosovitskiy, T. Brox, and M.A. Riedmiller. Striving for simplicity: The all convolutional net. *Computing Research Repository*, abs/1412.6806, 2014.

[91] C. Peng, X. Zhang, G. Yu, G. Luo, and J. Sun. Large Kernel Matters – Improve Semantic Segmentation by Global Convolutional Network. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1743–1751, July 2017.

[92] G. Chlebus, A. Schenk, J. H. Moltz, B. van Ginneken, H. K. Hahn, and H. Meine. Automatic liver tumor segmentation in CT with fully convolutional neural networks and object-based postprocessing. *Scientific Reports*, 8(1):15497, 2018.

[93] S. Ranjan and S. Senthamilarasu. *Applied Deep Learning and Computer Vision for Self-Driving Cars: Build autonomous vehicles using deep neural networks and behavior-cloning techniques*. Packt Publishing, 2020.

[94] H. Le and A. Borji. What are the receptive, effective receptive, and projective fields of neurons in convolutional neural networks? *Computing Research Repository*, abs/1705.07049, 2017.

[95] ZhiXuHao. unet, 2017. Available on https://github.com/zhixuhao/unet.

[96] Q. Huang, H. Ding, X. Wang, and G. Wang. Fully automatic liver segmentation in CT images using modified graph cuts and feature detection. *Computers in Biology and Medicine*, 95:198–208, 2018.

[97] T. Peters and K. Cleary. *Image-Guided Interventions: Technology and Applications*. SpringerLink Engineering. Springer US, 2008.

[98] V. Yeghiazaryan and I. Voiculescu. An overview of current evaluation methods used in medical image segmentation. Technical Report RR-15-08, Department of Computer Science, Oxford, UK, 2015.

[99] N. Aldoj, F. Biavati, F. Michallek, S. Stober, and M. Dewey. Automatic prostate and prostate zones segmentation of magnetic resonance images using densenet-like u-net. *Scientific Reports*, 10(1):14315, 2020.

[100] N. Mishra, S. Petrovic, and S. Sundar. A knowledge-light nonlinear case-based reasoning approach to radiotherapy planning. In *2009 21st IEEE International Conference on Tools with Artificial Intelligence*, pages 776–783, Nov 2009.

[101] J.A. Dowling, J. Fripp, S. Chandra, J.P.W. Pluim, J. Lambert, J. Parker, J. Denham, P.B. Greer, and O. Salvado. Fast automatic multi-atlas segmentation of the prostate from 3D MR images. In A. Madabhushi, J. Dowling, H. Huisman, and D. Barratt, editors, *Prostate Cancer Imaging. Image Analysis and Image-Guided Interventions*, pages 10–21, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[102] D. Mahapatra. Semi-supervised learning and graph cuts for consensus based medical image segmentation. *Pattern Recognition*, 63:700 – 709, 2017.

[103] D. R. White, A. S. Houston, W. F. Sampson, and G. P. Wilkins. Intra- and interoperator variations in region-of-interest drawing and their effect on the measurement of glomerular filtration rates. *Clinical Nuclear Medicine*, 24(3):177–81, 1999.

[104] S. Chandra, J. Dowling, K. Shen, J. Pluim, P. Greer, O. Salvado, and J. Fripp. Automatic segmentation of the prostate in 3D magnetic resonance images using case specific deformable models. In *2011 International Conference on Digital Image Computing: Techniques and Applications*, pages 7–12, Dec 2011.

[105] M. Shahedi, L. Ma, M. Halicek, R. Guo, G. Zhang, D.M. Schuster, P. Nieh, V. Master, and B. Fei. A semiautomatic algorithm for three-dimensional segmentation of the prostate on CT images using shape and local texture characteristics. In B. Fei and R.J. Webster III, editors, *Medical Imaging 2018: Image-Guided Procedures, Robotic Interventions, and Modeling*, volume 10576, pages 280 – 287. International Society for Optics and Photonics, SPIE, 2018.

