Automatic Segmentation of Healthy Liver in Abdominal Computed Tomography Scans

David Scicluna

Supervisor: Prof. Johann A. Briffa

Co-supervisor: Mr. Eric Pace

September, 2022

Submitted in partial fulfilment of the requirements for the degree of Master of Science in Medical Physics at the University of Malta.

L-Università ta' Malta
Faculty of Health Sciences

Medical Physics Unit

Acknowledgements

I would like to start this dissertation by first acknowledging and thanking the people who spurred me on throughout this journey. My supervisor, Prof. Johann A. Briffa for his guidance, patience and support, and for always believing in me.

There were difficult moments where I felt like throwing in the towel. However, with the help of my friends, I found the courage to persist and change my outlook on what seemed to be impossible tasks. I would like to thank Keith George Ciantar who dedicated several hours of his time to help me diagnose software to hardware conflicts, Stephanie Chetcuti who gave me a helping hand during my initial interactions with Python and, Noel Farrugia who got me started on LaTeX and supported me in any LaTeX related queries.

Commas, colons and semicolons fell under the department of Lisa Marie Galea, whose patience and dedication never seemed to run out. Her support helped me to carry on, the light at the end of the tunnel.

I would also like to thank my parents for their continuous support, and without whom, I would not be where I am today.

Abstract

Segmentation is the process of delineating regions of interest and this process is applied to medical scans to help with diagnosis of diseases as well as treatment planning and monitoring. At the date of writing this work, segmentation is primarily carried out manually by medical professionals, which adds a substantial workload.

Convolutional Encoder-Decoders (CEDs) currently dominate the medical image automatic segmentation field and many have produced satisfactory results, given the limited availability of training data. This work explores literature of some of these implementations and goes into detail about a state-of-the-art model called v16pUNet1.1C, which is an architecture based on VGG16, UNet and the Cascade Framework. The Combined Healthy Abdominal Organ Segmentation (CHAOS) Challenge database and its Task 2 framework are used to replicate and verify the state-of-the-art implementation. A modification of the architecture of v16pUNet1.1C was carried out with the purpose of increasing the performance. Modifications were also performed on the learning rate, context connections and the cascade framework, however, none seemed to lead to an increase in mean score performance, although they did narrow the interquartile range, which is a success in its own merit. The modified model, called v16pUNet1.1D, managed to achieve a mean score of 85.92, just 0.06 points shy from first place in Task 2 of the CHAOS Challenge.

Keywords: CHAOS challenge, segmentation, Computed Tomography (CT), liver

Contents

List of Figures viii

List of Tables ix

List of Definitions x

List of Acronyms xi

1 Introduction 1
- 1.1 Introduction 1
- 1.2 Problem Statement 1
  - 1.2.1 Difficulties in Automatic Segmentation 2
  - 1.2.2 Accurate Liver Volume Applications 2
- 1.3 Aims and Objectives 2
- 1.4 Research Design and Ethical Considerations 3
- 1.5 Conclusion 3

2 Literature Review 4
- 2.1 Introduction 4
- 2.2 Details on Challenge 4
- 2.3 Existing Implementations 5
  - 2.3.1 cGAN 7
  - 2.3.2 Modified Attention UNet 8
  - 2.3.3 DualTail-Net 8
  - 2.3.4 Ternaus-Net 8
  - 2.3.5 Link-Net34 9
  - 2.3.6 ResNet-50 9
  - 2.3.7 SE-ResNet-50 9
  - 2.3.8 Pixel Shuffle 10

2.3.9 SC-SegNet 10
2.3.10 Honourable Mentions 11
2.4 Building Blocks 12
2.4.1 UNet 12
2.4.2 VGG 13
2.5 Conclusion 13

3 Background **14**
3.1 Introduction 14
3.2 The CHAOS Challenge 14
3.2.1 Database 14
3.2.2 Evaluation 16
3.3 Selection Process 17
3.4 Starting Point 18
3.5 Building Blocks in Detail 19
3.5.1 VGG16UNet Architecture 19
3.5.2 The UNet Architecture 20
3.5.3 Cascaded Framework 22
3.5.4 Pre-Training 23
3.6 Introduction to Convolutional Encoder-Decoders 24
3.6.1 Implementation, Model and Architecture 24
3.6.2 What are CNNs? 25
3.6.3 What are Convolutional Encoder-Decoders? 26
3.7 Conclusion 26

4 Implementation **27**
4.1 Introduction 27
4.2 Getting Started 27
4.2.1 Database and Metrics 28
4.3 Replication and Verification 30
4.3.1 Ambiguities 32
4.3.2 Implementation and Parameters 34
4.4 Experimentation 35
4.4.1 Performance Tuning 35
4.4.2 Architecture Discrepancies 35
4.4.3 Training 37
4.4.4 Normalization Adjustment 38

4.4.5 Cascaded Framework Experiments 38
4.5 Conclusion 42

5 Results 44
5.1 Introduction 44
5.2 Submission Process 44
5.3 Replication and Verification of Results 45
5.4 Experimental Results 46
5.4.1 Comparison of the Modified Model 46
5.4.2 Experiments on the Modified Model 47
5.5 Conclusion 48

6 Discussion 51
6.1 Introduction 51
6.2 Replication and Verification Results 51
6.3 Comparison of Implementations 52
6.3.1 Training Methods 52
6.3.2 Modified Model 53
6.4 Experiments on Modified Model 53
6.4.1 Standard Category 53
6.4.2 Between Connections Category 54
6.4.3 Within Connections Category 54
6.4.4 Learning Rate Category 55
6.4.5 Size Adjustment Category 55
6.5 Analysis 56
6.6 Conclusion 56

7 Conclusion 57
7.1 Introduction 57
7.2 Summary of Conclusions 57
7.3 Limitations 57
7.4 Recommendations for Future Research 58
7.5 Conclusion 59

References 60

A Appendices 63
A.1 Database Technical Information 63

A.2 Additional Results 65

List of Figures

2.1 Architecture Relationships 7
2.2 Ideal Pixel Shuffle Example 11

3.1 Pre-Trained Cascaded Framework Architecture Diagram 19
3.2 VGG16 Architecture Diagram 20
3.3 VGG16UNetC Architecture Diagram 21
3.4 UNet Architecture Diagram 22
3.5 Cascaded Framework Diagram 23

4.1 Maximum Values for All Slices 30
4.2 Example of Anomaly 31
4.3 Cross-Validation Diagram 33
4.4 VGG16UNetD Architecture Diagram 36
4.5 Between Model Connections 40
4.6 Shallow Model Connections 41
4.7 Deep Model Connections 42
4.8 Shallow VGG16UNetD Model 43

5.1 Box Plots of Replication Results 46
5.2 Box Plots of Comparison Results 47
5.3 Box Plots of v16pUNet1.1D Variations' Results 49

A.1 Box Plots of Replication Results - Metrics 66
A.2 Box Plots of Comparison Results - Metrics 67
A.3 Box Plots of v16pUNet1.1D Variations' Results - Metrics 68

List of Tables

2.1 Summary of Results 12

3.1 CHAOS Metrics' Threshold Values. 17

4.1 Implementation Parameters (Conze et al., 2021) 31
4.2 Implementation Parameters (this work) 34

5.1 Replication Results - Summary 45
5.2 Comparison Results - Summary 48
5.3 Experimental Results - Summary 50

A.1 Training and Testing Databases' DICOM Parameters 64
A.2 Full Replication Results 69
A.3 Full Comparison Results 69
A.4 Full Experimentatal Results 70

List of Definitions

Architecture An architecture is a representation of the individual layers, clearly defining each layer in detail and how each is connected to the other. No training details are required when using the term architecture.

Database A database is a collection of organized datasets.

Dataset A dataset, as denoted by Kavur et al. (2021), is a collection of slices from a single scan.

Implementation An implementation includes the training parameters, and the architecture or model. When specifying an implementation all the details required to be able to replicate the building and train the architecture or model should be included, such as type of optimizer, learning rate, loss function, number of epochs, number of augmentations, etcetera.

Model A model is defined as the architecture after it has been trained. It contains information on how the model is built and the weights which have been calculated after training, but training parameters are not necessarily included[1]. For a given input, a model will always output the same result as its weights and biases are fixed. An architecture that is partially trained is also being considered a model because if the training is stopped, its bias and weights are still fixed at that point in time and will still give a deterministic result.

Segmentation Segmentation is the process of labelling each pixel in an image, where the label identifies the object to which each pixel belongs.

Slice A slice is defined as an individual two-dimensional image captured by the medical machine, within the context of a whole scan.

[1]https://developers.google.com/machine-learning/glossary#m

List of Acronyms

AAO All-At-Once.

ASSD Average Symmetric Surface Distance.

BCE Binary Cross Entropy.

CED Convolutional Encoder-Decoder.

cGAN Conditional Generative Adversarial Network.

CHAOS Combined Healthy Abdominal Organ Segmentation.

CNN Convolutional Neural Network.

CT Computed Tomography.

DICE Sørensen–Dice Coefficient.

ELU Exponential Linear Unit.

FC Fully Connected.

FCN Fully Convolutional Network.

GPU Graphics Processing Unit.

HU Hounsfield Unit.

LR Learning Rate.

MDCM Multi-Scale Dilated Convolutions Module.

MF UNet Multi-scale Fusion UNet.

MRI Magnetic Resonance Imaging.

MSSD Maximum Symmetric Surface Distance.

OAT One-At-a-Time.

PReLU Parametric Rectified Linear Unit.

RAVD Relative Absolute Volume Difference.

ReLU Rectified Linear Unit.

SC Shape Constrained.

1 Introduction

1.1 Introduction

This chapter presents the problem statement, background and context, objectives, ethical considerations, and relevance of the study.

1.2 Problem Statement

Segmentation helps extract objects of interest from scans, which enables medical professionals to achieve a better understanding prior to or during medical procedures through visualization and printing (Furtado, 2021b). This dissertation deals with the problem of segmentation, specifically, segmentation of healthy liver. Segmentation is the process of labelling each pixel in an image, where the label identifies the object to which each pixel belongs. Throughout this dissertation, the term object will refer to the object of interest, that is, the healthy liver, unless stated otherwise. However, it is important to recognize that the remaining pixels, also referred to as the background pixels, are considered as a separate object (Matcha, 2021). A typical scan can have tens, if not hundreds, of slices, and thus the process of segmentation can be very repetitive and exhaustive if carried out manually Alves et al. (2018); Li et al. (2021). Processes that alleviate this burden exist, one of them being automatic segmentation models.

Automatic segmentation models do not necessarily need to be used by themselves, but can also be used as an aid to alleviate the work load, increasing the reliability, accuracy and repeatability, and therefore, the quality of the segmentations (Lachinov, 2019; Li et al., 2021; Vladimir et al., 2019). This helps medical professionals make a more accurate diagnosis and, due to the increased efficiency, segmentation could also be applied to monitor medical conditions which can be scanned, such as tumour growth (Isensee et al., 2020; Tan et al., 2020).

Many challenges arose to prompt innovative solutions in the field of medical image segmentation. One of these challenges was the CHAOS Challenge (Kavur et al., 2021), which, as its title suggests, focuses on healthy abdominal organs. This challenge prompted the work being presented in this dissertation. A more in-depth introduction to this challenge can be found in Section 2.2.

1.2.1 Difficulties in Automatic Segmentation

Segmentation models try to find commonalities in colour, intensity, texture and shape to try to separate an image into its various regions of interest (Conze et al., 2021; Furtado, 2021b). Difficulties generally arise when there is lack of contrast between neighbouring regions or similarity between regions (Furtado, 2021b; Shuang and Wang, 2020). When speaking of liver in particular, there is a lack of contrast between it and the neighbouring organs, and inhomogeneous intensities within the liver area (Furtado, 2021b; Shuang and Wang, 2020).

1.2.2 Accurate Liver Volume Applications

The liver is the second-largest organ and the largest internal organ of the human body. The liver plays a role in digestion of food, detoxification of drugs and alcohol, and metabolism (Marieb and Keller, 2017). As per Kavur et al. (2021), “the liver volume is affected by several diseases including congestive heart failure, cancer, cirrhosis, infections, metabolic disorders, and congenital diseases”. According to data provided by the World Health Organization’s International Agency for Research on Cancer, there were 841,000 new cases of liver cancer alone in 2018 (Rumgay et al., 2022). The dimensions of the liver may be a good way of diagnosing the severity of the disease as well as help with treatment planning (Kavur et al., 2021; Shuang and Wang, 2020). Therein lies the importance of fast and accurate liver segmentation from which the liver volume can be determined.

1.3 Aims and Objectives

Due to the wide scope of the medical image segmentation problem, clear and conservative goals have to be set. The principal objective is to identify a state-of-the-art model of which the results are available so that the implementation could be replicated and verified using the same database and metrics. Additional objectives include modification of the state-of-the-art architecture, and experimentation by changing the training parameters with the aim to improve upon the exiting state-of-the-art implementation.

1.4 Research Design and Ethical Considerations

This study has been approved by the University Research Ethics Committee of the University of Malta. The dataset (Kavur et al., 2019) being used is anonymized, and no patient data is available at either the source, Kavur et al. (2021), or in the files themselves.

1.5 Conclusion

This chapter presented an introduction to the study. Chapter 2 provides a critical review of the literature. Chapter 3 goes deeper into the selection process and building blocks that will be used in Chapter 4. Chapter 4 describes the research design whilst Chapter 5 presents the results, discussed in Chapter 6. Chapter 7 summarizes the most important conclusions of the study, and proposes recommendations arising from the study and suggestions for future research.

2 Literature Review

2.1 Introduction

This review was initiated using the following electronic research databases: Google Scholar[1], HyDI[2], and the CHAOS Challenge leaderboard[3] until end of July 2022. The following keywords were used: “CHAOS, CT, Segmentation, Liver”. Only papers published in the last 5 years were considered (2017-2022), however, original published papers that were used as basis may be older.

2.2 Details on Challenge

The aim of this dissertation is embodied by one of the tasks which makes up the CHAOS Challenge (Kavur et al., 2021), specifically Task 2 Liver Segmentation (CT only). However, the aim of the CHAOS Challenge is broader than this dissertation, covering five tasks in total. In fact, the aim of the CHAOS Challenge is stated as follows “the segmentation of abdominal organs (liver, kidneys and spleen) from CT and Magnetic Resonance Imaging (MRI) data” (Kavur et al., 2021). The tasks are described in detail on the CHAOS website[4], however, due to the importance of the challenge within this dissertation, a brief overview of each task can be found below:

- Task 1 - Liver Segmentation (CT & MRI): to provide an implementation which can segment the liver in both CT and MRI modalities, also known as cross-modality.
- Task 2 - Liver Segmentation (CT only): to provide an implementation which can segment the liver using the provided CT datasets only.

[1]https://scholar.google.com/
[2]https://hydi.um.edu.mt/
[3]https://chaos.grand-challenge.org/evaluation/challenge/leaderboard/
[4]https://chaos.grand-challenge.org/

- Task 3 - Liver Segmentation (MRI only): to provide an implementation which can segment the liver using the provided MRI datasets only.
- Task 4 - Segmentation of abdominal organs (CT & MRI): an extension on Task 1. To provide an implementation which can segment liver, kidneys and spleen in the MRI datasets and liver only in the CT datasets.
- Task 5 - Segmentation of abdominal organs (MRI only): an extension to Task 3 where the goal is to provide an implementation which can segment the liver, kidneys and spleen in the MRI datasets only.

Due to time constraints, it was decided that this work would focus on the simplest task, that is, Task 2. In the next section, Section 2.3, reference is being made to the mean score of the CEDs. The mean score is based on the mean of the score obtained from four metrics: Sørensen–Dice Coefficient (DICE), Relative Absolute Volume Difference (RAVD), Average Symmetric Surface Distance (ASSD), and Maximum Symmetric Surface Distance (MSSD). Further detail about the scoring system and the metrics themselves will be given in Section 3.2.2.

2.3 Existing Implementations

The CHAOS Challenge (Kavur et al., 2021) was held at The IEEE International Symposium on Biomedical Imaging (ISBI) on April 11th, 2019, Venice, ITALY. Therefore, at the time of starting the literature review of this dissertation, the challenge is already a few years old. Although the challenge is officially closed, submissions can still be made, as the submission period has been extended, however, these cannot be published as part of the challenge itself. Since the challenge first launched, both during the official run and the extended period, several implementations have been applied to tackle Task 2 of the challenge. In this section, these architectures will be explored briefly to achieve a basic understanding of the several approaches which can be applied to address the liver segmentation problem.

The architectures which will be described in this chapter are all working models which had their results published on the challenge website[5] and achieved a higher mean score than the best score quoted by Kavur et al. (2021) for Task 2, or have been mentioned in the CHAOS publication (Kavur et al., 2021), or both. Therefore, these implementations were used as a starting point. A disclaimer about these implementations: these are implemen-

[5] https://chaos.grand-challenge.org/evaluation/challenge/leaderboard/

tations which have either been detailed in a publication, pre-prints or post-prints. The implementations applied to tackle Task 2 may either be an exact implementation or modified versions of the implementations. Unless explicitly stated in the submission to the CHAOS Challenge (Kavur et al., 2021), or a reference to literature is placed within the description, there is no way of knowing whether and how these were modified. An example of this are the following five architectures: DualTail-Net, Ternaus-Net, Link-Net34, ResNet-50, and SE-ResNet-50. These were applied to the CHAOS Challenge (Kavur et al., 2021) by team *MedianCHAOS*, four of which are UNet (Ronneberger et al., 2015) variants. Although Kavur et al. (2021) provide information regarding the configuration of each of the applied model, this information is insufficient to replicate such implementations in detail, unless these were implemented in the exact same way as the papers cited in their respective subsections below. Furthermore, in the results section, the implementations were listed as *MedianCHAOS* with an iterating number at the end (example *MedianCHAOS1*). Therefore, the ability to associate the score with the particular model was lost. A mean score for all the submissions by *MedianCHAOS* was calculated to be 75.50. Nonetheless, even if modified, the concept of the implementation being applied is still retained. Furthermore, the architectures presented below are in no way an exhaustive list of all the architectures which can be applied to Task 2. It is also worth noting that most approaches use some form of CEDs within their implementation. This will become more clear as the architectures are explored.

The architectures which are going to be discussed in this chapter are closely related to one another. Figure 2.1 presents an overview of how each architecture relates to its predecessors, and splits the architectures into four levels. These levels denote the subjective complexity level, from Level 1 representing the most basic, to Level 4 representing the most complex. It is also worth noting that Level 1 and Level 2 contain four architectures which were developed for classification purposes and serve as the base for the architectures presented in Level 3 and Level 4, and that these are segmentation architectures. Following along with Figure 2.1, from the nine presented segmentation architectures, UNet (Ronneberger et al., 2015) had an influence on seven, while ResNet (He et al., 2015) and VGG (Simonyan and Zisserman, 2014) had an influence on three. VGG also inspired the ResNet architecture and both architectures are very popular amongst machine learning enthusiasts. DenseNet (Huang et al., 2017) is younger than the other three classification architectures, and it has not yet enjoyed as much popularity. The coming sections will now go into further detail about each architecture.

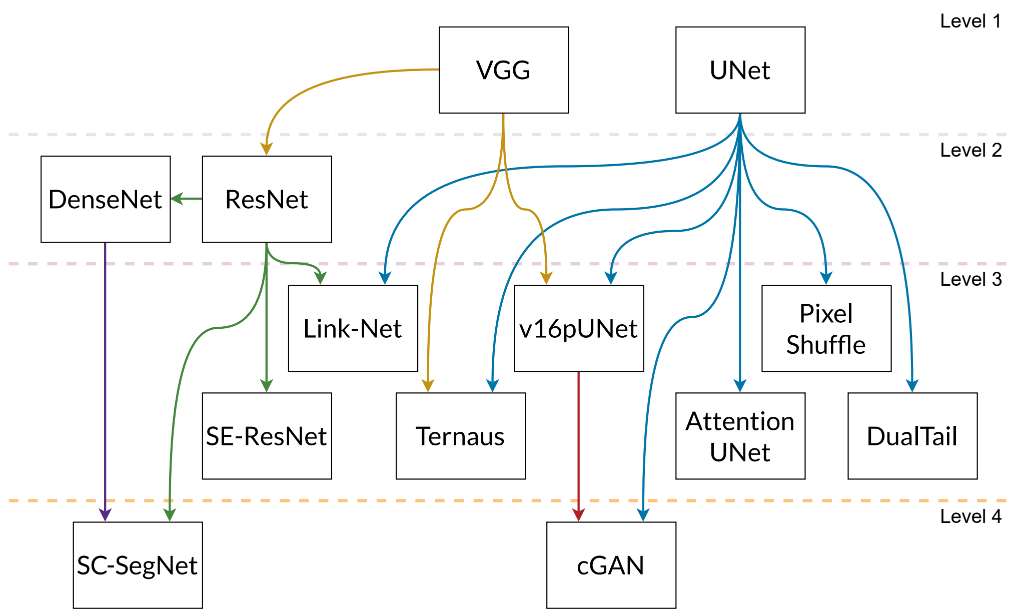

Figure 2.1: **Relationships** between the different architectures that will be reviewed in the coming sections. Levels on the right-hand side of the figure indicate the subjective complexity level.

