Final Year Project Report
Medical Image Segmentation using automatic adaptation and a CNN-Transformer hybrid

Nicholas Prowse - 28756401
npro0001@student.monash.edu
Supervisor: Mehrtash Harandi

October 23, 2021

MONASH University
Engineering

Department of Electrical and
Computer Systems Engineering

ECE4095 Final Year Project 2021 **Nicholas Prowse**

Medical Image Segmentation using automatic adaptation and a CNN-Transformer hybrid

Supervisor: Dr. Mehrtash Harandi

Overview

Using machine learning techniques, I have created a model that is able to accurately segment 3D medical images across a wide variety of tasks. The model is designed to automatically adapt to different datasets, allowing it to create highly accurate segmentations with no manual adjustment. This is achieved through a variety of preprocessing, post processing and data augmentation techniques which are adjusted based on the statistics of the given dataset. The design was tested on 11 different datasets, which represent a variety medical segmentation tasks from segmenting Brain Tumours and swelling to segmenting abdominal organs. The datasets are chosen to represent a variety of different challenges, such as class imbalances, small structures and small datasets.

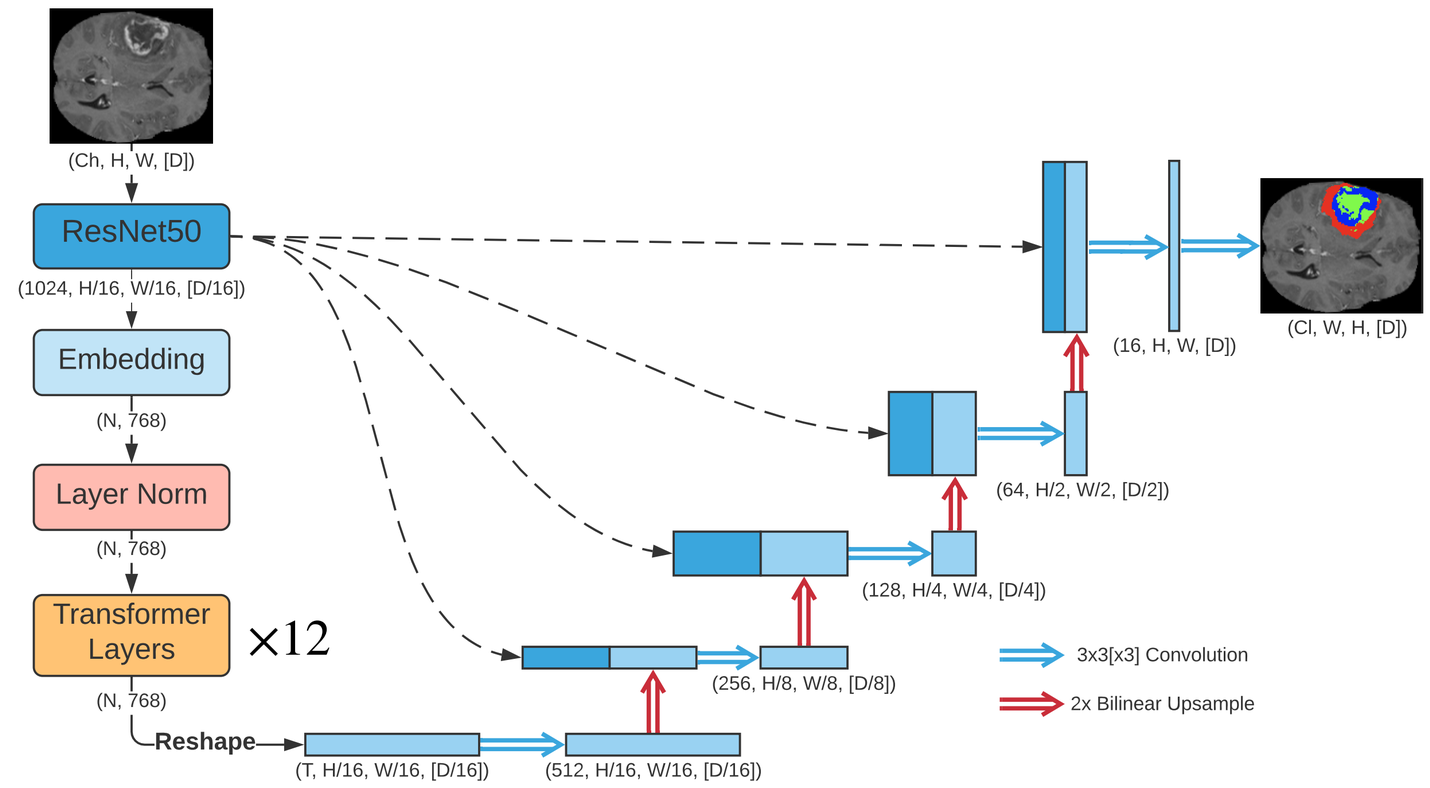


Figure 1: Schematic diagram of the model architecture.

Model Architecture

The model architecture is inspired by the TransU-Net and is a Convolutional Neural Network (CNN) and Transformer hybrid. The model consists of 3 main parts: the ResNet50, the Transformer, and the U-Net decoder. The images are first encoded into a latent feature space using a ResNet50. Feature encodings at different scales are used as skip connections to the U-Net decoder. This allows the model to transfer context to the decoder at different scales, which is crucial for the model to respond to global features, whilst also having high localisation accuracy. The transformer uses self attention mechanisms to extract global contextual information out of the encodings, which enables it to model large scale features much more accurately than a standard U-Net. Both a 2D and a 3D version of the model are trained, and the best one is chosen using cross validation accuracy.

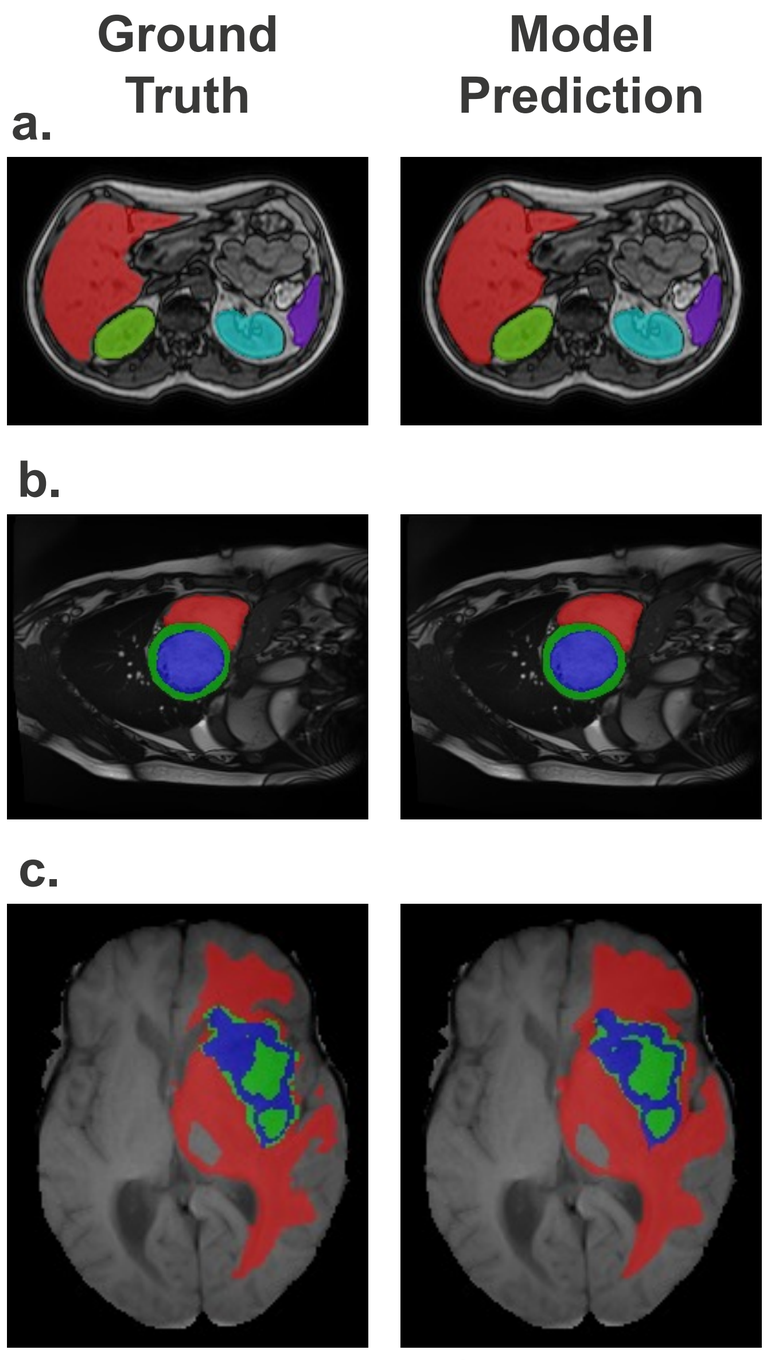


Figure 2: Model outputs of the validation set (right) along with the ground truth labels (left). **a.** Abdominal MRI scan with Liver (red), Right kidney (green), Left kidney (cyan) and Spleen (purple). **b.** MRI scan of the heart with Right ventricle (red), Left ventricular cavity (blue) and Myocardium (green). **c.** MRI scan of the brain with Edema (red), Non-enhancing tumour (green) and Enhancing tumour (blue).

Results

The results of my design vary greatly across the different datasets. For 5 of the datasets, the accuracy is comparable with the current state of the art (the nnU-Net). Furthermore, my design required 5 times fewer computational resources, and less than half the training time when compared to the nnU-Net. However, my design failed to produce reasonable results for four of the tested datasets due to the model's inability to adjust for extreme class imbalances in very large images. In the future, I plan to implement oversampling of classes, and a cascaded U-Net to help resolve these issues and improve the accuracy. So, while my design was not entirely successful, it shows promise as it is able to achieve comparable results to the current state of the art with far fewer resources.

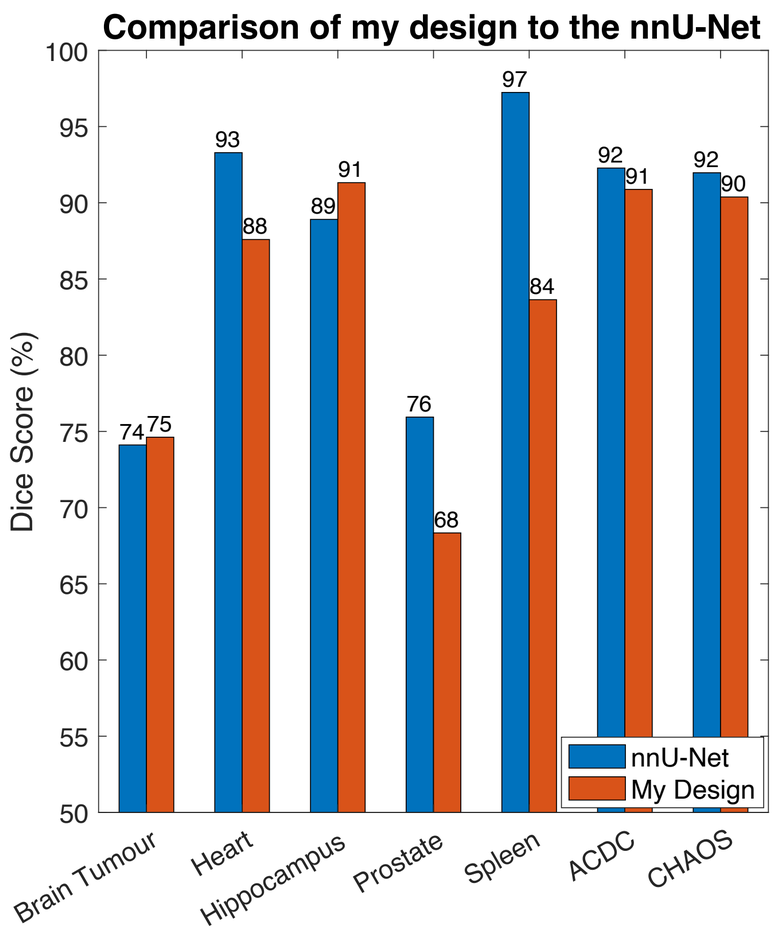


Figure 3: Comparison between state of the art (nnU-Net) and my design. Only datasets that produced successful results are shown here.

Significant Contributions

- Performed in depth analysis and research into the field of medical image segmentation to determine the current state of the art designs and techniques
- Designed the model and algorithms used for the image segmentation in the project, based on existing designs and research
- Implemented the entire design in Python, including writing the code to preprocess the data, train the model, evaluate the model, as well as the code for the model itself
- Extensive debugging, training, modification and tweaking was performed before the model was complete and ready for experimentation
- Tested the design extensively by performing ablation studies to determine the impact of changing various hyper parameters
- Tested the design on 11 datasets to evaluate the models ability to generalise
- Analysed the results, and determined explanations for the findings based upon the dataset statistics

1 Executive Summary

Medical image segmentation has the potential to revolutionise disease diagnosis and treatment planning. Recently, the U-Net has become the standard model used for image segmentation. However, there are a number of flaws in the U-Net which I aim to address in my design. In this report I outline the design, implementation and results for a new design that will solve two major problems with the original U-Net.