[106] S. S. Chandra, J. A. Dowling, K. Shen, P. Raniga, J. P. W. Pluim, P. B. Greer, O. Salvado, and J. Fripp. Patient specific prostate segmentation in 3-D magnetic resonance images. *IEEE Transactions on Medical Imaging*, 31(10):1955–1964, Oct 2012.

[107] S. Martin, J. Troccaz, and V. Daanen. Automated segmentation of the prostate in 3D MR images using a probabilistic atlas and a spatially constrained deformable model. *Medical physics*, 37(4):1579–1590, 2010.

[108] W. Wong, L.H. Leung, and D. Kwong. Evaluation and optimization of the parameters used in multiple-atlas-based segmentation of prostate cancers in radiation therapy. *The British Journal of Radiology*, 89(1057):20140732, 2016.

[109] Q. Zhu, B. Du, B. Turkbey, P. L. Choyke, and P. Yan. Deeply-supervised CNN for prostate segmentation. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 178–184, May 2017.

[110] Q. Xiangxiang, Z. Yu, and Z. Bingbing. Automated segmentation based on residual U-Net model for MR prostate images. In *2018 11th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*, pages 1–6, Oct 2018.

[111] Y. Zhu, R. Wei, G. Gao, L. Ding, X. Zhang, X. Wang, and J. Zhang. Fully automatic segmentation on prostate MR images based on cascaded fully convolution network. *Journal of Magnetic Resonance Imaging*, 49(4):1149–1156, Oct 2018.

[112] T. Hassanzadeh, L. G. C. Hamey, and K. Ho-Shon. Convolutional neural networks for prostate magnetic resonance image segmentation. *IEEE Access*, 7:36748–36760, 2019.

[113] Y. Yuan, W. Qin, X. Guo, M. Buyyounouski, S. Hancock, B. Han, and L. Xing. Prostate segmentation with encoder-decoder densely connected convolutional network (ed-densenet). In *2019 IEEE 16th International Symposium on Biomedical Imaging (ISBI 2019)*, pages 434–437, Apr 2019.

[114] F. Zabihollahy, N. Schieda, S.K. Jeyaraj, and E. Ukwatta. Automated segmentation of prostate zonal anatomy on T2-weighted (T2W) and apparent diffusion coefficient (ADC) map MR images using U-Nets. *Medical Physics*, 46(7):3078–3090, 2019.

[115] K. Yan, C. Li, X. Wang, A. Li, Y. Yuan, D. Feng, M. Khadra, and J. Kim. Automatic prostate segmentation on MR images with deep network and graph model. In *2016 38th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 635–638, Aug 2016.

[116] B. He, D. Xiao, Q. Hu, and F. Jia. Automatic magnetic resonance image prostate segmentation based on adaptive feature learning probability boosting tree initialization and CNN-ASM refinement. *IEEE Access*, 6:2005–2015, 2018.

[117] J.A. Dowling, J. Sun, P. Pichler, D. Rivest-Hénault, S. Ghose, H. Richardson, C. Wratten, J. Martin, J. Arm, L. Best, S.S. Chandra, J. Fripp, F.W. Menk, and P.B. Greer. Automatic substitute computed tomography generation and contouring for magnetic resonance imaging (MRI)-alone external beam radiation therapy from standard MRI sequences. *International Journal of Radiation Oncology*Biology*Physics*, 93(5):1144–1153, Dec 2015.

[118] G. Litjens, R. Toth, W. van de Ven, C. Hoeks, S. Kerkstra, B. van Ginneken, G. Vincent, G. Guillard, N. Birbeck, J. Zhang, R. Strand, F. Malmberg, Y. Ou, C. Davatzikos, M. Kirschner, F. Jung, J. Yuan, W. Qiu, Q. Gao, P. Edwards, B. Maan, F. van der Heijden, S. Ghose, J. Mitra, J. Dowling, D. Barratt, H. Huisman, and A. Madabhushi. Evaluation of prostate segmentation

algorithms for MRI: The PROMISE12 challenge. *Medical Image Analysis*, 18(2):359 – 373, 2014.