2.3.1 cGAN

Conze et al. (2021) with their team *PKDIA*, apply the concept of Conditional Generative Adversarial Networks (cGANs) to the CHAOS Challenge, which is an architecture composed of two separate components, the generator and discriminator, connected by a loss function. To build the generator Conze et al. (2021) combine a pre-trained VGG-19 (Simonyan and Zisserman, 2014) with UNet (Ronneberger et al., 2015) to form an amalgamation of both architectures. Conze et al. (2021) apply the auto-context paradigm (Yan et al., 2019), thus forming the generator part of the cGAN. The discriminator part of the cGAN tries to discern whether the generated image looks real enough to be considered (or mistaken) as ground truth. Therefore, the discriminator uses the source image to assess either the prediction from generator or the true ground truth, and outputs the likelihood, with 0 meaning that the presented input is considered to be fake, and 1 being the real ground truth (Conze et al., 2021). This, in turn, gives feedback to the generator (through the loss function) so that it learns whether it has done a good enough job to fool the discriminator or whether it requires further optimization. Note that both the discriminator and the generator are optimizing their model with each prediction presented. The mean score achieved by *PKDIA* is of 82.46, which is the best result recorded by Kavur et al. (2021).

2.3.2 Modified Attention UNet

Ernst et al. (2019), who participated in the CHAOS Challenge (Kavur et al., 2021) using team name *OvGUMEMoRIAL*, used a modified version of Attention UNet (Oktay et al., 2018). The modifications can be summarized in three points: the addition of three scaled inputs in addition to the original input at the encoder stage, loss calculation at the different depth levels of the model, and the use of Parametric Rectified Linear Unit (PReLU) (Ernst et al., 2019) instead of the more common Rectified Linear Unit (ReLU) (Fukushima, 1975). Similar to Leaky ReLU, PReLU allows values below zero to *leak* information to the next layers by multiplying these values with a fixed multiplier (Ernst et al., 2019). PReLU however, allows the model to adjust and learn the optimal value of such a multiplier (Ernst et al., 2019). The mean score obtained by *OvGUMEMoRIAL* is 61.13.

2.3.3 DualTail-Net

Vladimir et al. (2019) introduced a novel architecture called the DualTail-Net to the CHAOS Challenge (Kavur et al., 2021), which is the only architecture presented by *MedianCHAOS* which claims not to be a UNet variant. However, many similarities can be attributed to UNet. The encoder part of the architecture is quite typical, using convolution, Exponential Linear Unit (ELU), and max-pooling layers (Vladimir et al., 2019). The DualTail-Net gets its name from the decoder part of the architecture, where it uses two separate decoders working in parallel and one feeding its output into the other (Vladimir et al., 2019). One decoder, for which input is taken from the bottleneck output, is split into four blocks and is fed connections from both the encoder and the other decoder (Vladimir et al., 2019). The latter consists of three blocks and takes its input from the last encoder (Vladimir et al., 2019). The novel part of the DualTail-Net is, of course, the use of a second decoder which works in parallel with the main decoder.

2.3.4 Ternaus-Net

Iglovikov and Shvets (2018), using a UNet combined with VGG (Simonyan and Zisserman, 2014) encoder, particularly a VGG-11 encoder, introduced the technique of fine-tuning, a technique which was commonly used in image classification but had not yet been applied to image segmentation. Fine-tuning is the process of initializing the weights of the architecture using the weights of a pre-trained model (Iglovikov and Shvets, 2018). *MedianCHAOS*, similar to one of Iglovikov and Shvets (2018) implementations, used weights which were pre-trained on ImageNet (Kavur et al., 2021). Using this technique, the performance of the model is improved as well as achieving much faster convergence (Iglovikov and Shvets, 2018; Shvets et al., 2018).

2.3.5 Link-Net34

Link-Net, similar to Ternaus-Net (Iglovikov and Shvets, 2018), uses UNet but with a ResNet-type encoder (Shvets et al., 2018). Shvets et al. (2018) uses a pre-trained ResNet34 encoder (ResNet18 was used in Link-Net (Chaurasia and Culurciello, 2017)), however, slightly modifies the decoder by including batch normalization.

2.3.6 ResNet-50

ResNet originated after He et al. (2015) tried to answer whether stacking more layers, essentially deepening the architecture, has a direct effect towards increasing the performance of a model. He et al. (2015) gives an overview of the difficulties involved in solving issues such as vanishing and exploding gradients, and degradation of model accuracy. He et al. (2015) tries to tackle the latter problem by introducing a deep residual learning framework. Residual learning is done by introducing shortcut connections within the architecture (He et al., 2015). The input of a set of layers is processed through an identity map and the output of the identity map is propagated forward and added to the output of the stacked layers (He et al., 2015), essentially propagating and adding the input to the output of the two to three stacked layers (Furtado, 2021b). Other functions can be applied other than the identity map, however, He et al. (2015) stated that the performance achieved using an identity map is adequate, given that almost no computations are required other than the addition. Using this method, He et al. (2015) managed to show that deepening the model simply by stacking does not lead to an increase in performance, however, with the introduced shortcut connections, a deeper model does perform better. He et al. (2015) presents five different configurations of ResNet: 18, 34, 50, 101, and 152. *MedianCHAOS* implemented ResNet-50 (He et al., 2015) variant as a replacement to the UNet encoder and used a UNet decoder. No further information is given by Kavur et al. (2021).

2.3.7 SE-ResNet-50

The SE in SE-ResNet-50 stands for Squeeze-and-Excitation, which concept was conceived by Hu et al. (2017) in their paper *Squeeze-and-Excitation Networks*. Hu et al. (2017) try to exploit contextual information which is present within all channels by *squeezing* global spatial information into what they call a channel descriptor. This channel descriptor, also referred to as channel-wise statistics, is generated by using global average pooling on each channel (Hu et al., 2017). The channel descriptor is passed through a Fully Connected (FC) layer, a ReLU, another FC layer, and a sigmoid activation function, of which the output is then used by the model to gate the channels (a multiplication operation should suffice) (Hu et al., 2017). The FC and ReLU layers are added to limit architecture complexity and aid generalization (Hu et al., 2017). Apart from the explanation of their SE block, Hu et al. (2017) show how their proposed block can be applied to ResNet. Thus, by combining the knowledge gained through ResNet-50 with the novel approach proposed by Hu et al. (2017), a SE-ResNet-50 (He et al., 2015; Hu et al., 2017) architecture can be constructed. Also, as per Section 2.3.6, it is being assumed that *MedianCHAOS* implemented SE-ResNet-50 (He et al., 2015; Hu et al., 2017) as the encoder part of the architecture and that for the decoder part, a UNet approach was taken.

2.3.8 Pixel Shuffle

Pixel Shuffle was first introduced by Shi et al. (2016). Lachinov (2019) formed his own team for the CHAOS Challenge (Kavur et al., 2021) and introduced a combination of residual blocks (mentioned in Section 2.3.6) applied to UNet and pixel shuffle. Due to the introduction of pixel shuffle, a group normalization is being used with the number of groups being set to four for two-dimensional images (Lachinov, 2019). In an ideal scenario pixel shuffling uses a group of four images and stitches them into one higher resolution image as per Figure 2.2, hence why the group normalization is being set to four images. The concept shown in Figure 2.2 can be extended to three-dimensional volumes (Lachinov, 2019). At the decoding part of the architecture, instead of using up-convolutions, Lachinov (2019) uses the pixel shuffle operation to upsample the images. The mean score obtained by Lachinov (2019) is 39.86.

2.3.9 SC-SegNet

Tan et al. (2020) presented the SC-SegNet, which was not included in Kavur et al. (2021)'s paper, as this implementation was submitted during the extended period of the Challenge. The SC-SegNet requires two inputs, an input image and a constraint, more specifically, a shape constraint. This implementation thus contains two separate architectures, the SC-SegNet which generates the predicted segmentations and the constraint generator which is based on DenseNet (Huang et al., 2017). The SC in SC-SegNet therefore stands for shape-constrained (Tan et al., 2020). In order to train such an architecture, a cascaded

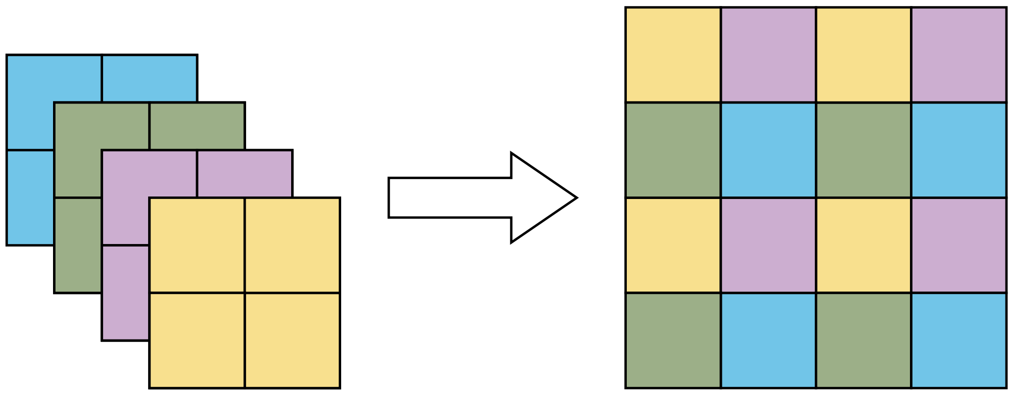

Figure 2.2: **An example of an ideal implementation of the pixel shuffle operation** applied to two-dimensional images, showing four channels in four colours on the left and the output of the operation on the right. The four channels were stitched together to form a higher resolution image (Lachinov, 2019).

learning strategy is employed, where first, the shape-constraining architecture is trained, then the SC-SegNet is trained, supervised by the ground truth images and the shape constraining model (Tan et al., 2020). The architecture of SC-SegNet is based on FCN (Long et al., 2015), DenseNet, and UNet. Apart from the SC-SegNet,Tan et al. (2020) also proposed a loss function which goes hand in hand with the architecture. The loss function considers three factors, adaptive weighted cross entropy, consistency between the learned shape constraint and the segmentation output, and edge preservation smoothness (Tan et al., 2020). The mean score obtained by SC-SegNet on the CHAOS Challenge is 83.52.

2.3.10 Honourable Mentions

Two other teams are listed as having participated in the CHAOS Challenge, with team names *ISDUE* and *IITKGP-KLIV*, however, very little detail is given about their implementations. For the sake of completeness their mean scores were as follows: *ISDUE* achieved a mean score of 55.79, and *IITKGP-KLIV* achieved a mean score of 55.35.

Apart from the implementations mentioned above, other implementations have made great strides since the challenge closed, achieving an even better score than the best mean score published in the CHAOS Challenge paper (Kavur et al., 2021) of 82.46. Unfortunately, most of these implementations did not have their methodology stated and were simply described by a short comment.

Table 2.1 summarizes the score of the implementations mentioned above. More de-

Table 2.1: **Summary of Results** for the models mentioned in Section 2.3 (Kavur et al., 2021).

Model	Mean Score	DICE Score	RAVD Score	ASSD Score	MSSD Score
cGAN	82.46	97.79	73.60	94.06	64.38
Modified Attention UNet Loss	61.13	90.18	44.35	81.03	28.96
Median CHAOS Models	80.45	97.55	69.19	94.02	61.02
Pixel Shuffle	39.86	68.00	22.67	53.28	15.47
SC-SegNet	83.52	97.57	67.99	94.77	73.75

tails can be found on the CHAOS paper (Kavur et al., 2021) the CHAOS and website[6].

2.4 Building Blocks

As seen in Figure 2.1, UNet and VGG have had a great impact on current Deep Convolutional Neural Networks (CNNs) and CEDs as well as the work which will be presented in this dissertation. Furthermore, many of the implementations mentioned in the previous section also made use of these architectures, either by directly using them as their base or indirectly by using architectures which were inspired by VGG or UNet architectures. Therefore, it was deemed important to review the work which had been carried out in these papers. A brief overview on their contributions will be given below and a more in-depth look into the inner workings of these architectures can be found in Chapter 3.

2.4.1 UNet

The UNet architecture, presented by Ronneberger et al. (2015), builds upon the concepts of Fully Convolutional Networks (FCNs) (Long et al., 2015), where Ronneberger et al. (2015) optimized UNet to work with few training images but, at the same time, yielding more precise results. Ronneberger et al. (2015) emphasize on feature localization by using what they call *skip-connections*, which connect the encoder parts of the architecture with the decoder parts at multiple levels by using the concatenate operation (Ronneberger et al., 2015; Vladimir et al., 2019). Another difference from FCN is that UNet does not have any FC layers, instead, it uses up-convolutions (Conze et al., 2021; Ronneberger et al., 2015). By concatenating the skip connections with the up-convolution output, the convolutions that follow can extract context from feature maps that are less processed and closer to the image, which enables the architecture to achieve a high resolution out-

[6]https://chaos.grand-challenge.org/evaluation/challenge/leaderboard/

put (Ronneberger et al., 2015). This architecture has been very popular amongst many biomedical and medical applications, achieving many state-of-the-art results (Lachinov, 2019) and many of the architectures presented in Section 2.3 used UNet as their base architecture (Conze et al., 2021; Furtado, 2021b; Lachinov, 2019), as seen in Figure 2.1.

2.4.2 VGG

The aim of Simonyan and Zisserman (2014) when writing their paper, was to study how CNN performance is affected by increase in depth and the feasibility of deepening CNNs for classification. With this in mind, Simonyan and Zisserman (2014) came up with several CNN configurations, presenting six configurations in total. All configurations use a combination of five stacks of convolutional layers and max pooling layers, followed by FC layers and a soft-max (Simonyan and Zisserman, 2014). Like UNet, the VGG architectures can be applied in various situations and can even be combined with UNet as seen in Section 2.3.1 and Section 2.3.4. Given its versatility, simplicity and effectiveness, the VGG architectures are widely used in conjunction with other architectures.

2.5 Conclusion

This chapter presented a critical review of the literature associated with the study. The next chapter shall describe and discuss the background concepts used in the study.

3 Background

3.1 Introduction

This chapter presents details on key concepts and background information on several topics, including the CHAOS Challenge, the selection process, and the building blocks, which are necessary to understanding the next Chapter.

3.2 The CHAOS Challenge

In Chapter 1, the CHAOS Challenge was briefly introduced. The results obtained in this work are a direct consequence of the CHAOS Challenge because the challenge provided resources, such as the database to train the architecture and a platform to submit segmentation results. This not only gives more robust results, but also enables the participant to compare their model with models implemented by other participants on a worldwide scale. Therefore, a more in-depth analysis of the resources provided by the Challenge and its metric will be presented below.

3.2.1 Database

The most important reason of why the CHAOS Challenge (Kavur et al., 2021) was chosen, apart from being one of the latest challenges to be published at the time of writing this dissertation, is its database (Kavur et al., 2019). The challenge provides two databases, one for CT and one for MRI. Given that this dissertation focuses on CT, only the CT database will be explained, however, more information can be found on either the challenge publication by Kavur et al. (2021) or their website[1].

The CT database contains 80 CT scans of 80 different potential liver donor patients, who have healthy (no tumours, lesions or any other diseases) liver and who were injected

[1]https://chaos.grand-challenge.org/

with contrast agent prior to taking the scans (Kavur et al., 2021). A single CT scan will be referred to as a dataset, as denoted by Kavur et al. (2021), and a dataset contains multiple slices. Slices are the individual two-dimensional images which, when combined and ordered, make up the dataset. Each dataset has a varying number of slices, as each dataset was taken using different CT machines or using different machine settings or both (Kavur et al., 2021). However, all patients, and thus all datasets, have the same orientation and alignment as well as bit-depth (16-bit) and slice size (512 by 512 pixels) (Kavur et al., 2021). The number of slices per dataset varies from 78 to 294, with an average number of 160 slices per dataset. The database was also described by (Kavur et al., 2021) as having the following characteristics:

- Similar Hounsfield value range of adjacent organs.
- Varying Hounsfield ranges for the same tissue across data sets due to the contrast media.
- Significant shape differences of anatomical structures across patients.
- 15% of the database contains atypical liver shapes (that is, unusual size or orientation of the liver).

The database is distributed into two sub-categories, consisting of 40 datasets for training and 40 datasets for evaluating the model (also referred to as testing) (Kavur et al., 2021). The training data contains both the source images, which are the images outputted by the scanner in the DICOM standard format (without the patient-related information), and the corresponding ground truth images (Kavur et al., 2021).

The ground truth images are images which contain segmentations of the DICOM images carried out by professionals (Kavur et al., 2021). During training, the ground truth images are used by the model to be able to learn, and to verify or validate that the proposed implementation is indeed converging towards an optimal solution. The testing datasets provided to the challenge participants only contains the DICOM images and no ground truth. (Kavur et al., 2019). The ground truth is reserved by the challenge organizer and, in order to achieve the final evaluation, the predicted segmentation images (or predictions) must be submitted to the challenge organizer. The predictions are the output of the model when processing the test dataset. These are compared to the ground truth using various metrics to be able to measure the model's performance. Detailed explanations of the metrics and evaluation will be given in the coming section, Section 3.2.2. This approach is taken to avoid foul play during the training and optimization process of the implementation being proposed (Maier-Hein et al., 2018).

3.2.2 Evaluation

In order to assess the performance of a proposed implementation, the model needs to be evaluated. Evaluation is carried out by mathematically measuring the performance of the predictions. By using mathematical evaluation methods, one is able objectively compare the proposed implementations, thus, optimizations and changes would reflect on that feedback (Goodfellow et al., 2016). It is also important to keep in mind that segmentations of organs can be used for various medical procedures (Kavur et al., 2021). Having a single metric to evaluate the proposed methods will not yield a robust evaluation that covers a broad spectrum of medical procedures. Thus, the CHAOS Challenge proposes four metrics (Maier-Hein et al., 2018) for evaluating the predictions and the implementations being proposed. The purpose of these metrics is to analyse results in terms of overlapping, volumetric, and spatial differences between a predicted solution and the ground truth (Kavur et al., 2021). The metrics chosen by the CHAOS Challenge (Kavur et al., 2021) are summarized in bullet form below. In the equations shown below, the term $\boldsymbol{S}$ represents the set of voxel labels as processed by the model, the term $\boldsymbol{G}$ represents the set of voxel labels in the ground truth, and $|.|$ denotes cardinality.

- DICE, a measure of the accuracy of the predictions. Ideal score 100 % (Goal: maximize).
$$\frac{2|(\boldsymbol{S} \cap \boldsymbol{G})|}{|\boldsymbol{S}| + |\boldsymbol{G}|}$$
- RAVD, compares the difference in volume between the ground truths and predictions. Ideal score 0 % (Goal: minimize).
$$\frac{abs(|\boldsymbol{S}| - |\boldsymbol{G}|)}{|\boldsymbol{G}|} \times 100$$
- ASSD, the average Hausdorff distance between the ground truths and predictions. Ideal score is 0 mm (Goal: minimize).
$$mean(\inf\{\varepsilon \geq 0 \mid \boldsymbol{G} \subseteq \boldsymbol{S}_\varepsilon \; and \; \boldsymbol{S} \subseteq \boldsymbol{G}_\varepsilon\})$$
- MSSD, the maximum Hausdorff distance between the ground truths and predictions. Ideal score is 0 mm (Goal: minimize).
$$\max(\inf\{\varepsilon \geq 0 \mid \boldsymbol{G} \subseteq \boldsymbol{S}_\varepsilon \; and \; \boldsymbol{S} \subseteq \boldsymbol{G}_\varepsilon\})$$

Although the above metrics can be used as an indicator of the performance of a model, there exists an inherent difficulty in comparing the four values simultaneously. Additionally, both the ASSD and MSSD can have values ranging from 0 mm up to an unknown

Table 3.1: CHAOS Metrics' Threshold Values.

Metric	Worst Value	Threshold
RAVD	∞	< 5%
ASSD	Δ	< 15 mm
MSSD	Δ	< 5 mm

Δ represents the largest possible distance (Kavur et al., 2021).

maximum quantity and similarly, RAVD can have values up to infinity (Kavur et al., 2021). This makes it difficult to discern performance as well as plot the data. Kavur et al. (2021) introduced a scoring system to deal with these issues and to help eliminate performance figures which are too poor to be of any value. Thus, Kavur et al. (2021) set maximum threshold values for all the metrics except the DICE Score[2]. Any values beyond the threshold are assigned a score of 0. The procedure to determine the thresholds is explained in detail by Kavur et al. (2021). The threshold values are tabulated in Table 3.1. Values within the threshold are normalized to obtain a score between 0 and 100, where 100 is the ideal score. The final score, also referred to as the mean score, is achieved by averaging the score over the four metrics to achieve a single point of reference for the overall performance of the model (Kavur et al., 2021). When submitting predictions to the Challenge, the scoring system is used to establish the rank of the model.

3.3 Selection Process

In the previous chapter, several implementations for the liver segmentation problem were explored. All the implementations are interesting and offer several concepts to explore, making it extremely hard to select a single implementation. Nonetheless, due to resource and time constraints, a point of reference had to be selected prior to starting experimentation, in order to achieve a successful implementation. A selection process was carried out in order to filter through the hundreds of papers which had been published on the subject.

Work in this dissertation was carried out to verify existing state-of-the-art implementations. In order to do so, the selected model had to have already published results using the CHAOS Challenge. Thus, the first criteria that was set in stone was that any work that would be selected had to have cited the CHAOS Challenge (Kavur et al., 2019) and, more

[2]Kavur et al. (2021) shows a threshold of DICE > 0.8 in their paper, however, upon investigating the received results, no difference in values could be noticed between the DICE values and the DICE score values. Therefore, it is being presumed that no processing is being carried out to compute the DICE score values.

importantly, the work had to have been evaluated using the CHAOS Challenge database (Kavur et al., 2019) and have its results published for Task 2. An implicit criterion of time was also present, given that the CHAOS Challenge was first published on April 11, 2019 and the selection of work was carried out around the month of October 2021[3]. The criteria presented above narrowed the search down to the CNNs, which were reviewed in Chapter 2. Other considerations which were taken into account during the selection process were the performance achieved by the proposed model, the information available about the implementation of such a model, as well as the difficulty of implementing same implementation.

