University of Glasgow | School of Computing Science

Honours Individual Project Dissertation

Application of Perceiver IO Network to Abdominal Multi-Class, Multi-Modality Medical Image Segmentation

Jim Carty

March 29th 2023

Abstract

Segmenting medical images into separate classes is the first step in diagnosing and treating many ailments. However, they are costly to produce since highly trained professionals must create them, and requiring lots of time to produce. As such, finding a way automatically create medical image segmentations would allow for faster treatment of patients while simultaneously reducing the workload on medical practitioners.

Current solutions suffer from several persistent generalisability issues and significant computation requirements. To tackle both of these, this paper introduces a novel application of the Perceiver IO architecture to biomedical segmentation and evaluates its effectiveness and generalisability. The model is compared to the state-of-the-art using a standardised dataset and public online leaderboard.

Evaluations show that the model is receptive to the novel extensions and derivative-free parameter optimisation implemented in this paper. Recursive inferences improved the macro-averaged Dice coefficient (DSC) by 176.7%; overlapping inferences decreased Hausdorff distance by 24.9%; and parameter optimisation decreased a combined loss of cross-entropy and Dice coefficient by 48.7%. However, this paper's application is not competitive compared to other models, placing last on the AMOS22 leaderboard with a DSC of 0.0114. This paper, however, acts as a successful exploratory work, paving the way for future work that may prove to be more effective than the state-of-the-art.

Acknowledgements

Thanks to my partner Caitlin for her continuous support throughout my entire project; without her, I would have left this dissertation until the last minute if I finished it at all.

I also extend my thanks to my advisor Dr. Ali Gooya who helped me with my journey through the incredibly dense forest that is medical image analysis.

Finally, I would like to thank my parents who offered their help despite me (foolishly) never taking it.

Education Use Consent

Contents

1 Introduction

In medicine, segmentations of medical images are the first step in quickly and accurately analysing patients for several ailments (Chen et al. 2018). These segmentations can be, for example, of brains and brain lesions (Monteiro et al. 2020) inside of cancer diagnosis, or the abdomen in the diagnosis of diseases of the central organs (Zhao et al. 2021).

Image segmentations such as these contain vast amounts of information that allow for several complex analyses. In the medical imaging field, segmentations can be used to compute quantitative metrics and measures to describe and recognise symptoms of diseases. From these measures, a correct diagnosis can be given, and treatments can be quickly planned (Chen et al. 2018).

1.1 Motivations

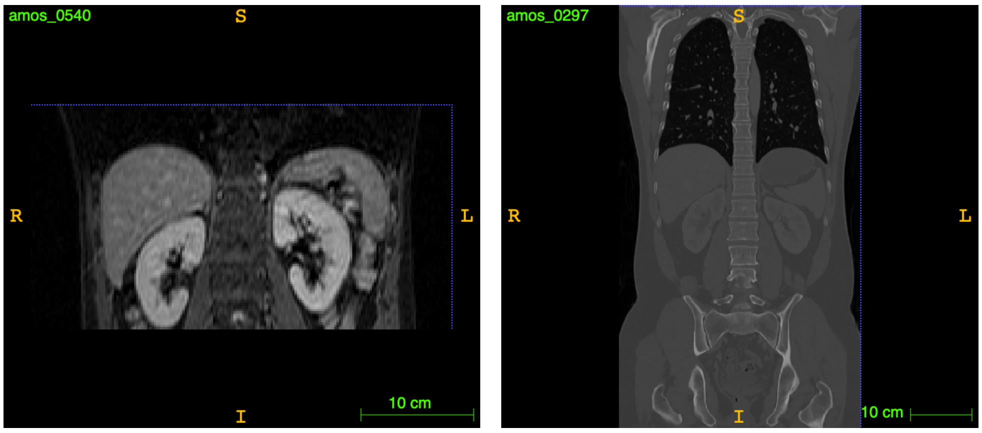
Creating these segmentations has been an area of active research for many years. This is because creating these segmentations for analysis is an incredibly time-consuming activity and must be completed by highly trained medical professionals. Decreasing the workload for these professionals is essential as it allows them to spend time on other important tasks. Further, different professionals may segment medical images differently, decreasing the objectivity of their labellings (Patil and Deore 2013). In turn, this reduces the accuracy of the diagnoses. Due to these factors, segmentations must be accurate and objective so that the efficacy of their use in diagnosis and treatment is not compromised. As such, identifying a method to accurately and automatically compute these segmentations is essential.

Furthermore, medical imaging has improved vastly throughout the last decade, allowing for higher resolution and less noisy images to be produced. Additionally, a range of techniques are used to create medical images; the most notable - and widely used - are Computed Tomography (CT) and Magnetic Resonance Imaging (MRI). However, different scans are produced from these different modalities, such as with differing resolutions and intensity ranges. Image properties can also vary between different makes and models of medical imaging machines; an MRI machine with more powerful magnets will produce scans of a higher resolution than one with less. Such possible differences between scans can be seen in Figure 1.1.

Physicians may also take scans of parts of a patient instead of larger scans of the whole patient, as larger scans take more time to produce. Putting forward an example, in the case of liver pain, a medical professional would only complete a partial abdominal scan - which would include organs between as the lungs and intestines - instead of a full scan that could include unnecessary areas such as the bladder. This would result in a “shorter” scan - say 30 “slices” tall instead of 300. Considering these different image properties, creating a bespoke system for each possible machine configuration and scan size is not viable (Ji et al. 2022). This described difference in scan size can also be seen in Figure 1.1.

1.2 Aims

From the above motivations, it can be seen that segmentations require vast amounts of domain-specific knowledge to create accurately. Further, creating a generic solution that can be transfer-



(a) MRI *(b) CT*

Figure 1.1: *Example MRI and CT scans centred on the kidneys showing possible differences in resolutions and scan heights; images 0540 and 0297 respectively from the AMOS22 dataset (Ji et al. 2022), visualised using ITK-SNAP (Yushkevich et al. 2006)*

able to different modalities and different configurations is difficult.

As such, this project seeks to:

- Employ a new method to create accurate segmentations that is transferable to multiple modalities and properties. We discuss the implementation and integration of this method in Chapter 5.
- Evaluate the above platform by providing metrics of the accuracy of the implemented architecture in these multiple circumstances. Comparisons to state-of-the-art systems will be given to show the relative effectiveness of the chosen system in Chapter 6

2 | Background

2.1 Classical Approaches

Upon first inspection, the problem of creating medical image segmentations appears simple. There are highly accurate high-resolution greyscale images that depict different regions of interest (ROI); why can we not simply use a classical image processing technique to extract these regions?

A threshold on these intensity values would be a high-speed and effective method for finding these different ROIs. Employing multithresholding to segment multiple classes would be a trivial extension. However, in practice, thresholding has been found to be incredibly susceptible to noise in the scans (Pham et al. 2000; Sharma et al. 2010; Rogowska 2000) and requires noticeable borders between ROIs that are not present in all circumstances (Patil and Deore 2013). Additionally, thresholding does not consider the spatial positions of classes, meaning that organs may be detected in areas that are not morphologically correct; or possible (Pham et al. 2000) - for example, confusing a right kidney with a left kidney, as shown in Figure 2.1.

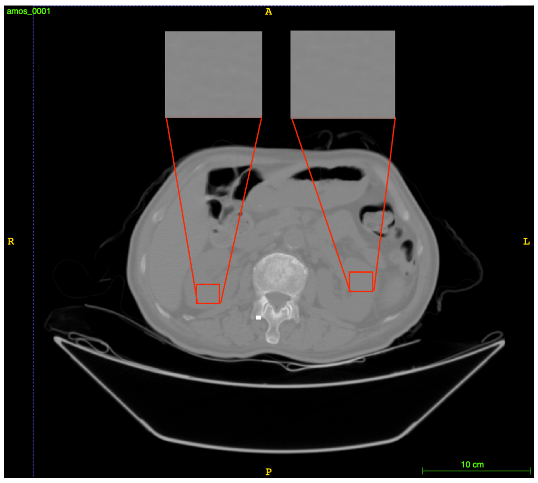


Figure 2.1: *Zoomed in left and right kidneys from image 0001 of the AMOS22 dataset (Ji et al. 2022), visualised using ITK-SNAP (Yushkevich et al. 2006)*

These advantages and disadvantages of thresholding are true for most other techniques used in early segmentation computation. Techniques such as k-means clustering and region-growing also suffer from these pitfalls relating to image noise and naivete to the spatial and morphological characteristics of the classes (Patil and Deore 2013; Sharma et al. 2010; Rogowska 2000; Pham et al. 2000).

Due to this, more complex and generalisable solutions in the form of machine learning and neural networks have had to be employed for this task. Additionally, with the recent explosion in the machine learning field, thanks to computation acceleration on GP-GPUs and techniques such as backpropagation, complex neural networks have become viable (Lee et al. 2017).

2.2 Convolutional Neural Network

A convolutional neural network (CNN) is a type of deep learning network that has been readily used and applied to computer vision problems, such as bounding box computation and semantic segmentation. These high-level problems usually require trained individuals to complete successfully and effectively - akin to physicians completing medical segmentations. However, CNNs have automated these processes to great effect (Wang et al. 2022; Briot et al. 2018). As such, CNNs have been identified as good candidates for generalisable medical image segmentation computation (Kayalibay et al. 2017).