First, convolutions are intrinsically local operations, meaning that purely convolutional segmentation networks such as the U-Net will struggle be able to model large scale, global features. I have used a transformer U-Net hybrid design, to improve the models ability to model global features, and improve the models accuracy when modelling large features.

Secondly, the design of segmentation models is highly dependant on properties of the dataset, meaning that segmentation models are often designed to work on a specific task, but do not transfer to other tasks very well. My design uses automatically adapting preprocessing and postprocessing tasks to ensure the model will work well on a wide range of datasets. I have also implemented two model designs, a 2D and 3D model, and the best performing one based on cross validation accuracy is used.

My design has been tested on 11 different datasets representing a variety of challenges. Unfortunately, my design did not perform well on all datasets, as it was unable to adapt to datasets with very large class imbalances. However, on datasets without extreme class imbalances, the model provided highly accurate segmentation's, that are comparable to current state of the art models. With a few modifications to improve the models ability to learn with class imbalances, it is likely that this design will be able to generalise to many different tasks, and provide similar accuracy to state of the art designs. The code used in this project is available at `https://github.com/nicholasprowse/FYP`

Contents

List of Figures

List of Tables

2 Introduction

Semantic image segmentation is the process of splitting an image up into different, meaningful regions. It transforms raw image data into meaningful information about the type and location of content within the image, and thus plays an essential role in scientific discovery. It achieves this by assigning each pixel within the image with a class, corresponding to the object or structure that that pixel belongs to within the image. Image segmentation is particularly useful for medical applications, as it can be used to identify abnormalities in medical images such as tumours or legions. The ability to precisely locate abnormalities can help with diagnosis and treatment planning. There are a wide variety of applications for medical image segmentation, such as segmenting tumours and swelling within MRI scans of the brain[1, 2], segmenting organs within CT scans of the abdomen[3] and even identifying neural pathways within electron microscopy images[4].

Image segmentation has been a large focus of research for the last couple of decades. Early techniques used classical computer vision techniques such as edge detection and mathematical methods. However, in the 2000's, due to improvements in hardware, machine learning techniques began to increase in popularity[5]. These use supervised learning techniques, meaning a machine learning model is trained by being exposed to a large number of example images along with the correct segmentation, called the ground truth. However, early methods were very limited in the size of the input image, and required a long time to train. In 2015, the U-Net was introduced by O. Ronneberger et al.[6], which provided a segmentation technique which was much faster, less memory intensive and more accurate than previous techniques. Moreover, they also introduced a tiling strategy, which allowed them to segment much larger images by splitting the images into patches, and segmenting them separately. Due to the exceptional representational power of the U-Net it has become the most popular network configuration for image segmentation.

Despite the success of the U-Net it still had some major limitations that recent research has been attempting to solve. The U-Net is a Fully Convolutional Network (FCN) meaning every layer is a convolutional layer. But convolutions are inherently local operations, so the U-Net has limited ability to model long range relations and large scale features. Recent studies have shown that transformers can be incorporated into the traditional U-Net structure to help mitigate these limitations[7]. Transformers where originally designed for Natural Language Processing (NLP), which involves tasks such as interpreting and translating languages. They excel at this task as they are able to identify long range dependencies in the input data, which is crucial in NLP, as the meaning of a single word can vary greatly depending on the context of the sentence[8]. Inspired by the transformers ability to model long range features, many researchers attempted to apply them to computer vision tasks to varying success. It was A. Dosovitskiy that managed to create a computationally efficient and accurate model, called the Vision Transformer, by incorporating the transformer into a Convolutional Neural Network (CNN)[9]. The Vision Transformer was created for classification tasks, but it has also been modified to perform image segmentation by incorporating the transformers into a U-Net, creating the TransU-Net. The TransU-Net has been shown to provide much higher accuracy than the standard U-Net, due to its ability to model long range features, allowing it to identify and segment large objects[7].

But, the TransU-Net was designed to work on a specific dataset. There are so many

different applications for image segmentation, especially in the field of medicine, so a design that is able to automatically adapt itself to the given dataset would be incredibly useful. This has been done recently in the nnU-Net ("no new U-Net"), which uses various preprocessing and postprocessing techniques, as well as automatic network configuration to enable accurate training across a wide variety of medical segmentation tasks[10]. In this report, I have designed and implemented a new design, which modifies the TransU-Net so that it automatically adapts to a new dataset, similarly to the nnU-Net, in such a way that it is able to provide accurate image segmentation's regardless of the dataset. This will result in a model that can accurately model both small and large features in images, due to the combination of convolutions and transformers, all while being able to generalise to many different tasks.

3 Literature Review

3.1 Model Architecture

Background on Convolutional Neural Networks

Convolutional Neural Networks (CNN) are a type of machine learning model that primarily consist of a sequence of convolutional layers. They are very important in the field of computer vision, especially for classification tasks, but have many other uses including image segmentation. The first layer is the input layer, and is directly connected to the input image. Subsequent layers are convolutional layers, and result in the convolution of the input with a kernel. The size, stride and other parameters of the kernel are decided by the network designer, but the values in the kernel are optimised during the training phase to provide the best output[5]. Parameters that are adjusted during the training phase are called learnable parameters. Each pixel in the output of a convolution responds only to a specific area of the input to previous layer, which is determined by the kernel size. This area is called the receptive field. Convolutions are linear operators, so to model non-linear functions, activation layers are used after every convolutional layer. Pooling layers such as max pooling or average pooling can be used to reduce the size or dimensionality of the output, while activation functions such as ReLU can be used if no change in size is needed[5]. These convolutional layers have been found to learn to extract meaningful features out of the images, so the outputs of the convolutional layers are commonly called the features of the image[11]. Lastly, a fully connected layer is often used to take these features and map them to the desired output for the specific task. A fully connected layer is one where every pixel in the output is connected to every pixel in the input via learnable weights and biases. The learnable parameters are optimised during the training phase, through a process called back propagation, whose goal is to minimise a loss function[5]. This is the structure typically used in CNN's for image classification, however there are many task specific variations, some of which I shall discuss in the following sections.

ResNet

As computing hardware improved, deeper and deeper CNN's where developed, which provided better accuracy in classification tasks. However, deeper networks result in much slower training. During training, each of the network's weights are updated proportionally to the partial derivative of the loss function with respect to the current weight. However, with very deep CNN's the gradient in the early layers becomes very small as the gradient has to propagate backwards from the output. This results in the first few layers learning very slowly, in a problem known as the vanishing gradient problem[12].

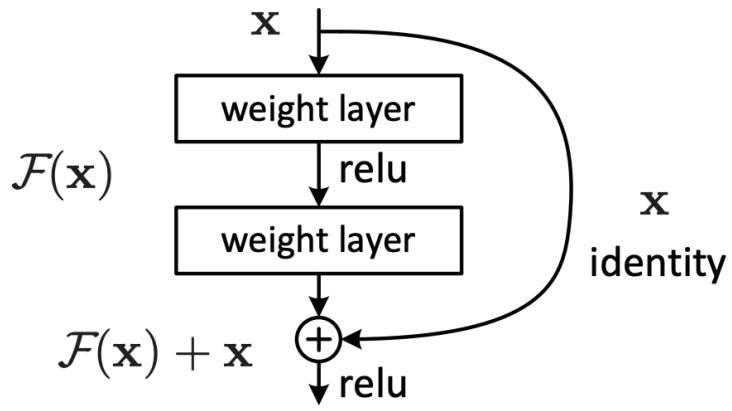


Figure 1: Example structure of a residual connection within the ResNet. $\mathbf{x}$ is passed through two standard layers, and then added to the original value of $\mathbf{x}$[12]

The ResNet introduced a solution to this problem in the form of residual connections. A residual connection is when there is a connection that skips a number of layers, as shown in Figure 1. These skip connections allowed the gradient to skip a number of layers, which allowed the gradient to reach the early layers through a shorter path. The ResNet provided faster training and better accuracy than comparable designs without skip connections. It also allowed for much deeper networks than previously possible as the vanishing gradient problem was no longer an issue. The ResNet comes in a few different variations depending on the number of layers within the network. Deeper networks provide better accuracy, but take longer to train[12]. The most commonly used in the ResNet50 (named so because there are 50 layers) as it provides a good trade off between accuracy and training times. The ResNet50 consists of a 7×7 convolution layer at the beginning followed by a stride 2, 3×3 max pool. This is then followed by 16 convolutional blocks. Each block has 3 convolutional layers, a 1×1, followed by a 3×3, followed by a 1×1. Each block has a skip connection, connecting the input directly to the output of the block. After specific blocks, the spatial dimensions are reduced, and the channels increased by using stride 2 convolutions instead of stride 1. This is done 4 times, so the resulting features are 32 times smaller than the input along each spatial dimension. Finally, there is an average pool layer followed by a fully connected layer to create the output classification.

The original ResNet uses Batch normalisation (BN) between every convolutional layer. Batch normalisation involves normalising each channel within a batch separately. So, if a

batch has dimension (N, C, H, W), where N is the batch size and C is the number of channels, then the mean and variance is computed for each channel of dimension (N, H, W), and then each channel is normalised. That is, the normalisation is done across the entire batch, hence the name. However, BN performs poorly with small batch sizes, since the normalisation is only performed over a small set of data. As such, Wu et al. proposed Group normalisation (GN) as an alternative for small batch sizes[13]. Group normalisation normalises each image separately, so it is not affected by batch size. To get over the problem of having a small set over which the normalisation occurs, they suggest normalising over a group of channels rather than just a single channel. The size of the group is a hyper parameter decided by the designer, but is was found that any size of group above 4 provides good performance. By training a ResNet50 on ImageNet[14], they found that GN provided significant improvements to the models accuracy for small batch sizes less than 16. The biggest improvement was for a batch size of 2, where they observed a 10.6% improvement over BN. While BN did provide better accuracy for large batch sizes, the improvements where minor, with BN being only 0.5% more accurate than GN.

Qiao et al. observed similar results to Wu et al., but they also showed that weight normilisation can provide a similar improvement[15]. Weight normalisation simply involves normalising the weights of each convolutional kernel to zero mean and unit variance before applying the convolution. They showed both theoretically and empirically that this smooths the loss landscape, resulting in faster and more robust learning.

U-Net

Early image segmentation techniques involved using a classification model trained to classify a small patch of an image. Then, a sliding window is used to classify each pixel by providing the network with the small patch surrounding the pixel. However, this was both slow, and inaccurate. The network is required to run separately for each pixel, and there is a lot of overlap between the patches, resulting in redundancy. Also, the patch sizes create a trade off. Larger patches provide the network with more context, so the classification is more accurate, but this comes at the cost of reduced localisation accuracy. On the other hand smaller patch sizes provide less context, but more localisation[6].

These problems were solved by Ronneberger et al., who introduced the U-Net. The U-Net is capable of segmenting an entire image with a single pass through the network. Figure 2 shows the design they used. It is a fully convolutional network (FCN) meaning every layer is convolutional. It consists of four sets of convolutional blocks, each with three 3×3 convolutions. Each block is connected through a 2×2 max pool which downsamples the features by a factor of 2. This path is called the encoder, as it encodes the image into a small feature vector that should describe the image at a high level. The other half is called the decoder, as it decodes the features into the required segmentation. It is identical in structure, except the max pooling layers are replaced with 2×2 "up-conv" layers, more commonly known as deconvolution or transposed convolution layers. These increase the size by 2, resulting in the features being restored to their original size at the end of the decoder. At each scale, a skip connection connects the encoder to the decoder, allowing localisation information to be transferred to the encoder. The U-Net provided very good segmentation performance, and was much more computationally efficient than previous designs as it is

fully convolutional and only requires a single pass to segment an entire image[6].