In the end, Conze et al. (2021) was chosen as the main reference paper for the following reasons: Conze et al. (2021) implemented 16 different models, starting from basic and advancing to more complex implementations, which meant one could understand simple concepts first and steadily build up to bridge the gap towards the state-of-the-art implementation. For each of the models implemented, the results obtained from the CHAOS Challenge were clearly listed in a table. Overall, the paper is well explained and enough detail about the architectures implemented is presented. Where previously published architectures were used, Conze et al. (2021) made clear references to the published work, avoiding any misunderstandings. Furthermore, although it was not a requirement, the models implemented were tested not just on Task 2 of the CHAOS Challenge but on all tasks. This gives an understanding of the robustness of the implemented model.

3.4 Starting Point

As mentioned in Section 2.3.1, the state-of-the-art implementation by Conze et al. (2021) uses the cGAN architecture. However, the best score for Task 2 of the CHAOS Challenge was achieved by a model called v16pUNet1.1C, which was also present in Conze et al. (2021)'s work. Given that v16pUNet1.1C achieved the best performance, it is being considered as state-of-the-art, at least in terms of Task 2. Thus, the focus of this dissertation was placed on this architecture, given its performance versus complexity advantage. The name of this architecture can be split into four parts: 'v16' representing the VGG16 encoder, p short for pre-trained (Conze et al. (2021) pre-trained on ImageNet (Russakovsky et al., 2015)), 'UNet' meaning that the architecture is based on a UNet architecture, '1.1' meaning that the architecture is cascaded and 'C' to denote Conze et al. (2021)'s architecture. Each of these components are explained in Section 2.4. The overall architecture

[3]Although the initial selection was carried out in the month of October 2021, another review was also carried out in the during the month of May 2022 to include any new implementations.

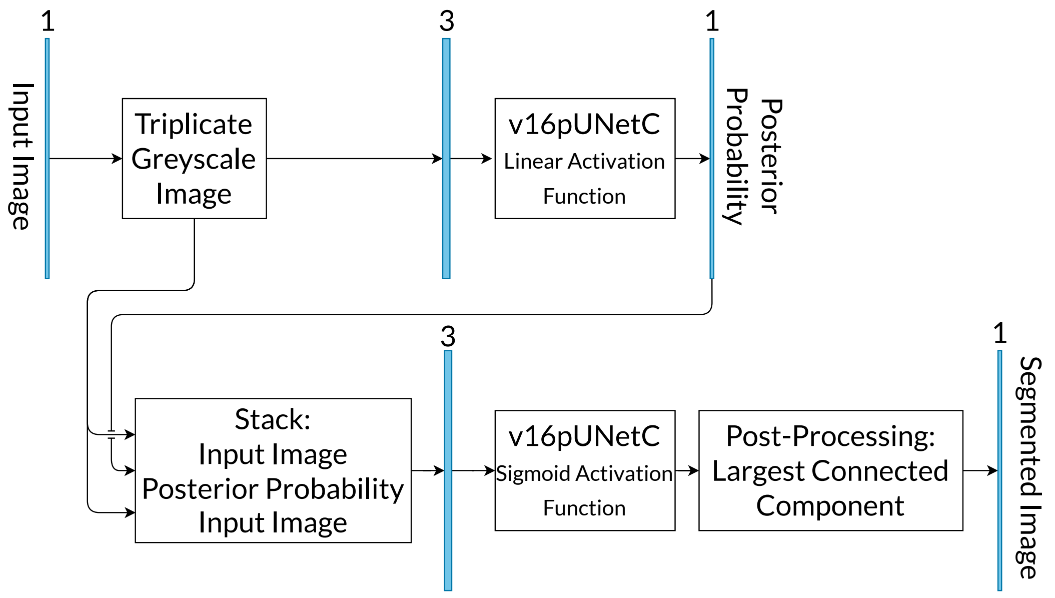

Figure 3.1: Pre-Trained Cascaded Framework Architecture Diagram (Conze et al., 2021)

can be observed in Figure 3.1. Due to the fact that the model being used is pre-trained on RGB images, the input (being greyscale) image needs to be triplicated and concatenated in a stack format to mimic the three channels of an RGB image. A similar concept is applied to the input of the second model but instead of triplicating the posterior probabilities of the first model, Conze et al. (2021) concatenated the input image, the posterior probabilities and the input image again, stacking the three in a sandwich-like manner, with the posterior probabilities in between two copies of the input image. This enables the second model to obtain context, not only from the output of the previous model, but also from the input.

3.5 Building Blocks in Detail

3.5.1 VGG16UNet Architecture

The VGG16UNet Architecture is a combination of two architectures, VGG16 as an encoder and UNet as a decoder. Since the UNet architecture will be explained in Section 3.5.2, only the VGG16, and the combination of UNet and VGG16 will be explored in this section. Inspired by Krizhevsky et al. (2012), in their paper, Simonyan and Zisserman (2014) propose six configurations of what today are generally known as VGG architectures. Given that in this dissertation only the VGG16 architecture has been implemented,

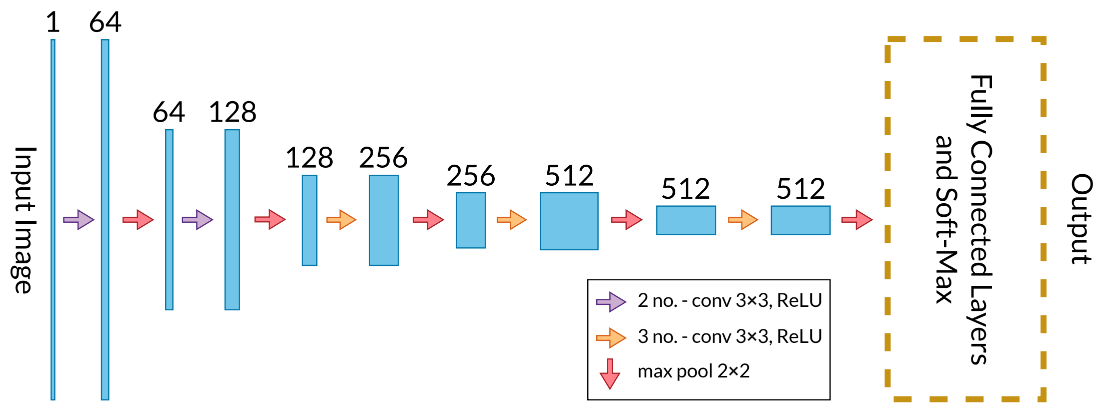

Figure 3.2: VGG16 Architecture Diagram (Simonyan and Zisserman, 2014)

the explanation shall focus only on VGG16. However, the other five implementations share a similar approach, with a different number of layers. To aid with the explanation, a diagram of the architecture was prepared and can be seen in Figure 3.2. The architecture is built on five sets of stacks of 3×3 convolutions with ReLU, with a 2×2 max pool operation at the end of each stack. Stack 1 and Stack 2 (the stacks closest to the input) contain two convolutions, while Stacks 3 to 5 contain three convolutions. After the last max pool operation, Simonyan and Zisserman (2014) use two 4096-D FC layers and one 1000-D FC layer for a 1000 way classification. The output of the last FC layer is passed through a soft-max activation layer to obtain the classification output.

Observing both the encoder part of UNet and the VGG16 architecture, one can notice several similarities in the construction of the architecture such as the structuring of the convolution layers followed by a max pool operation. Focusing more on the differences, VGG16 increases the number of convolutions at deeper levels and at the deepest level, the architecture does not go beyond 512 feature maps. By taking note of these changes and updating the UNet decoder to match these changes, the VGG16UNetC can be constructed. The architecture diagram of the VGG16UNetC can be observed in Figure 3.3.

3.5.2 The UNet Architecture

The UNet architecture builds upon the FCN architecture by Long et al. (2015). UNet is made up of five types of CNN layers, three convolution types, copy and crop operation (previously referred to as skip connection), and max pool layers. The UNet architecture takes its name after the shape which the architecture forms when presented as a diagram,

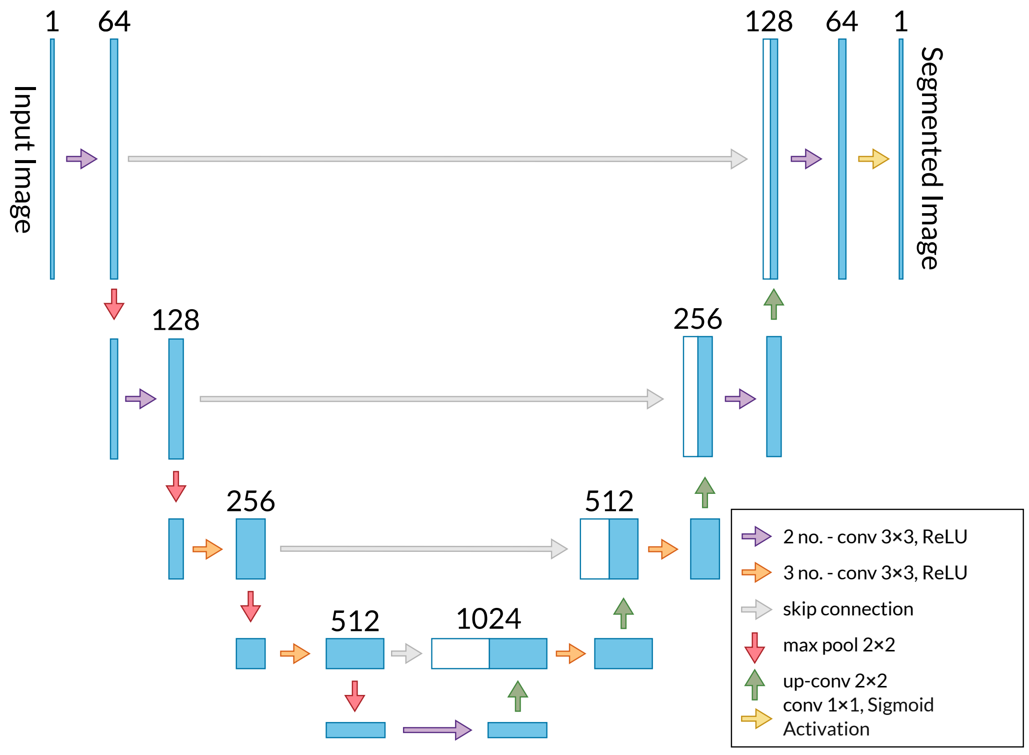

Figure 3.3: VGG16UNetC Architecture Diagram (Conze et al., 2021)

as seen in Figure 3.4. The architecture consists of two 3 $\times$ 3 convolution operations, each followed by a ReLU, where the first convolution operation doubles the number of features. These are followed by a 2 $\times$ 2 max pooling operation with a stride of 2 for down sampling. The convolution, ReLU and max pooling operations are repeated for four times in total to make up the contracting path. Before each max pooling operation, the output is stored to later be used by the expanding path. The expanding path follows similar repetitions to the contracting path, but the max pooling operations are replaced by 2 $\times$ 2 up-convolution operations. The up-convolution halves the number of features, but doubles the resolution. After the up-convolution, a concatenation operation is carried out to double the number of features prior to convolution operations. Linking the contracting and expanding paths, is the bottleneck, which consists of two 3 $\times$ 3 convolution operations each followed by a ReLU. The final operation of the expanding path is a 1 $\times$ 1 convolution to map the 64 remaining feature maps to the desired number of classes. On the bottom right of the figure, there is a legend showing the five types of layers.

The diagram shown in Figure 3.4 has one key difference from the diagram presented by Ronneberger et al. (2015), this being the use unpadded convolutions. As a conse-

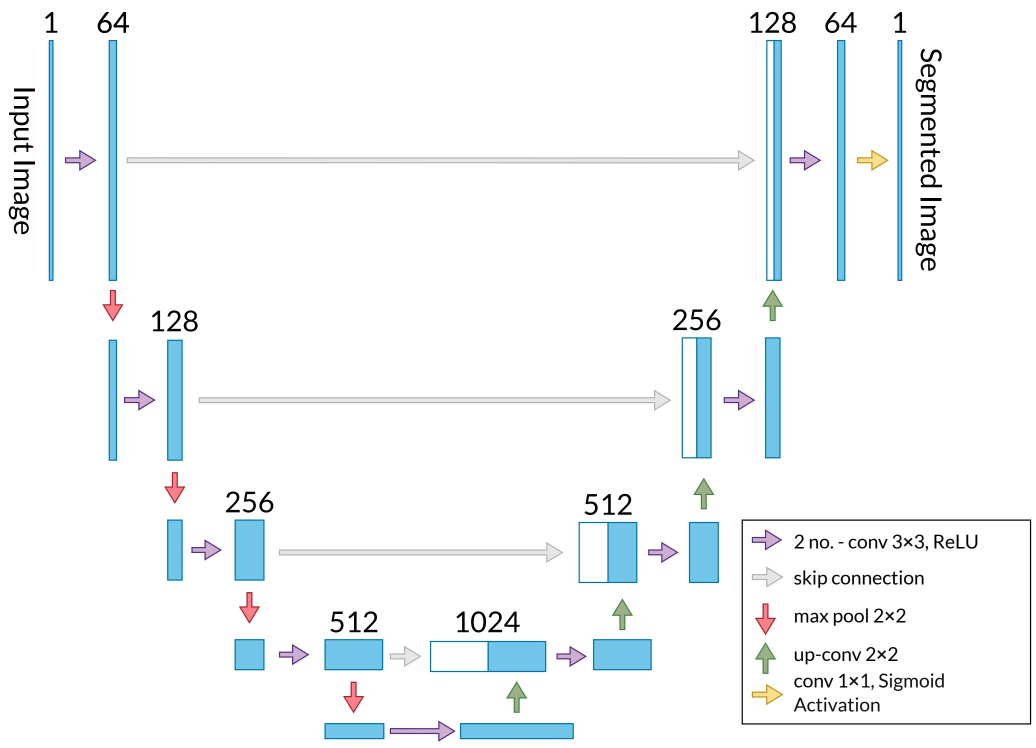

Figure 3.4: UNet Architecture Diagram (Ronneberger et al., 2015)

quence, the skip connections would also require a crop operation and the size of the output images (segmented images) would not be the same as the input. Depending on the type of input images being fed to the model, unpadded convolutions can be replaced by padded ones to, essentially, achieve a same resolution input-output (Ronneberger et al., 2015). At the time, Ronneberger et al. (2015) opted for unpadded convolutions, as the input image had to be split into smaller patches since the original image was too large to be processed at one go by the Graphics Processing Units (GPUs) available then. In order to achieve a complete output, a process of stitching had to take place after all the patches were processed (Ronneberger et al., 2015). Due to the limited availability of training data, Ronneberger et al. (2015) also performed data augmentation by applying elastic deformations, essentially increasing the potential invariance learning of the model.

3.5.3 Cascaded Framework

The cascaded framework works by having two models, one after the other, with the first model being used to give context to the second model. This is what Yan et al. (2019) called long-range spatial context. In their implementation, Yan et al. (2019) used different

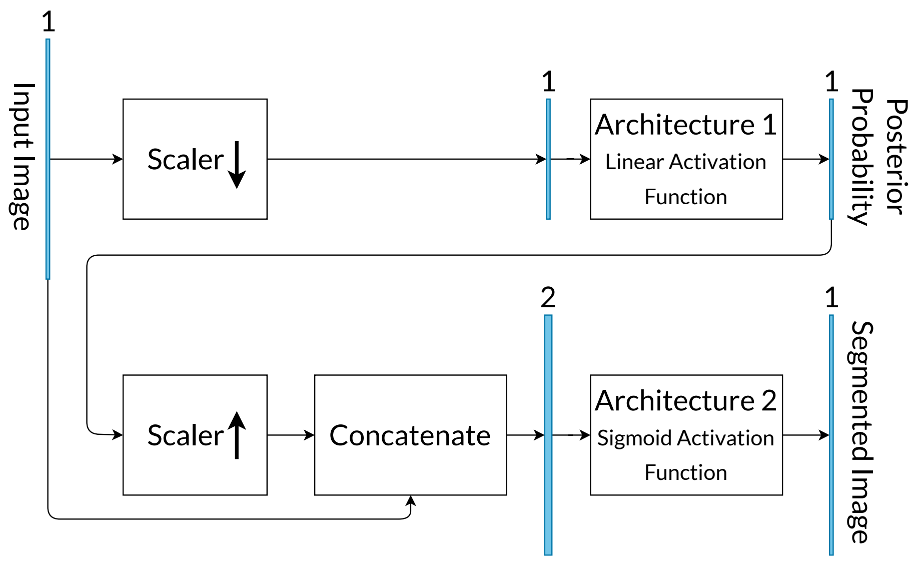

Figure 3.5: Cascaded Framework Diagram (Yan et al., 2019)

sized models in their cascade, the first processed a scaled down, lower resolution version of the input image. The posterior probabilities of the first model are then scaled to the original resolution and concatenated with the input image. This stack is then fed to the full resolution model, of which the output is the segmented image. This framework can also be implemented without scaling, meaning that the input to the first model is the same resolution as the input to the second. Figure 3.5 shows the cascaded framework as implemented by Yan et al. (2019). In their work, Conze et al. (2021) applied the cascaded framework to several architectures and reported an overall mean score increase of approximately 2 percent.

3.5.4 Pre-Training

Another feature which Yan et al. (2019) used was transfer learning for the first architecture (see Figure 3.5). Transfer learning is not to be confused with pre-training. Pre-training occurs when the architecture is first trained on a mock database and the weights and biases are saved. The mock database does not necessarily follow the same types of images or tasks which the target database does. Thus, pre-training is considered as a general term. The process of transferring the weights and biases to the architecture which will be trained on the target database, is known as transfer learning. Transfer learning is generally considered as such when the mock database has a similar task to that of the target database, which helps the architecture to familiarize itself with the task at hand prior to start training on the target database. When the task of the mock database is not similar to that of the target database, the term used is simply that the model has been pre-trained, stating the database on which the model has been pre-trained. The weights and biases learned from the mock database are then transferred to the architecture training on the target database. During the training process on the target database, a process referred to as fine-tuning (Goodfellow et al., 2016; Iglovikov and Shvets, 2018; Kavur et al., 2021; Zhu et al., 2022) occurs, which is the process of further adjusting the weights to the target data.

Two caveats of using pre-trained models are that the architecture being used must match the pre-trained model's architecture exactly and, unless pre-training is carried out by the architecture developer, only a limited selection of models are available, given that pre-trained models are made available by third-parties. Most pre-trained models are pre-trained on classification tasks, therefore, when translating to a segmentation task, pre-training can only be applied to the encoder part (see Section 3.6.3) of an architecture. Nonetheless, pre-training still plays an important role, as it can compensate for a lack of training data and increases the variety of the data which helps to avoid overfitting.

3.6 Introduction to Convolutional Encoder-Decoders

CEDs are built on the concepts which were laid down by CNNs. To fully understand the progress that has been made in artificial neural network architecture, it is important to understand what CNNs are. The sections that follow present brief definitions of what is understood by the terminology used by CNNs and a brief overview of CNNs and CEDs.

3.6.1 Implementation, Model and Architecture

In this work, there are several references to the terms Implementation, Model, and Architecture. A brief clarification on how these terms are being used will follow. When the term implementation is used in this work, it is understood to include the training parameters, and the architecture or model. All the details required to be able to replicate the building and train the architecture or model should be included, such as type of optimizer, learning rate, loss function, number of epochs, number of augmentations, etcetera. An architecture is a representation of the individual layers, clearly defining each layer in detail and how each is connected to the other. No training details are required when using the term architecture. A model is defined as the architecture after it has been trained. It contains information on how the model is built and the weights which have been calculated after training, but training parameters are not necessarily included[4]. For a given input, a model will always output the same result as its weights and biases are fixed. An architecture that is partially trained is also being considered a model because if the training is stopped, its bias and weights are still fixed at that point in time and will still give a deterministic result.

3.6.2 What are CNNs?

CNNs are a sequence of layers which compute feature maps (Furtado, 2021b). To explain what feature maps are, consider a greyscale image of size 512 by 512 pixels and this image will be used as an input to a CNN (Furtado, 2021b). Then it can be said that the CNN will be fed a feature map of depth 1 and a resolution of 512 by 512. As the CNN processes the image, it computes matrices having an increased depth but lower resolution. These lower resolution feature maps try to capture features from the input that are particular enough to be able to correctly classify or segment the input, but general enough that if a different input is fed, they are still able to output a satisfactory answer. An important part of CNNs and also part of their name, are the convolution operations or convolutional layers (Hu et al., 2017). Convolutions are mathematical product-sum operations (Furtado, 2021b). They have a kernel size of (n $\times$ n) and a bias. The kernel passes over the feature maps, producing an output of which the size depends on the stride and padding selected. This output is then processed by other mathematical operations or layers such as batch normalization, ReLU, pooling layers, activation layers, and FC layers.