2.2.1 U-Net

For segmentation tasks, CNNs have been applied using an encoder-decoder (autoencoder) architecture (Zhou et al. 2019). This architecture consists of multiple decreasing resolution convolutions to a single bottleneck, then near-symmetrical increasing resolution convolutions back to the original image resolution (Badrinarayanan et al. 2017). This allows the network to utilise the context of the neighbouring regions around the convolutions while maintaining high fidelity in the outputs.

An extension to CNN-based encoder-decoders has been the U-Net architecture. This network architecture relies on the regular encoder-decoder, with the addition of "skip connections". These skip connections allow the preservation of image data between the network's encoder and decoder (Ronneberger et al. 2015). This helps the network's decoder to be more informed when creating the resulting segmentation, improving accuracy. Figure 2.2 shows the U-Net architecture's encoder-decoder with skip-connections.

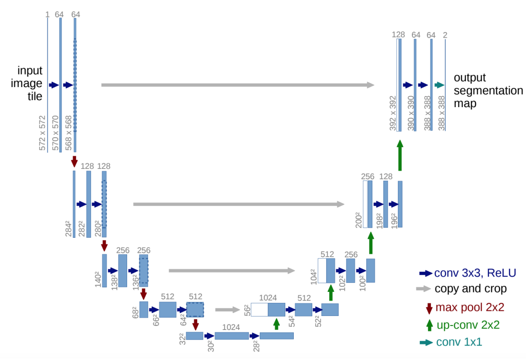


Figure 2.2: *The U-Net architecture as used by Ronneberger et al. (2015)*

In turn, the U-Net architecture is effective even when trained on fewer images (Ronneberger et al. 2015). This is incredibly important inside the medical imaging field as labelled data is challenging to collect due to the time required to label an image and the qualifications needed to do so accurately. Further, due to accelerations provided by GP-GPUs, the U-Net architecture is able to compute accurate segmentations in under a second.

However, due to how CNNs process data, very little information of the surrounding area typically persists. This is an issue as with no knowledge of the context around an area, the network cannot give an accurate prediction of what it is in the area. This lack of contextual information causes issues similar to those seen in the previously referenced example of determining left and right kidneys in Figure 2.1.

Despite being mitigated by the encoder-decoder and U-Net architectures, the sheer amount of data to process limits these networks (Tian et al. 2019). To this end, much research has been conducted on optimisations and improvements to the U-Net architecture to help it process more contextual and spatial information (Rahman et al. 2022; Zhang et al. 2021).

2.3 Spatial Awareness

This lack of contextual information available to the cutting-edge U-Net networks can also be seen as a lack of spatial awareness. Due to the prevalence of this issue, and with specific regard to large abdominal medical images, a lack of spatial awareness could reasonably decrease any network's ability to classify areas correctly. This lack of long-range contextual information that the network depends on would reasonably decrease the network's efficacy in correctly classifying regions.

2.3.1 Attention Maps

As humans, we cannot pay attention to everything we see all the time. Doing so would put undue strain on our brains to attempt to perceive and understand visual information that is not relevant. We, humans, have evolved to have limited attention that is spent only on the salient regions in a scene (Boynton 2005).

Applying this idea of "attention" to neural networks is enticing. Just as in humans, attention will help with utilising contextual information to limit what a network needs to focus on. This will help it use its processing power to predict the segmentations better as it would only be attending to what the segmentations are dependant on.

Attention takes many forms inside image processing; however, all pertain to creating an attention map. Attention maps can be thought of as heatmaps that describe where in the inputs the network should attend to. Typically, this requires a first bout of processing to create the map from the input image, then a second bout to compute the actual outputs. This second set of processing acts on the same inputs, either after having been transformed by the attention map (Wang, Zhou, Shen, Park, Fishman and Yuille 2019) or after having the intermediary states of the network transformed by the attention map (Kaul et al. 2019; Zhang et al. 2019).

Attention maps have shown competitive results with baseline U-Net models (Kaul et al. 2019), performing better in cases (Sinha and Dolz 2020), making them an exciting solution to the spatial awareness/long-range dependency issue.

2.3.2 Recurrent Networks

An additional method humans use to help create and preserve contextual information about a scene is instead of perceiving it in one single perception, we look around and move our attention throughout the scene, while remembering what we have seen before (Wang, Yu, Hugonot, Fua and Salzmann 2019).

This idea of recursively building up a knowledge of the inputs - built upon by previous knowledge - has also been applied to biomedical image segmentation. Specifically, recursive networks have been devised that iteratively take in previous inference results alongside the current input image to help it understand what it is currently perceiving based on what it has previously seen (Wang, Yu, Hugonot, Fua and Salzmann 2019).

In biomedical image segmentation, where scans can be extensive, often multiple network inferences must be completed on sets of individual slices to segment a full scan (Cao et al. 2023). Recurrently preserving contextual and spatial information on previous slices, and passing that information to the network has been found to improve a network's classifications (Mosinska et al. 2018).

2.4 Transformers

With these many extensions to existing architectures (adding recurrency and attention maps to the U-Net architecture) to allow them to perform tasks more effectively, it begs the question: Would an entirely different network architecture perform better than any form of U-Net?

A new network architecture has gained massive popularity recently, the transformer. Proposed initially by Vaswani et al. (2017), the transformer architecture has been effectively applied to several natural language processing tasks, becoming the backbone of modern language models such as BERT and GPT-3 (Devlin et al. 2018; Brown et al. 2020).

2.4.1 High-Level Transformer Architecture

Transformer models are based on self-attention layers which require three trainable matrices, a query, a key, and a value matrix. Attention is computed by multiplying the queries (input word vectors multiplied by the query matrix) and the keys (input word vectors multiplied by the key matrix), divided by a normalisation factor. This matrix is then normalised again using softmax and multiplied by the values (input word vectors multiplied by the value matrix). This computes an attention representation of the input by comparing all values to one another (Vaswani et al. 2017). A feed-forward layer - a multi-layer perceptron/fully connected neural network - can then use this self-attention representation to process the inputs into a set of outputs. This self-attention idea is used readily throughout the transformer architecture, used in both the input encoder and decoder, attending to the previous outputs of the feed-forward layer.

The transformer architecture allows for recursive inferences on previous outputs because the inputs and outputs are of the same size. This means many repetitions of the attention and feed-forward layers can be used; the number of these “transformer layers” is generally limited by available computer memory. BERT, for example, utilises 12 transformer layers for its “base” model (Devlin et al. 2018). This recurrency allows for multiple sets of matrix weights to be used, allowing for a deep and expressive set of latent values in the final layers of the network, allowing for impressive results, such as seen from the GPT-3 models (Brown et al. 2020).

Additionally, due to the latent representations used by transformers, using pre-trained models and fine-tuning them for specific tasks has been found to work incredibly effectively. Notably, the latent spaces commonly do not have to be pre-trained on the same style of data, for example, a language model pre-trained on translating languages, could have its latent space used as the basis of a masked language model (Dosovitskiy et al. 2020). This effective utilisation of pre-trained models allows the research into new applications for transformers to be accelerated as pre-trained models can be utilised in almost all cases, decreasing the requirement for large multi-day training times.

2.4.2 Vision Transformers

The effectiveness of these models has begged the question of if they could be applied to vision tasks. This was successfully done by Dosovitskiy et al. (2020) who described the now well-renowned vision transformer architecture. Vision transformers split the input image into multiple areas, which the transformer then processes as individual image patches - used like word vectors in natural language processing - to produce outputs (Dosovitskiy et al. 2020).

Since all areas on the image are processed simultaneously as image patches, the network can compute expressive self-attentions for the entire picture. This allows the vision transformers to be computationally faster than CNN-based networks, as transformers require far fewer operations to process the same image. Additionally, although faster, transformers can still effectively utilise the powerful attention mechanism. This has allowed vision transformers to outperform similarly sized, purely CNN-based models and even CNNs with attention maps in several computer vision tasks (Dosovitskiy et al. 2020). This performance increase is likely due to the transformer's ability to preserve long-range contextual information through self-attention and its ability to build an expressive latent space.

2.4.3 Application In Medical Segmentation

Due to the success of vision transformers in computer vision, their application in biomedical imaging segmentation was inevitable. Most initial applications applied the transformers alongside additional convolutional layers: TransUNet applied transformer layers to the lower-resolution convolutional layers, those closest to the bottleneck, inside the U-Net architecture. This application allows long-range features and context to persist throughout the lower-resolution layers to improve the network's contextual information. In turn, this resulted in impressive results over similar CNN-only U-Net architectures (Chen et al. 2021), helping to highlight that transformers can have a good application and work effectively within the biomedical segmentation setting (Huang et al. 2021).

Further applications of transformers within the U-Net architecture allowed for additional improvements in results. UNETR by Hatamizadeh et al. (2022) applies transformers by replacing the convolutions in the encoder area of the U-Net architecture, fully utilising the expressive of its latent values throughout the network. This produced exciting results over the baseline CNN U-Net and other transformer-assisted U-Nets, such as TransUNet (Hatamizadeh et al. 2022).

2.4.4 Swin Transformer

Alongside this integration of transformers into medical image segmentation, work to further refine vision transformers inside computer vision continued. Specifically, a research team inside of Microsoft created the Swin transformer as a different approach to transforming images (Liu et al. 2021).

The corresponding paper notes that vision attention differs from natural-language attention. As such, applying natural-language style attention architectures to visual processing tasks would be less effective than what transformers allow for. Due to this, the Swin-transformer paper proposed a technique for applying the network's transformers in a sequential hierarchy. The paper found that this novel method can improve vision transformers' performances in several computer vision tasks, including semantic segmentation (Liu et al. 2021).