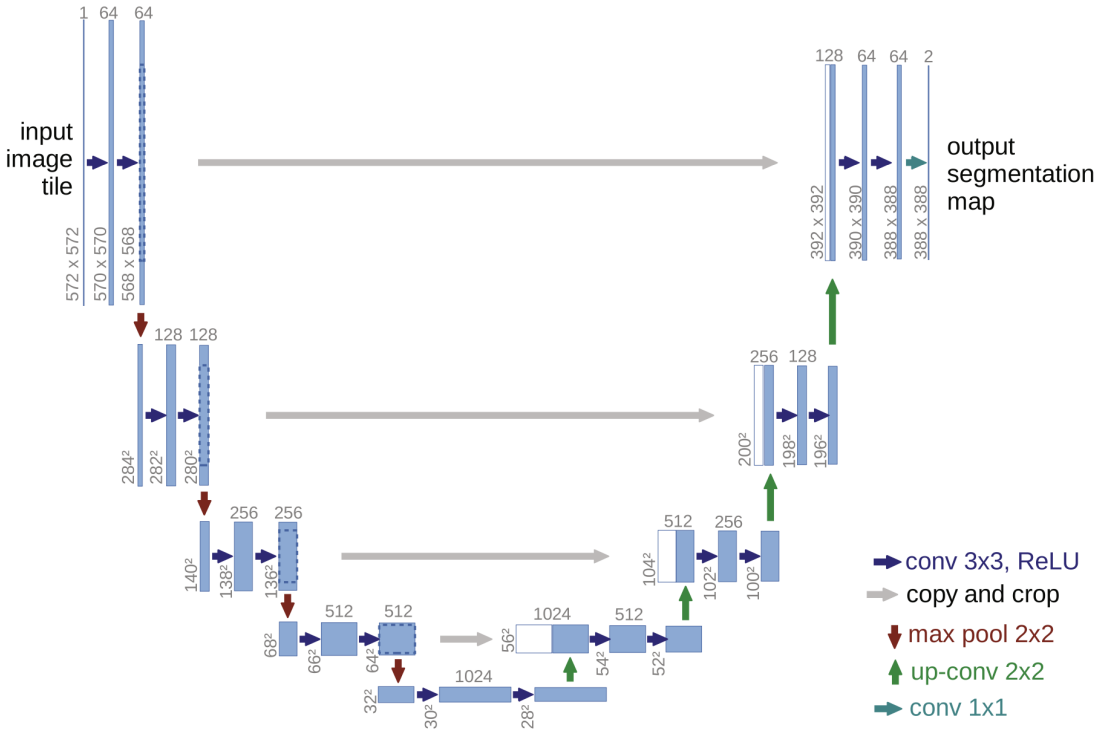


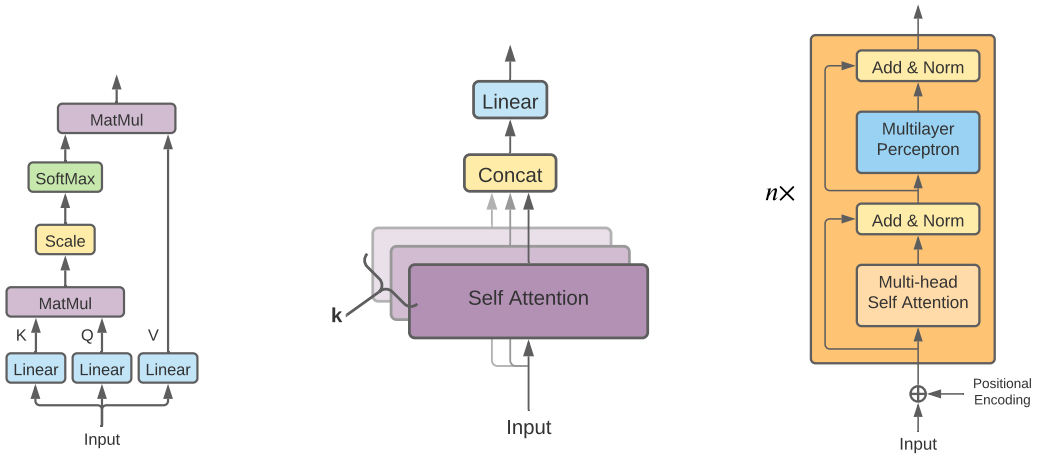
Figure 2: The network architecture in the U-Net, as originally proposed by Ronneberger et al.[6]

Transformers

One issue with FCN's such as the U-Net is that convolutions are inherently local operations. This means that the U-Net can perform poorly when large scale features are present within an image. If an object requires context from opposite sides of the image in order to be classified correctly, then the U-Net will not be able to do this, because convolutions have a limited receptive field[9].

The transformer architecture, first introduced by Vaswani et al., provides a solution for this problem[8]. Vaswani et al. originally designed the transformer for use in natural language processing (NLP) tasks. It is designed to encode a sequence of words into an encoding with greater context, where this context can come from anywhere within the sequence of words. In this case, context refers to the meaning of a word in relation to the words around it. Words often carry different meaning depending on other words within the sentence. For example "bank" could refer to a financial establishment or the shoreline of a river, depending on how it is used within a sentence. The transformer uses contextual information from the rest of the sequence of words to create an output that contains more contextual information. It does this using a mechanism called Multi headed Self Attention (MSA). Self attention is a technique of relating different positions of a sequence based on how similar the information they carry is. As can be seen in Figure 3a it contains 3 linear transformations, controlled by the learnable matrices Q, K and V, allowing the model to learn the optimal way to apply

attention. Multi headed self attention (Figure 3b simply performs self attention multiple times in parallel, with each attention mechanism being referred to as a head. It has been found that the different heads learn to focus on different types of attention, so having multiple heads allows more contextual information about each token to be obtained. The different heads are combined by concatenating them and passing them through a linear layer, resulting in the input and output having the same dimensionality.



(a) Self Attention **(b)** Multihead Self Attention **(c)** Transformer Encoder

Figure 3: Architecture of the Transformer Encoder

Figure 3c shows the design of the Transformer as described by Vaswani et al. The full transformer contains an encoder and decoder, but I will just be focusing on the encoder half of the transformer, so throughout this report, I will refer to the transformer encoder just as a transformer. The transformer consists of an MSA layer, followed by a multi-layer perceptron (MLP) layer. Skip connections are present across each of these layers to allow the gradient to propagate backwards through the layers. Since the output dimensionality is equal to the dimensionality of the input, these transformer layers can be stacked one after the other as many times as is required. Due to the attention mechanism, which is able to gather contextual information between two token regardless of their relative position, the transformer has proven to be superior to previous NLP architectures such a recurrent neural networks. However, since the transformer does not use any positional information, it is important to add a positional encoding before the transformer, which is a learnable parameter that allows the model to add positional information before the transformer. This allows the transformer to respond to positional information, but it is not constrained by it in the same way convolutions are.

Vision Transformer

While transformers were designed for NLP, they have been proven to also be very powerful when used in conjunction with a CNN in computer vision tasks. The Vision Transformer, developed by Dosovitskiy et al. modifies the transformer to be used for image classification

tasks, including ImageNet[9]. This is done by splitting the image up into patches, using a learnable, linear projection to project each patch into a $1 \times T$ vector, and using these as the input tokens. These are then fed through $n = 12$ transformer layers before being passed through a final MLP to perform the classification. This architecture was able to match the performance of the ResNet152 with less than 10% of the training time required by the ResNet. This is because the transformer layers are able to obtain contextual information from the entire image in each layer, but in a CNN such as the ResNet, you need a very deep network in order to increase the receptive field of the convolution sufficiently to obtain the same result[9].

TransU-Net

This architecture was modified by Chen et al. to create the TransU-Net, a U-Net achitucture with a transformer to perform image segmentation[7]. They did this by replacing the final MLP on the Vision Transformer with the decoder half of the U-Net. However, they found that this approach did not provide optimal results, since the encoding generated by the transformers was significantly lower resolution than the output, so there is a loss of low level details as it is upsampled. So, rather than using a pure transformer for the encoder, they used a CNN-transformer hybrid. A ResNet50 with the final linear layer removed is used to generate the input tokens to the transformer. Then, the features after each downsampling in the ResNet50 are used as skip connections to the corresponding layer in the decoder in a similar fashion to the original U-Net. This provides the model with high resolution details as the image is upsampled, providing much better performance. Using this architecture they were able to beat the state of the art performance on the ACDC dataset by over 2% (using the Dice score metric)[7].

3.2 Model Generalisation

Machine learning models are often very task specific. They will be optimised to perform well for a specific dataset, but may not generalise well to other datasets. The nnU-Net is designed to address this problem, and is currently one of the most successful designs in the field of medical image segmentation[10]. It is ranked number one on the Medical Segmentation Decathlon[2], a medical image segmentation challenge that involves performing semantic segmentation on 10 different datasets, without any manual adjustment of the model. Every dataset has different challenges, such as class imbalances, small datasets, and small features, so to perform well, a model needs to be able to overcome a wide variety of challenges automatically.

The nnU-Net achieves this by altering a number of aspects of its design automatically based on rules and empirical observations. The given dataset is analysed an a "dataset fingerprint" is obtained. This contains things such as the median pixel spacing, median image size and how isotropic the images are. This dataset fingerprint is used to determine how the images are pre-processed, and how data augmentation is performed. Empirical observations are also used to optimise the model. Both a 3D and 2D U-Net are trained and the one that provides the best cross validation performance is used. Intuitively, one would expect the 3D model to perform better, as it has access to contextual information along

the third axis while the 2D model doesn't. However, due to GPU memory limitations, the 3D images need to be split up into patches in order to be segmented. This can reduce the performance of the model as the model gets to see less of the image in each pass. Due to this, the 2D model does occasionally perform better than the 3D model, but the 3D model is superior for most datasets[10].

The depth of two U-Nets are automatically configured to provide optimal performance, so that larger images have deeper networks. This allows the receptive field of the encoded features to be larger on larger images, allowing for accurate segmentation's regardless of size. The combination of network configuration, as well as automatic adaptation of pre-processing, post-processing and data augmentation routines allows the U-Net to provide accurate results on many medical image segmentation tasks. It was tested on 23 different datasets, and while it didn't beat state of the art architectures for all of them, it did prove to generalise well to many different tasks which it was not initially optimised on.

3.3 Training the model

Loss Function

One of the most important factors determining how well a model is able to learn is the choice of loss function. The loss function is a measure of how well the model is performing, with lower losses being better, and the optimiser minimises the loss by computing the gradient of the loss with respect to the model parameters. The most common loss function used in machine learning is the Cross Entropy loss. If $\mathbf{x}$ is the output of the network, and $\mathbf{p}$ is the ground truth, then the cross entropy loss is found by first applying a Softmax to $\mathbf{x}$, then applying negative log likelihood:

$$\text{Softmax}(\mathbf{x}) = \frac{\exp(\mathbf{x})}{\sum_i \exp(\mathbf{x}_i)} \tag{1}$$

$$\mathcal{L}_{CE} = -\sum_i \mathbf{p}_i \ln(\text{Softmax}(\mathbf{x})_i) \tag{2}$$

While the Cross Entropy loss does provide good performance, there are loss functions specifically designed for image segmentation which can provide different performances. The dice score is calculated separately for each class in the output, and is defined as

$$\text{DSC} = \frac{2\text{TP}}{\text{TP} + \text{FN} + \text{FP}} \tag{3}$$

where $\text{TP} = \sum_i \mathbf{x}_i\mathbf{p}_i$ is the true positive rate, $\text{FP} = \sum_i \mathbf{x}_i(1-\mathbf{p}_i)$ is the false positive rate and $\text{FN} = \sum_i(1-\mathbf{x}_i)\mathbf{p}_i$ is the false negative rate. This provides a measure of the overlap between the ground truth and the prediction. A larger dice score provides a better segmentation, so in practice $\mathcal{L}_{dice} = 1 - \text{DSC}$ is used as a loss function. A generalisation of the dice score is the Tversky index, which weights the false positive and false negative values differently.

$$\text{TI} = \frac{2\text{TP}}{\text{TP} + \alpha\text{FN} + (1-\alpha)\text{FP}} \tag{4}$$

This is advantageous as it allows you to penalise false negatives more than false positives. Commonly the background class is much more frequent than others, so models often predict the background more than they should, leading to high precision but low recall. Precision refers to the proportion of pixels the model predicts to be in a class that are actually in that class, while sensitivity refers to the proportion of pixels from a given class that the model correctly identifies. To prevent this, we can penalise the false negatives more than the false positives, by setting $\alpha > 0.5$[16].

A further improvement that was suggested by Abraham et al. is to add some non linearity to the loss, by defining the Focal Tversky Loss as

$$\mathcal{L}_{FTL} = (1 - TI)^{\gamma} \tag{5}$$

If $\gamma > 1$, then this increases the gradient for cases where TI < 0.5 which forces the model to focus on harder examples. On the other hand, if $\gamma < 1$, the gradient is increased for cases where TI > 0.5 resulting in the model focusing on easier cases. This can lead the model to learn slower in the beginning, but allows the model to continue learning even when TI is nearing convergence.

Optimiser

There are many different types of optimisers, but the two most commonly used are Stochastic Gradient Descent (SGD) and Adam. SGD works by computing the gradient of the loss with respect to each parameter in the network. Then, each parameter is updated in the opposite direction to the gradient, scaled by a learning rate. This results in a reduction in the loss. If w are the weights of the model, η is the learning rate and $\mathcal{L}$ is the loss, then one iteration of SGD is given by the following update rule.

$$w \to w - \eta \nabla_w \mathcal{L} \tag{6}$$

SGD uses the same learning rate for all parameters within the model, however Adam computes different learning rates for different parameters based on first and second moments of the gradient. It adaptively adjusts the learning rate throughout training to provide the best training. Adam is very popular and is commonly believed to be superior to standard SGD, since it is adaptive. However, recent research by Wilson et al. shows that while Adam does provide slightly faster training, and does reach a higher accuracy during training, it does not generalise well to the test dataset. Using the CIFAR-10 dataset, they found that SGD provided almost 5% lower top 1 error than Adam on the test dataset, even though they both reach the same accuracy on the training dataset[17].