Up till this point, data has been considered to move forward only (from the input to the output). However, for a model to learn, information needs to be propagated backwards. This is done using an algorithm called back-propagation (Furtado, 2021b). During the training process, the model is allowed access to both the training data and the ground truth values. By using the ground truth values, the model is able to mathematically calculate the difference between what was predicted and what the value should be. This information (error or loss) is then passed backward (also applying a learning rate to limit the magnitude of change) to update the convolutional kernels and biases as well as other layers which benefit from such information (Furtado, 2021b). The model iterates this process several times and, ideally, the loss starts to become so small, that the changes applied by back-propagation become negligible (Furtado, 2021b). At this point, the model has successfully learned a function that best approximates the input-output relationship of the training data.

[4]https://developers.google.com/machine-learning/glossary#m

3.6.3 What are Convolutional Encoder-Decoders?

CEDs can be defined by three main parts: the encoder, the bottleneck, and the decoder. The encoder is the contracting part of the architecture, it computes and codes the original input, CT slices in the case of this dissertation, into feature maps. The feature maps are generally of lower resolution but higher depth than the original input. The encoder's architecture is usually inspired by a CNN architecture (Conze et al., 2021; Li et al., 2021; Ronneberger et al., 2015; Vladimir et al., 2019). This, however, is where the similarities between CNNs and CEDs end, because CNNs are generally used for classification, whereas CEDs are generally applied to image segmentation. At this point the CNNs can prepare to output their results, whereas CEDs still have two parts remaining for them to be able to output the segmented image.

The decoder, also known as the expanding path of the architecture, tries to find an optimal solution for the output from the coded input, that is, it tries to guess the desired output from the information which was computed in the feature maps. The decoder is generally made up of convolution, batch normalization, ReLU, and activation layers. The link between the end of the encoder and the start of the decoder is known as the bottleneck. The bottleneck bridges the output codes from the encoder and prepares them for the decoder for further processing. The architecture may also have other intermediate connections between the contracting path and the expanding path, such as dense connections or residual connections. (Conze et al., 2021; Li et al., 2021; Ronneberger et al., 2015; Vladimir et al., 2019).

3.7 Conclusion

This chapter presented background information of the concepts associated with this study. The next chapter shall describe and discuss the research design used in this study.

4 Implementation

4.1 Introduction

As discussed in Section 3.4, the work carried out during this dissertation follows very closely the work carried out by Conze et al. (2021). In their work, Conze et al. (2021) presented multiple models with varying implementation difficulty. Although the main focus of this dissertation was implementing the v16pUNetC, this helped with the familiarization of several architectures. The research approach which has been taken to tackle the research question is a quantitative research method with experimental research design, as the research involves the modelling of data and calculation of metrics using a statistical approach to make sense of the data (Kamiri and Mariga, 2021). This chapter is split into three main sections: Section 4.2 explains the difficulties encountered that are CHAOS related, Section 4.3 where part of the work carried out by Conze et al. (2021) was verified, and Section 4.4 where variations on a modified model inspired by Conze et al. (2021) were carried out.

4.2 Getting Started

The work carried out prior to running a training session with an architecture, is vast and very important. This includes tasks such as setting up the environment, importing and processing the dataset, implementation of part of the pre-processing, as well as dealing with any anomalies. The synergy between these components was tested using the UNet architecture (Ronneberger et al., 2015), given its simplistic nature, with slight modifications. These modifications included the padded convolutions and the batch normalization after each convolution. The addition of the padded convolutions was explained in Section 3.5.2, but in short, these were used to keep the output resolution the same as the input. Batch normalization is carried out in batches during training to speed up the optimization process of the model. It introduces additive and multiplicative noise on the hidden units, which has a regularization effect and should make drop out layers unnecessary (Goodfellow et al., 2016). During this process, no architecture changes were carried out, using only the UNet architecture to test out the various components.

4.2.1 Database and Metrics

Specific details about the composition of the Database and the metrics used by the CHAOS Challenge were discussed in Section 3.2. In this section, the practical aspects of the database will be discussed. The two X-ray tube physical parameters that govern the quality of the slices are the tube current over time (mAs) and the peak tube voltage (kVP). Higher tube current over time generally translates to lower noise while the peak tube voltage affects the penetrating power of the X-Rays; as peak tube voltage is increased, the X-rays' penetrating power increases. This can affect the contrast of the slice as well as its Hounsfield Unit (HU) values. Let δ be a measurement of a patient's tissue at a location $(x,\ y,\ z)$ in HU. It can be said that $\delta_{\mathrm{HU}(x,y,z)}$ is a measurement based on the linear attenuation coefficient, μ, of the patient's tissue at a location $(x,\ y,\ z)$ in space (at a particular voxel) (Mahesh, 2013). To convert the value of $\mu(x,y,z)$ to HU the following equation can be used:

$$\delta_{\mathrm{HU}(x,y,z)} = 1000\frac{\mu(x,y,z) - \mu_w}{\mu_w}$$

where μ_w is the linear attenuation coefficient of water. The HU range for a human body CT scan generally ranges from -1,000 HU for free air to around 1,200 HU for bone. The HU range does not have an upper limit for CT scans, therefore, the limit for a slice is typically set either by what is being scanned or the physical limitations of the sensor, that is, by the bit-depth of the scanning machine and the saturation levels of the sensor itself. This gives a practical range to work with. HUs are used by medical professionals as their values are easier to remember than linear coefficient unit values. Most CT scans provided by the CHAOS Challenge are stored in 16-bit, unsigned integer format (expect for dataset 39) and, depending on the tool which is used to read the DICOM images, these have to be converted into HUs. In order to be able to carry out this operation, a DICOM file stores a slope and an intercept value. These are then applied to the array of values stored within the DICOM file, which values are converted to HUs. These are applied as follows:

$$\delta_{\mathrm{HU}} = \delta_{\mathrm{raw}} \times \text{slope} + \text{intercept}$$

where δ_{raw} is the value stored in the DICOM image and δ_{HU} is the converted value. The tube current over time, tube peak voltage, slope and intercept values contained within the training and testing databases were read[1] to check for any anomalies. Dataset 39 in

[1]Details about the databases parameters can be found in Appendix A.1, Table A.1.

the Testing database had different peak voltage than all other datasets.

During the process of reading the datasets, a medical professional may apply a window filter. The window filter helps to overcome the sensitivity limitations of the human eye, as it can only differentiate between a limited number of light intensities. Therefore, windowing helps medical professionals to view only the areas of interest required. When changing the kVP, the window levels also need to be adjusted to match the new HU values of interest. A similar approach can be applied to an implementation to facilitate training. In fact, it is almost essential. Initially, the process of windowing was not being applied, and the architectures were being fed the whole range of each image. However, normalization without windowing was found to be problematic because each individual slice contains different maximum and minimum values. Thus, when normalizing each image individually, the relationship between the physical value (linear attenuation coefficient) and the values being computed, is lost. Furthermore, some DICOM images contained what will be referred to as anomaly values, which are not removed by a normalization operation. The anomaly values are single pixels within the slice that contain values magnitudes higher than the generally accepted range for human tissue. These anomaly values were generally located at the top leftmost part of the image, which is not within the chest area of the patient. These anomaly values caused two major problems to the model. The first problem was that the anomaly values significantly slowed down the optimization process of the model during training. The slowdown was attributed to the model being unable to optimize towards a specific range of HU values, causing the model to have to re-adjust completely whenever the anomaly is present. Secondly, when the model encountered the anomaly values within the validation datasets only, the model was unable to detect any liver.

Solving the normalization issue was tedious. All the maximum and minimum values from all the DICOM slices were read and visualized for all slices. The graph, seen in Figure 4.1, shows that 90% of the slices have a maximum value of less than 1,906 HU. Values above 1,906 HU then jump to 16,600 HU which is too dense to be considered as human tissue. A maximum saturating threshold was therefore set to 2,000 HU, as visualized in Figure 4.1 by the green line. A similar exercise was carried out for the minimum values. The minimum values ranged from -1,200 HU to -1,000 HU. The saturating threshold for the minimum values was set to -1024 HU. The slices were then capped at values ranging from -1024 HU to 2,000 HU. Values beyond the thresholds were set to the threshold and the images were then normalized to [0, 1]. Figure 4.2 shows an example of a normalized image prior to applying the windowing on the left-hand side, and post-application of the windowing on the right-hand side.

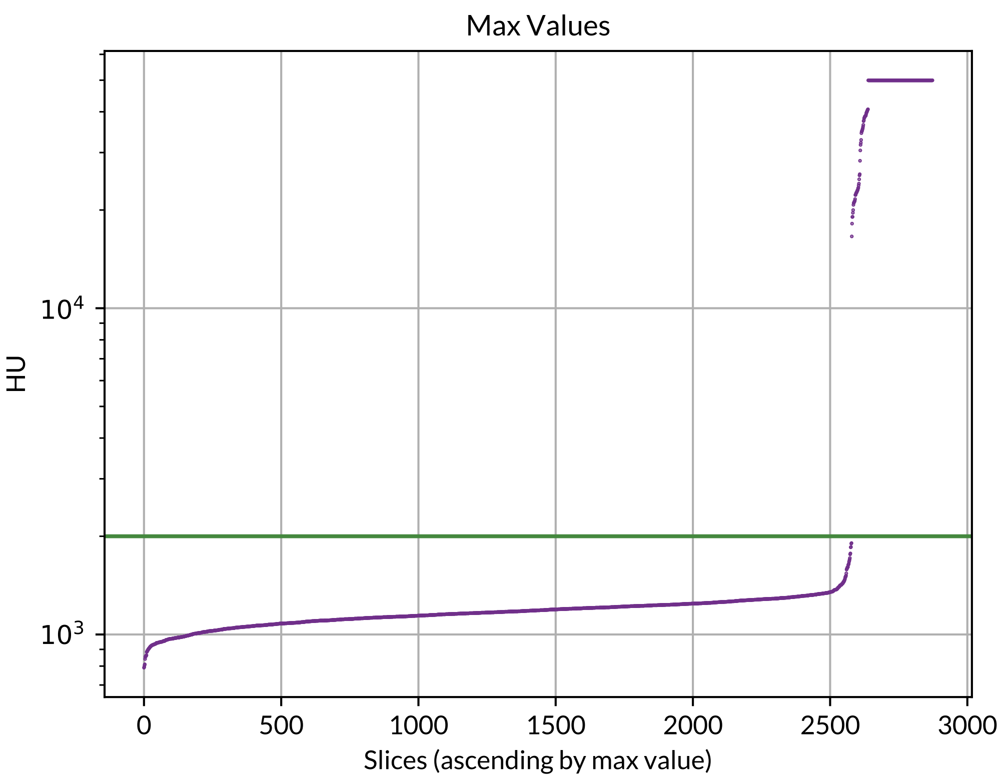

Figure 4.1: **A graph showing the maximum value for every slice** within the training database sorted in ascending order of maximum value. Superimposed in orange is the maximum threshold value.

4.3 Replication and Verification

One of the key objectives of this dissertation was to replicate the v16pUNet1.1C model (Conze et al., 2021) which achieved the best result for Task 2 of the CHAOS Challenge (Kavur et al., 2021). Deeper VGG models were also implemented by Conze et al. (2021), such as v19pUNet1.1, which uses a VGG19 encoder. However, for liver segmentation, the deeper models did not lead to performance gains. Therefore, these architecture versions were not replicated. Training parameters as stated by Conze et al. (2021) are listed in Table 4.1.

The replication of v16pUNet1.1C was split into three steps: first the implementation of v16UNetC, then applying pre-training on ImageNet to achieve v16pUNetC and the last step was to cascade the architecture to achieve the v16pUNet1.1C. With the exception of the normalization step suggested by PyTorch[2] when using pre-trained models, the de-

[2]https://pytorch.org/vision/0.8/models.html

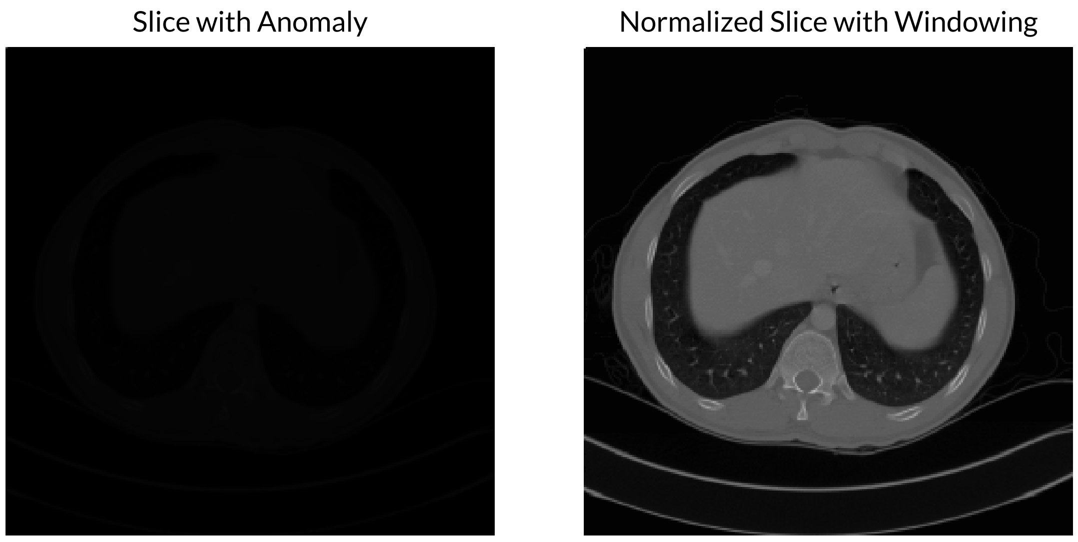

Figure 4.2: **Before and after range restricted normalization** - Left: Image normalized using its min and max values, and Right: the same slice after normalization using -1024 HU and 2000 HU as min max values.

Table 4.1: Implementation Parameters (Conze et al., 2021)

Parameter	Value/Type
Number of Augmentations per Slice	100
Augmentation Type	Random scaling, rotation, shearing and shifting
Number of Training Epochs	6
Batch Size	3
Optimizer	Adam
Learning Rate	10^{-5}
Loss Function	Fuzzy DICE Loss
Pre-Training Dataset	ImageNet(Russakovsky et al., 2015)
Post-Processing	Largest connected segmented area
IDE	Keras
GPU	Nvidia GeForce 1080 Ti

tails of each of these steps are included in Chapter 3. The normalization parameters as suggested by PyTorch are as follows: mean equal to [0.485, 0.456, 0.406] and standard deviation equal to [0.229, 0.224, 0.225].

4.3.1 Ambiguities

There were a few parameters of which the setup values were not clear, namely the transformation values for augmenting the database, the loss function, and the training percentage (and validation percentage). Starting with the former issue, little literature clearly state the exact properties of the applied transformations. In their paper, Furtado (2021a) state that they applied “random translations up to 10 pixels, random rotations up to 10 degrees, shearing up to 10 pixels and scaling up to 10% up and down as well”. Thus, except for the scaling transformation (unless stated otherwise), these values were used throughout all the training sessions. Another transformation which was added was the elastic transformation, as it yields “biologically plausible images” (Çiçek et al., 2016; Eaton-Rosen et al., 2018; Isensee et al., 2020; Kavur et al., 2021; Ronneberger et al., 2015; Zhang et al., 2021).

Moving on to discuss the loss function, Conze et al. (2021) mention the *fuzzy DICE Loss* in their work, but no detail or reference is given about the implementation of such a function. Furthermore, no literature could be found on how to go about implementing *fuzzy DICE Loss*. Thus, the loss function was replaced, first by a Binary Cross Entropy (BCE) loss function, and later by a variation on the DICE loss function. Both loss functions provide very good performance for CT liver segmentation, but the DICE loss function should perform slightly better (Furtado, 2021a). The choice of an optimal loss function is a critical process, as fundamentally, it measures the quality[3] of the output of the implementation (Furtado, 2021a). If a loss function fails to reveal the problems in the model’s output, then the model cannot be optimized correctly (Furtado, 2021a; Li et al., 2021). The smooth DICE loss function is very similar to the standard DICE loss but contains an additional smooth parameter to not only prevent dividing by zero, but to also act as a generalization parameter. The smooth parameter is generally set to 1 (RNA, 2021). The equation for the smooth DICE loss can be found below, where (1) represents the smooth parameter:

$$1 - \frac{2|(\boldsymbol{S} \cap \boldsymbol{G})| + (1)}{|\boldsymbol{S}| + |\boldsymbol{G}| + (1)}$$

[3]It is widely known that using accuracy as a loss function gives great importance to the background pixels, which are a large portion of the image (Furtado, 2021b; Li et al., 2021). This problem can be solved by choosing an appropriate loss function (Furtado, 2021b; Li et al., 2021).

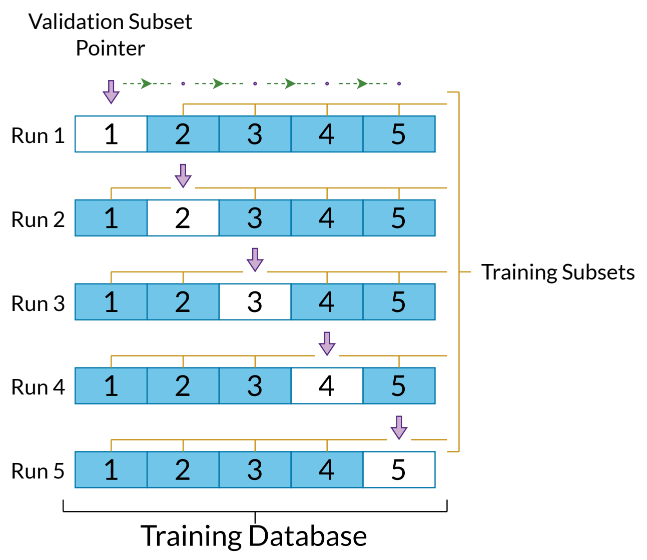

Figure 4.3: **Cross-Validation Diagram** for 20% validation subset, five runs, indicating the validation subset using a purple pointer.

There is no right or wrong answer when choosing which data is going to be used for training and which data is going to be used for validation. The ideal and best approach is cross-validation. Cross-validation is a process where the data is split into a given number of subsets. A single subset is assigned to the validation set and the remaining subsets are assigned to the training set. The architecture is then trained and validated using the chosen configuration of subsets. After completing the training process, the validation metrics are saved. The validation subset is then merged with the training subset whilst extracting another validation subset from the training subset, systematically. This process is repeated until all the subsets have been assigned to both the training and validation sets. The results from each training run are then compared, and the variance is calculated to check how robust the implementation is to a particular database. Figure 4.3 shows an example diagram of cross-validation using subsets of 20% of the training database. Although cross-validation is the ideal method, it is also a very lengthy process, as it requires the architecture to be trained multiple times. On average, each training session took around three days. Furthermore, other limitations exist, such as resource availability to train and validate the models and challenge limitations on the number of submissions per day. Due to these limitations, it is therefore more common practice to split the training database into two subsets, 80% training and 20% validation, at random (Furtado, 2021a,b; Goodfellow et al., 2016; Wen et al., 2021), at the cost of not knowing the expected variance.

Table 4.2: Implementation Parameters (this work)

Parameter	Value/Type
Number of Augmentations per Slice	100
Training % (Validation %)	80% (20%)
Augmentation Library	Albumentations
Augmentation Type	Random translation 0.02% $\approx$ 10 pixels, rotation $\pm 10^\circ$, shearing $\pm 10^\circ$ and elastic transformation $\sigma = 10$ $\alpha = 10$
Number of Training Epochs	6
Batch Size	3
Optimizer	Adam
Learning Rate	10^{-5}
Loss Function	Smooth DICE Loss
Pre-Training Dataset	ImageNet(Russakovsky et al., 2015)
Post-Processing	Three-Dimensional Largest-Connected-Component Filter
IDE	Pytorch
GPU/s	Nvidia GeForce 3060 Ti 8GB and Nvidia GeForce 3060 12GB

4.3.2 Implementation and Parameters

Due to the nature of the way the augmentations are applied during training, a clarification note is necessary. Typically, augmentations are applied independently of whether the input image belongs to a particular group or set. However, in this particular case, the images being augmented are CT slices which have a specific sequence. It was therefore decided that the transformations applied for data augmentation would be kept uniform for each dataset. This means that the transformations were applied by first generating a random seed per dataset, per augmentation number and this seed was then applied (and re-used) for all the slices of the particular dataset for the particular augmentation number. This means that if the dataset were to be visualized in three dimensions, the visualization would show a homogeneous and organic-like volume of the liver, avoiding any discontinuities between slices. The parameters used to replicate the results achieved by Conze et al. (2021) are listed in Table 4.2. Training was carried out using the Networks Laboratory computers provided by the Department of Communications and Computer Engineering on computers having the GPUs listed in Table 4.2, while testing was carried out using the personal computer on an Nvidia GeForce 1070 8GB. Other computer specifications have little effect on the training and testing performance given that the processes are being run on the GPUs.

4.4 Experimentation

4.4.1 Performance Tuning

Following the Performance Tuning Guide[4], one of the optimizations that PyTorch recommend, is to turn off the bias for convolutional layers which are immediately followed by a batch normalization operation. They state that removing the convolution bias should have no effect on the output result. They also state that the pre-trained models which they offer, also use this principle, as it frees up memory, reduces computation time and the model should theoretically achieve the same performance.

4.4.2 Architecture Discrepancies

In their work, Conze et al. (2021) presented several diagrams, one of which shows the architectures of their implementations for UNet, v19UNet and v19pUNet. The diagrams, however, differ from the original source for both UNet and VGG19 implementations. These discrepancies will be discussed in the following sections.