2.4.5 Swin-Unet

These improved Swin vision transformers, specialised for working on image data, have also been applied successfully to working with medical imaging data. This has been done by replacing the convolutions in a U-Net model with Swin transformer layers as part of Swin U-Net (Cao et al. 2023).

This replacement replaces all convolutional layers with the hierarchical Swin-transformers. Patch-merging layers were used to decrease the image sizes through the network towards the bottleneck, replacing the max-pooling operations used by the traditional CNN-based U-Nets. Conversely, patch-expanding layers were used to increase the latent image resolutions when decoding the images, then passed through Swin-transformers in a symmetrical U-Net fashion (Cao et al. 2023). The architecture showing this relationship can be seen in Figure 2.3 and compared to the CNN-based U-Net architecture in Figure 2.2.

Overall, this architecture allowed for the use of long-range semantic information throughout the network and for the expressive latent factors of transformers to be utilised fully. These factors resulted in the Swin-UNet outperforming fully CNN-based and transformer-assisted

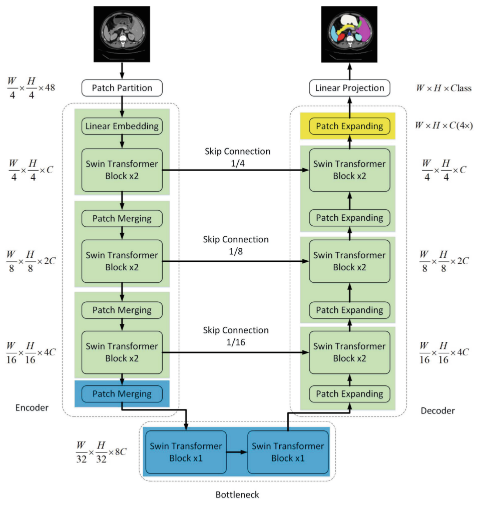


Figure 2.3: The Swin U-Net architecture as proposed by Cao et al. (2023)

approaches on a biomedical segmentation dataset, demonstrating the effectiveness of transformers in biomedical segmentation (Cao et al. 2023).

2.5 Perceiver

Transformers have excellent computational efficiency, allowing large models with billions of parameters to be feasible on current technology (Dosovitskiy et al. 2020). However, naturally, as transformer-based architectures got more complex and more transformer layers were used, even transformers required vast amounts of computational power. In BERT's case, training the model originally required 3 days to produce acceptable results (You et al. 2019).

This requirement for compute largely stems from the self-attention layer's query-key-value computations in transformers requiring quadratic time to compute; due to the matrix multiplication operations that make them up (Vaswani et al. 2017). Several transformer architectures have utilised domain-specific processing techniques to achieve linear time complexity w.r.t the inputs; however, they are not readily generalisable to other domains (Jaegle, Gimeno, Brock, Vinyals, Zisserman and Carreira 2021).

To combat this quadratic time complexity generally, the Perceiver model architecture was proposed (Jaegle, Gimeno, Brock, Vinyals, Zisserman and Carreira 2021).

2.5.1 Difference to Transformers

The Perceiver architecture is similar to classical transformers, with one key difference: Instead of propagating latent arrays, the same size as the inputs throughout the network, propagate a separated N-sized latent array throughout the transformer layers, incorporating the M-sized inputs with the latent array in a separate, prior layer.

These inputs are incorporated into the latent array using the latent array, a query vector from the latent array, and key and value vectors from the input array. The layer that completes this operation has been named the cross-attention layer. Figure 2.4 shows the interaction of the latent array, the inputs, and their associated query-key-value vectors. As with classic transformers, cross-attention and "latent" transformers layers can be sequentially stacked, allowing for deeper and more expressive results to be computed.

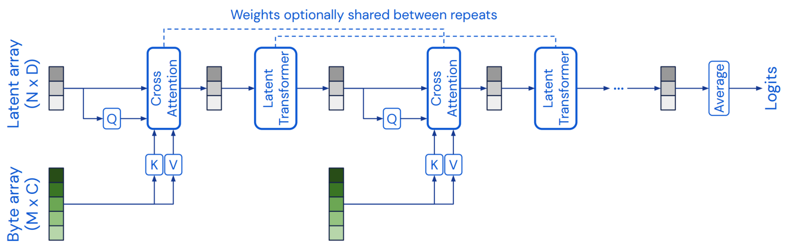


Figure 2.4: *The Perceiver architecture as proposed by Jaegle, Gimeno, Brock, Vinyals, Zisserman and Carreira (2021)*

Calculating the complexity of the Perceiver cross-attention layers results in $O(MN)$ required computations. From this value, and since the latent array is used as the input to the latent transformer (which still utilises quadratic self-attention layers), the overall time complexity for a Perceiver "layer" is $O(MN + LN^2)$; where L is the number of layers that make up the latent transformer layers.

From this new time complexity, no longer quadratic on the input size, adding more transformer layers - increasing L - does not increase compute requirements as much as classic transformers with $O(LM^2)$ time complexity, assuming $N < M$. This allows deeper networks to be built, allowing more expressive and novel processing of the input data, resulting in higher-quality outputs. This can be done using the same number of computations as shallower classical transformer models.

2.5.2 Exisiting Applications

Despite the improved computational complexity of the system through reduced computations, Perceiver-based networks perform similarly to other generic architectures. In an image recognition task, Perceiver produces competitive results compared to similarly sized CNN-based and vision transformer-based networks (Jaegle, Gimeno, Brock, Vinyals, Zisserman and Carreira 2021).

In a point cloud object classification task, Perceiver outperformed similarly sized CNN-based and transformer-based models. However, it did not achieve better results than the state-of-the-art models (Jaegle, Gimeno, Brock, Vinyals, Zisserman and Carreira 2021).

As with the demonstration of the transformer's capabilities in language and then image-based tasks, the question arose of if Perceiver networks could be similarly applied to semantic segmentation tasks.

2.5.3 Perceiver IO

The issue with applying the Perceiver model is that, as shown in Figure 2.4 the network's outputs results from an aggregation of the latent array - after being processed by the latent transformers. This means that extracting a segmentation of size M (assuming the same size as the inputs) from a latent array when $N \ll M$, is infeasible.

After the release of the Perceiver architecture, a general-purpose network architecture was proposed by Jaegle, Borgeaud, Alayrac, Doersch, Ionescu, Ding, Koppula, Zoran, Brock, Shelhamer et al. (2021). This Perceiver IO architecture describes a Perceiver-based network that can input and output independently sized-arrays, as shown in Figure 2.5.

These generic inputs and outputs are possible due to separate encode, process, and decode steps. Aside from the byte/input and latent arrays used by the Perceiver architecture, Perceiver IO also requires an "Output query array" (Jaegle, Borgeaud, Alayrac, Doersch, Ionescu, Ding, Koppula,

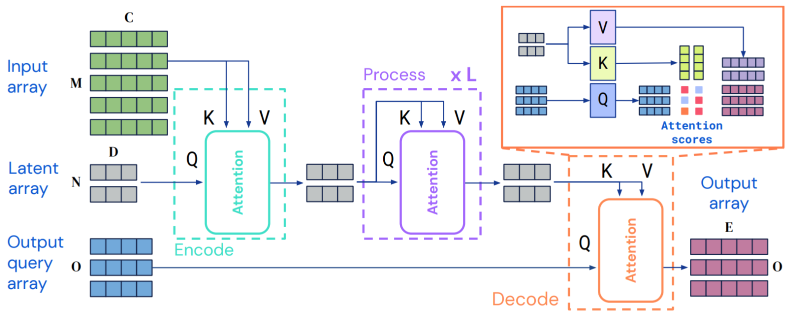


Figure 2.5: *The PerceiverIO architecture as proposed by Jaegle, Borgeaud, Alayrac, Doersch, Ionescu, Ding, Koppula, Zoran, Brock, Shelhamer et al. (2021)*

Zoran, Brock, Shelhamer et al. 2021). Using these arrays, the inputs can be encoded into the latent array using the same cross-attention layer described by Jaegle, Gimeno, Brock, Vinyals, Zisserman and Carreira (2021), with key and value arrays from the input array, and the query array from the latent array. This latent array can then be processed using multiple cross-attention and transformer layers, akin to the processing performed by the original Perceiver; only changed by the fact that the query-key-value arrays originate from only the latent array. Finally, the latent array is decoded by extracting key and value arrays from it and combining them with a query array from the output-sized output array. This forms an encoder-decoder-style structure, akin to that of a U-Net.

However, unlike U-Net, the Perceiver IO architecture is independent of its inputs. This allows the network to use impressively expressive latent arrays whilst computing requirements scale linearly with the size of the inputs and outputs. Additionally, due to the latent representations central to Perceiver-based architectures, it has been found that in a semantic segmentation task, Perceiver IO suffers less output degradation when generalised from one dataset to another (Choi and Ha 2022; Tan et al. 2022) than classical CNN-based approaches. In the case of medical image segmentation, Perceiver IO has yet to be applied.