4 Overview

I have designed an image segmentation model based on the TransU-Net, and using elements of the nnU-Net in an attempt to make a transformer based model that is able to generalise to a wide variety of medical image segmentation tasks. The TransU-Net was created specifically for two datasets, the Automated Cardiac Diagnosis Challenge (ACDC)[18], and the Synapse

multi-organ segmentation dataset. Due to this, there are a number of limitations on the TransU-Net which mean that it is unable to translate well to other tasks.

The original TransU-Net only worked on square images, whose side lengths were a multiple of 16. Furthermore, it was a 2D model, meaning that in order to process a 3D medical image, it has to process each slice separately. However, Isensee et al. found that 3D U-Nets often perform better than equivalent 2D U-Nets when segmenting 3D medical images[10]. My design is inspired by the TransU-Net, but it is able to segment any size image, including both 2D and 3D images.

I have also used a task independent method of choosing design parameters, so to provide good model performance regardless of the dataset used to train the model. Design decisions are determined through three different techniques

1. Task independent decisions, that do not require adaptation between tasks are fixed across different iterations of the model (fixed parameters)

2. Heuristic rules are used for as many of the remaining choices as possible. These rules use information about the dataset to determine appropriate values (rule based parameters)

3. Remaining decisions are decided empirically based on model performance. The best option for these decisions is decided based upon which option provides the best cross validation accuracy for the model (empirical parameters)

Fixed parameters include things such as the learning rate, choice of optimiser, the loss function, as well as hyper parameters relating to network architecture such as the number of transformer layers and the number of attention heads. Rule base parameters include how images are normalised, the size and method used for resampling, the patch and batch size, and certain aspects of data augmentation. These parameters are determined by factors such as the modality of the images (MRI or CT), the distribution of voxel spacings, and whether the images are isotropic or not. The empirical parameters are the choice of the 2D or 3D model, and whether to perform non-largest connected component suppression on the output prediction. Further details about how these parameters are determined is included in the Methods section. The nnU-Net uses a standard U-Net, however they adaptively choose the depth of the U-Net to ensure that the encoded features are as small as possible, without being smaller that $4 \times 4 (\times 4)$. This is done to ensure the encoded features have a large enough receptive field to obtain global contextual information, however it results in larger models and slower training for larger images. My design does not adaptively choose the depth of the network. Instead it is a fixed parameter. This is due to the fact that the transformer is very good at obtaining global context, so it is not necessary to downsample the features to such a small size.

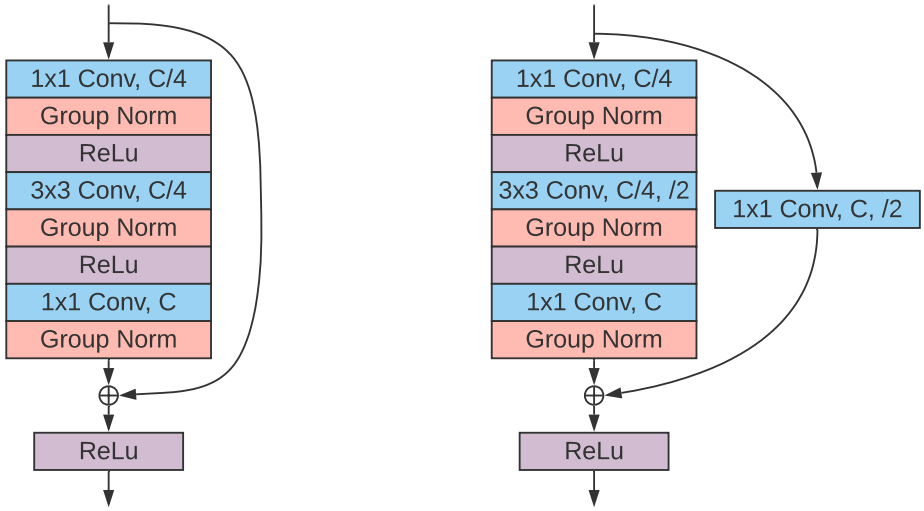
The fixed parameters have been determined using a variety of ablation studies, aimed at finding optimal values for certain network hyper parameters. In order to test the ability of the model to generalise to a wide variety of datasets, the model has been tested on 11 datasets, chosen to represent a variety of common challenges present in the field of medical image segmentation such as class imbalances, small training samples and varied feature sizes and shapes.

5 Methods

5.1 Model Architecture

The model consists of 3 main components: the ResNet50, to downsample the input into feature encodings; the transformer, to add contextual information to the encodings; and the decoder, which is similar to the original U-Net decoder and upsamples the image to generate the output segmentation. I have designed the same model in both a 2D and 3D version, so throughout this discussion, I will use the square brackets to denote that something is present in the 3D version, but not the 2D version (i.e. the image dimensions $(H, W, [D])$, indicates that the third dimension D is not present in the 2D model, but is present in the 3D one).

ResNet50



(a) Regular Block **(b)** Downsampling block

Figure 4: Design of the ResNet convolutional units. The number of output channels of each convolutional layer is shown as the variable after the Conv, and the /2 indicates a stride of 2

The ResNet50 I have implemented is very similar to the original ResNet50 proposed by He et al.[12], but contains some important differences. The below description describes the ResNet used in the 2D model, however, the 3D model is identical, except the 2D convolutions are replaced with equivalent 3D convolutions. The ResNet50 primarily consists of a series of convolutional blocks, the structure of which is shown in Figure 4. Each block consists of 3 convolutional layers, each with a Group Norm and a ReLU activation function afterwards. This is in contrast to the original ResNet50, which uses Batch Norm. Group Norm has been used instead, as I have used very small batch sizes, and Group Norm provides superior performance with low batch sizes[13]. The ReLU is the Rectified Linear Unit, as is defined as

$$\text{ReLU}(\mathbf{x}) = \max(0, \mathbf{x}) \tag{7}$$

There are two subtly different variants of this block. The first is the regular block, in which the spatial dimensions of the input and output are identical, and the second is the downsampling block, in which the spatial dimensions of the output are halved. This is achieved by using a stride of two in the 3×3 convolution, and also adding a stride 2, 1×1 convolution to the skip connection so the output and the skip connections have the same size.

These convolutional blocks contain a "bottleneck", meaning the number of channels is reduced in the block to $C/4$, then restored to C afterwards. This forces the network to find a representation of the data that requires a smaller size, resulting in it finding a better representation of the features. It is also a good regularisation technique. It prevents overfitting by reducing the complexity of the network. The 1×1 convolutional layers are there to facilitate this bottleneck. The first one is to reduce the number of channels to $C/4$, and the second is to restore them back to C.

Figure 5 shows how these convolutional blocks are used to create the full ResNet50. This consists of a 7×7 convolution, followed by a max pooling layer, followed by 16 convolutional blocks, split into 3 different groups. The second two groups both begin with a downsampling block, resulting in a total of 4 downsamplings through the ResNet. So, given an input of dimension (C, H, W), the output will have dimension of $(1024, H/16, W/16)$. This design differs from the original ResNet50 in two important aspects. First, the original ResNet50 had a final fully connected layer since it was designed for classification tasks. This is not required in this case, as we just need the encoded features for the transformer. Secondly, the original ResNet50 had 4 groups of convolutional blocks, rather than 3, resulting in 5 downsampling layers. Since some of the images I am using are relatively small, downsampling 5 times will result in too much spatial information lost, so I have combined the final two groups into one, meaning there are only 4 downsampling layers. However, the total number of layers is still the same as the original. I have also implemented weight standardisation to the weights of all convolutions in the ResNet, as this has been shown to smooth the loss landscape, leading to more robust learning[15]. This is achieved by taking the weights learned by the model, the normalising them to zero mean and unit variance before applying the convolution.

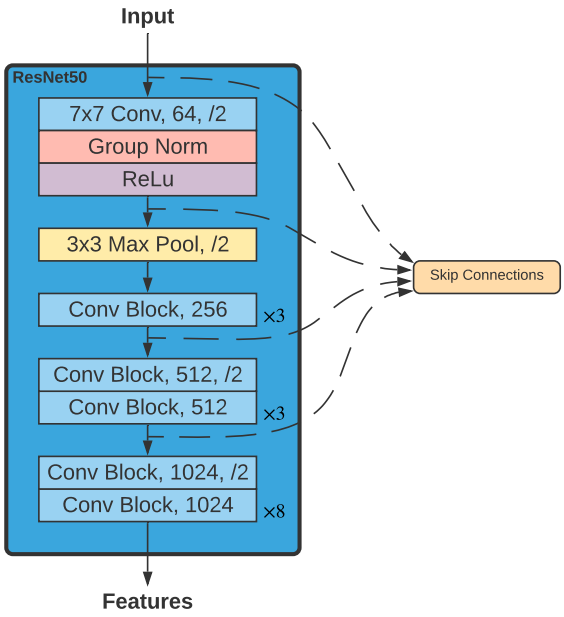


Figure 5: Design of the ResNet50 used in my model.

At each spatial resolution, there are skip connections to the decoder to provide higher resolution context to the decoder. This can be seen in more detail in Figure 6.

Transformer

In order to convert the output of the ResNet into tokens that can be used in the Transformer, we use an embedding layer. Let the output of the ResNet be $\mathbf{x} \in \mathbb{R}^{1024 \times h \times w[\times d]}$, where $(w, h, [d]) = (W, H, [D])/16$ is the original image dimensions divided by 16. To embed this, we split the image up into N patches of size $P \times P[\times P]$. Each patch is converted to a $(1, T)$ vector using a linear mapping. Then, each vector is stacked together along the first dimension to create embeddings of shape (N, T). In practice, this is achieved with a convolutional layer with a kernel size P and stride P, meaning there is no overlap between each convolution. The spatial dimensions are then flattened to get the embeddings.

This procedure of embedding the features was used in the TransU-Net[7], however, they used a patch size of 1, meaning N is equal to the spatial size of the ResNet features. Initially, this is what I did, however, I found that for larger images, this did not work. The TransU-Net uses a fixed size of image, which results in N always being the same. However, I am using many different datasets, which results in a wide variety of different image sizes. For larger images, particularly 3D images, this can result in a very large value for N if a patch size of 1 is used. Due to the complexity of the transformer, excessively large values of N can cause the program to quickly run out of memory. As such, my design adaptively chooses the patch size, P, to ensure that the number of patches is as close as possible to $4 \times 4[\times 4]$, without being less than this. This results in larger patch sizes for larger images, which makes the GPU memory usage more reasonable. Since the patch size may not be divisible by the size of the features, the features are padded accordingly so that no information is lost during the process of embedding.

These embeddings are then normalised with a Layer Norm, and passed through n transformer layers. The transformer layers are defined exactly as originally described by Vaswani et al.[8]. The structure is shown in figure 3c. Each transformer layer consists of a MSA layer, with a skip connection around it, followed by an MLP, also with a skip connection around it. If $\mathbf{z}_0$ is output of the embeddings procedure, then the transformer layers transform $\mathbf{z}_0$ as follows

$$\begin{aligned} \mathbf{z}'_i &= \mathrm{LN}(\mathrm{MSA}(\mathbf{z}_{i-1}) + \mathbf{z}_{i-1}) && i = 1, \dots n \\ \mathbf{z}_i &= \mathrm{LN}(\mathrm{MLP}(\mathbf{z}'_i) + \mathbf{z}'_i) && i = 1, \dots n \end{aligned}$$

The MLP layer is a fully connected neural network with 2 layers using dropout of 0.1, and the ReLU activation function. The MSA layer is implemented as shown in figure 3b. For an input $\mathbf{z} \in \mathbb{R}^{N \times D}$, self attention of $\mathbf{z}$ is defined as follows

$$[\mathbf{q}, \mathbf{k}, \mathbf{v}] = \mathbf{z}\mathbf{U} \qquad \mathbf{U} \in \mathbb{R}^{D \times 3D_h}, \mathbf{q}, \mathbf{k}, \mathbf{v} \in \mathbb{R}^{N \times D_h} \tag{8}$$

$$\mathbf{A} = \mathrm{softmax}\left(\frac{\mathbf{q}\mathbf{k}^T}{\sqrt{D_h}}\right) \qquad \mathbf{A} \in \mathbb{R}^{N \times N} \tag{9}$$

$$\mathrm{SA}(\mathbf{z}) = \mathbf{A}\mathbf{v} \qquad \mathrm{SA}(\mathbf{z}) \in \mathbb{R}^{N \times D_h} \tag{10}$$

The Multihead Self-Attention is created by taking k self attention operations, concatenating them together and projecting the result to the output shape

$$\mathrm{MSA}(\mathbf{z}) = [\mathrm{SA}_1(\mathbf{z}), \mathrm{SA}_2(\mathbf{z}), \dots, \mathrm{SA}_k(\mathbf{z})]\mathbf{W} \qquad \mathbf{W} \in \mathbb{R}^{kD_h \times D} \tag{11}$$

I have set $D_h = \lfloor D/k \rfloor$. This results in the value kD_h being approximately constant if the number of heads changes. This results in the number of parameters in $\mathbf{W}$ being approximately constant and, since there are k different $\mathbf{U}$ matrices, the total number of parameters across all $\mathbf{U}$ matrices is approximately constant. This is done so that we can examine the effect of the number of heads without having a large impact on the number of parameters in the model.