UNet discrepancy

In the Background Chapter, reference was made to the original UNet diagram which was presented in Figure 3.4. It is understood that in their diagram, Conze et al. (2021) presented the UNet Architecture, however there are several discrepancies from the original. The number of feature channels are all different; in the original (Ronneberger et al., 2015), they follow the order presented in Figure 3.4 of [64, 128, 256, 512] and 1024 for the bottleneck. Conze et al. (2021) present the channels as follows: [32, 64, 128, 256] and 256 for the bottleneck. Another discrepancy is the implementation of the bottleneck for UNet. Whereas in the original architecture the bottleneck doubles in number of features, in the diagram shown by Conze et al. (2021), the bottleneck maintains the same number of features as the previous convolution operation. This may have been implemented as such to mimic the architectures presented by Simonyan and Zisserman (2014). To clarify, throughout this dissertation, when a reference is made to UNet, reference is being made to the original architecture of Ronneberger et al. (2015) as presented in Figure 3.4, whereas when referring to the architecture presented by Conze et al. (2021), the nomenclature changes to UNetC.

[4]https://pytorch.org/tutorials/recipes/recipes/tuning_guide.html

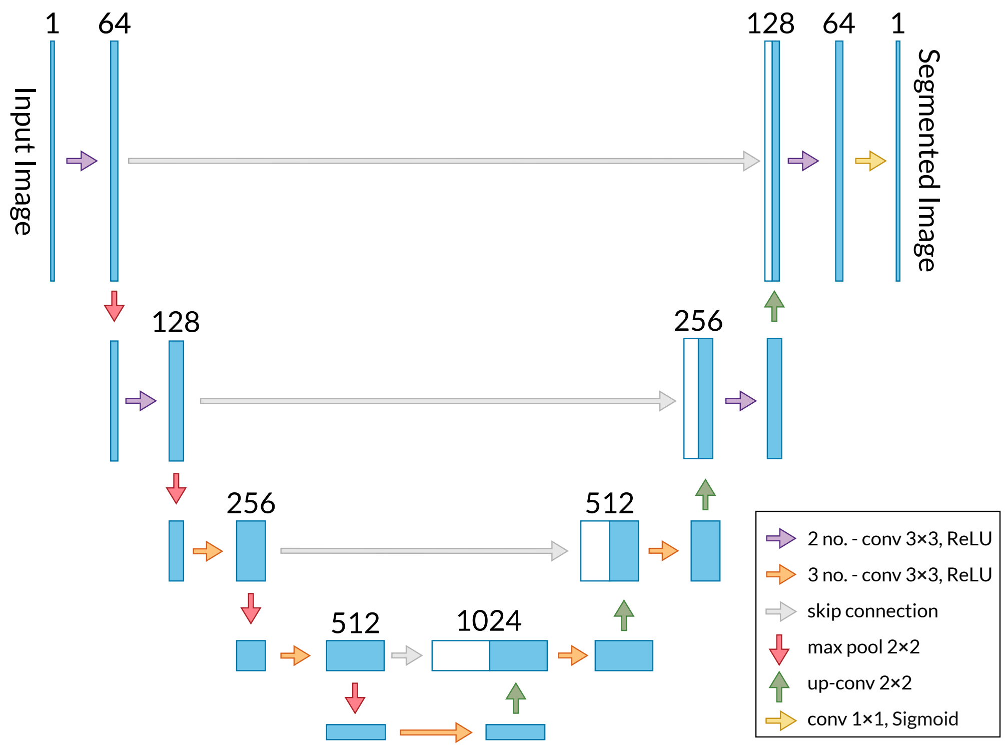

Figure 4.4: VGG16UNetD Architecture Diagram

VGG19-UNet discrepancy

In their diagram, Conze et al. (2021) presented their implementation of the v19UNet and v19pUNet. Although these two architectures were not implemented during the course of this dissertation, these diagrams were used as reference when implementing their VGG16 counterparts. Consider the VGG19 architecture as presented by Simonyan and Zisserman (2014). Using the following representation [number of convolutions $\times$ number of features, ...], VGG19 consists of the following sets of convolutions: [2×64, 2×128, 4×256, 4×512, 4×512]. However, the diagram presented by Conze et al. (2021) shows that the bottleneck layers were reduced by 1 convolution to [3×512]. Translating this interpretation to VGG16, one can end up with two versions of architectures: [2×64, 2×128, 3×256, 3×512, 3×512] which will be referred to as v16UNetD and any architectures or models which are a derivative of v16UNetD will be denoted by the letter 'D' (for example v16pUNetD); and [2×64, 2×128, 3×256, 3×512, 2×512] v16UNetC (v16pUNetC). The diagram for the latter architecture was presented in Figure 3.3, whereas the diagram for v16UNetD can be seen in Figure 4.4.

4.4.3 Training

As mentioned in Section 4.3, Conze et al. (2021) trained their architecture for six epochs. It is therefore being assumed that Conze et al. (2021) first applied the transformations on the database (100 per slice) and then fed the augmented database to the architecture and trained for six epochs. There is no mention of whether different approaches were taken for training the architecture, thus two different methods were applied separately to determine which is better.

Validation Shuffle

To make the most of training an architecture using a small database, the validation shuffle method was tested to check whether this would improve performance. The validation shuffle method is very simple in its implementation. Initially, the dataset directory numbers are shuffled to generate a random sequence of the datasets. This sequence is then split into two subsets, 16 datasets for training and 4 datasets for validation, approximating the 80% and 20% respectively, as discussed in Section 4.3.1. This process can be repeated after a set number of epochs to achieve updated training and validation datasets subsets. In theory, this should expose the architecture to all the possible datasets, however, this may also cause the model to decrease its generalization capabilities. Nonetheless, the implementations were tested to verify this training method.

Augmentation Sequence

During the course of this dissertation, two methods were tested on how to go about feeding the model with the augmented datasets. The first method is what is considered closest to the method applied by Conze et al. (2021). This involves augmenting the database at once with the required number of augmentations per slice then feeding all the data (in batches) to the model. Validation only takes place after the whole database and the augmentations are fed to the model. For example, consider the CHAOS database, containing 2,874 slices, then splitting the training and validation datasets using 80% training, 20% validation (2,299 slices[5]) and applying 10 augmentations for each slice on the training sets, totalling to 22,990 slices. After the latter number of slices have been fed to the model, the validation starts. After the validation, the model has trained for one epoch. This method shall be referred to as All-At-Once (AAO) or the 6-epoch method.

Another method of feeding the model can be applied by training the model for six epochs at a time per augmentation. Referring to the previous example, this means that at

[5]Note that this value is only indicative. The program splits the database based on whole datasets.

every epoch, the model is trained only on 2,299 slices and then validated. After training on the same slices for six epochs, the augmentation transformations are applied on the original database for the training slices only. These transformed slices are then fed to the model, which is trained for another six epochs. This process is repeated for the number of augmentations required. This method shall be referred to as One-At-a-Time (OAT). It is expected that OAT is more time-consuming, given that the validation step is repeated multiple times, depending on the number of augmentations required.

4.4.4 Normalization Adjustment

Taking the windowing filter, explained in Section 4.2.1, one step further and applying the same principles as medical professionals, the thresholds can be adjusted such that windowing is carried out over the set of HU values that fall within the ground truth region of the DICOM slice. This restrics the HU values to the required values only. These values were found by first applying the ground truth to each corresponding DICOM slice and then reading the minimum and maximum values. After carrying out this process for all the training and testing databases, the minimum threshold of -700 HU and the maximum threshold of 1,600 HU were chosen. The range was reduced from 3,024 HU to 2,300 HU. The windowing process is being applied through a normalization function, hence this threshold tuning is being called Normalization Adjustment.

4.4.5 Cascaded Framework Experiments

Apart from different training configurations, experimentation was carried out on the cascaded framework model. The experiments can be categorized in five parts, experimentation with: the connections between the two models, the connections from one model to the other at the shallow levels and at the deep levels, different learning rates for the two architectures in the same run, and different model sizes. Each of the mentioned parts will be explored in more detail below.

Connections Between the Two Models

When using a cascaded framework, an inherent connection is required between the first and second models. In their paper, Conze et al. (2021) show the connection between the two models as a stack of the input image, posterior probability of the first model and another copy of the input image, but no details are given about how this configuration was decided. Figure 4.5 shows three alternative approaches to these connections. Figure 4.5 (a) shows a similar configuration to the one used by Conze et al. (2021) changing only the configuration of the stack, where instead of having a stack with duplicate input image and a single copy of the posterior probabilities of the first model, the stack contains two copies of the posterior probabilities of the first model and a single copy of the input. This configuration should give greater importance to the posterior probabilities of the first model rather than the original input. The model will be referred to as $[v,\ x,\ v]$ Context, referring to the posterior probabilities as v and the input slice as x.

Figure 4.5 (b) and (c) show a completely different approach. Inspired by residual connections, the posterior probabilities are added to the input image and then fed to the second model. In (b), Model 1's posterior probabilities are added to the input image. The result is then triplicated and fed to the pre-trained model, Model 2. This architecture will be referred to as v16pUNet1S1D, the 'S' instead of '.' signifying a combined but Single source. In (c), Model 1 is configured to output three posterior probabilities (channels). Each one of these is then added to a copy of the input image and are then fed to Model 2. This network will be referred to as v16pUNet1T1D, where T signifies the Three posterior probabilities.

Connections at Shallow and Deep Levels

Taking inspiration from both UNet and the cascade framework, three approaches were implemented to share context between the two networks. Two of the methods take into context shallow connections whilst the other approach focuses on the deep connections.

Figure 4.6 and Figure 4.7 show the second part of the architecture only within a cascade framework (Model 2). The feature maps are copied from Model 1, transferred from Model 1 to Model 2, and concatenated to the feature maps which the encoder of Model 2 generates, keeping the same depth during all steps. Figure 4.6 shows two implementations in a single figure, where the only difference between the two is the number of feature maps, one having 64 and the other 128, at the penultimate convolutional layer. Following Model 2, the encoder part of the model for both the shallow and deep connection re-configuration remains unchanged. This cannot be changed since the model being used is pre-trained and, therefore, the layers are fixed.

For the shallow connection implementations, the bottleneck and the two deepest layers of the model decoder also remain unchanged. The part where the model differs from v16pUNet1.1D is at the three shallow connections of the decoder. Retaining the same spirit as UNet, Model 2 takes context from the encoder part of the same model as well as the encoder part of Model 1. Thus, by having the additional connections, instead of doubling the number of feature maps, these are tripled, giving the network more context from Model 1. A similar line of thought was followed when implementing the variation of

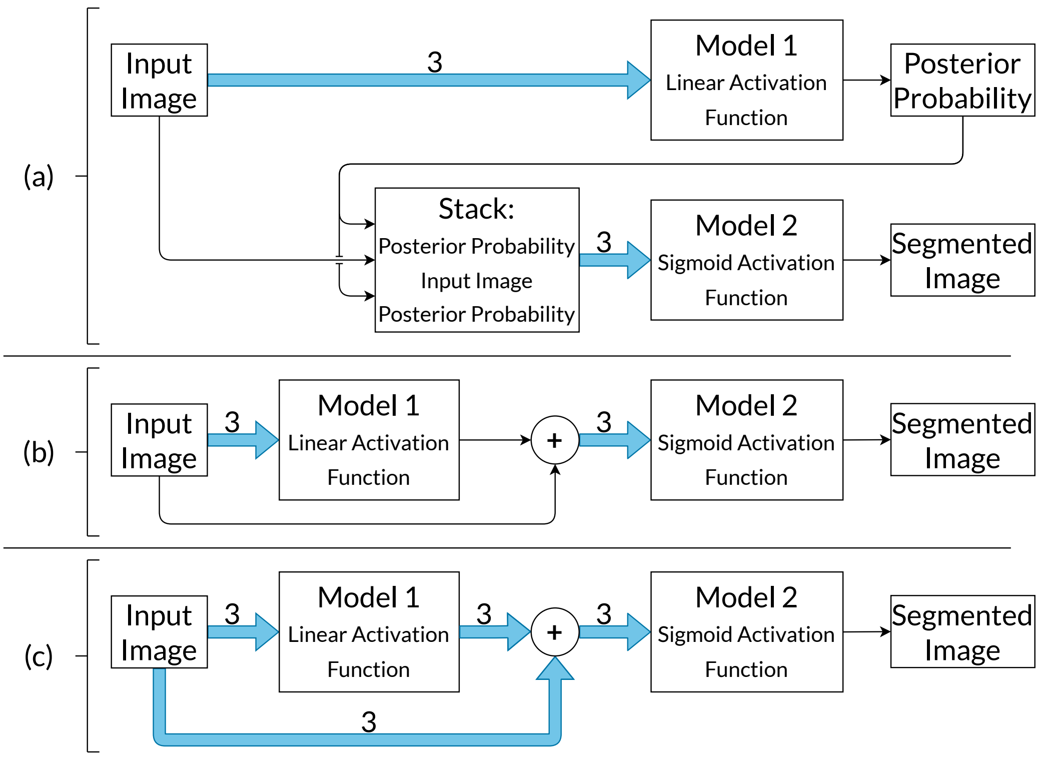

Figure 4.5: **Alternative configurations of connection between the first and second models of the cascade framework**, where (a) shows a variation in the stack configuration by having duplicate posterior probabilities instead of the standard duplicate input image, (b) shows a configuration where the posterior probability of the first model is added to the input image and then triplicated, and in (c) the first model is configured to output three channels and the input is added to each channel.

keeping additional feature maps at the penultimate layer. Tests were carried out to verify whether adding more features would result in better discrimination.

The approach towards giving more context to the deeper levels of the network, also follows the same concept as that of the shallow connections. As seen in Figure 4.7, the deepest three layers, these being the bottleneck and the two layers that follow in the decoder part of the Model 2, are given context from the encoder parts of Model 1.

Varying Learning Rates

The cascade framework inherently contains at least two cascaded models, the learning rates of which can be altered separately from one another. Let LR1 be the learning rate of the first model, Model 1 and LR2 the learning rate for the second model, Model 2. LR2 was kept constant at 1×10^{-5} whilst changing LR1 and two implementations were run, one having LR1 = 1.25 $\times$ LR2 and the other having LR1 = 0.75 $\times$ LR2.

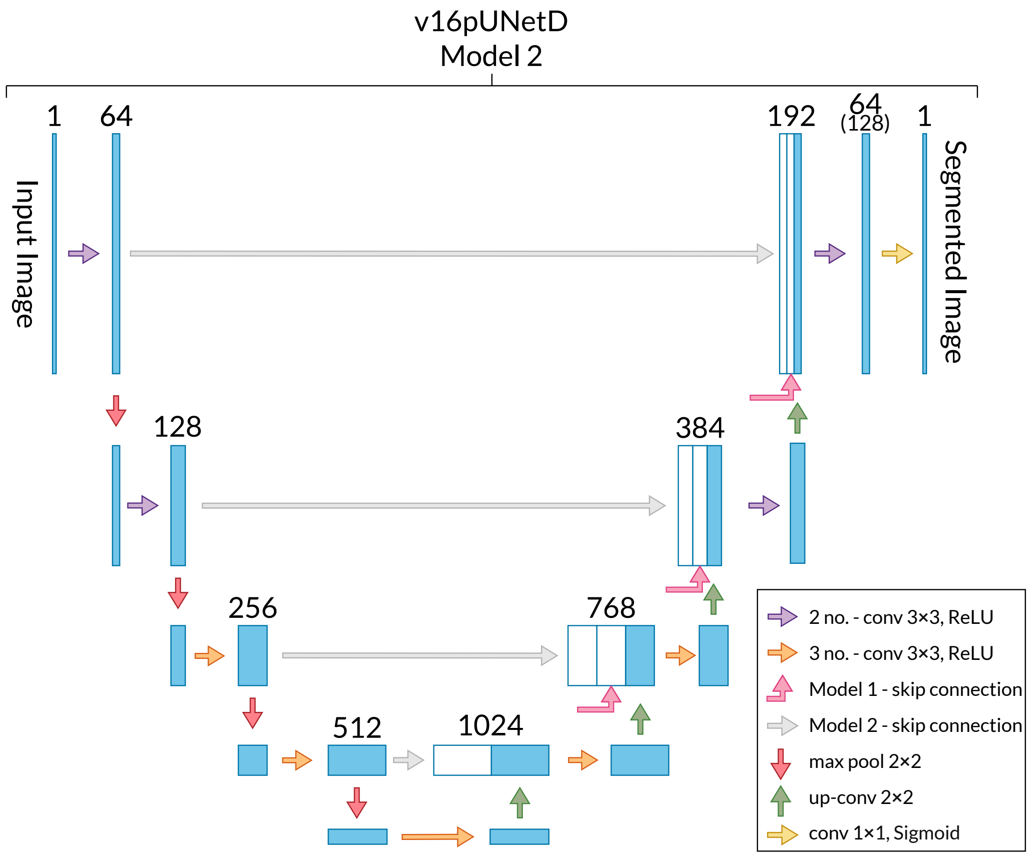

Figure 4.6: **Two configurations of the second model within the cascade framework**, Model 2, where the pink connections are context (skip) connections from the first model, Model 1. The difference between the two configurations is at the last 2 no. convolution. One configuration compresses the 192 feature maps to 64 and the other configuration compresses the 192 feature maps to 128 before the final convolution layer.

Adjusting Model Sizes

So far, when discussing the cascade framework, there has been an underlying assumption that the architectures for the two models were the same. Additionally, as discussed in previous sections, the first model is used to produce posterior probabilities which are used as context for the second model. Given the pre-text that the first model is only used to give context to the second model, the question arises as to whether the complexity and depth of the first model needs to be the same as that of the second. An architecture with a shallower first model was implemented, where Model 1 is an implementation of v16pUNetD with reduced depth such that it arrives at a bottleneck after three levels of depth as seen in Figure 4.8. Model 2 follows the architecture of v16pUNetD with shallow connections and 64 feature maps at the penultimate layer, as seen in Figure 4.6.

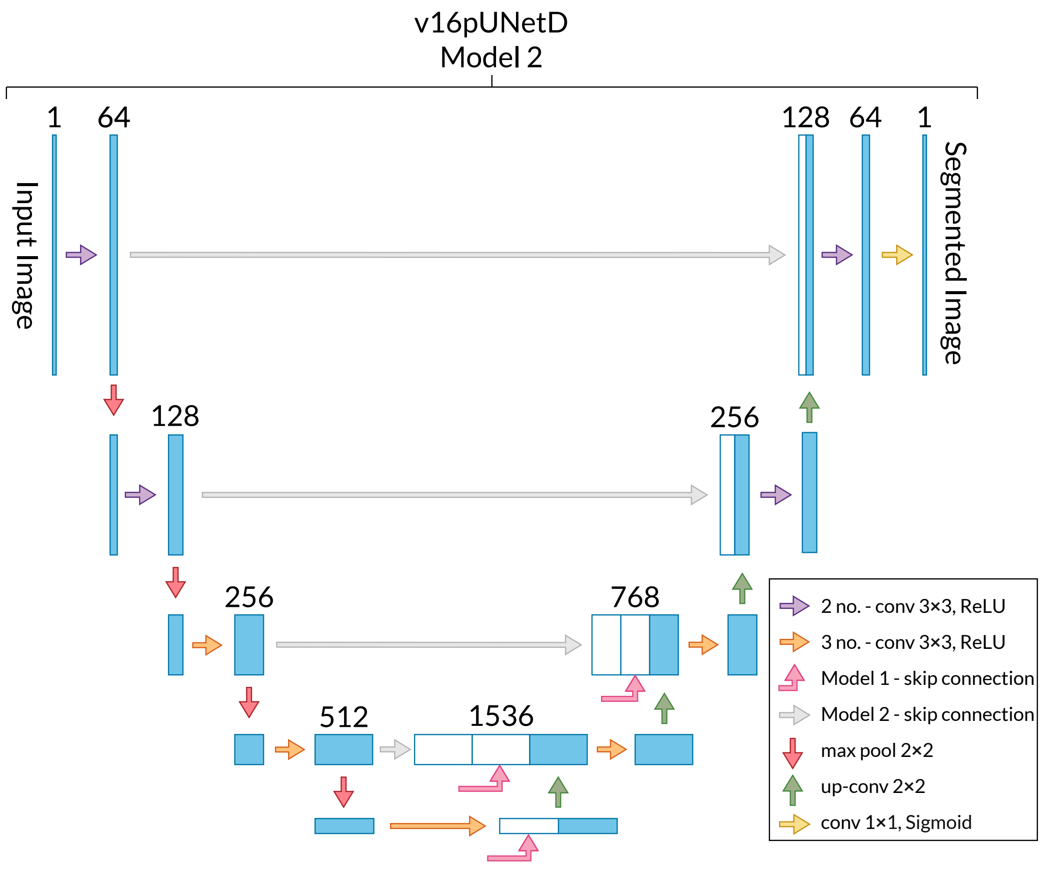

Figure 4.7: **Context connections** from Model 1 to Model 2 within a cascade framework at deep levels of the two models.

4.5 Conclusion

This chapter presented the methods applied and the reasoning behind the experiments carried out during the course of this dissertation. The next chapter presents the results of these experiments.