3 Requirements

3.1 Perceiver IO

Considering prior work in the field of biomedical segmentations, and work that has been completed in adjacent areas, we can identify several requirements for a successful and generalisable system based on the issues faced in creating one:

- The system has to be able to process long-range spatial and contextual information (Section 2.1)
- There have been known success in the application of attentive, deep, transformer-based models (Section 2.4)
- There exists difficulties when generalising one model to multiple modalities (Section 1.1, Section 2.5.3)
- There is a severe lack of large datasets in comparison to other fields (Section 1.1)
- A deep, extensive, and complex level of intelligence is required to label the data accurately (Sections 1.1 and 2.4.5); and,
- The large size of individual scans makes employing network's whose shape is reliant on the inputs difficult (Section 1.1, Section 2.3.2)

With these points in mind, the Perceiver IO architecture is a good candidate for solving several of these issues currently faced in medical image segmentation while maintaining effectiveness compared to the state-of-the-art. Specifically, the Perceiver IO architecture allows:

- Long-range information to be utilised using the processes of self and cross attention
- The use of deep transformers, allowing for expressive latent values to be utilised, not normally allowed using conventional transformers
- For effective generalisation from one domain/dataset to another due to its latent representations
- For pretraining on larger datasets, and fine tuning on specific tasks - as allowed by its underlying transformer-style architecture
- For model computations to scale linearly with inputs, unlike transformers and CNN-based approaches that scale quadratically, limiting the complexity of their architectures

Directly addressing these issues highlights that Perceiver IO's application in medical image segmentation must be investigated. This application is the primary goal of this project: To apply this architecture to the field of medical image segmentation, and to evaluate its application in comparison with other state-of-the-art systems. Further, to assess the generalisability of such a system, we will test its application with multi-modality data, tackling the issue of innumerable bespoke models required by other architectures.

3.2 AMOS

To apply the Perceiver IO architecture to the case of medical image segmentation, a dataset must be used to train it on high-quality, multi-modality data robustly. Additionally, having access to other architectures' performances is required to accurately compare Perceiver IO's performance with state-of-the-art systems.

To this end, we identified the Abdominal Multi-Organ Segmentation (AMOS) dataset. The AMOS dataset consists of 500 CT and 100 MRI scans of varying sizes and resolutions containing up to 16 different segmentation classes of various sizes. These various class sizes help test a network's high fidelity and spatial awareness capabilities, including classes such as the liver and aorta.

This dataset was also used in the "Multi-Modality Abdominal Multi-Organ Segmentation Challenge 2022" (Ji et al. 2022), which utilised a standardised evaluation procedure to compare test dataset performance across different submissions robustly. The results of this evaluation procedure are made public on the AMOS22 challenge's leaderboards, helping facilitate direct comparisons. Additionally, the AMOS22 challenge requires the submission of a Docker container that can run inferences given a set of files to segment. This helps to require the reproducibility of results and helps to facilitate the productionisation of these models, hopefully helping to instigate their inclusion in end-to-end biomedical systems.

The AMOS22 challenge is also split into CT-only and CT and MRI submissions. Each submission requires the models to be trained and validated using different data, specified by the AMOS challenge's accompanying `dataset.json` metadata file. This allows the specialised and generalised performances of the model to be compared, showing a model architecture's usefulness as a generic tool for computing segmentations.

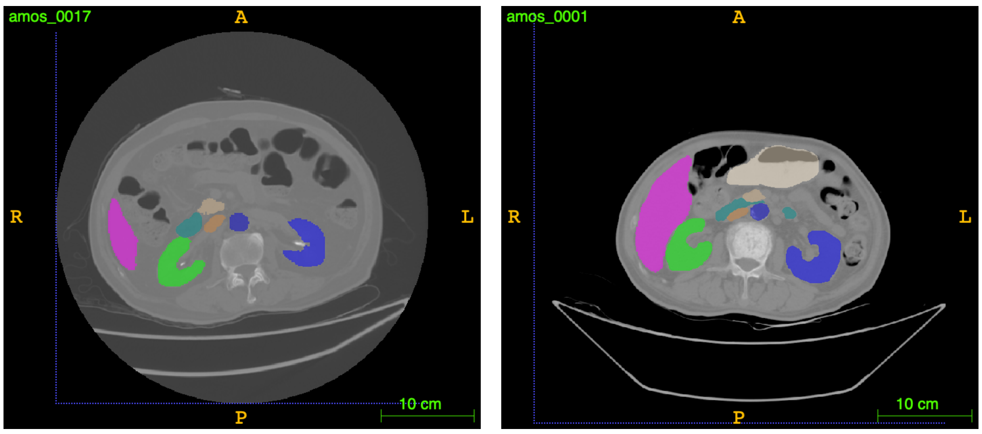
3.3 Contributions of This Paper

Overall, this paper seeks to novelly apply the Perceiver IO architecture to the field of medical image segmentation and evaluate its effectiveness and generalisability with a multi-modality dataset. This application will be made reproducible using data from the AMOS dataset. The AMOS dataset will also allow accurate comparisons with other state-of-the-art systems using standardised test data and docker image submissions.

4 Design

4.1 Preprocessing

The data provided by the AMOS22 challenge consists of raw, unnormalised and unprocessed scans. This means that the intensity ranges can vary between scans, the locations and orientations of organs can be different, the scans can be many different sizes, and the scans can have different resolutions. Figure 1.1 highlights the differences in heights, intensities and resolutions. Figure 4.1 has been included to highlight the issues with different patient and organ poses.



(a) *Figure showing a scan of a patient lying with a left lean*

(b) *Figure showing a scan of a patient lying with a slight right lean*

Figure 4.1: *Scans 0017 and 0001 from the AMOS dataset (Ji et al. 2022), visualised using ITK-SNAP (Yushkevich et al. 2006). Screenshots centred on the kidneys to highlight the different poses that patients and organs can have between scans*

Notoriously, machine learning requires good quality and consistent data to form meaningful representations and generalisation of the data; garbage in, garbage out. Due to the nature of the raw data, preprocessing it to produce consistent and valuable data is imperative.

4.1.1 Coregistration

To improve the consistency of the scans to allow for the utilisation of spatial information by the Perceiver, all scans were coregistered to a single scan. This target scan that all others were coregistered to was determined by identifying the largest scan in the dataset.

This was, importantly, not done by the number of voxels, but by true "world" size. This is because each dataset file consisted of the scan's voxels and a description of the conversion from the image to real life: the spacings between the pixel values/sizes of the voxels in x, y, and z. Since scans can differ noticeably in their resolutions, they can, in turn, differ massively in their spacings. This is most noticeable in the vertical axis, where scans can differ from 2 mm up to 5 mm between CT scans and even more when considering MRI scans as well.

With the scan containing the most features, the largest scan, all other scans could have a transformation from their frame to the largest scan's frame computed to match the scans as closely as possible. Using this transformation, all scans could have their spatial positions and orientations for inputting into the network.

4.1.2 Intensity Normalisation

After inspecting the dataset, many scans had two different background values, inside and outside the machine's scanning area - seen in Figure 4.1a. These two background values were made to be identical to avoid confusing the different types of backgrounds. This was done naively by capping the minimum intensity of all scans to the second lowest occurring value. After this, the scans were then normalised to between 0.0 and 1.0, before being fed into the model.

4.1.3 Multi-Modality

Additionally, the AMOS dataset combines all CT and MRI scans in one directory. The challenge's website states that MRI scans are scans with an ID value greater than 500. Therefore, splitting CT and MRI data can be easily done by checking the ID in the filename whilst iterating through the train, validation, and test image directories.

4.1.4 Pipeline

The preprocessing pipeline can be described diagrammatically, shown in Figure 4.2.

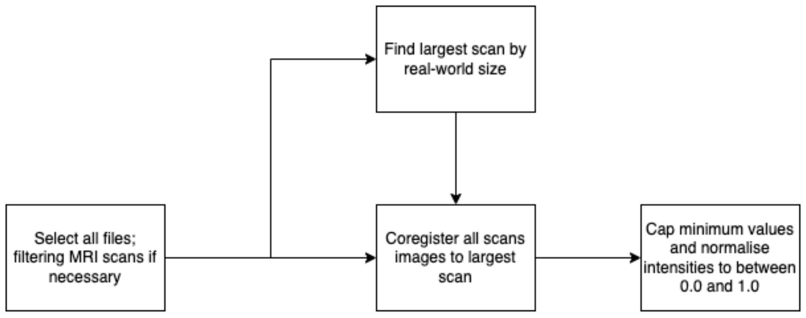

Find largest scan by real-world size
Select all files; filtering MRI scans if necessary
Coregister all scans images to largest scan
Cap minimum values and normalise intensities to between 0.0 and 1.0

Figure 4.2: *Diagram showing the preprocessing pipeline designed for transforming the raw AMOS CT and MRI scans to standardised and normalised inputs to the Perceiver network*

4.2 GPGPU Acceleration

Although the Perceiver architecture allows for better memory and computational efficiency while providing deep transformer layers, the proposed application for the Perceiver system will still require a large amount of computation to train and infer. Due to this, the network inferences and training will have to use GP-GPU acceleration to make training larger networks viable. Additionally, due to the depth of the network, a GPU with sufficient compute power and memory capacity will have to be used to train the network and compute model inferences.

4.3 Docker Inference Image

For submission to the AMOS challenge, and to allow reproducible training and inferencing, docker images will have to be created. To simplify the process of creating and using them, separate images will be created for each task: training, inferencing, and one for each ablation study conducted.

As mentioned in Section 4.2, these tasks will have to be acceleratable using a GPU. The NVIDIA Container Runtime allows for GPUs to be accessed and used by a container. Using this runtime, we can containerise and accelerate all tasks associated with the Perceiver network.