Decoder

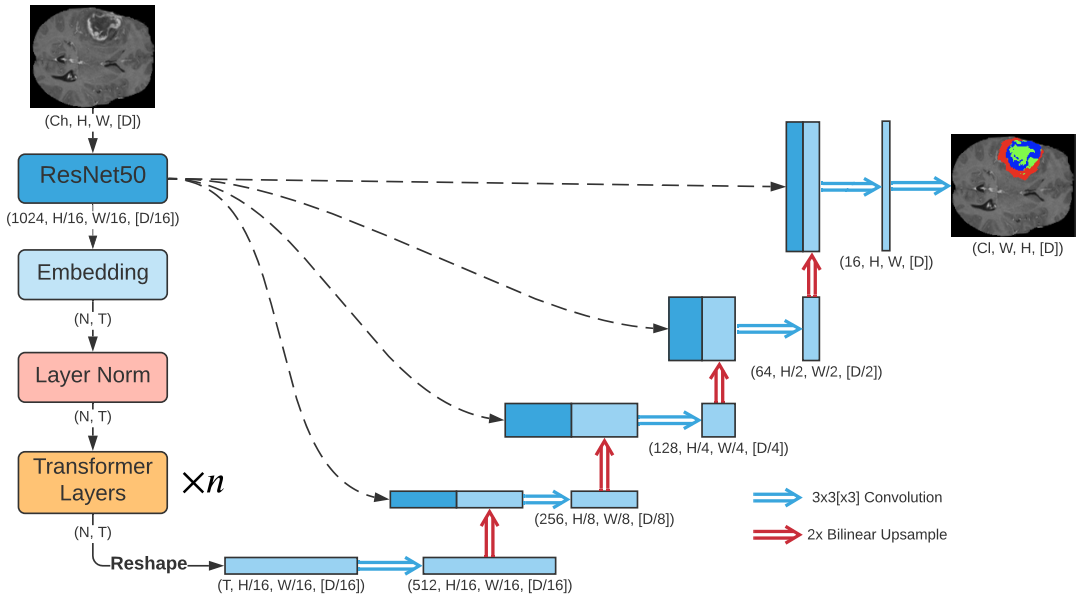


Figure 6: Overview of the entire Transformer U-Net architecture. The encoder on the left is the ResNet50 followed by the transformer, while the decoder in the right is the standard U-Net decoder. The decoder consists of repeated 2x upscaling followed by $3 \times 3[\times 3]$ convolutions, with skip connections from the ResNet50. The dimensions displayed assume a patch size of $P = 1$. With larger patch sizes, the dimensions of the encoded features will be smaller.

For the decoder of the model I am using a design very similar to the original U-Net architecture. The main difference is that instead of using transposed convolutions for the upsampling stage, I have instead decided to use Bilinear (or Trilinear in the 3D model) interpolation to upsample the features. This is done because the up sampling is not always exactly $2\times$. For example, if there is a stage in the CNN where the features are 15×15, then the stride 2 convolution will result in a 8×8 feature space. So that the sizes of the skip connections match, we would then need this 8×8 tensor to be resized to 15×15, instead of 16×16, which is what we would obtain with the U-Net design. Since the U-Net operated on images with sizes divisible by 16, it did not have this issue, but since I am using images of arbitrary size, I am forced to use upsampling, instead of transposed convolutions.

After the transformer layers, the encoded features are first reshaped into the shape they had before they were flattened into embeddings. Then a single $3 \times 3[\times 3]$ convolution is used

to convert the number of channels from T to 512.

Apart from the upsampling layers, the rest of the decoder design is identical to the U-Net. For input $\mathbf{x}_0$, each layer is defined as follows

$$\text{ConvLayer}(\mathbf{x}) = \text{ReLU}(\text{GN}(\text{conv}(\mathbf{x}))) \tag{12}$$

$$\mathbf{x}_i = \text{ConvLayer}(\text{ConvLayer}(\text{UP}(\mathbf{x}_{i-1}))) \tag{13}$$

where $\text{conv}(\cdot)$ is a $3 \times 3[\times 3]$ convolution and $\text{UP}(\cdot)$ is a 2x upsample. After the features have been upsampled to the image size, we do one final $3 \times 3[\times 3]$ convolution to change the channels from 16 to C where C is the number of classes in the segmentation. This is the output of the segmentation model, and will represent a semantic segmentation of the input image. The complete model architecture is shown in Figure 6.

5.2 Preprocessing

There are a number of preprocessing tasks that are completed before segmentation. All preprocessing is performed before training, and the resulting images are saved onto permanent storage, so that preprocessing is only done once for each image. This greatly increases performance as the preprocessing is very computationally intensive. First, the image is cropped to the smallest non-zero region. For certain datasets, such as Task 1 of the Medical Segmentation Decathlon (Brain Tumours), this results in much smaller images which allows for faster training. Next we determine if the image is anisotropic or not. For the purposes of image preprocessing, we will deem an image to be anisotropic if the ratio of the largest axis pixel spacing to the smallest axis pixel spacing is greater than 3.

Intensity Normalisation

If the dataset consists of CT scans then each scan is first percentile clipped, meaning any values below the 0.5% percentile and above the 99.5% percentile are clipped to be within this range. Then the scans are normalised to unity standard deviation and zero mean. This is done because CT scans can contain large outlying voxel values that skew the distribution of values in a scan.

If the dataset does not consist of CT scans, then they are just normalised to unity standard deviation and zero mean. Intensity normalisation is done per image.

Image target spacing

We require all images input to the network to have the same pixel spacing. By ensuring the voxel spacing of all images are equal, we ensure that all images have the same scale, meaning the model is more easily able to infer the size of features within the images. This improves learning speed and accuracy[10]. Since some datasets have variable spacing between each image, we resample all images to have the same target voxel spacing.

The target spacing is chosen to be the median spacing across all training images, computed independently for each axis. However, if the image is anisotropic, the target spacing along the anisotropic axis is chosen to be the tenth percentile of the voxel spacings across all training samples for that image.

Anisotropic axes are treated differently as there is typically a very large variation in pixel spacing across the dataset in anisotropic axes, so choosing the median can cause interpolation artefacts and loss of information (if the median was chosen, there will likely be images with spacing significantly less than this, which would result in a large loss of information as it would result in significant down sampling). As such, we use the tenth percentile so that the pixel spacing is smaller than the median, resulting in less loss of information.

Image Resampling Strategy

The images are then resampled such that they match the target voxel spacing. If the image is isotropic then third order spline is used to resample images.

If the image is anisotropic, then third order spline is used for in plane resampling and nearest neighbour is used for out of plane resampling.

Label Resampling strategy

The label is first converted to one hot encoding, then, if the image is isotropic, linear interpolation is used. If the image is anisotropic linear interpolation is used for the in plane resampling and nearest neighbour is used for out of plane resampling.

Image size

The model requires all input images to be of identical size, to ensure that the number of embeddings is the same for every sample. Since many of the datasets contain images of various different sizes, we must pad the images to a uniform size. The model has been designed so there are no constraints on this size, so the maximum size of all training images (after preprocessing) is determined independently for each axis. This maximum size is then used as the size for all input images, and smaller images are zero padded on the fly to achieve this size.

Batch size and Patch size

The size of the batches and patches are determined during the preprocessing stage. The batch size determines the number of samples used in each mini-batch of training, while the patch size represents the size of each sample. Due to GPU memory constraints, for certain datasets, it is impossible to pass entire images through the model. So, the images are split up into several patches, passed through the model separately, and then the segmentation's of each patch are reconstructed into a single image.

The patch size and batch size are determined simultaneously by taking into account GPU memory constraints. The patch size is first initialised to be the total size of the images. Then, we iteratively halve the largest dimension of the patch until we can achieve a batch size of 2 given the memory constraint. With the use of Group Norm rather than Batch Norm, a batch size of 2 has been shown to be enough to provide robust training[13, 10]. If there is space for a batch size larger than 2, then the batch size is increased until the memory constraint is maxed out, up to a maximum batch size of 5% of the total size of the dataset. The calculation of patch and batch size is done separately for the 3D and 2D model, and

while I have implemented the possibility for the 2D slices to be separated into patches, the images in the datasets I have used are never large enough for this to occur.

The model is trained on the Nvidia Tesla P100 GPU, which contains 16GB of memory. Through trial and error, I have determined that using a batch size containing a total of 10MB of data uses most of the available memory without using it all. Much more than this, and I found that training would occasionally crash due to insufficient memory.

5.3 Postprocessing

After the individual patches have been passed through the model, they are combined together using patch based inference. Due to a lack of contextual information near the edge of the patches, we expect the accuracy of the segmentation to be lower close to the edge of a patch. As such, when generating the patches, it is ensured that there is at least a 20 voxel overlap between each patch. Gaussian importance weighting is then used to stitch the patches together. This involves scaling each voxel in the overlapping region by a Gaussian, with a standard deviation of half of the patch overlap. These Gaussian's are normalised, so there is no net change in the mean voxel value. This results in voxels close to the edge receiving a very low weighting, since they are more likely to be inaccurate, and voxels closer to the centre receiving a higher weighting. This also prevents stitching artefacts by smoothly interpolating from one patch to the next, meaning the resulting segmentation does not have clearly visible patches.

Often in medical imaging, an image will only contain one instance of a structure. For example, if the task is to segment the Spleen, then we know there can only be one spleen in each image. In this situation, it is beneficial to remove all but the largest connected component in the output segmentation, since any smaller components must be false positives. However, in other tasks there may be multiple instances of an object, meaning non largest component suppression would result in poor performance. As such, both techniques are used, and the best performing one is chosen through cross validation performance.

5.4 Data Augmentation

A range of data augmentation strategies have been utilised to artificially inflate the size of the datasets. Augmentations are applied to the entire image before they are split into patches, and for the 2D model, the augmentations are applied separately for each 2D slice.

Random Flip

Flip the images with a probability of 0.5 along each axis

Random Rotation

A random rotation is applied to the images with a probability of 0.2. The rotation angles for each axis of rotation is sampled from $U(-30, 30)$. If the patch size is anisotropic, then large rotations can result in a lot of voxels rotating out of the image. So, in this case, the rotation angle of the axis with the largest patch size remains the same, while the other two angles are drawn from $U(-15, 15)$.

Random Scaling

The images are scaled with a probability of 0.2. The scaling factor is drawn from $U(0.7, 1.4)$. Scaling factors less than 1 result in a "zoom out" effect, while scales larger than 1 result in a "zoom in" effect.

Random Gaussian Blur

A Gaussian blur is applied to images with a probability of 0.2. If blurring is applied, then each axis is blurred independently with probability of 0.5. The standard deviation (in voxels) of the Gaussian kernel is drawn from $U(0.5, 1.5)$ and is chosen independently for each axis.

Random Gaussian Noise

Random Gaussian noise is added to the image with a probability of 0.2. The noise is sampled from a distribution with standard deviation σ sampled from $U(0, 0.1)$. Note that the images have been normalised to zero mean and unit variance before this noise is added, so the noise will have a significant effect, despite the small standard deviation.

Random Brightness Adjustment

The brightness of the images is adjusted with a probability of 0.2. In this case, the voxel intensities are multiplied by a constant value drawn from $x \sim U(0.6, 1.3)$, which simulates both brightness reductions ($x < 1$) and brightness increases ($x > 1$).

5.5 Loss Function

The loss function used during training is the average of the Focal Tversky Loss[16] and the Cross Entropy loss. If $\mathbf{p}$ is the one hot encoded label, and $\mathbf{x}$ is the model prediction, then the loss is defined as

$$\mathrm{TI} = \frac{\sum_i \mathbf{x}_i \mathbf{p}_i + \epsilon}{\sum_i \mathbf{x}_i \mathbf{p}_i + \alpha \sum_i \mathbf{x}_i (1 - \mathbf{p}_i) + (1 - \alpha) \sum_i (1 - \mathbf{x}_i) \mathbf{p}_i + \epsilon} \tag{14}$$

$$\mathcal{L}_{FTL} = (1 - \mathrm{TI})^{\gamma} \tag{15}$$

$$\mathcal{L}_{CE} = -\sum_i \mathbf{p}_i \ln \left(\frac{\exp(\mathbf{x})}{\sum_i \exp(\mathbf{x}_i)} \right) \tag{16}$$

$$\mathcal{L} = \mathcal{L}_{FTL} + \mathcal{L}_{CE} \tag{17}$$

where α and γ are hyper parameters and $\epsilon = 10^{-5}$ is used to prevent exploding gradients. The Focal Tversky Loss is based on the dice loss, which is beneficial to use as the metric we are optimising is the dice score. This is averaged with the Cross Entropy Loss, since it has been empirically shown that this provides better performance than using either loss function in isolation[10]. The parameter α determines how much the loss penalises false positives relative to false negatives, while γ determines the non-linearity.