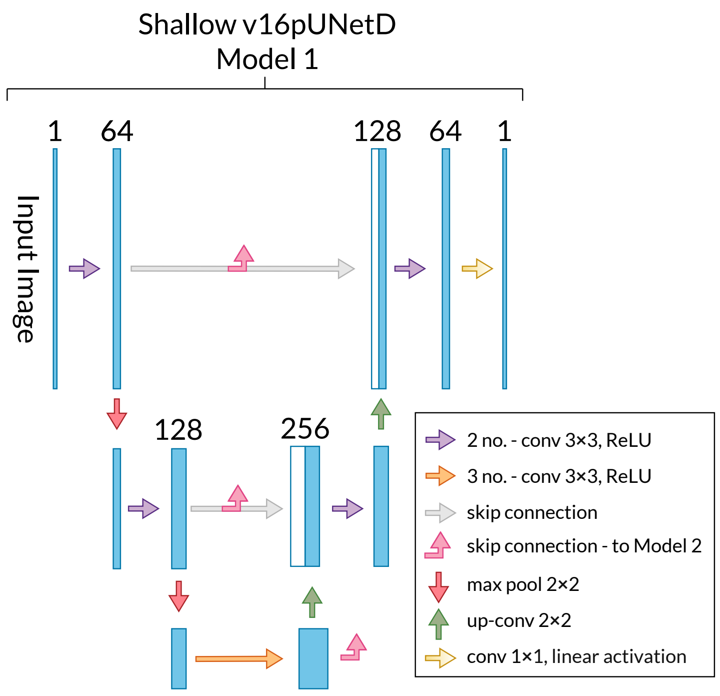

Figure 4.8: **Shallow VGG16UNetD model**, configured to be used as Model 1, showing context connections to be fed to the encoder of Model 2. Refer to Figure 4.6 for Model 2 configuration.

5 Results

5.1 Introduction

In this chapter, the results obtained from the experiments which were carried out were tabulated as well as visualized using the box plot format.

5.2 Submission Process

The results being presented in this chapter are all results which were submitted to the CHAOS Challenge. That is, after training the model on the training database, the model is then put into an evaluation state and the test database is fed to the model. As stated in Section 3.2.1, the ground truth for the test database is not disclosed to the public to avoid over-optimization. The model outputs the segmented datasets using the requirements specified by the challenge. Prior to submission, Conze et al. (2021) used a largest connected segmented area filter as listed in Table 4.1. Similar to Conze et al. (2021)'s approach, in this work, each of the test dataset segmentations are passed through a three-dimensional largest-connected-component filter having 26-way connectivity. This can be applied given the premise that the liver is a single volume. It was observed that the mean score after applying the filter rises by approximately 10% overall. Therefore, all results presented in this chapter will be the results achieved after applying the filter. Another requirement for submitting to the challenge is a sheet stating the important details of the implementation. After uploading the submission files[1], the metrics and scores are automatically calculated by the challenge and these are published on the challenge website[2].

[1]Segmented outputs can be submitted at:
https://chaos.grand-challenge.org/evaluation/challenge/submissions/create/
[2]https://chaos.grand-challenge.org/evaluation/challenge/leaderboard/

Table 5.1: **Replication Results** - results within this table are from the v16pUNet1.1C Model. The standard result, labelled v16pUNet1.1C, uses the parameters stated in Table 4.2. The BCE Loss variation uses the same parameters as standard other than the loss function and scaling transformation result uses the same parameters as standard but includes the scaling transformation when augmenting the database. The results presented in this table are a summary of the full results, which can be found in Table A.2

Variation	Mean Score	DICE Score	RAVD Score	ASSD Score	MSSD Score
Standard	85.49	97.80	79.12	94.72	70.35
BCE Loss	83.87	97.87	72.88	94.97	69.77
Scaling	84.92	97.55	75.85	94.13	72.16

5.3 Replication and Verification of Results

The objective of the results being presented in this section was to verify the results achieved by Conze et al. (2021). Due to the ambiguities which were mentioned in Section 4.3.1, three results are being presented to make sure that the selected loss function and transformations applied, as listed in Table 4.2, are satisfactory.

The results achieved are summarized in Figure 5.1 and Table 5.1. Figure 5.1 shows three box plots, one for the implementation of v16pUNet1.1C using the standard training parameters (listed in Table 4.2), another using the standard training parameters, but with a BCE loss function instead of DICE loss function and the third result also uses the standard training parameters but, additionally, includes a scaling transformation to the transformation list. The box plot shows the minimum and maximum values in the form of whiskers, bottom whisker being the minimum and top whisker being the maximum and the lower and upper quartiles are shown by the lower and upper bounds of the box respectively. The box plot also shows the median values for each plot as a dashed, red line, the mean marked with $\times$ and the mean score achieved by Conze et al. (2021) as a solid, black line.

The mean score achieved by Conze et al. (2021) was of 85.53, which is very close to the mean achieved by the replication model of 85.49, observed in Table 5.1. Changing the loss function from a smooth DICE loss function to a BCE loss functions, the mean score decreases to 83.87. A slight decrease in the mean score is also observed when including the scaling transformation in the transformation list. The RAVD score shows a similar trend to the mean score. Minute changes can be observed in the DICE score and the ASSD score. The MSSD score shows a different trend, where an increase can be observed after the application of the scaling transformation.

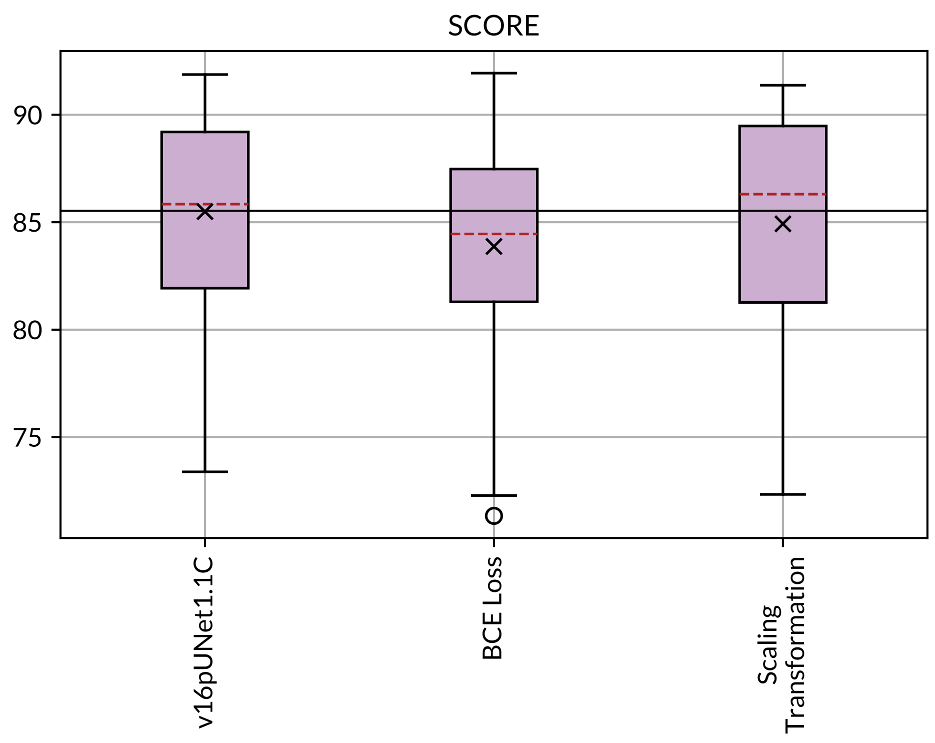

Figure 5.1: **Replication results** for the v16pUNet1.1C Model plotted in box plot format, where the median values for each plot as a dashed, red line, the mean marked with × and the mean score achieved by Conze et al. (2021) as a solid, black line. The standard result, labelled v16pUNet1.1C, uses the parameters stated in Table 4.2. The BCE Loss variation uses the same parameters as standard other than the loss function and scaling transformation result uses the same parameters as standard but includes the scaling transformation when augmenting the database. The plots for the metrics' scores can be found, which can be found in Figure A.1.

5.4 Experimental Results

5.4.1 Comparison of the Modified Model

A comparison between the model proposed by Conze et al. (2021), the v16pUNet1.1C, and the model which originated from the discrepancy stated in Section 4.4.2, called v16pUNet1.1D, was carried out to test which model has the better performance.

The results achieved are summarized in Figure 5.2 and Table 5.2. Figure 5.2 shows six box plots, comparing implementations of v16pUNet1.1C and v16pUNet1.1D using the standard training parameters (listed in Table 4.2), using OAT training, and OAT training with shuffled validation (discussed in Section 4.4.3). The figure includes the median values

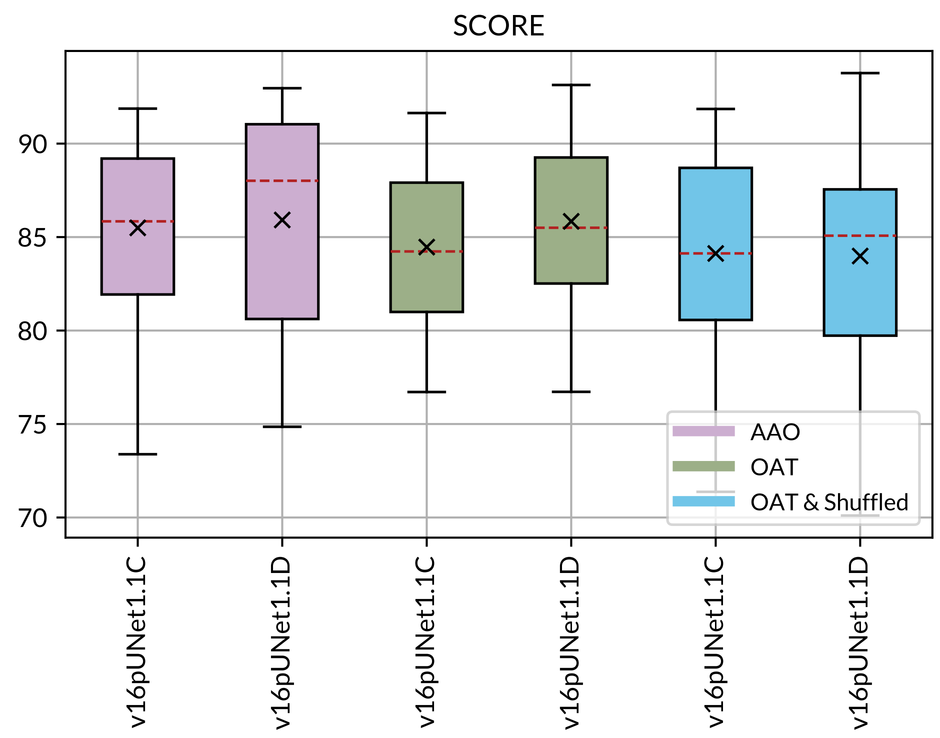

Figure 5.2: **A comparison of v16pUNet1.1C and v16pUNet1.1D models' results** in box plot format, where the median values for each plot as a dashed, red line, the mean marked with $\times$. The parameters common to both models are listed in Table 4.2. The legend splits the training method of the results by colour while the x-axis ticks show which result is associated to which model. The plots for the metrics' scores can be found, which can be found in Figure A.2.

for each plot shown as a dashed, red line and the mean marked with $\times$.

Comparing the mean score results of v16pUNet1.1C and v16pUNet1.1D models presented in Figure 5.2, v16pUNet1.1D achieves a better mean score for AAO and OAT training and slightly lower performance when using OAT with shuffle. Observing the median result, v16pUNet1.1D obtained better performance for all training categories. Observing the metric results in Table 5.2, the mean score, DICE Score, RAVD Score, and ASSD score show almost identical performance. The main discriminator was the MSSD.

5.4.2 Experiments on the Modified Model

Figure 5.3 and Table 5.3 present a summary of most of the experiments which were carried out on the v16pUNet1.1D model. In total, 13 variations were tested on the modified model. The standard implementation, as per previous sections, uses the parameters

Table 5.2: **Comparison Results** - results within this table are a direct comparison of the v16pUNet1.1C and the v16pUNet1.1D models under various training methods. The parameters common to both models are listed in Table 4.2. The results are distinguished by the first three columns: the model, the type of training, and whether the validation set was shuffled during training. The results presented in this table are a summary of the full results, which can be found in Table A.3.

Architecture	Training Type	Shuffled	Mean Score	DICE Score	RAVD Score	ASSD Score	MSSD Score
v16pUNet1.1C	AAO	No	85.49	97.80	79.12	94.72	70.35
v16pUNet1.1D	AAO	No	85.92	97.78	78.01	94.77	73.09
v16pUNet1.1C	OAT	No	84.46	97.67	79.90	94.39	65.90
v16pUNet1.1D	OAT	No	85.84	97.85	80.33	94.80	70.38
v16pUNet1.1C	OAT	Yes	84.12	97.77	73.66	94.69	70.37
v16pUNet1.1D	OAT	Yes	83.99	97.77	73.53	94.63	70.01

listed in Table 4.2. The results of the standard implementation are slightly different, as the results being shown are from another run. All the variations on the architecture are explained in detail in Section 4.4. Figure 5.3 groups the variations into five categories by colour: minor changes with the heading Standard are represented in purple, the green colour represents the variations in the connections between the two models in a cascaded framework, the blue colour represents the variations of the within model connections, changes in Learning Rate (LR) are represented by the yellow colour, and the pink colour represents size adjustment models. Below each plot is a unique reference for that particular result. The metrics' score plots and the full variations' results table with standard deviation values can be found in Appendix A.2, titled Additional Results.

Observing the mean scores from Figure 5.3 and Table 5.3, there is an overall decrease in performance when applying any variation except the Normalization Adjustment result and v16pUNet1T1D. A similar trend can also be noted when observing the median results with the addition that all within connections achieve a better median for the mean score.

5.5 Conclusion

This chapter presented the results obtained from the experiments which were run during the course of this dissertation. The next chapter will discuss in detail these results.

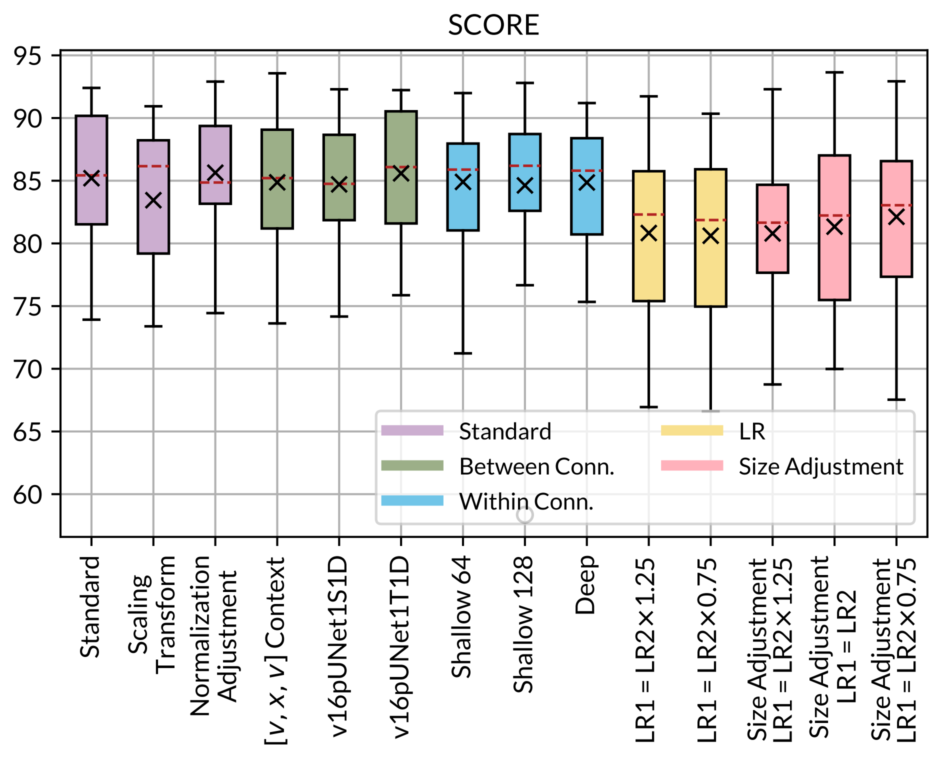

Figure 5.3: **v16pUNet1.1D Variations' Results** plotted in box plot format, where the median values for each plot as a dashed, red line and the mean marked with ×. The standard result, labelled v16pUNet1.1D, uses the parameters stated in Table 4.2. The variations applied to the architecture are detailed in Section 4.4. The plots for the metrics' scores can be found, which can be found in Figure A.3.

Table 5.3: **Experimentation Results** - results within this table are from the v16pUNet1.1D Model. The standard result uses parameters detailed in Table 4.2 and is a different run from the result presented in Table 5.2. The variations applied to the architecture are detailed in Section 4.4. The results presented in this table are a summary of the full results, which can be found in Table A.4.

Variation	Mean Score	DICE Score	RAVD Score	ASSD Score	MSSD Score
Standard	85.20	97.77	78.31	94.58	70.14
Scaling	83.44	97.63	69.80	94.44	71.92
Normalization Adjustment	85.64	97.78	80.03	94.59	70.16
$[v, x, v]$ Context	84.87	97.70	77.26	94.37	70.15
v16pUNet1S1D	84.71	97.74	77.86	94.20	69.03
v16pUNet1T1D	85.59	97.77	80.15	94.58	69.86
Shallow Connections - 64	84.91	97.77	79.18	94.53	68.15
Shallow Connections - 128	84.63	97.71	78.64	94.05	68.11
Deep Connections	84.86	97.83	74.32	94.90	72.39
LR1 = LR2 $\times$ 1.25	80.83	97.47	61.09	93.80	70.95
LR1 = LR2 $\times$ 0.75	80.61	97.50	64.02	93.70	67.23
Size Adjustment LR1 = LR2 $\times$ 1.25	80.80	97.61	63.48	94.12	67.98
Size Adjustment LR1 = LR2	81.34	97.56	65.43	94.08	68.28
Size Adjustment LR1 = LR2 $\times$ 0.75	82.12	97.60	67.24	94.24	69.41

6 Discussion

6.1 Introduction

In this chapter, the results presented in Chapter 5 are interpreted. This includes a comparison of the several implementations and architectures which were presented as well as an objective and subjective opinion on the performance achieved.

6.2 Replication and Verification Results

The first set of results in the previous chapter were presented to show that the goal of replicating the state-of-the-art model presented by Conze et al. (2021) was successfully replicated and that any ambiguities with the design of the architecture were settled. Three separate implementations are being presented in Figure 5.1 and Table 5.1: the v16pUNet1.1C which is being referred to as the standard for this set of results, the BCE Loss result where the architecture was trained using a BCE loss function instead of a smooth DICE, and the Scaling Transformation result, which should be an exact replica of what was presented by Conze et al. (2021).

Observing Figure 5.1 and comparing the standard result with the result achieved by Conze et al. (2021) (85.53, solid, black line), the mean score and median score achieved are almost exactly identical to the mean score achieved by Conze et al. (2021). Comparing to the Scaling Transformation result, which uses all the data transformations as per Conze et al. (2021), similar values are observed, although having a slightly lower mean score and a slightly higher median score. Quantifying the differences in mean scores from Table 5.1, the standard model achieved a result of 84.49, 0.04 less than that reported by Conze et al. (2021) and the Scaling Transformation achieved a mean score of 84.92, 0.61 less than that reported by the same. The difference observed is minute and can be attributed to different training and validation datasets. Observing the implementation of the BCE Loss, a decrease in mean score of 1.62 can be noted, very similar in magnitude to what was reported by Furtado (2021b).

Looking at the results achieved, and comparing them to the data which was provided by Conze et al. (2021), the replication and verification of their work was, overall, successful. The scaling transform, although included by Conze et al. (2021), was found to have very little effect on the mean score result and, if anything, it widened the interquartile range indicating an increase in variability of the mean scores of the individual testing datasets' results. This translates to lower precision, thus, the scaling transform was excluded from the implementations that followed, as reported in Section 4.3.1.

6.3 Comparison of Implementations

The results presented in Section 5.4.1 serve a two-fold purpose, to compare the modified model, the v16pUNet1.1D, with the v16pUNet1.1C model and to compare the different approaches towards training the architectures. To further clarify, the architecture as presented by Conze et al. (2021) is being referred to as v16pUNet1.1C.

6.3.1 Training Methods

Starting from the training methods, three methods were proposed in this dissertation: the AAO, the OAT, and the OAT with shuffled validation. The AAO, OAT methods as well as the validation shuffle were explained in Section 4.4.3. In Figure 5.2, these have been categorized using three colours, purple, green and blue respectively. In Table 5.2, the results have been sorted using the Training Type and Shuffled columns.

Analysing the mean score achieved by OAT with shuffled validation, for both models, the results are almost identical and lower than the other two training methods. Analysing the metrics' score, however, the results achieved are more consistent, meaning that this training method did not discriminate between the two models. Using OAT only as the training method yielded better results, v16pUNet1.1D showing an increase of almost 2 points. A similar trend, but to a lesser degree, was observed when switching to AAO. In terms of mean score points, taking the mean score of both models, AAO scored 85.71, OAT scored 85.15 and OAT with shuffled validation scored 84.06. Shifting the attention to the medians presented in Figure 5.2, a similar trend can be observed where AAO achieved the highest score and decreases when switching to OAT and OAT with shuffled validation.