4.4 Slab-Wise Inferencing

Although the Perceiver IO architecture scales linearly with input and output sizes, hardware capabilities still limit how extensive the network's inputs can be. The higher the resolution of the scans, the more high-fidelity information can be passed to the network, allowing it to detect smaller organs better and more accurately. To allow for as high a resolution per inference as possible, we have chosen not to pass the full scan to the network.

This will reduce the network's ability to generate full-scan-level long-range attentions; however, the increased ability to identify smaller organs was deemed more critical. To help facilitate attention still being built up across multiple "slices", we decided that inputting "slabs" - multiple stacked slices - was the best approach. This allows high-resolution scans to be inputted without losing the ability to build up critical inter-slice, intra-slab attention.

4.5 Segmentation Specific Implementation

4.5.1 Recurrent Inputs

To help the Perceiver architecture perform competitively with other state-of-the-art networks, additional features will be implemented to optimise it for this particular application. Just as recursive CNN networks recursively take in prior inputs to provide additional context for the network, see Section 2.3.2, prior outputs will also be passed to the Perceiver network for each slab. This will be done by stacking the outputs from the prior slab on top of the current slab. This stacked prior output and current input should facilitate inter-slab dependencies being built up, further minimising the attention loss required when using higher-resolution scans.

4.5.2 Overlapping Inputs

In early prototypes of the system, large amounts of noise were seen on the edges of labels. To decrease this noise, the idea of combining multiple inferences over slices was put forward: Instead of each slab starting at the end of the last one, each slab would start a certain number of slices into the previous slab, overlapping the slabs. Unlike recursive slices, no additional inputs must be added, as only the start slice for the current slab is changing. This idea of overlapping slabs should also assist inter-slab attention further, although indirectly through the aggregation of inference outputs.

4.5.3 Input Diagram

To better describe these concepts, Figure 4.3 shows how both the recurrent outputs and overlapping scans work with respect to an input scan.

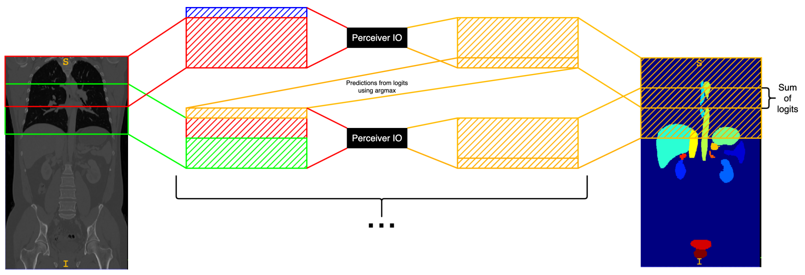


Figure 4.3: Diagram showing how the proposed ideas of overlapping and recursive inputs operate on a segmentation map. The image and labels used are of scan 0297 from the AMOS dataset (Ji et al. 2022), visualised using ITK-SNAP (Yushkevich et al. 2006). Red and green indicates the first and second scans being inferred; blue indicates a zero-valued matrix; and orange indicates network outputs.

4.6 Parameter Optimisation

To ensure that the network is evaluated fairly against other network architectures, the parameters for the network must be optimised. This will be done by training and evaluating the network on many sets of parameters and finding the most optimal. There are several parameters to optimise in the Perceiver architecture, as well as in this specific application of the architecture:

- The resolution of the scans being inputted to the network
- The number of slices per slab
- The number of recurrent slices
- The number of overlapping slices between slabs
- Number of latents
- Number of latent channels
- Number of encoder frequency bands
- Number of encoder cross-attention layers
- Number of encoder cross-attention heads
- Number of encoder self-attention blocks, and the layers per block
- Number of encoder self-attention heads
- Number of decoder cross-attention heads
- Number of query and key channels for each cross and self-attention head
- Number of value channels for each cross and self-attention head

All of these parameters must be optimised. We suspected that all variables would map to a logarithmic gain with their value and the efficacy of the model. However, increases in many of the parameters also result in significant increases in the memory required to store and train the network. As such, all values must be optimised together as increases in some parameters may result in more significant gains than others. For example, two models that take up the same amount of memory may have vastly different accuracies. Since model inferences must be completed on GPUs, the memory required to train the model has a hard limit, limiting how complex the model can be.

Many effective algorithms can optimise complex problems such as these. However, many of the most effective, such as gradient descent, require gradients to discover the topology of the parameter space to find the optimal solution. However, utilising these gradients may lead to non-optimal answers due to the inherent non-convexity of the parameter space; there may be many local minima that a gradient-based approach would get stuck in. Additionally, gradient-based approaches require significant amounts of memory to store the gradients to optimise the parameters correctly. As such, using a gradient-based approach may produce a non-optimal solution, with the additional cost of decreasing the network size. To avoid these concerns, a derivative-free approach was determined to be the best for optimising the network parameters.

5 Implementation

5.1 Perceiver IO

To avoid having to reimplement the Perceiver IO architecture, a GitHub repository was identified with an easily extensible implementation, developed originally by Krasser and Stumpf (2023). A fork of this repository was made, and extensions were implemented to allow the model to take in, process, analyse, and output biomedical scans and their segmentations.

5.1.1 Encoder

To allow the network to take in segmentations, an encoder had to be defined that would be able to take in images of a specific size. This size depended on the scans' resolution and the slabs' depth, which were set to be $256x256$ and four, respectively. This resolution was chosen as lower resolutions could not show the smaller organs required for good segmentations, and larger resolutions required far more memory than was available to compute. Additionally, this depth was chosen as it allowed for good inter-slice attention to be utilised, but with the memory still available to have extensive recursive inputs and deep models used.

5.1.2 Decoder

A decoder also had to be specified. This was far simpler, as the output was the same size as the inputted scan. Since there were multiple organs to classify, each pixel was treated like a separate classification task, requiring a vector of logits per pixel. This vector was the same length as the number of classes to classify, plus an unclassified/background class. In the case of the AMOS dataset, this resulted in vectors of length 16. To extract the predicted segmentation from these outputs, the $argmax$ of the $softmax$ was taken for each pixel.

5.1.3 Loss and Metrics

To compute the loss of the model's predictions, inspiration was taken from the Swin-UNet paper by Cao et al. (2023): A combined loss of cross-entropy loss and the Dice coefficient was used. In the Swin-UNet implementation, a weighted sum was used to combine the scores, shown in Equation 5.1.

$$loss(x, y) = 0.4 * cross\_entropy(x, y) + 0.6 * dice(argmax(softmax(x)), y), \tag{5.1}$$

where x is the model outputs with pixel-wise logits, y is the ground truth segmentation, $dice(w, y)$ is the Dice coefficient (described in Equation 6.1) between a predicted segmentation w and a ground truth segmentation y, and $cross\_entropy(x, y)$ is the cross-entropy loss between a segmentation with pixel-wise logits x, and a ground truth segmentation y.

The cross-entropy loss identifies the difference between each pixel's logits/probability and the ground truth value. This is useful for promoting more confident classification from the model during training. The Dice loss helps to exemplify the inaccuracies between the model's prediction and the ground truths. This is useful for training the model to predict accurately, more so than confidently.

Due to the small size of the dataset, we decided to adjust this combined loss function to use different weights to the Swin-UNet implementation, shown in Equation 5.2. This was done as we found that the network trained well using purely cross-entropy loss; however, we want the network to still train for what it aimed to minimise - inaccuracies in the segmentation.

$$loss(x, y) = 0.9 * cross\_entropy(x, y) + 0.1 * dice(argmax(softmax(x)), y), \tag{5.2}$$

where all variables have the same definition as in Equation 5.1.

This loss was used by the `AdamW` optimisation algorithm with a learning rate of $5e^{-4}$ for all results included in the evaluation. This loss was determined in early prototyping stages as it more rapidly converged a model to the same minimum as when trained with a learning rate of $1e^{-4}$.

In addition to loss, accuracy metrics were computed for the model, including a Dice coefficient and a multi-class, global accuracy measure. These allowed a sense of the model's performance to be gleaned while training was occurring. After training, when evaluating the model, the segmentations' Hausdorff distances for each class were also computed for the model's segmentations. This is another common metric used to evaluate models in the literature. It describes errors with the edges of segmentation by computing the largest difference between a point on a segmentation's edge and the closest on another segmentation's edge (Cao et al. 2023). This was especially useful for evaluating the effectiveness of the overlapping slabs in decreasing the edge noise, shown in Section 6.5.

5.1.4 Pretraining

Pretraining is an important concept used by transformer-based networks as it allows them to utilise preformed conceptual latent representations of the input media. It has given impressive results for vision transformers and even the Perceiver architecture. However, due to the recency of the development of the Perceiver IO architecture, unfortunately, no prertrained networks are readily available for this application. Considering the parameter optimisation to be completed - described in Section 5.7 - a wide array of pretrained models would have to be available to cover the innumerable possible model configurations. As such, all training started from scratch without the optimisation of pretrained models.

5.2 PyTorch Lightning

The implementation of the underlying Perceiver IO network, extended as part of this paper, utilised the PyTorch Lightning library (Falcon and The PyTorch Lightning team 2019) to handle the network training. This library provides an easy-to-use process for network training tasks, and supplies several integrations to help accelerate the model training. Further, PyTorch Lightning works consistently inside and outside a docker container, meaning it can be utilised effectively when deployed as a containerised service.