Class weights have also been implemented to resolve the issue of class imbalances. Prior to training the model, the frequency of each class over the entire dataset is determined. If

$\mathbf{c} = [c_0, c_1, \ldots, c_n]$ is the frequency of each of the n classes, where $\sum_i c_i = 1$, then the class weights have been assigned as follows

$$w_i = \frac{c_i^{-\beta}}{\sum_j c_j^{-\beta}} \tag{18}$$

Note, that for $\beta = 0$, the class weights are all equal, and for $\beta = 1$, the class weights are inversely proportional to the class frequency. We want less frequent classes to be weighted higher, which this achieves. The larger the value of β is, the larger the weights will be for less frequent classes.

The loss function contains three hyper parameters, α, γ and β. The values for these have been determined using ablation studies, the results of which are in Section 6.

5.6 Datasets

Many segmentation models are designed for a specific application, and can not easily adapt to new datasets[10]. In order to ensure my model is able to adapt to a wide variety of tasks, I have trained my model on a number of different datasets. The datasets are chosen so that they represent a variety of challenges, which are summarised in Table 1. Datasets D1 to D9 are available at `http://medicaldecathlon.com/`[2], D10 is available at `https://acdc.creatis.insa-lyon.fr/`[18] and D11 is available at `https://chaos.grand-challenge.org/`[19].

ID	Name	Description	Classes	Challenge
D1	Brain Tumour	Multimodal MRI Scans of the brain	- Edema - Non-enhancing tumour - Enhancing tumour	Complex and heterogeneously located targets
D2	Cardiac	Mono-modal MRI scans of the heart	- Left Atrium	Small training dataset with large variability
D3	Hippocampus	Mono-modal MRI scans of the brain	- Anterior Hippocampus - Posterior Hippocampus	Segmenting two neighbouring small structures with high precision
D4	Prostate	Multi-modal MR scans of the prostate	- Peripheral zone - Transitional zone	Segmenting two adjoint regions with large inter-subject variations
D5	Lung	CT Scans of the lung	- Cancer	Segmentation of a small target (cancer) in a large image
D6	Pancreas	Portal venous phase CT scans of the pancreas	- Pancreas - Cancer	Label unbalance with large (background), medium (pancreas) and small (tumour) structures
D7	Hepatic Vessels	CT scans of the liver	- Vessel - Tumour	Tubular small structures next to heterogeneous tumour
D8	Spleen	CT scans of the spleen	- Spleen	Large ranging foreground size
D9	Colon	CT scans of the colon	- Colon cancer	Heterogeneous appearance
D10	ACDC	The Automated Cardiac Diagnosis Challenge (ACDC) consists of cine-MRI scans of the heart	- Right Ventricle - Myocardium - Left ventricular cavity	Highly anisotropic images
D11	CHAOS	The Combined Healthy Abdominal Organ Segmentation consists of abdominal MRI scans	- Liver - Right Kidney - Left Kidney - Spleen	Large number of foreground classes

Table 1: Summary of the 11 datasets used

6 Results and Discussion

The model was trained on the MonARCH high performance computing cluster, using a single Nvidia Tesla P100 GPU. These GPU's have 16GB of memory, and through trial and error, it was found that 10MB was the largest a single batch could be without exceeding this memory limit. The model was trained using SGD, as this has been shown to provide better generalisation to unseen data than the Adam optimiser[17]. The optimiser is used with a Nesterov momentum of $\mu = 0.99$, and a poly learning rate of $\alpha = \alpha_0(1-\text{epoch}/\text{epoch}_{\text{max}})^{0.8}$[10]. Since the size of the datasets varies a lot, I have decided to train for a fixed number of minibatches, rather than a fixed number of epochs. The model is trained for 10,000 minibatches, which results in training taking about 20 hours, regardless of the dataset size. Ideally one would train for a fixed number of epochs, but this would have resulted in prohibitively long training times for the large datasets, especially considering I tested this on 11 different datasets. The initial learning rate is chosen to be quite high at $\alpha_0 = 0.1$, so that initial learning is fast, but the decaying learning rate still allows the model to fine tune the parameters to find a good local optimum. All datasets are split into a validation set and a training set, with the validation set being 10% of the original dataset. The test datasets provided with the datasets do not contain labels, and so I cannot evaluate the accuracy of the design on the test set directly. Instead the test set predictions are uploaded to evaluation programs provided by the dataset providers. However, my results have not been reviewed at the time of writing, so I do not have access to the test set results. As such, all results here are from the validation dataset, however, this should be indicative of the performance that would be obtained on the test set.

6.1 Ablation Studies

Ablation studies where performed on seven hyper parameters within the network to determine the optimal values for them. The seven parameters studied are

- α - The weighting of false negatives in the Focal Tversky Loss
- γ - The focal factor in the Focal Tversky Loss
- β - Influences how the class weights are determined from the class frequencies
- The number of hidden neurons within the MLP in the transformer layers
- n - The number of transformer layers
- k - The number of self attention heads within the MSA layers
- D - The number of channels created for the embeddings used in the transformer layers

A lot of the results from these studies are not very enlightening, so I will not include all of them here, however all results of the ablation studies are in Appendix A. Due to time limitations, ablation studies have only been performed on dataset D3 (Hippocampus), and the training has been reduced to 2000 minibatches. Since this dataset is relatively small, training converges relatively quickly, so the reduction in training time doesn't impact the performance too much, and still gives an indication of the effect of varying the hyper parameters

Effect on α

Table 2 shows the results of the ablation study on α. As we can see, lowering the value of α consistently produces better segmentation performance. This means that it is beneficial to penalise false positives more than false negatives. This is contrary to what Hashemi et al. found[20]. They obtained the best performance when using a value of $\alpha = 0.7$. Hashemi et al. used the Tversky Index to resolve the issue of class imbalances without the use of class weights, so I also performed an ablation study with $\beta = 0$, which results in equal weighting for each class. Table 3 shows the results of this. As we can see, lower values of α still provide far superior performance. As such, the value $\alpha = 0.1$ will be used for the rest of the experiments in this report, as it provided the best model performance

Table 2: Model performance on D3 for a range of α values with $\beta = 0.5$. The bold line indicates the value that will be used for the final design

	Dice Score		
α	Anterior	Posterior	**Average**
0.0	0.881	0.864	0.873
0.1	**0.880**	**0.866**	**0.873**
0.2	0.879	0.863	0.871
0.3	0.871	0.857	0.864
0.7	0.859	0.842	0.851
0.9	0.842	0.825	0.832

Table 3: Model performance on D3 for a range of α values with $\beta = 0$

	Dice Score		
α	Anterior	Posterior	**Average**
0.5	0.762	0.740	0.744
0.3	0.878	0.863	0.870
0.1	0.873	0.860	0.866

Effect on γ

Table 4 shows the results of the ablation study on the value of γ. γ determines the non-linearity of the Focal Tversky Loss function. The results of this study show that changing γ has little effect on the performance of the model, but values in the range 0.5 to 0.8 show a slight advantage over other values for γ. However, it was found when training the model on other datasets, that values of γ less than one can result in exploding gradients. If we let x be the Tversky index and y be the Focal Tversky Loss, then they are related by

$$y = (1 - x)^{\gamma} \tag{19}$$

So, the gradient of the FTL with respect to the TI is

$$\frac{\mathrm{d}y}{\mathrm{d}x} = -\gamma(1 - x)^{\gamma - 1} \tag{20}$$

If the model is very close to perfect for a specific batch, then the Tversky Index is approximately 1. If γ is less than 1, then we get an exploding gradient as the Tversky index approaches 1

$$\lim_{x\to 1}\left(\frac{\mathrm{d}y}{\mathrm{d}x}\right) = \lim_{x\to 1}\left(-\gamma(1-x)^{\gamma-1}\right) = \infty \tag{21}$$

However, if $\gamma > 1$, we get a vanishing gradient when $x \approx 1$, which is desirable, as the model should not take large steps when it is already close to convergence. The model did not have an issue on D3, as the batch size is quite large on this dataset, meaning it is very unlikely to find an example where the TI is very close to 1. However, on datasets with larger images, the batch size is only 2, so it is possible to get batches that are very easy, resulting in a value of TI very close to 1.

Thus, in practice, we require $\gamma \geq 1$ to ensure the model is stable. As such, $\gamma = 1.3$ will be used for the rest of this report, as it still provided good results, and it has been shown to provide high accuracy on other datasets as well[16]

Table 4: Model performance on D3 for a range of γ values. The bold line indicates the value that will be used for the final design. Note that $\gamma < 1$ cannot be used as it has exploding gradient problems

	Dice Score		
γ	Anterior	Posterior	**Average**
0.4	0.878	0.861	0.869
0.5	0.878	0.865	0.871
0.6	0.882	0.867	0.874
0.7	0.877	0.864	0.871
0.8	0.881	0.867	0.874
0.9	0.876	0.863	0.870
1.0	0.876	0.861	0.868
1.1	0.869	0.855	0.862
1.2	0.876	0.860	0.868
1.3	**0.874**	**0.861**	**0.868**

Effect on β

Table 5 shows the results of the ablation study on β. β influences the relative weightings of each class. For $\beta = 0$, each class is weighted equally, and we can see this is not optimal. This is because the background class occurs far more than any other class, resulting in the model focusing too much on the background. Increasing the value of β increases the weighting of the rarer classes, and this provides better performance, but only up to $\beta = 0.5$. So, the optimal performance is not using the inverse of the class frequencies ($\beta = 1$) as is commonly used. Instead, weighting the rare classes less than this provides a significant boost to performance.

Table 5: Model performance on D3 for a range of β values. The bold line indicates the value that will be used for the final design

	Dice Score		
β	Anterior	Posterior	**Average**
0.0	0.854	0.830	0.840
0.25	0.871	0.861	0.866
0.5	**0.882**	**0.867**	**0.874**
0.75	0.869	0.850	0.857
1.0	0.835	0.815	0.822
1.25	0.800	0.788	0.790
1.5	0.755	0.740	0.746
1.75	0.700	0.672	0.681
2.0	0.690	0.632	0.661

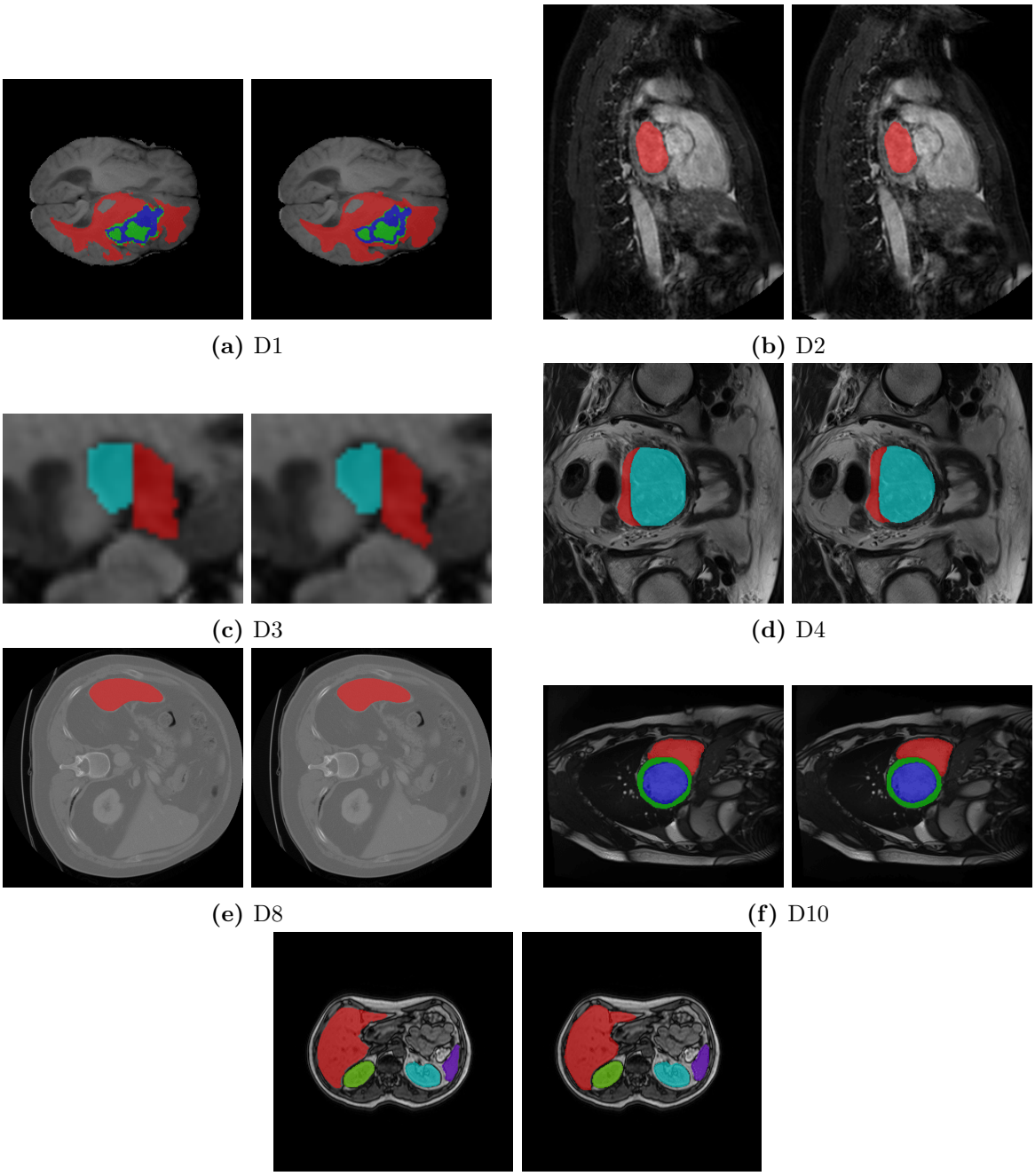
Other ablation studies

The other four ablation studies did not obtain any interesting results. There was no significant impact to the performance of the model when changing these parameters. These four parameters all pertain to the transformer layers, which are important to adding global context to the features. So, it is possible that this is not important to dataset D3 as accurate segmentation's can be achieved using relatively local features. I found that comparable accuracy can be achieved even if the transformer layers are removed entirely. However, it is expected that the transformer will play an important role in larger images, where more global context is required for an accurate segmentation. More research is required to examine the effects of the transformer layers datasets with larger images. As such, I will simply use the following values, which were found to produce good results in the TransU-Net[7]

- Number of hidden neurons in the MLP: 3072
- Number of heads (k): 12
- Number of transformer layers (n): 12
- Number of embeddings (T): 768

6.2 Model performance on the datasets

The model was trained on all eleven datasets shown in Table 1 and no manual adjustments were made to the design between each dataset. Unfortunately, the model was only successfully able to learn on 7 on the datasets, while on the other four no meaningful learning was observed.