6.3.2 Modified Model

As discussed in Section 4.4.2, investigating the discrepancies and departing from the work presented by Simonyan and Zisserman (2014), a modified architecture is being presented, called v16pUNet1.1D. Observing the mean score results achieved in Figure 5.2 across the three training methods, v16pUNet1.1D achieved a higher mean score when using both AAO and OAT, and a lower mean score when using OAT with shuffled validation. This particular run of v16pUNet1.1D also achieved the second-highest mean score of 85.92 for Task 2 of the CHAOS Challenge, with the highest score being 85.98 at the time of writing this dissertation. The performance advantage of using v16pUNet1.1D is even more clear when observing the median results, where v16pUNet1.1D achieved much higher median scores for every training method when compared to v16pUNet1.1C. This shows that adding the additional convolutional layer in the bottleneck has enough impact to increase the overall median score. Looking at the mean score values presented in Table 5.2 and taking the difference between the two models for each training method, a difference of 0.43, 1.38 and 0.13 can be observed for AAO, OAT, and OAT with shuffled validation respectively. When v16pUNet1.1D had a worse performance than v16pUNet1.1C, the difference in performance was much lower than when v16pUNet1.1D performed better than v16pUNet1.1C, that is, the significance of the performance difference when v16pUNet1.1D performed better is higher. Therefore, this architecture was chosen for further experimentation.

6.4 Experiments on Modified Model

The results presented in Figure 5.3 and Table 5.3 are the results of the experiments which were carried out on the modified model, the v16pUNet1.1D. As stated in Section 5.4.2 the results are categorized into five groups, which are an indication of the type of modification which was carried out. In the coming section, each group will be discussed individually. The standard result in Figure 5.3 and Table 5.3 is a result from a different run, and therefore has different results than the run presented in Figure 5.2 and Table 5.2.

6.4.1 Standard Category

The standard category contains three results: the standard, the scaling transform, and the normalization adjustment. Analysing the effect of the scaling transform on v16pUNet1.1D, a decrease in mean score of 1.76 was observed. This decrease in performance is similar to what was reported in Section 6.2. The median score on the other hand is higher, while overall interquartile results are both lower, thus, it was deemed that the scaling transform does not provide any significant performance increase. The Normalization Adjustment result, on the other hand, shows a higher mean score and a lower median score, while the lower quartile shows a higher result and a reduced range between quartiles. Due to the reduced quartile range, the mean score result was deemed to be significant, meaning that reducing the windowing range does improve performance.

6.4.2 Between Connections Category

The standard between model connection, as published by Conze et al. (2021), uses a $[x,\ v,\ x]$ connection, where v represents the posterior probabilities of Model 1 and x represents the slice or input. Three modifications were proposed in Section 4.4.5: $[v,\ x,\ v]$ Context, v16pUNet1S1D, and v16pUNet1T1D, the results of which were presented in Figure 5.3 and Table 5.3. All three results showed performance close to the standard result, with $[v,\ x,\ v]$ Context achieving a mean score of 84.87, v16pUNet1S1D with a mean score of 84.71, and v16pUNet1T1D with a mean score of 85.49. The median score followed the mean score very closely. Interpreting the results shown on Figure 5.3, the performance of $[v,\ x,\ v]$ Context looks very similar to the standard result. v16pUNet1S1D shows promise because, while the mean score and median score are slightly lower than the standard result, the interquartile range has been reduced by around 1.5 points. v16pUNet1T1D did just the opposite, that is, a higher mean score was achieved, while increasing the interquartile range. However, the interquartile range was increased from the top end and only very slightly. Further tests are required to confirm the repeatability of such results.

6.4.3 Within Connections Category

The within connections category was an experiment to test whether context from one model to the other, in a cascaded framework, provided any benefits. The initial attempt was to connect all encoder outputs from Model 1 to the decoder of Model 2, however, due to GPU memory limitations (an estimate of 20 GB was required), this could not be tested. Thus, three variations were tested based on a similar concept. These variations were described in Section 4.4.5 and were labelled as follows: Shallow 64, Shallow 128, and Deep. All models show good performance reducing the interquartile range by anywhere from 0.7 to 1.3 points. The mean score results are also close to the standard result, at worst achieving a difference in performance of 0.57 points and although 0.57 is not small, the median results achieved are all above the median score of the standard model.

From what was observed, adding context between the two models when using a cascade framework, does seem to provide a significant improvement, especially in the interquartile range. Although Shallow 128 has the lowest mean score, overall, it has obtained the best performance when considering the median and interquartile results.

6.4.4 Learning Rate Category

The Learning Rate category consists of two results labelled: LR1 = LR2$\times$1.25 and LR1 = LR2$\times$0.75. The methodology to producing the LR results is explained in Section 4.4.5, where the LR of Model 2 is being kept constant at 1×10^{-5} and a ratio is being applied to obtain the LR of Model 1. The model performance digressed when changing the LR of Model 1 in either direction. The results shown in Figure 5.3 are an indication that the model is very susceptible to changes made to the learning rate parameter. Overall, a performance decrease of 4.48 points in the mean score can be observed. An interesting observation can be made by looking at the metrics' score results in Table 5.3. Increasing the LR of Model 1 increased the MSSD score while reducing the RAVD score while. The opposite can be observed when reducing the LR of Model 1. Further, testing would be required to verify the claims being made in this paragraph, however, the overall result obtained was still a decrease in performance.

6.4.5 Size Adjustment Category

The size adjustment category, explained in Section 4.4.5, was trained using the standard training parameters (tabulated in Table 4.2), except for the learning rate. Three different learning rates were applied, using the same configuration explained in Sub-section Varying Learning Rates, and the nomenclature used is self-explanatory. Figure 5.3 shows that the spread of the mean scores for the Size Adjustment models is very large, and all models performed worse than the Standard model. The median scores, although slightly higher than the mean scores, follow the same trend. Size Adjustment LR1 = LR2$\times$1.25 did manage to achieve a narrow interquartile range, however, this in itself was not good enough to be considered as a step forward. Table 5.3 shows that the mean score of the three models is 81.42. Although the results achieved were not an improvement when compared to their counterparts, there is one aspect in which they performed excellently. The average training time of the three models was around two days, a huge improvement when compared to the overall average of three days. This in itself does not outweigh the performance benefits gained by the additional layers and training time, but it may be something which can be applied if time restrictions are more important than the 3.78 points gained when using the Standard model.

6.5 Analysis

After analysing and discussing the results achieved, and giving a subjective overview of all models, there is still a degree of uncertainty as to which model performed best. Considering only the mean score and median score, the best model is the standard v16pUNet1.1D model with a mean score of 85.92 points and a median score of 88.01 points, achieving second place in Task 2 of the CHAOS Challenge. However, considering only the interquartile range, the best model would be the Shallow 128 with an interquartile range of 6.13 points. Experiments can also be carried out using a combination of the variations presented above. However, based on the results obtained so far, and considering the variance observed, the best model is the Normalization Adjustment model, achieving a narrow interquartile range of 6.20 points and a good mean score of 85.64, achieving 13th place in Task 2 of the CHAOS Challenge.

6.6 Conclusion

In this chapter, the performance of the several architectures which were implemented was discussed through the interpretation of their results. The next chapter will present the conclusions and any future work which may be carried out.

7 Conclusion

7.1 Introduction

This chapter presents the outcomes of this work, its limitations, and recommendations for future work.

7.2 Summary of Conclusions

The main objective of this dissertation was to replicate a state-of-the-art implementation that is able to segment healthy liver using CT scans. This goal was successfully achieved as discussed in Section 6.2. Going beyond the baseline objective, by applying a modification on the state-of-the-art architecture, a model improvement was achieved, obtaining second place in the CHAOS Challenge for Task 2. Further variations were carried out on the modified architecture, including ones which were not published in this work, however, these did not perform any better than the modified architecture, v16pUNet1.1D.

7.3 Limitations

The field of image segmentation is vast, with thousands of different architectures and even more implementations. Medical image segmentation, although a subset of image segmentation, also shares the same characteristics. Exploring all architectures and implementations is next to impossible, given the time limitations. This imposed a limitation on the number of variations which could be explored. There is also an issue of resource availability and energy use, for example, it is not viable to test every architecture using all the possible combinations of batches, epochs and learning rates, as this would be a huge waste of energy and resources, especially when interpolation can be carried out to predict the outcome. Furthermore, some architectures, although they may achieve state-of-the-art performance, are simply not feasible to run, either because of resource limitations (such as using too much GPU Memory) or the time and power consumption required versus performance gained is simply not feasible. The GPU Memory limitation was encountered and overcome several times throughout this work, by tweaking the architecture to limit the use of GPU Memory to the available resources.

The database being used also has its own set of limitations. As a general rule for any deep-learning architecture, the greater the pool of images, the more generalized the result and, usually, the better the performance achieved, assuming that the images being presented are of the same nature. The CHAOS Challenge provided 20 training datasets (2,874 Slices) and 20 testing datasets (3,533 Slices) and although these might seem to be a lot, this is actually quite small when compared to, for example, ImageNet which has approximately 14 million images. Nonetheless, training on a small database is a challenge in itself, and thus, was considered as part of the task. Another limitation of the database was found when reading the information off the DICOM images. By reading the information contained within the DICOM files it was found that the peak tube voltage for the training database was kept constant at 120 kVP, while for the testing database, for dataset 39, the peak tube voltage changes from 120 kVP to 100 kVP. This creates inconsistencies in the HU values which the model was not trained on, making it more susceptible to errors.

Another limitation, tied to time restrictions, was the use of a single challenge to verify the performance of the models. One way ensure that the implementation being used is not optimized on a particular database, is to apply it to several databases, which might have different metrics, and test its performance. This would therefore serve as a test to verify that the implementation is generalized enough to work on different databases.

7.4 Recommendations for Future Research

There are several future endeavours which can be tackled or experimented upon. This list presents some of the thoughts which were written down during the experimentation phase, but there was simply not enough time to put them into practice.

Starting from presentation of the input data, only two pre-processing techniques were applied (excluding data transformations for augmentation purposes), namely windowing and normalization. However, there is a plethora of image pre-processing techniques and combinations which can be applied prior to feeding the data to the implementation, which may lead to improved performance. Similar to pre-processing techniques, post-processing techniques can also be applied to the output of the implementations to improve the score achieved. Examples of such workflows were presented by Furtado (2021b).

The recommendations mentioned in the previous paragraph all had to do with data manipulation, however, other experiments can be carried out on the model architecture. The encoder part of the model can be changed from VGG16 to other encoders such as ResNet (Furtado, 2021b; Lachinov, 2019), ResNeXt etc., and have their performance compared. Although the smooth DICE loss function has been found to be the better option, other loss functions can be implemented to better represent the problems within the segmented outputs (Furtado, 2021b; Li et al., 2021). The in-between convolution activation function being used by this work's architecture is ReLU. Other types of activation functions such as Leaky ReLU or PReLU (Ernst et al., 2019) can be applied. Although currently limited by the GPU memory, training can be carried out on more powerful hardware to test whether increasing the batch size, increases the generalization ability of the architecture.

7.5 Conclusion

In this chapter, the outcomes of this work, its limitations, and recommendations for future work were discussed. The main objective of this dissertation has been successfully reached. Beyond the main objective several other experiments were carried out to test various different implementations.

References

Alves, J. H., Neto, P. M. M., and de Oliveira, L. F. Extracting lungs from ct images using fully convolutional networks, 2018.

Chaurasia, A. and Culurciello, E. Linknet: Exploiting encoder representations for efficient semantic segmentation. 2017. doi: 10.48550/ARXIV.1707.03718.

Çiçek, Ö., Abdulkadir, A., Lienkamp, S. S., Brox, T., and Ronneberger, O. 3d u-net: Learning dense volumetric segmentation from sparse annotation, 2016.

Conze, P.-H., Kavur, A. E., Gall, E. C.-L., Gezer, N. S., Meur, Y. L., Selver, M. A., and Rousseau, F. Abdominal multi-organ segmentation with cascaded convolutional and adversarial deep networks. *Artificial Intelligence in Medicine*, 117:102109, July 2021. doi: 10.1016/j.artmed.2021.102109.

Eaton-Rosen, Z., Bragman, F., Ourselin, S., and Cardoso, M. J. Improving data augmentation for medical image segmentation. 1st Conference on Medical Imaging with Deep Learning (MIDL 2018), June 2018. URL `https://openreview.net/pdf?id=rkBBChjiG`.

Ernst, P., Soumick Chatterjee, Speck, O., and Nürnberger, A. Chaos challenge - team ovgu memorial. 2019. doi: 10.13140/RG.2.2.27960.49928.

Fukushima, K. Cognitron: A self-organizing multilayered neural network. *Biological Cybernetics*, 20(3-4):121–136, 1975. doi: 10.1007/bf00342633.

Furtado, P. Testing segmentation popular loss and variations in three multiclass medical imaging problems. *Journal of Imaging*, 7(2):16, January 2021a. doi: 10.3390/jimaging7020016.

Furtado, P. Loss, post-processing and standard architecture improvements of liver deep learning segmentation from computed tomography and magnetic resonance. *Informatics in Medicine Unlocked*, 24:100585, 2021b. doi: 10.1016/j.imu.2021.100585.

Goodfellow, I., Bengio, Y., and Courville, A. *Deep Learning*. The MIT Press, 2016. ISBN 0262035618.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition, 2015.

Hu, J., Shen, L., Albanie, S., Sun, G., and Wu, E. Squeeze-and-excitation networks, 2017.

Huang, G., Liu, Z., Maaten, L. V. D., and Weinberger, K. Q. Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, July 2017. doi: 10.1109/cvpr.2017.243.

Iglovikov, V. and Shvets, A. Ternausnet: U-net with vgg11 encoder pre-trained on imagenet for image segmentation, 2018.

Isensee, F., Jaeger, P. F., Kohl, S. A. A., Petersen, J., and Maier-Hein, K. H. nnU-net: a self-configuring method for deep learning-based biomedical image segmentation. *Nature Methods*, 18(2):203–211, December 2020. doi: 10.1038/s41592-020-01008-z.

Kamiri, J. and Mariga, G. Research methods in machine learning: A content analysis. *International Journal of Computer and Information Technology(2279-0764)*, 10(2), March 2021. doi: 10.24203/ijcit.v10i2.79.

Kavur, A. E., Gezer, N. S., Barış, M., Aslan, S., Conze, P.-H., Groza, V., Pham, D. D., Chatterjee, S., Ernst, P., Özkan, S., Baydar, B., Lachinov, D., Han, S., Pauli, J., Isensee, F., Perkonigg, M., Sathish, R., Rajan, R., Sheet, D., Dovletov, G., Speck, O., Nürnberger, A., Maier-Hein, K. H., Akar, G. B., Ünal, G., Dicle, O., and Selver, M. A. CHAOS challenge - combined (CT-MR) healthy abdominal organ segmentation. *Medical Image Analysis*, 69:101950, April 2021. doi: 10.1016/j.media.2020.101950.

Kavur, A. E., Selver, M. A., Dicle, O., Barış, M., and Gezer, N. S. Chaos - combined (ct-mr) healthy abdominal organ segmentation challenge data, 2019.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In Pereira, F., Burges, C., Bottou, L., and Weinberger, K., editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

Lachinov, D. A. Segmentation of thoracic organs using pixel shuffle. In *SegTHOR@ ISBI*, 2019. URL http://refhub.elsevier.com/S1361-8415(20)30314-5/sbref0036.

Li, J., Cheng, L., Xia, T., Ni, H., and Li, J. Multi-scale fusion u-net for the segmentation of breast lesions. *IEEE Access*, 9:137125–137139, 2021. doi: 10.1109/access.2021.3117578.

Long, J., Shelhamer, E., and Darrell, T. Fully convolutional networks for semantic segmentation. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2015. doi: 10.1109/cvpr.2015.7298965.

Mahesh, M. The essential physics of medical imaging, third edition. *Medical Physics*, 40(7):077301, June 2013. doi: 10.1118/1.4811156.

Maier-Hein, L., Eisenmann, M., Reinke, A., Onogur, S., Stankovic, M., Scholz, P., Arbel, T., Bogunovic, H., Bradley, A. P., Carass, A., Feldmann, C., Frangi, A. F., Full, P. M., van Ginneken, B., Hanbury, A., Honauer, K., Kozubek, M., Landman, B. A., März, K., Maier, O., Maier-Hein, K., Menze, B. H., Müller, H., Neher, P. F., Niessen, W., Rajpoot, N., Sharp, G. C., Sirinukunwattana, K., Speidel, S., Stock, C., Stoyanov, D., Taha, A. A., van der Sommen, F., Wang, C.-W., Weber, M.-A., Zheng, G., Jannin, P., and Kopp-Schneider, A. Why rankings of biomedical image analysis competitions should be interpreted with care. *Nature Communications*, 9(1), December 2018. doi: 10.1038/s41467-018-07619-7.

Marieb, E. N. and Keller, S. M. *Essentials of Human Anatomy & Physiology, eBook, Global Edition*. Pearson Deutschland, 2017. URL https://elibrary.pearson.de/book/99.150005/9781292216201.

Matcha, A. C. N. A 2021 guide to semantic segmentation, 2021. URL https://nanonets.com/blog/semantic-image-segmentation-2020/.

Oktay, O., Schlemper, J., Folgoc, L. L., Lee, M., Heinrich, M., Misawa, K., Mori, K., McDonagh, S., Hammerla, N. Y., Kainz, B., Glocker, B., and Rueckert, D. Attention u-net: Learning where to look for the pancreas, 2018.

RNA. Loss function library - keras & pytorch, July 2021. URL https://www.kaggle.com/code/bigironsphere/loss-function-library-keras-pytorch/notebook.

Ronneberger, O., Fischer, P., and Brox, T. U-net: Convolutional networks for biomedical image segmentation. In *Lecture Notes in Computer Science*, pages 234–241. Springer International Publishing, 2015. doi: 10.1007/978-3-319-24574-4_28.

Rumgay, H., Ferlay, J., de Martel, C., Georges, D., Ibrahim, A. S., Zheng, R., Wei, W., Lemmens, V. E., and Soerjomataram, I. Global, regional and national burden of primary liver cancer by subtype. *European Journal of Cancer*, 161:108–118, jan 2022. doi: 10.1016/j.ejca.2021.11.023.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, April 2015. doi: 10.1007/s11263-015-0816-y.

Shi, W., Caballero, J., Huszár, F., Totz, J., Aitken, A. P., Bishop, R., Rueckert, D., and Wang, Z. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network, 2016.

Shuang, Y. and Wang, Z. A novel approach for automatic and robust segmentation of the 3d liver in computed tomography images. *Measurement Science and Technology*, 31(11):115701, September 2020. doi: 10.1088/1361-6501/ab95db.

Shvets, A. A., Rakhlin, A., Kalinin, A. A., and Iglovikov, V. I. Automatic instrument segmentation in robot-assisted surgery using deep learning. In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, December 2018. doi: 10.1109/icmla.2018.00100.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition, 2014.

Tan, M., Mao, X., Wu, F., and Kong, D. Accurate liver segmentation using 3d cnns with high level shape constraints, 2020. URL https://rumc-gcorg-p-public.s3.amazonaws.com/evaluation-supplementary/178/7c179371-455e-4c2b-aa7b-a0b6a485d1b5/segnet-crf.pdf.

Vladimir, G., Johan, B., and Michael, A. Dualtail-net for liver segmentation on abdominal ct images, April 2019. URL https://mediantechnologies.com/wp-content/uploads/2019/04/20190408_ISBI2019_MedianTechnologies.pdf.

Wen, Y., Chen, L., Deng, Y., and Zhou, C. Rethinking pre-training on medical imaging. *Journal of Visual Communication and Image Representation*, 78:103145, July 2021. doi: 10.1016/j.jvcir.2021.103145.

Yan, Y., Conze, P.-H., Decenciere, E., Lamard, M., Quellec, G., Cochener, B., and Coatrieux, G. Cascaded multi-scale convolutional encoder-decoders for breast mass segmentation in high-resolution mammograms. In *2019 41st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*. IEEE, July 2019. doi: 10.1109/embc.2019.8857167.

Zhang, X., Xie, W., Huang, C., Wang, Y., Zhang, Y., Chen, X., and Tian, Q. Self-supervised tumor segmentation through layer decomposition, 2021.

Zhu, S., Du, B., Zhang, L., and Li, X. Attention-based multiscale residual adaptation network for cross-scene classification. *IEEE Transactions on Geoscience and Remote Sensing*, 60:1–15, 2022. doi: 10.1109/tgrs.2021.3056624.

A . Appendices

A.1 Database Technical Information

Appendix A.1 includes a table showing the CT Scan parameters and the conversion parameters that were contained within the DICOM files. Table A.1 shows the parameters for the Training database and Testing databases.

Table A.1: **Training and Testing Databases** - Tube Current Over Set Amount of Time, Tube Voltage Peak, Slope & Intercept

	Dataset No.	Tube Current (mAs)	Tube Voltage Peak (kVP)	Slope	Intercept
Training	1	200	120	1	-1200
	2	240	120	1	-1000
	5	260	120	1	-1000
	6	180	120	1	-1200
	8	260	120	1	-1000
	10	300	120	1	-1000
	14	200	120	1	-1000
	16	260	120	1	-1000
	18	260	120	1	-1000
	19	240	120	1	-1000
	21	469	120	1	-1024
	22	300	120	1	-1000
	23	305	120	1	-1024
	24	296	120	1	-1024
	25	375	120	1	-1024
	26	300	120	1	-1000
	27	305	120	1	-1024
	28	300	120	1	-1000
	29	296	120	1	-1024
	30	287	120	1	-1024
Testing	3	220	120	1	-1000
	4	220	120	1	-1000
	7	240	120	1	-1000
	9	260	120	1	-1000
	11	240	120	1	-1000
	12	220	120	1	-1000
	13	240	120	1	-1000
	15	240	120	1	-1000
	17	260	120	1	-1000
	20	260	120	1	-1000
	31	296	120	1	-1024
	32	313	120	1	-1024
	33	305	120	1	-1024
	34	305	120	1	-1024
	35	313	120	1	-1024
	36	305	120	1	-1024
	37	391	120	1	-1024
	38	296	120	1	-1024
	39	227	100	1	0
	40	305	120	1	-1024

A.2 Additional Results

Appendix A.2 is split into two parts, the first part showing additional box plots related to the metrics' scores and the second part show additional results in table form. The figures are split into the four quadrants representing the metrics used by the CHAOS challenge: top left, DICE score; top right, RAVD score; bottom left, ASSD score; and bottom right, MSSD score. The tables are organized using columns to represent each of the individual metrics and giving information not only on the score but also on the actual metric values.