5.2.1 Model Implementation

The Perceiver IO for biomedical segmentation was implemented as a PyTorch Lightning `LightningModule`. This allowed the model to take in parameters from the Lightning `Trainer` (described in Section 5.2.2) and for training actions to be more explicitly defined. Taking in parameters from the `Trainer` allowed for the parameters to be passed to the abstract representation of a Perceiver IO network defined by the original repository (Krasser and Stumpf 2023) in the form of encoder and decoder configurations. Additionally, several model specific parameters could be passed to the network, including ones that specified how many and what slabs the model should operate on. These "slab-parameters" allowed for the model to operate on specific areas in scans, such as the liver and kidneys, instead of full scans. This allowed for faster model training and as such evaluations, used namely for parameter optimisation, multi-modality evaluation, and ablation studies.

The model's `forward` method, responsible for the model's inference was implemented to take in a Python dictionary containing the `image` to be inferred and the `label`/ground truth values to compare the inference with (which could be `None` in cases of inferencing). As only the metric computations requires the label, it was mearly cropped to be of the area being inferred by the model. This was so it could be used accurately by the implemented `step` functions that called the `forward` method.

To infer the `image`, firstly a tensor to store the entire prediction results for the area defined by the slab-parameters was created. The Perceiver IO network was then inferred iteratively on each input, with its predictions stored in the previously defined tensor. This network inference was enclosed in as a PyTorch checkpoint to allow the model to backpropogate based on its loss, without requiring the full network's gradients to stored during all iterations. Storing these gradients would greatly increase the memory requirements to train the model, making it infeasible. As such, although checkpointing requires recomputing the gradients during backpropogation, slowing the training process, it is required by the implemented model.

Instead of defining a singular `forward` method, as is done by default PyTorch `Modules`, `LightningModules` require additional `training\_step`, `validation\_step`, and `test\_step` methods called during the respective model operations by the Lightning `Trainer`. This is because PyTorch Lightning strictly adheres to the principles of the train-test-validate dataset splits to avoid inaccurate metric computation which can lead to overfitting. These functions are called to compute the loss of the model used for backpropagation and model weight optimisation. We implemented this loss computation, and other metric computations used for process monitoring, into an additional `step` function. This was to decrease the repetition present in the code by abstracting the repeated use functionality.

5.2.2 Dynamic Model Configuration

PyTorch Lightning uses a `Trainer` class to handle all model training operations, including data loading, batching, forward passes, backward passes, and optimisation. The framework allows for several configuration parameters, such as the number of GPU devices to be used for acceleration, the learning rate, and the optimisation algorithm to be specified. Alongside training and optimiser parameters, model parameters can be specified to configure the model. These parameters are passed directly to the model as `kwargs`.

Using these model parameters, different models can be constructed and tested. Implementation for the Perceiver IO parameters already existed in the repository; however, medical image segmentation-specific parameters had to be implemented. Namely, slices per slab, overlapping slices, and recurrent slices. The remaining parameter, the resolution of the scans, was configured in the custom data loading class described in Section 5.6 but was used as a constant $256x256x220$ for all training and inferencing. Additionally, the slab-parameters were set to constant values depending on the model's task, as noted in each experiment in Chapter 6.

The dynamically configurable model parameters also allowed for runtime changes to the model to be made. This feature was most helpful in the automated derivative-free parameter optimisation (discussed in Section 5.7). Specifically, this feature allowed the optimisation algorithm to be applied to the network parameters to refine the architecture for segmentations without the need for manual restarts.

5.3 Multi-GPU Cluster Training

PyTorch Lightning also provides state-of-the-art methods for running and parallelising model training and inference operations. The PyTorch library (Paszke et al. 2019), the basis of the Lightning library, allows for multiple GPUs to be used for machine learning processes. Lightning extends this functionality, making implementing and integrating this accelerator parallelisation far easier. Instead of considering what accelerator devices to run what operation, for every operation, Lightning allows the specification of "strategies". These strategies handle model training on multiple devices by splitting the input data into smaller batches or subsets to be run on each GPU in parallel. After each inference, the strategies combine these cross-device results to compute the required metrics and loss.

Initially, development was completed on a system with a single GPU, rendering these strategies optimisations not applicable. However, due to the scale of the models that had to be trained, a compute cluster was used to complete the later parameter optimisation task, detailed in Section 5.7. This compute cluster, provided by the University of Glasgow School of Computer Science, ran docker images with access to multiple GPUs, allowing for this "Data-Parallel" strategy to be used. Figure 5.1 displays the strategy utilised pictographically.

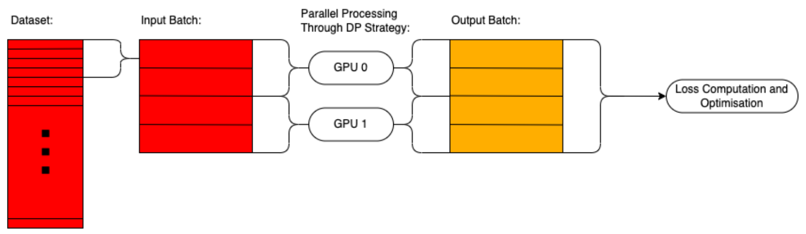

Dataset:
Input Batch:
Parallel Processing Through DP Strategy:
Output Batch:
GPU 0
GPU 1
Loss Computation and Optimisation

Figure 5.1: *Diagram showing the Data-Parallel (DP) strategy splitting batches of inputs across GPUs to allow multi-GPU systems to be used effectively*

5.3.1 Details of Hardware Used

The compute cluster provided by the University of Glasgow School of Computer Science gave this paper access to two NVIDIA 3090 GPUs, with 24GB of VRAM each. As mentioned, this cluster was utilised for the parameter optimisations. The other system used for development had a single NVIDIA RTX 3080 with 10GB of VRAM. This system was used for the early development and testing of the models, the ablation studies on the recursive and overlapping slices, the multi-modality evaluations, and the final model training.

5.4 Recursive Slices

As mentioned in Chapter 4, to help promote inter and intra-slab attention and dependency, recursive outputs will be used to persist information between otherwise independent network inferences. This was implemented by stacking the previous predictions of the network on top of the current scan data.

This caused an increase in the computation required to encode the inputs. However, due to the Perceiver architecture, this increase was only linear; smaller than the quadratic increase if included in a CNN or a classic-transformer-based network. Due to the minor increase in the computation requirements and how the model can be dynamically configured, the number of recursive slices was added to the list of parameters to optimise during the parameter optimisation.

5.5 Overlapping Slices

Since the overlapping slice do not increase the network's input size, they will not cause an increase in the computational requirements for each slab's network inference. However, since less of the scan is covered per slab, more network inferences are required to predict the entire scan.

With the inclusion of the recursive slices, each slab relies on the previous one (except the first slab). As such, although the scans are independent with respect to the scan data in the inputs, they cannot be run in parallel. As such, more network inferences do increase the time for the network to infer a scan, slowing down training and inferences. Despite this, we deemed the accuracy of the network to be of the greatest importance. From this, we decided the number of overlapping slices per slab should be an optimisable parameter.

5.6 Custom Dataloader

The AMOS dataset is relatively recent and is only available through a ZIP file download of NIFTI formatted files. As such, there is no easy way to include the dataset in a machine learning project, as there is no library support for it; like HuggingFace's `datasets` library that allows for easy and reproducible downloading of the MNIST dataset (Deng 2012). As part of this Perceiver IO implementation, a dataset loader was created. This loader served a dual purpose, allowing for reproducible data loading from files to memory, as described above, and preprocessing the dataset, as mentioned in Section 4.1.

The dataset loading was implemented as a PyTorch Lightning `LightningDataModule`. Like the model, this module can take in parameters from the Lightning `Trainer`, such as where to load the data from and if it should be shuffled. Additionally, like the model, the data module requires specific methods for loading the training, test, and validation datasets, each returning a Lightning `DataLoader`. These `DataLoaders` are constructed with a `Dataset` class, and handle the methods for batching and inputting the dataset to the model training, validation, and test processes.

To produce these `Datasets` for each type of dataset, a `MICCAILoader` class was created. This class can read in the AMOS dataset scans and labels, coregistering both to the largest scan (as described in 4.1.1). The class then returns the correct splits for each dataset, as defined by the accompanying `dataset.json`. The loaded dataset can also optionally exclude MRI scans from being loaded to allow for a single modality dataset instead. This single modality is used for the ablation studies and parameter optimisations completed in this paper, discussed in Chapter 6.

For splitting the AMOS dataset, one exception is made when following the `dataset.json` file. Since the AMOS22 challenge uses a public online leaderboard for ranking, it provides its test dataset without any ground-truth labels. However, we determined that a test dataset with labels is required to evaluate the models in this paper. This is because important metrics, such as Hausdorff distance, are not provided by the online leaderboard. From this, the training dataset was split into 70% and 30%; the latter was used as the test dataset. This means that less training data was available to train the models; however, the granularity of the metrics is imperative for the accurate evaluation of the Perceiver IO architecture.

5.6.1 Coregistration and Preprocessing

When coregistering scans to the largest, the largest must first be prepared to ensure it is the correct size for inputting to the network and that each scan's coregistration resulted in as little data loss as possible. First, all unprocessed scans are loaded to determine the smallest spacings – the highest resolution of the scans – and the largest world width, height, and depth. With these spacings, the largest scan is interpolated up to this resolution. Next, padding is added to the width and height dimensions if smaller than the largest real sizes found for them. This is to avoid images that are not exact "sub-scans" of the largest scan being cropped. The depth is then padded so that when resized down, the image size matches the size of the input images to the network. This image size is a constant parameter set at 256*x*256*x*220 for all model evaluations. Finally, the largest scan is interpolated down to match the input image size. When resizing the image, spline interpolation of order three is used to prevent extensive data loss. The same process is followed for resizing the corresponding label; however, a spline interpolation of order zero prevents segmentation boundaries from being interpolated to different labels incorrectly.