[119] S.S. Chandra, J.A. Dowling, P.B. Greer, J. Martin, C. Wratten, P. Pichler, J. Fripp, and S. Crozier. Fast automated segmentation of multiple objects via spatially weighted shape learning. *Physics in Medicine and Biology*, 61(22):8070–8084, Oct 2016.

[120] J.E. Iglesias and M.R. Sabuncu. Multi-atlas segmentation of biomedical images: A survey. *Medical Image Analysis*, 24(1):205 – 219, 2015.

[121] V. S. Sheng, J. Zhang, B. Gu, and X. Wu. Majority voting and pairing with multiple noisy labeling. *IEEE Transactions on Knowledge and Data Engineering*, 31(7):1355–1368, 2019.

[122] François Chollet et al. Keras, 2015. Available on https://keras.io.

[123] Itseez. Open source computer vision library. https://github.com/itseez/opencv, 2015.

[124] B. Boilly, S. Faulkner, P. Jobling, and H. Hondermarck. Nerve dependence: From regeneration to cancer. *Cancer Cell*, 31(3):342–354, 2017.

[125] C. Magnon, S. J. Hall, J. Lin, X. Xue, L. Gerber, S. J. Freedland, and P. S. Frenette. Autonomic nerve development contributes to prostate cancer progression. *Science*, 341(6142):1236361, 2013.

[126] J. Pundavela, S. Roselli, S. Faulkner, J. Attia, R. J. Scott, R. F. Thorne, J. F. Forbes, R. A. Bradshaw, M. M. Walker, P. Jobling, and H. Hondermarck. Nerve fibers infiltrate the tumor microenvironment and are associated with nerve growth factor production and lymph node invasion in breast cancer. *Molecular Oncology*, 9(8):1626–35, 2015.

[127] T. Iwasaki, N. Hiraoka, Y. Ino, K. Nakajima, Y. Kishi, S. Nara, M. Esaki, K. Shimada, and H. Katai. Reduction of intrapancreatic neural density in cancer tissue predicts poorer outcome in pancreatic ductal carcinoma. *Cancer Science*, 110(4):1491–1502, 2019.

[128] C.W. Rowe, T. Dill, N. Griffin, P. Jobling, S. Faulkner, J.W. Paul, S. King, R. Smith, and H. Hondermarck. Innervation of papillary thyroid cancer and its association with extra-thyroidal invasion. *Scientific Reports*, 10(1):1539, 2020.

[129] J. Bründl, S. Schneider, F. Weber, F. Zeman, W. F. Wieland, and R. Ganzer. Computerized quantification and planimetry of prostatic capsular nerves in relation to adjacent prostate cancer foci. *European Urology*, 65(4):802–8, 2014.

[130] R. Ganzer, A. Blana, A. Gaumann, J. U. Stolzenburg, R. Rabenalt, T. Bach, W. F. Wieland, and S. Denzinger. Topographical anatomy of periprostatic and capsular nerves: quantification and computerised planimetry. *European Urology*, 54(2):353–60, 2008.

[131] M. Bizrah, S.C. Dakin, L. Guo, F. Rahman, M. Parnell, E. Normando, S. Nizari, B. Davis, A. Younis, and M.F. Cordeiro. A semi-automated technique for labeling and counting of apoptosing retinal cells. *BMC Bioinformatics*, 15(1):169, 2014.

[132] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587, 2014.

[133] H. Jia, Y. Song, H. Huang, W. Cai, and Y. Xia. Hd-net: Hybrid discriminative network for prostate segmentation in mr images. In *Medical Image Computing and Computer Assisted Intervention – MICCAI 2019*, page 110–118, Berlin, Heidelberg. Springer-Verlag.