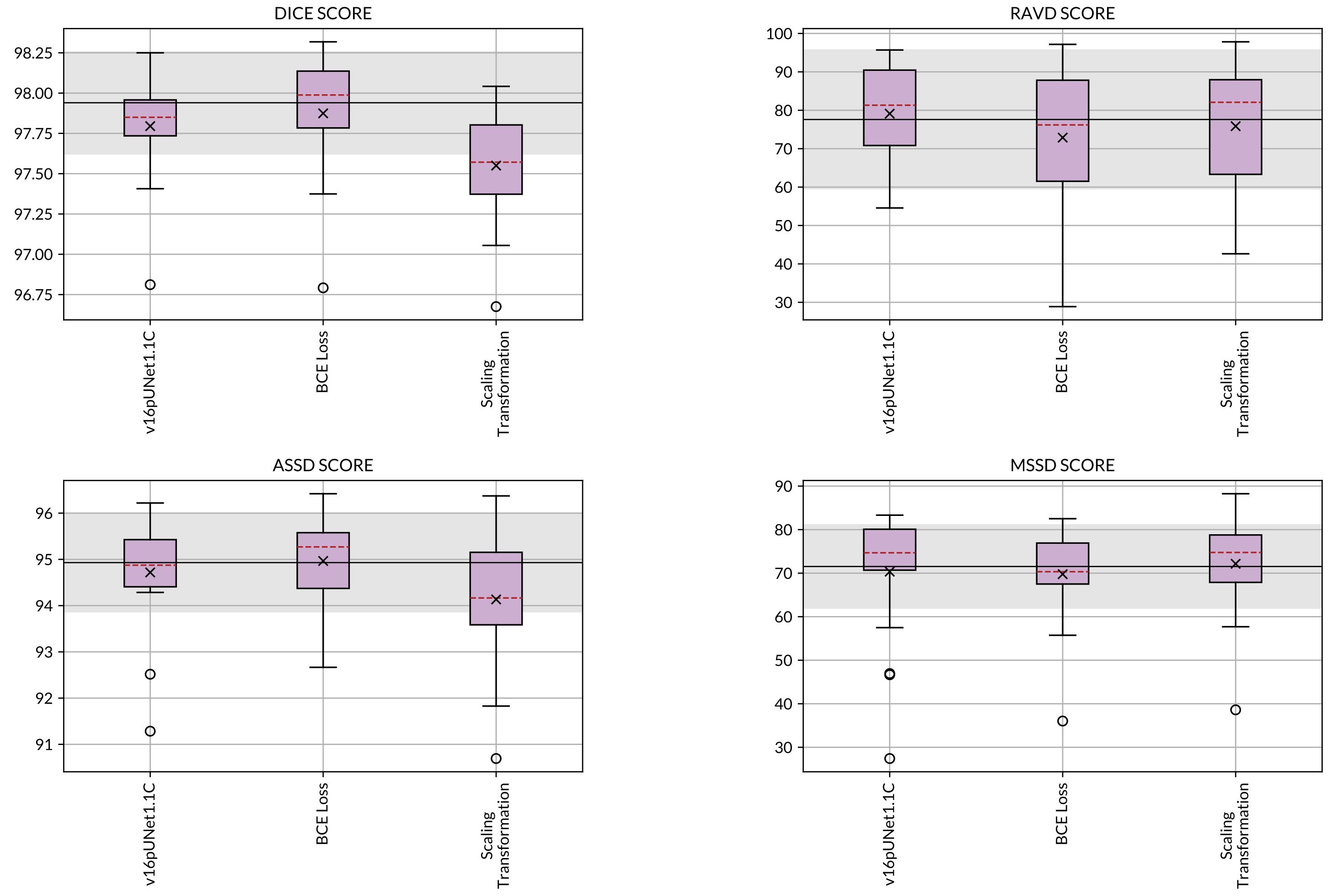

Figure A.1: **Metrics' Replication results** for the v16pUNet1.1C Model plotted in box plot format, where the median values for each plot as a dashed, red line, the mean marked with ×, the mean score and standard deviation achieved by Conze et al. (2021) as a solid, black line and grey shading respectively. The standard result, labelled v16pUNet1.1C, uses the parameters stated in Table 4.2. The BCE Loss variation uses the same parameters as standard other than the loss function and scaling transformation result uses the same parameters as standard but includes the scaling transformation when augmenting the database.

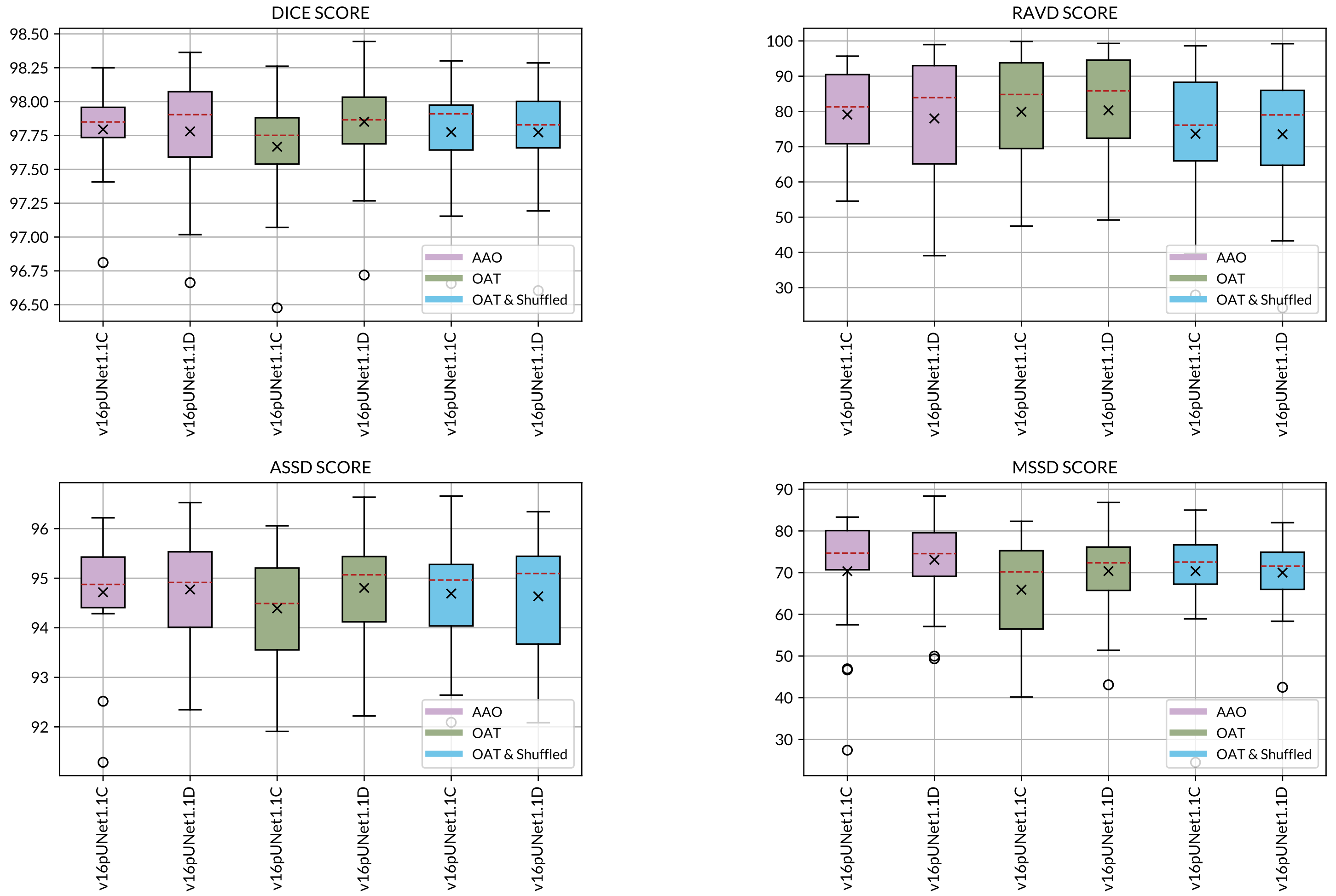

Figure A.2: **A comparison of v16pUNet1.1C and v16pUNet1.1D metrics' results** in box plot format, where the median values for each plot as a dashed, red line, the mean marked with $\times$. The parameters common to both models are listed in Table 4.2. The legend splits the training method of the results by colour while the x-axis ticks show which result is associated to which model.

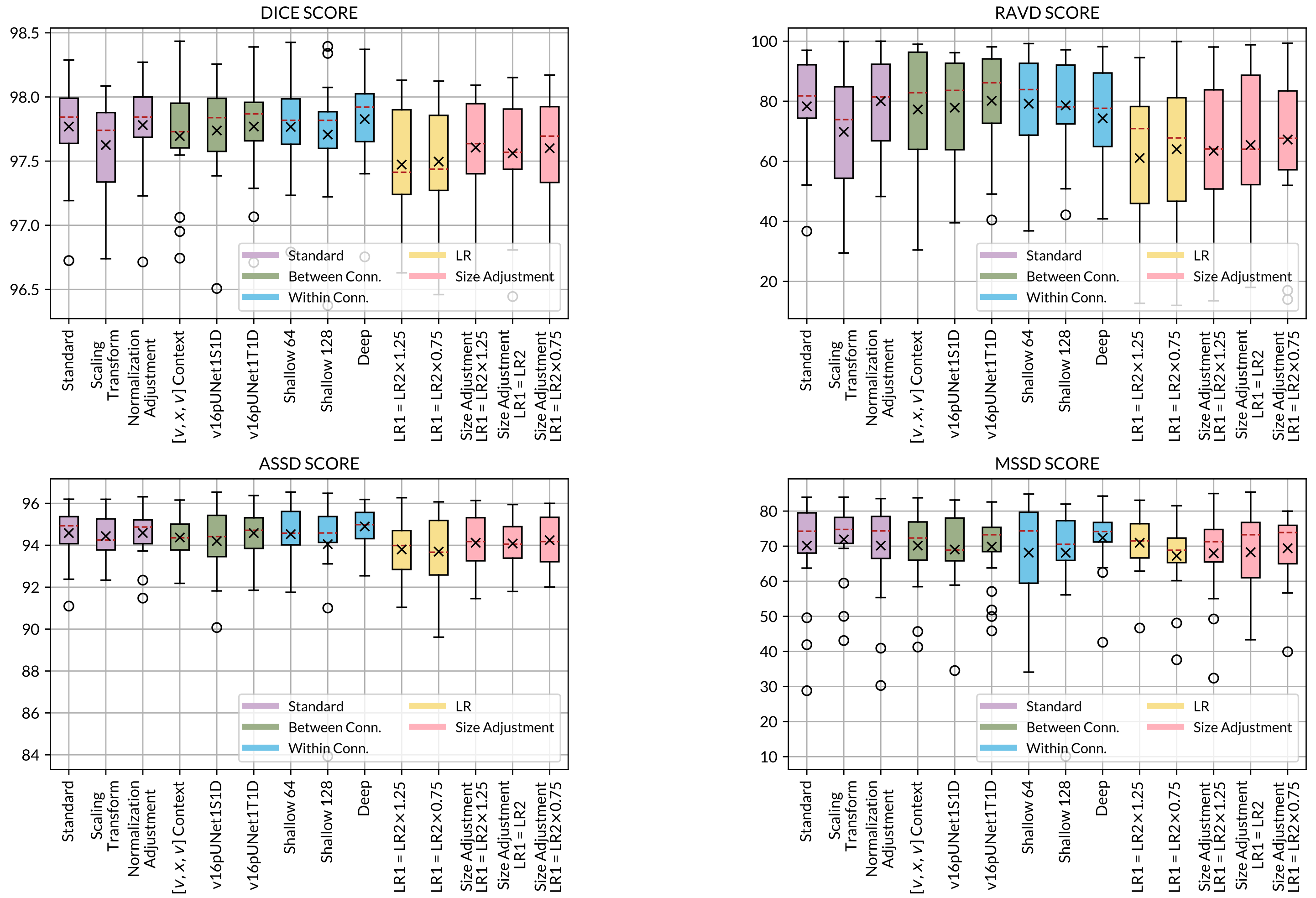

Figure A.3: **v16pUNet1.1D Variation Metrics' Results** plotted in box plot format, where the median values for each plot as a dashed, red line and the mean marked with ×. The standard result, labelled v16pUNet1.1D, uses the parameters stated in Table 4.2. The variations applied to the architecture are detailed in Section 4.4.

Table A.2: **Replication Results - Full** - results within this table are from the v16pUNet1.1C Model. The standard result, labelled v16pUNet1.1C, uses the parameters stated in Table 4.2. The BCE Loss variation uses the same parameters as standard other than the loss function and scaling transformation result uses the same parameters as standard but includes the scaling transformation when augmenting the database. The results presented in this table include both the actual values for the metrics, the score and the standard deviation of each.

Variation	Mean Score	DICE Score	RAVD (%)	RAVD Score	ASSD	ASSD Score	MSSD	MSSD Score
Standard	85.49 ± 5.06	97.80 ± 0.32	1.04 ± 0.67	79.12 ± 13.50	0.79 ± 0.17	94.72 ± 1.14	17.79 ± 8.83	70.35 ± 14.72
BCE Loss	83.87 ± 5.53	97.87 ± 0.37	1.36 ± 0.95	72.88 ± 19.07	0.76 ± 0.15	94.97 ± 1.00	18.14 ± 6.41	69.77 ± 10.69
Scaling	84.92 ± 5.63	97.55 ± 0.35	1.21 ± 0.82	75.85 ± 16.30	0.88 ± 0.21	94.13 ± 1.38	16.70 ± 6.45	72.16 ± 10.75

Table A.3: **Comparison Results - Full** - results within this table are a direct comparison of the v16pUNet1.1C and the v16pUNet1.1D models under various training methods. The parameters common to both models are listed in Table 4.2. The results are distinguished by the first three columns: the model, the type of training, and whether the validation set was shuffled during training. The results presented in this table are a summary of the full results, which can be found in Table A.3. The results presented in this table include both the actual values for the metrics, the score and the standard deviation of each.

Architecture	Training Type	Shuffled	Mean Score	DICE Score	RAVD (%)	RAVD Score	ASSD	ASSD Score	MSSD	MSSD Score
v16pUNet1.1C	AAO	No	85.49 ± 5.06	97.80 ± 0.32	1.04 ± 0.67	79.12 ± 13.50	0.79 ± 0.17	94.72 ± 1.14	17.79 ± 8.83	70.35 ± 14.72
v16pUNet1.1D	AAO	No	85.92 ± 6.07	97.78 ± 0.42	1.10 ± 0.88	78.01 ± 17.69	0.78 ± 0.16	94.77 ± 1.05	16.14 ± 6.43	73.09 ± 10.72
v16pUNet1.1C	OAT	No	84.46 ± 4.35	97.67 ± 0.42	1.01 ± 0.90	79.90 ± 17.97	0.84 ± 0.16	94.39 ± 1.08	20.46 ± 8.05	65.90 ± 13.42
v16pUNet1.1D	OAT	No	85.84 ± 4.43	97.85 ± 0.38	0.98 ± 0.81	80.33 ± 16.29	0.78 ± 0.16	94.80 ± 1.05	17.77 ± 6.71	70.38 ± 11.18
v16pUNet1.1C	OAT	Yes	84.12 ± 5.49	97.77 ± 0.40	1.32 ± 0.97	73.66 ± 19.41	0.80 ± 0.17	94.69 ± 1.15	17.78 ± 7.74	70.37 ± 12.90
v16pUNet1.1D	OAT	Yes	83.99 ± 5.73	97.77 ± 0.39	1.32 ± 1.00	73.53 ± 20.06	0.80 ± 0.20	94.63 ± 1.32	17.99 ± 5.60	70.01 ± 9.33

Table A.4: **Experimentation Results - Full** - results within this table are from the v16pUNet1.1D Model. The standard result uses parameters detailed in Table 4.2 and is a different run from the result presented in Table A.3. The variations applied to the architecture are detailed in Section 4.4. The results presented in this table include both the actual values for the metrics, the score and the standard deviation of each.

Variation	Mean Score	DICE Score	RAVD (%)	RAVD Score	ASSD	ASSD Score	MSSD	MSSD Score
Standard	85.20 $\pm$ 5.67	97.77 $\pm$ 0.36	1.08 $\pm$ 0.84	78.31 $\pm$ 16.86	0.81 $\pm$ 0.20	94.58 $\pm$ 1.30	17.92 $\pm$ 8.71	70.14 $\pm$ 14.52
Scaling	83.44 $\pm$ 5.66	97.63 $\pm$ 0.36	1.51 $\pm$ 1.06	69.80 $\pm$ 21.15	0.83 $\pm$ 0.16	94.44 $\pm$ 1.03	16.85 $\pm$ 6.09	71.92 $\pm$ 10.14
Normalization Adjustment	85.64 $\pm$ 4.87	97.78 $\pm$ 0.35	1.00 $\pm$ 0.73	80.03 $\pm$ 14.58	0.81 $\pm$ 0.18	94.59 $\pm$ 1.20	17.91 $\pm$ 8.24	70.16 $\pm$ 13.74
$[v, x, v]$ Context	84.87 $\pm$ 5.82	97.70 $\pm$ 0.40	1.14 $\pm$ 1.05	77.26 $\pm$ 21.03	0.84 $\pm$ 0.17	94.37 $\pm$ 1.11	17.91 $\pm$ 6.76	70.15 $\pm$ 11.26
v16pUNet1S1D	84.71 $\pm$ 5.24	97.74 $\pm$ 0.38	1.11 $\pm$ 0.86	77.86 $\pm$ 17.22	0.87 $\pm$ 0.24	94.20 $\pm$ 1.59	18.58 $\pm$ 6.43	69.03 $\pm$ 10.72
v16pUNet1T1D	85.59 $\pm$ 4.81	97.77 $\pm$ 0.40	0.99 $\pm$ 0.89	80.15 $\pm$ 17.77	0.81 $\pm$ 0.18	94.58 $\pm$ 1.22	18.09 $\pm$ 6.35	69.86 $\pm$ 10.58
Shallow Connections - 64	84.91 $\pm$ 5.22	97.77 $\pm$ 0.36	1.04 $\pm$ 0.84	79.18 $\pm$ 16.81	0.82 $\pm$ 0.19	94.53 $\pm$ 1.25	19.11 $\pm$ 8.59	68.15 $\pm$ 14.32
Shallow Connections - 128	84.63 $\pm$ 7.37	97.71 $\pm$ 0.44	1.07 $\pm$ 0.78	78.64 $\pm$ 15.57	0.89 $\pm$ 0.40	94.05 $\pm$ 2.65	19.14 $\pm$ 9.23	68.11 $\pm$ 15.38
Deep Connections	84.86 $\pm$ 4.85	97.83 $\pm$ 0.36	1.28 $\pm$ 0.93	74.32 $\pm$ 18.68	0.77 $\pm$ 0.14	94.90 $\pm$ 0.91	16.56 $\pm$ 5.19	72.39 $\pm$ 8.66
LR1 = LR2 $\times 1.25$	80.83 $\pm$ 6.63	97.47 $\pm$ 0.44	1.95 $\pm$ 1.18	61.09 $\pm$ 23.62	0.93 $\pm$ 0.19	93.80 $\pm$ 1.27	17.43 $\pm$ 4.91	70.95 $\pm$ 8.18
LR1 = LR2 $\times 0.75$	80.61 $\pm$ 6.89	97.50 $\pm$ 0.43	1.80 $\pm$ 1.20	64.02 $\pm$ 23.96	0.95 $\pm$ 0.25	93.70 $\pm$ 1.68	19.66 $\pm$ 5.92	67.23 $\pm$ 9.86
Size Adjustment LR1 = LR2 $\times$ 1.25	80.80 $\pm$ 6.68	97.61 $\pm$ 0.39	1.83 $\pm$ 1.24	63.48 $\pm$ 24.89	0.88 $\pm$ 0.20	94.12 $\pm$ 1.36	19.21 $\pm$ 7.14	67.98 $\pm$ 11.90
Size Adjustment LR1 = LR2	81.34 $\pm$ 7.09	97.56 $\pm$ 0.44	1.73 $\pm$ 1.21	65.43 $\pm$ 24.27	0.89 $\pm$ 0.17	94.08 $\pm$ 1.15	19.03 $\pm$ 7.10	68.28 $\pm$ 11.84
Size Adjustment LR1 = LR2 $\times$ 0.75	82.12 $\pm$ 6.56	97.60 $\pm$ 0.43	1.64 $\pm$ 1.15	67.24 $\pm$ 23.04	0.86 $\pm$ 0.18	94.24 $\pm$ 1.18	18.35 $\pm$ 6.01	69.41 $\pm$ 10.02