(a) D1 **(b)** D2

(c) D3 **(d)** D4

(e) D8 **(f)** D10

(g) D11

Figure 7: Example segmentation's produced by the model (right), along with the corresponding ground truth label (left). All of these examples are from the validation dataset, meaning the model was not trained on these examples, so they are a good indication of the models performance. **a)** edema (red), non-enhancing tumour (green), enhancing tumour (blue)[2]. **b)** left atrium (red)[2]. **c)** anterior hippocampus (red), posterior hippocampus (cyan)[2]. **d)** peripheral zone (red), transitional zone (cyan)[2]. **e)** spleen (red)[2]. **f)** right ventricle (red), myocardium (green), left ventricular cavity (blue)[18]. **g)** liver (red), left and right kidney (cyan and green respectively), spleen (purple)[3]

Table 6 shows the accuracy of the model on the successful dataset, and Figure 7 shows example visualisations of the model output, along with the ground truth labels. I have compared my design to both the nnU-Net and the TransU-Net, as these are current state of the art, and my design is inspired by both of these models. My design outperforms the nnU-Net on D1 (Brain Tumours) and D3 (Hippocampus) and is very close to the nnU-Net for D10 (ACDC) and D11 (CHAOS). My model also outperforms the TransU-Net on D10 (ACDC) (the only dataset I can compare with the TransU-Net). The model performs reasonably well on D2 (Heart), when compared to the nnU-Net, and for D4 (Prostate) and D8 (Spleen), my model is significantly worse than the nnU-Net, but still provides reasonably accurate segmentation's.

	Dice Score (%)		
Dataset	My design	nnU-Net	TransU-Net
D1	74.62	74.11	-
D2	87.59	93.29	-
D3	91.32	88.91	-
D4	68.34	75.94	-
D8	83.64	97.24	-
D10	90.88	92.28	89.71
D11	90.38	91.97	-

Table 6: Model results on the datasets that showed meaningful learning, compared with current state of the art designs. All results are shown using the Dice Score Coefficient (%). Full results for each class within each dataset is shown in Appendix B

While this design does show promising results on the datasets discussed above, unfortunately it failed to produce segmentation's on four of the tested datasets. The reason for this is primarily due to large class imbalances, and large image sizes. Table 7 shows the relative frequency of the foreground classes to the background class, as well as the image size for both the successful and unsuccessful datasets. All of the unsuccessful datasets have a very low frequency of the foreground class, and very large image sizes. Dataset D2 and D8 also have low foreground frequencies, which is likely the reason the model did not perform as well as the nnU-Net on these datasets.

To understand why the model fails to learn on these datasets, first consider just the 3D model. When the images are very large, they must be split up into a lot of patches in order to meet memory constraints for the batches. This results in each patch covering a small percentage of the total image volume. So, if the foreground class is rare, and the patches only cover a small part of the image, it is likely that any particular patch will not contain any foreground class. Given that the batch size can be as small as 2, this results in a significant amount of batches containing no foreground class at all.

For the 2D model, the situation is similar. The 3D image is split up into a series of 2D slices, and because the foreground objects are relatively small, very few of the slices contain the foreground class, resulting in a significant number of batches containing no or very little foreground class. This makes it very difficult for the model to learn how to segment the foreground classes, since it sees them so rarely. This results in the model learning to always

predict the background class, which results in a dice score of 0.00.

Table 7: Relative frequency of the foreground classes, as well as the image size for each of the datasets. Very low frequencies of the foreground class and large image sizes result in poor performance for the model

Successful Datasets		
Dataset	Foreground frequency	Image size
D1	1.16%	$160 \times 187 \times 149$
D2	0.40%	$256 \times 320 \times 130$
D3	5.28%	$43 \times 59 \times 47$
D4	2.75%	$370 \times 370 \times 24$
D8	0.47%	$631 \times 631 \times 324$
D10	3.73%	$275 \times 307 \times 22$
D11	4.61%	$249 \times 313 \times 47$
Unsuccessful Datasets		
Dataset	Foreground frequency	Image size
D5	0.04%	$637 \times 637 \times 512$
D6	0.23%	$623 \times 623 \times 210$
D7	0.47%	$626 \times 626 \times 478$
D9	0.06%	$640 \times 640 \times 304$

7 Future Work

There are a number of improvements that can be made to the model to improve its accuracy. Firstly, class imbalances in large images are a big problem for the model, and can result in no meaningful learning occurring. This can be resolved by implementing class oversampling. Class oversampling involves selecting patches which contain the foreground class with a higher probability than patches without any foreground class. The probabilities are determined such that a certain proportion of all patches trained with contain the foreground class. Isensee et al. suggests oversampling such that 2/3 of all patches contain the foreground class[10], so this is were I would start, and then this could be tweaked to ensure it produces good results. This will resolve the issue of class imbalance, as the data the model is trained on will no longer see such an extreme class imbalance. It is important that some patches still contain no foreground class, so that the model still learns how to identify the background.

Another improvement that could used is implementing a cascaded U-Net design[10]. A Cascaded U-Net design will improve the accuracy of the model when the images are very large. If the images are large, then they will be split up into a lot of patches in order to meet the GPU memory constraints. Thus, each patch only covers a small portion of the total image, and the model is unable to obtain enough context from a single patch. This results in poor performance for larger images.

A cascaded U-Net design uses two networks, each with the same design that I have used in this report. The image is first downsampled to a low resolution image, and passed through

the first model. The first model is trained to produce a low resolution segmentation. Since the image is now smaller, the patches cover a larger portion of the total image, meaning the model can obtain more global context, resulting in better accuracy. This low resolution segmentation is then upsampled to the original image size, and concatenated with the original image. This is then passed through the second model, by splitting it up into patches. Again, the patches will be quite small relative to the size of the image, however, now that the model has access to the low resolution segmentation, it can perform well despite this. This is because it just has to fine tune the segmentation using local contextual information.

The Cascaded U-Net design requires two models, so it will take twice as long to train. As such, it would only be used when necessary, if the images are very large. The use of the cascaded design over the regular design would be triggered automatically during the preprocessing stage.

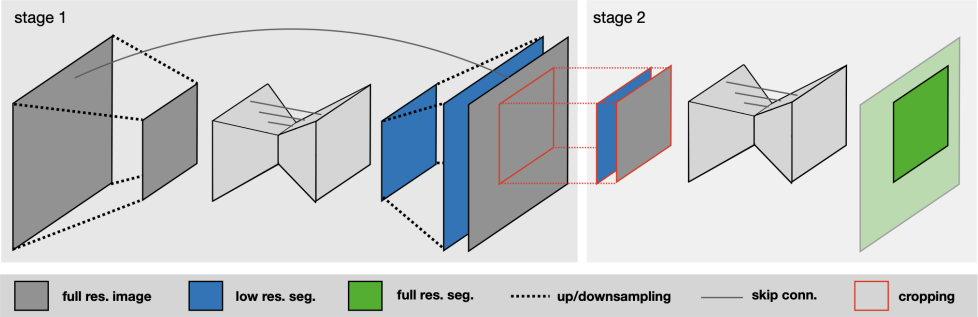


Figure 8: Schematic diagram of the Cascaded U-Net[10]

There a a few other minor changes which may result in improved accuracy. The first is to train for more epochs. I was limited in how long I could train the model for, due to time limitations, but using a lower learning rate and longer training time may result in a marginal improvement in performance. Most datasets had reached convergence by the time training had finished, however some of the more difficult datasets had not completed training. For example, the validation accuracy on dataset D1 (Brain Tumours) was still increasing when training finished.

The final improvement would be to train multiple models on different folds of the dataset. Isensee et al. used five fold cross validation in the nnU-Net, meaning five models were trained using different folds, and the resulting test set is segmented using the consensus from the five models[10]. If multiple GPU's are available, then each model can be trained in parallel on different GPU's, meaning the increase in training time will be negligible. However, if only one GPU is available, this technique may not be viable, and just one model would need to be used. As such, I would implement this as an optional technique, which could be used depending on the hardware available.

Also, when performing the ablation studies on dataset D3, it was found that the transformer provided little to no benefit. It is likely that on larger images, the transformer did in fact provide improved accuracy, but more studies would be required to determine if this is the case. As such, in the future, more ablation studies should be conducted on different datasets to determine how the transformer impacts the models accuracy, and how changing the different hyper parameters related to the transformer influences the model performance.

8 Conclusion

Image segmentation is a rapidly advancing field. A lot of research has focused on fully convolutional networks such as the U-Net, but recent studies have shown that incorporating transformers can provide excellent global feature recognition. My design harnesses the features of the transformer to create a U-Net based model that can extract more global contextual information than a FCN, providing superior segmentation accuracy. My design also automatically adapts to the given dataset, allowing it to perform well on a range of datasets. It does this by automatically adjusting the preprocessing and postprocessing tasks, as well as the model architecture based upon the dataset. However, due to the models inability to adjust for extreme class imbalances in large images, the model is unable to learn on some datasets. Future work will involve implementing class oversampling and a cascaded U-Net design, as these measures have been shown to provide robust training despite class imbalances in large images. While the model under performs in datasets with low foreground class frequency, I have found that it does perform well on all other datasets tested, and even outperformed the current state of the art on a couple of the datasets. As such, if the class imbalance issue can be resolved, this design may provide comparable or even superior performance to current state of the art techniques.

Bibliography

[1] Amber L. Simpson et al. *A large annotated medical image dataset for the development and evaluation of segmentation algorithms*. 2019. arXiv: 1902.09063 [cs.CV].

[2] *Medical Segmentation Decathlon*. URL: http://medicaldecathlon.com/.

[3] A. Emre Kavur et al. "CHAOS Challenge - combined (CT-MR) healthy abdominal organ segmentation". In: *Medical Image Analysis* 69 (Apr. 2021), p. 101950. ISSN: 1361-8415. DOI: https://doi.org/10.1016/j.media.2020.101950. URL: http://www.sciencedirect.com/science/article/pii/S1361841520303145.

[4] *MICCAI Challenge on Circuit Reconstruction from Electron Microscopy Images*. URL: https://cremi.org/.

[5] Mohammad Hesam Hesamian et al. "Deep Learning Techniques for Medical Image Segmentation: Achievements and Challenges". In: *Journal of Digital Imaging* 32 (May 2019). DOI: 10.1007/s10278-019-00227-x.

[6] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *CoRR* abs/1505.04597 (2015). arXiv: 1505.04597. URL: http://arxiv.org/abs/1505.04597.

[7] Jieneng Chen et al. "TransUNet: Transformers Make Strong Encoders for Medical Image Segmentation". In: (2021). arXiv: 2102.04306 [cs.CV].

[8] Ashish Vaswani et al. "Attention Is All You Need". In: *CoRR* abs/1706.03762 (2017). arXiv: 1706.03762. URL: http://arxiv.org/abs/1706.03762.

[9] Alexey Dosovitskiy et al. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *CoRR* abs/2010.11929 (2020). arXiv: 2010.11929. URL: https://arxiv.org/abs/2010.11929.

[10] Fabian Isensee et al. "nnU-Net: a self-configuring method for deep learning-based biomedical image segmentation". In: *Nature Methods* 18 (2021), pp. 203–211. DOI: https://doi.org/10.1038/s41592-020-01008-z.

[11] Quanshi Zhang et al. "Interpreting CNN Knowledge via An Explanatory Graph". In: *Association for the Advancement of Artificial Intelligence* (2018), pp. 4454–4463.