Using this transformed scan of the correct size and optimal spacing, all other scans can be coregistered to help standardise the dataset. Coregistration has been integrated using the SimpleITK python library (Lowekamp et al. 2013; Yaniv et al. 2018; Beare et al. 2018) and its `ImageRegistrationMethod`. A gradient descent optimiser was used three times, each with a learning rate of 2.0 and 500 iterations, to compute the optimal transformation between the fixed - the constant, largest scan - and the moving scan. This transformation was then applied with linear interpolation to the moving image to place it in the fixed image's space. The same was done for the scan's label, instead with nearest neighbour interpolation used to avoid incorrect labels at boundaries.

Each scan took around five seconds to coregister on an Intel 12600K CPU. To avoid having to compute the coregistration every time the dataset is loaded, the coregistered scans and their transformations are saved to the AMOS dataset's directory. Before the `MICCAILoader` loads the scans, it checks if they have been previously coregistered and will load the coregistered scans and labels instead.

After loading the scans to memory, their two background values are made the same, and their intensities are normalised to prepare the scans for input to the network. These operations are represented as composed TorchVision transforms applied to each scan when accessed from the `Datasets` (maintainers and contributors 2016).

5.6.2 Shared Memory

The AMOS dataset is approximately 24.2 GB. The coregistered versions of the scans are smaller than the whole dataset, however, are in total approximately 6 GB. When loading this data repeatedly during development, training, and parameter optimisation, a significant amount of time is lost. To minimise the data loading time, the dataset was copied to Linux's shared memory at `/dev/shm` before using the data loader. This allowed the files to be stored and loaded from memory instead of hard drives. This helped to approximately third the data loading times. This was especially useful when the data had to be loaded multiple times, such as in the parameter optimisation.

5.7 Derivative-Free Model Parameter Optimisation

As discussed throughout and in Section 4.6, there are several parameters that can be tuned to improve model performance. Some parameters are dependant on one another, meaning that they can be described by a single parameter value. Namely, the query-key-value channels which are dependant on the number of heads for their respective cross or self attentions. Additionally, several parameters have been give a constant value to preserve the model's generalisability, avoid slow preprocessing operations, and to make the parameter optimisation feasible in the time available. These constant values include:

- The resolution of the input and output scans: 256*x*256*x*220
- Whether cross attention residuals are used in the decoder: `True`
- The encoder and decoder dropouts: 0.3
- The slice that the inference slabs begin on: 80

- The number of slices that the inference is ran on: 12
- The number of slices per slab: 4
- The number of epochs the model was trained for: 20
- The optimisation algorithm used: AdamW
- The optimisation learning rate used: $5e^{-4}$

From these constraints and the parameter interactions, the following list of parameters to optimise was made:

- Number of latents in the model's latent array
- Number of latent channels in the model's latent array
- Number of overlapping slices per slab
- Number of recursive slices per slab
- Number of frequency bands in the encoder
- Number of cross attention heads in the encoder
- Number of cross attention heads in the decoder
- Number of cross attention layers
- Number of self attention blocks
- Number of self attention layers per block
- Number of self attention heads

5.7.1 Algorithm Choice

Approaches like grid-search and random search are commonly applied to explore the parameter space without the possibility of being stuck in local minima while possibly returning a better set of parameters than the baseline model. However, these approaches do not consider the possibility that the parameters may be linked and that there may be a way to make more reasoned approximations of the shape of the parameter space. Additionally, the number of iterations to fully explore the parameter space required by the model is often rather large; exponential with respect to the number of parameters in the case of grid-search. Exponential time complexity is particularly problematic in the case of tuning the Perceiver IO parameters as training a single model requires significant amounts of time, taking multiple days in many cases.

To avoid exponential time complexity, and to explore the parameter space in a more intelligent way, Bayesian optimisation approaches have been used to predict the optimal parameters based on sparse observations. In the case of this paper, the Nevergrad library (Rapin and Teytaud 2018) has been chosen as to optimise the network parameters. The Nevergrad optimisation algorithm can explore a parameter space and narrow down on an optimal parameter set within a defined "budget". This budget allows for a limit to be set on how long to optimise the parameters for, preventing any exponential time complexity. A larger budget often allows for better results, however, Nevergrad has been found to be effective even in low budget scenarios (Rapin et al. 2019). As such, it has been chosen as the algorithm to optimise the model's parameters.

5.7.2 Algorithm Integration

The Nevergrad optimiser is initialised using an `Instrumentation` of the variables. This is made up of descriptions of each variable's range and type. In this case, all the parameters only take integer values. Additionally, reasonable limits were set on each of the values to avoid having the algorithm only operate in impossibly high or low values. The specific values are included in the parameter optimisation evaluation in Section 6.2. The optimiser is also intialised with the budget that it has to explore the parameter space; 50 was used for the optimisation in this paper.

After initialising the optimiser, the optimiser was `suggested` the baseline model parameters. These parameters were determined through very minor manual experimental work during the prototyping stage. The optimiser is then interacted with through an `ask` and `tell` interface. This interface requires that we `ask` the optimiser for a parameter set to find the loss for, then `tell` it said loss so that it can update its knowledge of the parameter space. The first `ask` operation returns the values we originally `suggested` to the model, allowing us to always compare to the baseline when optimising. After the budget of the Nevergrad optimiser is fulfilled, a `recommendation` can be generated based on what it has learned. The loss value told to the optimiser after each model was trained was the lowest loss achieved by the model on the validation set after each epoch.

6 Evaluation

6.1 Metrics

Two metrics have been used to evaluate the models trained to verify Perceiver IO's usefulness in biomedical image segmentation. These metrics act on the segmentations produced by the models and act to find the scale of the differences between the ground truth and the predicted labels.

The first metric used is the Dice coefficient (DSC) which finds the proportion of true-positive labellings against the number of true positives, false positives and false negatives. Equation 6.1 shows the exact calculation used. This description of the Dice coefficient normally only describes binary data; however, it can be applied to multi-class problems by averaging each class against the other classes. These scores are then often averaged to find a single Dice coefficient value to describe the whole segmentation, often done through macro-averaging across all scans for each class, weighting each class equally. The Dice coefficient is a popular metric used in biomedical segmentation for scoring a segmentation model; being used by the AMOS challenge. Additionally, using similar metrics to other state-of-the-art research allows us to compare the Perceiver model with other models. The Dice coefficient ranges from 0 to 1, with 1 being a perfect match between a target and a predicted segmentation.

$$DSC = \frac{2TP}{2TP + FP + FN}, \tag{6.1}$$

where TP indicates true-positive values between the target and predicted segmentation, FP is the false positive, and FN is the false negative

The second metric used to evaluate the Perceiver IO model is the Hausdorff Distance (HD). This metric is also commonly used to evaluate models and is computed by finding the largest distance from one segmentation area to the closest point in another. Since this metric works on the largest difference between two maps' edges, it helps to describe the error in the edges of segmentations. This is useful as the Dice coefficient alone cannot describe this as it operates on global quantities of values. The calculation for the Hausdorff distance is also typically computed on binary classes; however, it can also be macro-averaged to find a single value to describe an entire dataset.

6.2 Parameter Optimisation

The Nevergrad optimiser was given a budget of 50 to optimise the parameters. Additionally, limits were set on the parameter's ranges to avoid extreme values from being tested. These limits for each parameter can be found in Table 6.1.

Using these values, the parameters were optimised by training the Perceiver IO model using CT-only data on the compute cluster with two 3090 GPUs with the Data-Parallel strategy. Models were trained for 20 epochs, which did not allow for full convergence; however, fully training the Perceiver IO network commonly takes over a day on partial scans due to not using pretrained models.

Table 6.1: Table containing the parameters, the limits put on them for parameter optimisation using the Nevergrad algorithm, the values used as the baseline model, and the values recommended after the optimisation

Parameter	**Upper**	**Lower**	**Baseline**	**Optimised**
Number of Latents	1	2048	512	1090
Number of Latent Channels	1	1024	128	334
Number of Overlapping Slices	0	4	2	2
Number of Recursive Slices	0	4	1	2
Number of Encoder Frequency Bands	1	128	4	70
Number of Encoder Cross Attention Heads	1	4	2	2
Number of Decoder Cross Attention Heads	1	4	2	2
Number of Cross Attention Layers	1	4	2	2
Number of Self Attention Blocks	1	4	2	2
Number of Self Attention Layers per Block	1	4	4	3
Number of Self Attention Heads	1	4	2	2

Table 6.2: The macro-averaged Dice coefficient and Hausdorff distances across all organ labels in the validation set for the baseline model and the model with optimised parameters

	Average DSC	**Average HD**
Baseline Model	0.031	**28.832**
Parameter Optimised Model	**0.031**	36.231

6.2.1 Results

After fulfilling the optimiser's budget, the recommendation was generated and the results of which can be seen in Table 6.1. This parameter optimisation decreased the model's loss value from the baseline of 0.076 to 0.039. To further compare the baseline and optimised models, each was trained for 350 epochs. The macro-averaged Dice coefficient (higher is better) and Hausdorff distance (lower is better) were computed across the models' labels. The results are shown in Table 6.2 with the emboldened results showing the best.