[134] D.C. Cireşan, A. Giusti, L.M. Gambardella, and J. Schmidhuber. Mitosis detection in breast cancer histology images with deep neural networks. In K. Mori, I. Sakuma, Y. Sato, C. Barillot, and N. Navab, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2013*, pages 411–418, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[135] H. Wang, A. Cruz-Roa, A. Basavanhally, H. Gilmore, N. Shih, M. Feldman, J. Tomaszewski, F. Gonzalez, and A. Madabhushi. Mitosis detection in breast cancer pathology images by combining handcrafted and convolutional neural network features. *Journal of Medical Imaging (Bellingham)*, 1(3):034003, 2014.

[136] Y. Song, L. Zhang, S. Chen, D. Ni, B. Lei, and T. Wang. Accurate segmentation of cervical cytoplasm and nuclei based on multiscale convolutional network and graph partitioning. *IEEE Transactions on Biomedical Engineering*, 62(10):2421–33, 2015.

[137] D.J. Bora, A.K. Gupta, and F.A. Khan. Comparing the performance of L*A*B* and HSV color spaces with respect to color image segmentation. *Computing Research Repository*, abs/1506.01472, 2015.

[138] P.M. Waghmare and S.S. Shetkar. Guided filter for grayscale, RGB and HSV color space. *International Journal of Engineering and Computer Science*, 5(12), Nov 2016.

[139] J. Schindelin, C. T. Rueden, M. C. Hiner, and K. W. Eliceiri. The ImageJ ecosystem: An open platform for biomedical image analysis. *Molecular Reproduction and Development*, 82(7-8):518–29, 2015.

[140] R. Masteling, L. Voorhoeve, J. Ijsselmuiden, F. Dini-Andreote, W. de Boer, and J.M. Raaijmakers. Discount: computer vision for automated quantification of striga seed germination. *Plant Methods*, 16(1):60, 2020.

[141] A. Hamidinekoo, G.A. Garzón-Martínez, M. Ghahremani, F.M. K. Corke, R. Zwiggelaar, J.H. Doonan, and C. Lu. DeepPod: a convolutional neural network based quantification of fruit number in Arabidopsis. *GigaScience*, 9(3), 2020.

[142] H. Harzallah, F. Jurie, and C. Schmid. Combining efficient object localization and image classification. In *2009 IEEE 12th International Conference on Computer Vision*, pages 237–244, 2009.

[143] P.F. Felzenszwalb, R.B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(9):1627–1645, 2010.

[144] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. In *2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings*, 2014.

[145] J.R. Uijlings, K.E. Sande, T. Gevers, and A.W. Smeulders. Selective search for object recognition. *International Journal of Computer Vision*, 104(2):154–171, September 2013.

[146] K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(9):1904–1916, 2015.

[147] A. Cruz-Roa, A. Basavanhally, F. González, H. Gilmore, M. Feldman, S. Ganesan, N. Shih, J. Tomaszewski, and A. Madabhushi. Automatic detection of invasive ductal carcinoma in whole slide images with convolutional neural networks. In Metin N. Gurcan and Anant Madabhushi, editors, *Medical Imaging 2014: Digital Pathology*, volume 9041, pages 1 – 15. International Society for Optics and Photonics, SPIE, 2014.

[148] A. Cruz-Roa, H. Gilmore, A. Basavanhally, M. Feldman, S. Ganesan, N. Shih, J. Tomaszewski, A. Madabhushi, and F. González. High-throughput adaptive sampling for whole-slide histopathology image analysis (HASHI) via convolutional neural networks: Application to invasive breast cancer detection. *PLOS ONE*, 13(5):1–23, May 2018.

[149] H. Chang, J. Han, A. Borowsky, L. Loss, J.W. Gray, P.T. Spellman, and B. Parvin. Invariant delineation of nuclear architecture in glioblastoma multiforme for clinical and molecular association. *IEEE Transactions on Medical Imaging*, 32(4):670–82, 2013.

[150] H. Irshad. Automated mitosis detection in histopathology using morphological and multi-channel statistics features. *Journal of pathology informatics*, 4(1):10, 2013.