[12] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

[13] Yuxin Wu and Kaiming He. "Group Normalization". In: *CoRR* abs/1803.08494 (2018). arXiv: 1803.08494. URL: http://arxiv.org/abs/1803.08494.

[14] Jia Deng et al. "ImageNet: A large-scale hierarchical image database". In: (2009), pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

[15] Siyuan Qiao et al. "Micro-Batch Training with Batch-Channel Normalization and Weight Standardization". In: *CoRR* abs/1903.10520 (2019). arXiv: 1903.10520. URL: http://arxiv.org/abs/1903.10520.

[16] Nabila Abraham and Naimul Mefraz Khan. "A Novel Focal Tversky loss function with improved Attention U-Net for lesion segmentation". In: *CoRR* abs/1810.07842 (2018). arXiv: 1810.07842. URL: http://arxiv.org/abs/1810.07842.

[17] Ashia C. Wilson et al. *The Marginal Value of Adaptive Gradient Methods in Machine Learning*. 2018. arXiv: 1705.08292 [stat.ML].

[18] Olivier Bernard et al. "Deep Learning Techniques for Automatic MRI Cardiac Multi-Structures Segmentation and Diagnosis: Is the Problem Solved?" In: *IEEE Transactions on Medical Imaging* 37.11 (2018), pp. 2514–2525. DOI: 10.1109/TMI.2018.2837502.

[19] Ali Emre Kavur et al. *CHAOS - Combined (CT-MR) Healthy Abdominal Organ Segmentation Challenge Data*. Version v1.03. Zenodo, Apr. 2019. DOI: 10.5281/zenodo.3362844. URL: https://doi.org/10.5281/zenodo.3362844.

[20] Seyed Raein Hashemi et al. "Tversky as a Loss Function for Highly Unbalanced Image Segmentation using 3D Fully Convolutional Deep Networks". In: *CoRR* abs/1803.11078 (2018). arXiv: 1803.11078. URL: http://arxiv.org/abs/1803.11078.

Appendix A

Ablation Studies

The following tables show the complete results of the ablation studies.

Table A.1: Model performance on D3 for a range of α values

	Dice Score		
α	Anterior	Posterior	**Average**
0.0	0.881	0.864	0.873
0.1	0.880	0.866	0.873
0.2	0.879	0.863	0.871
0.3	0.871	0.857	0.864
0.7	0.859	0.842	0.851
0.9	0.842	0.825	0.832

Table A.2: Model performance on D3 for a range of γ values

	Dice Score		
γ	Anterior	Posterior	**Average**
0.4	0.878	0.861	0.869
0.5	0.878	0.865	0.871
0.6	0.882	0.867	0.874
0.7	0.877	0.864	0.871
0.8	0.881	0.867	0.874
0.9	0.876	0.863	0.870
1.0	0.876	0.861	0.868
1.1	0.869	0.855	0.862
1.2	0.876	0.860	0.868
1.3	0.874	0.861	0.868

Table A.3: Model performance on D3 for a range of β values

	Dice Score		
β	Anterior	Posterior	**Average**
0.0	0.854	0.830	0.840
0.25	0.871	0.861	0.866
0.5	0.882	0.867	0.874
0.75	0.869	0.850	0.857
1.0	0.835	0.815	0.822
1.25	0.800	0.788	0.790
1.5	0.755	0.740	0.746
1.75	0.700	0.672	0.681
2.0	0.690	0.632	0.661

Table A.4: Model performance on D3 for a range of MLP Hidden layer dimensions

	Dice Score		
MLP Hidden layer dimension	Anterior	Posterior	**Average**
1024	0.881	0.866	0.873
2048	0.879	0.866	0.872
3072	0.882	0.867	0.874
4096	0.880	0.865	0.873
5120	0.880	0.865	0.872

Table A.5: Model performance on D3 for a range of T values

	Dice Score		
T	Anterior	Posterior	**Average**
480	0.878	0.861	0.870
576	0.881	0.866	0.873
672	0.879	0.864	0.871
768	0.882	0.867	0.874
864	0.878	0.861	0.870

Table A.6: Effect of model performance on changing the number of heads, k

	Dice Score		
k	Anterior	Posterior	**Average**
1	0.884	0.866	0.875
2	0.881	0.864	0.873
4	0.880	0.866	0.873
6	0.883	0.867	0.875
8	0.877	0.864	0.870
10	0.879	0.864	0.871
12	0.882	0.867	0.874
14	0.878	0.862	0.870
16	0.878	0.862	0.870
18	0.881	0.866	0.873

Table A.7: Effect of model performance on changing the number of transformer layers, n

	Dice Score		
n	Anterior	Posterior	**Average**
0	0.878	0.864	0.871
1	0.878	0.863	0.870
2	0.880	0.866	0.873
4	0.879	0.864	0.871
6	0.881	0.868	0.874
8	0.878	0.862	0.870
10	0.879	0.863	0.871
12	0.882	0.867	0.874
14	0.878	0.865	0.872
16	0.878	0.863	0.871

Appendix B

Performance on each dataset

D1 – Brain Tumour

Table B.1: Summary of dataset statistics for dataset D1. Median and maximum shape refer to the shapes at target spacing. The maximum shape is the shape all images are padded to before being passed through the model

	2D	3D
Median Shape	138×169	$138 \times 169 \times 138$
Maximum Shape	160×187	$160 \times 187 \times 149$
Patch Size	160×187	$80 \times 47 \times 75$
Batch Size	21	2
Relative class frequency	98.84%, 0.73%, 0.20%, 0.23%	
Training set size	436	
Validation set size	48	

Table B.2: Model performance on dataset D1

	Dice Score (%)			
Model	Edema	Non-Enhancing Tumour	Enhancing Tumour	**Average**
My Design	76.60	64.07	83.20	74.62
nnU-Net	81.01	61.99	79.34	74.11

D2 – Heart

Table B.3: Summary of dataset statistics for dataset D2. Median and maximum shape refer to the shapes at target spacing. The maximum shape is the shape all images are padded to before being passed through the model

	2D	3D
Median Shape	232×320	$232 \times 320 \times 115$
Maximum Shape	256×320	$256 \times 320 \times 130$
Patch Size	232×320	$128 \times 80 \times 65$
Batch Size	32	2
Relative class frequency	99.60%, 0.40%	
Training set size	18	
Validation set size	2	

Table B.4: Model performance on dataset D2

	Dice Score (%)	
Model	Left Atrium	**Average**
My Design	87.59	87.59
nnU-Net	93.29	92.29

D3 – Hippocampus

Table B.5: Summary of dataset statistics for dataset D3. Median and maximum shape refer to the shapes at target spacing. The maximum shape is the shape all images are padded to before being passed through the model

	2D	3D
Median Shape	36×50	$36 \times 50 \times 35$
Maximum Shape	43×59	$43 \times 59 \times 47$
Patch Size	43×59	$43 \times 59 \times 47$
Batch Size	463	13
Relative class frequency	94.72%, 2.75%, 2.54%	
Training set size	234	
Validation set size	26	

Table B.6: Model performance on dataset D3

	Dice Score (%)		
Model	Anterior	Posterior	**Average**
My Design	91.74	90.89	91.32
nnU-Net	89.75	88.07	88.91

D4 – Prostate

Table B.7: Summary of dataset statistics for dataset D4. Median and maximum shape refer to the shapes at target spacing. The maximum shape is the shape all images are padded to before being passed through the model

	2D	3D
Median Shape	319×320	$319 \times 320 \times 20$
Maximum Shape	370×370	$370 \times 370 \times 24$
Patch Size	370×370	$93 \times 185 \times 24$
Batch Size	9	2
Relative class frequency	97.25%, 0.70%, 2.05%	
Training set size	29	
Validation set size	3	

Table B.8: Model performance on dataset D4

	Dice Score (%)		
Model	Peripheral Zone	Transitional Zone	**Average**
My Design	53.36	83.31	68.34
nnU-Net	66.11	85.77	75.94

D5 – Lung

Table B.9: Summary of dataset statistics for dataset D5. Median and maximum shape refer to the shapes at target spacing. The maximum shape is the shape all images are padded to before being passed through the model

	2D	3D
Median Shape	512×512	$512 \times 512 \times 252$
Maximum Shape	637×637	$637 \times 637 \times 512$
Patch Size	637×637	$69 \times 138 \times 111$
Batch Size	6	2
Relative class frequency	99.96%, 0.04%	
Training set size	57	
Validation set size	7	

Table B.10: Model performance on dataset D5

	Dice Score (%)	
Model	Cancer	**Average**
My Design	0.00	0.00
nnU-Net	72.41	72.41

D6 – Pancreas

Table B.11: Summary of dataset statistics for dataset D6. Median and maximum shape refer to the shapes at target spacing. The maximum shape is the shape all images are padded to before being passed through the model

	2D	3D
Median Shape	512×512	$512 \times 512 \times 96$
Maximum Shape	623×623	$626 \times 626 \times 210$
Patch Size	623×623	$92 \times 92 \times 124$
Batch Size	6	2
Relative class frequency	99.77%, 0.2%, 0.03%	
Training set size	253	
Validation set size	28	

Table B.12: Model performance on dataset D6

	Dice Score (%)		
Model	Pancreas	Cancer	**Average**
My Design	0.00	0.00	0.00
nnU-Net	82.14	54.28	68.21

D7 – Hepatic Vessels

Table B.13: Summary of dataset statistics for dataset D7. Median and maximum shape refer to the shapes at target spacing. The maximum shape is the shape all images are padded to before being passed through the model

	2D	3D
Median Shape	512×512	$512 \times 512 \times 150$
Maximum Shape	623×623	$626 \times 626 \times 478$
Patch Size	623×623	$70 \times 140 \times 107$
Batch Size	6	2
Relative class frequency	99.53%, 0.11%, 0.36%	
Training set size	273	
Validation set size	30	

Table B.14: Model performance on dataset D7

	Dice Score (%)		
Model	Vessel	Tumour	**Average**
My Design	0.00	0.00	0.00
nnU-Net	64.85	72.50	68.67

D8 – Spleen

Table B.15: Summary of dataset statistics for dataset D8. Median and maximum shape refer to the shapes at target spacing. The maximum shape is the shape all images are padded to before being passed through the model

	2D	3D
Median Shape	512×512	$512 \times 512 \times 187$
Maximum Shape	631×631	$631 \times 631 \times 324$
Patch Size	631×631	$80 \times 160 \times 82$
Batch Size	6	2
Relative class frequency	99.53%, 0.47%	
Training set size	37	
Validation set size	4	

Table B.16: Model performance on dataset D8

	Dice Score (%)	
Model	Spleen	**Average**
My Design	83.64	83.64
nnU-Net	97.24	97.24

D9 – Colon

Table B.17: Summary of dataset statistics for dataset D9. Median and maximum shape refer to the shapes at target spacing. The maximum shape is the shape all images are padded to before being passed through the model

	2D	3D
Median Shape	512×512	$512 \times 512 \times 150$
Maximum Shape	640×640	$640 \times 640 \times 304$
Patch Size	640×640	$82 \times 82 \times 156$
Batch Size	6	2
Relative class frequency	99.94%, 0.06%	
Training set size	113	
Validation set size	13	

Table B.18: Model performance on dataset D9

	Dice Score (%)	
Model	Cancer	**Average**
My Design	0.00	0.00
nnU-Net	49.37	49.37

D10 – Automated Cardiac Diagnosis Challenge

Table B.19: Summary of dataset statistics for dataset D10. Median and maximum shape refer to the shapes at target spacing. The maximum shape is the shape all images are padded to before being passed through the model

	2D	3D
Median Shape	237×208	$237 \times 208 \times 18$
Maximum Shape	275×307	$275 \times 307 \times 22$
Patch Size	275×307	$275 \times 154 \times 22$
Batch Size	31	2
Relative class frequency	96.27%, 1.22%, 1.26%, 1.25%	
Training set size	180	
Validation set size	20	

Table B.20: Model performance on dataset D10

	Dice Score (%)			
Model	Right Ventricle	Myocardium	Right Ventricular Cavity	**Average**
My Design	88.87	87.30	96.45	90.88
nnU-Net	91.45	90.59	94.79	92.28
TransU-Net	88.86	84.53	95.73	89.71

D11 - Combined Healthy Abdominal Organ Segmentation

Table B.21: Summary of dataset statistics for dataset D11. Median and maximum shape refer to the shapes at target spacing. The maximum shape is the shape all images are padded to before being passed through the model

	2D	3D
Median Shape	195×262	$195 \times 262 \times 45$
Maximum Shape	249×313	$249 \times 313 \times 47$
Patch Size	249×313	$125 \times 79 \times 47$
Batch Size	16	2
Relative class frequency	95.39%, 3.29%, 0.37%, 0.37%, 0.58%	
Training set size	36	
Validation set size	4	

Table B.22: Model performance on dataset D11

	Dice Score (%)				
Model	Liver	Right Kidney	Left Kidney	Spleen	**Average**
My Design	94.23	90.03	92.90	84.38	90.38
nnU-Net	93.45	92.89	92.12	89.40	91.97