[151] Z. Swiderska-Chadaj, H. Pinckaers, M. van Rijthoven, M. Balkenhol, M. Melnikova, O. Geessink, Q. Manson, M. Sherman, A. Polonia, J. Parry, M. Abubakar, G. Litjens, J. van der Laak, and F. Ciompi. Learning to detect lymphocytes in immunohistochemistry with deep learning. *Medical Image Analysis*, 58:101547, 2019.

[152] T. de Bel, M. Hermsen, B. Smeets, L. Hilbrands, J. van der Laak, and G. Litjens. Automatic segmentation of histopathological slides of renal tissue using deep learning. In J.E. Tomaszewski and M.N. Gurcan, editors, *Medical Imaging 2018: Digital Pathology*, volume 10581, pages 285 – 290. International Society for Optics and Photonics, SPIE, 2018.

[153] Z.H. Zhou. A brief introduction to weakly supervised learning. *National Science Review*, 5(1):44–53, Aug 2017.

[154] C.W. Rowe, T. Dill, N. Griffin, P. Jobling, S. Faulkner, J.W. Paul, S. King, R. Smith, and H. Hondermarck. 4 micron sections of human thyroid cancer stained with PGP9.5 and counterstained with haematoxylin, January 2021.

[155] P. Bankhead, M.B. Loughrey, J.A. Fernández, Y. Dombrowski, D.G. McArt, P.D. Dunne, S. McQuaid, R.T. Gray, L.J. Murray, H.G. Coleman, J.A. James, M. Salto-Tellez, and P.W. Hamilton. Qupath: Open source software for digital pathology image analysis. *Scientific Reports*, 7(1):16878, 2017.

[156] P.V. Krishna, S. Gurumoorthy, and M.S. Obaidat. *Social Network Forensics, Cyber Security, and Machine Learning.* Springer Publishing Company, Incorporated, 1st edition, 2018.

[157] A. Mortazi and U. Bagci. Automatically designing cnn architectures for medical image segmentation. In Y. Shi, HI. Suk, and M. Liu, editors,

Machine Learning in Medical Imaging, pages 98–106, Cham, 2018. Springer International Publishing.

[158] Y. Sun, B. Xue, M. Zhang, G.G. Yen, and J. Lv. Automatically designing cnn architectures using the genetic algorithm for image classification. *IEEE Transactions on Cybernetics*, 50(9):3840–3854, 2020.

[159] J. Schlemper, O. Oktay, M. Schaap, M. Heinrich, B. Kainz, B. Glocker, and D. Rueckert. Attention gated networks: Learning to leverage salient regions in medical images. *Medical Image Analysis*, 53:197 – 207, 2019.

[160] J. Hu, L. Shen, S. Albanie, G. Sun, and E. Wu. Squeeze-and-excitation networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(8):2011–2023, 2020.

[161] L. Rundo, C. Han, Y. Nagano, J. Zhang, R. Hataya, C. Militello, A. Tangherloni, M. Nobile, C. Ferretti, D. Besozzi, M.C. Gilardi, S. Vitabile, G. Mauri, H. Nakayama, and P. Cazzaniga. USE-Net: incorporating squeeze-and-excitation blocks into U-net for prostate zonal segmentation of multi-institutional MRI datasets. *Neurocomputing*, 365:31–43, Nov 2019.

[162] M. Kolařík, R. Burget, V. Uher, K. Riha, and M. Dutta. Optimized high resolution 3D Dense-U-Net network for brain and spine segmentation. *Applied Science*, 9:404, Jan 2019.

[163] L. Rundo, C. Militello, G. Russo, A. Garufi, S. Vitabile, M.C. Gilardi, and G. Mauri. Automated prostate gland segmentation based on an unsupervised Fuzzy C-Means clustering technique using multispectral T1w and T2w MR imaging. *Information*, 8(2):49, 2017.

[164] P. Lapa, M. Castelli, I. Gonçalves, E. Sala, and L. Rundo. A hybrid end-to-end approach integrating conditional random fields into CNNs for prostate cancer detection on MRI. *Applied Science*, 10(1):338, Jan 2020.