Since the optimised and baseline were different sizes, the resources required to train and infer them differed. As such, the average inference times and their number of parameters have also been noted in Table 6.3.

6.3 Multi-Modality Generalisability

Another critical area to test the Perceiver IO architecture's capabilities on is its generalisability to handle multiple modalities of data. As noted by the AMOS challenge, this factor is key to simplifying and enabling the deployment of machine learning segmentation systems to medical imaging scenarios (Ji et al. 2022). Further, the Perceiver model's ability to generalise - enabled by its expressive and deep latent arrays - was a key reason for its selection to be used in this paper (Choi and Ha 2022).

Table 6.3: The average inference time and number of parameters for the baseline and parameter optimised models

	Inference Time (s)	**Number of Parameters (M)**
Baseline Model	**0.209**	**4.5**
Parameter Optimised Model	0.303	5.7

Table 6.4: *The macro-averaged Dice coefficient and Hausdorff distances across all organ labels in the validation set for the CT-only trained model and multi-modality (CT plus MRI) trained model*

	Average DSC	**Average HD**
CT-Only Model (baseline)	0.031	28.832
Multi-Modality Model	**0.056**	**14.429**

Two models with identical parameters were trained and validated to evaluate the model's performance. One was trained and validated on only the CT scans of the AMOS dataset, and the other was trained and validated on both CT and MRI scans. The models used the same parameter values as the baseline model described in the parameter optimisation test. This was because these models required fewer training resources - due to the fewer parameters - and their results should match what would be seen in larger models since they follow the same general architecture. The models were also trained to only infer on 12 slices to decrease the time to train them. After training for 350 epochs, the models were evaluated on the validation dataset. The macro-averaged DSC and HD metrics were then computed, shown in Table 6.4.

6.4 Full-Scan Effectiveness

All prior results have been computed using parts of the scans to decrease model training time. These have given robust results. However, they cannot be used to compare the Perceiver IO model to other model architectures using the AMOS online leaderboard, as it computes scores based on entire scans. As such, the Perceiver IO model must be trained to compute full scan inferences. This greatly increases training times as each step in the epoch can take upwards of 15 seconds, compared to around two seconds when training on partial scans. As such, the results of the prior evaluations were used to inform the decisions of what model was to be trained and how it was to be trained.

From the parameter optimisation results, we decided to train the baseline model instead of the parameter-optimised model. This was because the marginal improvements offered by the parameter-optimised model were deemed not significant enough to warrant the significantly greater resources needed to train its 1.2 million extra parameters. Additionally, we chose to train the full-scan model using MRI and CT data as the improvements to the DSC and HD scores are great enough to justify the increased training time required to train on more data. Furthermore, instead of training for 350 epochs, which has been shown to allow for model convergence, the model was trained until it appeared it converged. Convergence was determined using the graph of validation loss against the real-world time - shown in Figure 6.1. Using this graph, the model training was halted after 212 epochs.

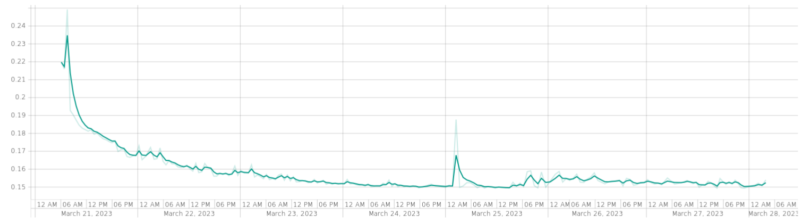


Figure 6.1: *Graph showing validation dataset loss against the real-world time. Graph generated using Tensorboard from logs generated by PyTorch Lightning (Falcon and The PyTorch Lightning team 2019)*

Table 6.5: *Macro-averaged Dice coefficient and Hausdorff Distance metrics computed for each organ across all scans in the validation and test datasets for the full-scan Perceiver IO model using baseline parameters and multi-modality data*

	Validation		**Test**	
	Avg. DSC	**Avg. HD**	**Avg. DSC**	**Avg. HD**
Spleen	0.023	12.656	0.003	5.999
Right Kidney	0.079	33.951	0.041	24.170
Left Kidney	0.059	35.131	0.026	25.534
Gall Bladder	0.000	27.061	0.000	6.594
Esophagus	0.002	5.777	0.000	0.000
Liver	0.068	30.462	0.012	30.592
Stomach	0.010	18.651	0.000	3.613
Arota	0.008	24.014	0.004	24.697
Postcava	0.036	37.489	0.013	28.612
Pancreas	0.010	34.097	0.002	23.038
Right Adrenal Gland	0.001	7.354	0.000	0.833
Left Adrenal Gland	0.002	11.247	0.000	8.785
Duodenum	0.027	46.845	0.005	29.035
Bladder	0.000	4.198	0.000	0.000
Prostate/Uterus	0.008	11.819	0.000	3.894
Macro	0.022	22.717	0.007	14.360

6.4.1 Results

After training the model, averaged DSC and HD scores were computed on the validation set for each organ using the model from the last epoch. Computing the scores per organ allows for a higher-fidelity examination of the results to see what organs the model is failing on. This information could be used to further refine the model in future. Additionally, macro-averaged DSC and HD scores were computed to compare with the model computed in earlier evaluations. These results can be seen in Table 6.5.

Since this model is to be compared with literature, test dataset scores were also computed in the same way, using the data split off from the defined training set - described in Section 5.6. Additionally, inferences were computed for each scan in the AMOS-defined test dataset so that they could be evaluated and listed on the AMOS leaderboard. The final place for the Perceiver IO model on the multi-modality AMOS leaderboard was third, and the computed average DSC for the AMOS test set was 0.011.

On top of these evaluations, the model's average inference time for a single inference of batch size 1 was found to be 2.368 seconds. This was achieved on a single NVIDIA RTX 3080 GPU. A comparison of an example inference and the ground truth values is shown in Figure 6.2.

6.5 Ablation Studies

To further investigate and evaluate the features added to make the Perceiver IO architecture more effective at medical image segmentation, ablation studies were performed on the inclusion of recursive and overlapping slices. The models trained used the same parameters as the baseline model, as training was faster and the results were comparable. We also trained the models for 350 epochs, allowing them to converge and inference optimally. The models were also trained on only 12 slices to allow for faster training while still having long-range relationships in the data. Additionally, the models with the highest DSC on the validation set were used for the inferences as opposed to the last model trained, which may have begun overfitting to the training set.

Table 6.6: Macro-averaged Dice coefficients across all organ labels in the validation set for each model trained during the ablation study

	Two Recursive Slices	**Zero Recursive Slices**
Two Overlapping Slices	0.032	0.030
Zero Overlapping Slices	**0.041**	0.023

Table 6.7: Macro-averaged Hausdorff Distances across all organ labels in the validation set for each model trained during the ablation study

	Two Recursive Slices	**Zero Recursive Slices**
Two Overlapping Slices	28.758	38.302
Zero Overlapping Slices	26.291	**14.786**

6.5.1 Results

The models trained and compared include the baseline model but with two recursive and two overlapping slices, zero recursive and two overlapping, two recursive and zero overlapping, and zero recursive or overlapping slices. This forms a Punnett square for these parameters, shown in Tables 6.6 and 6.7 with macro-averaged Dice coefficient and Hausdorff distances on the validation set for each model, respectively.

Further, the average inference times for each model were computed to see if any improvements in performance may be at the cost of time to inference. These times are shown in Table 6.8.

6.6 Additional Observations

During the evaluation of the model, particularly anomalous results were identified multiple times. When there is no organ data in certain areas, such as a region of the body that does not contain any labelled organs, the network still infers that organs are present. This can be seen in Figure 6.2f where the network attempts to infer that the spleen is present in the lungs. This could result from the network wrongly learning where to attend to and attending to areas other than the spleen for information about the spleen.

However, this style of mis-inferencing has also been found to be present in areas with no surrounding data. This can be seen in Figure 6.3 where several labels are mis-inferenced, with large distances between them, showing that no information should be passed into the network through the inputs. However, namely with the Prostate/Uterus classification, the Perceiver IO's classifications are approximately where the organ would have been located had they been included in the scan.

Table 6.8: Mean inference times in seconds across all scans in the validation set for each model trained during the ablation study; smaller is better

	Two Recursive Slices	**Zero Recursive Slices**
Two Overlapping Slices	0.212	0.217
Zero Overlapping Slices	0.150	**0.141**

6.7 Discussion

6.7.1 Parameter Optimisation

It can be seen from Table 6.2 that the parameter optimisation has improved Perceiver IO's results in the Dice coefficient metric and decreased its loss. This shows that the method for optimising the parameters was applied effectively. However, the optimised model does not perform better in the Hausdorff distance metric, showing that although there was a small increase in the DSC scores, the optimised model is not significantly better than the baseline model. Additionally, due to the increase in the number of parameters and, by extension, the inference time, the optimised model is not worth using for medical image segmentation over the baseline model.

Although the parameter optimisation did not lead to further refinement in this paper, this paper proves that this technique can be used to optimise the parameters of the Perceiver IO network. Allowing the parameter optimisation to run longer would further improve the results and, as discussed in Section 7.2, could lead to substantial improvements in the network's performance.

6.7.2 Multi-Modality

The results of the multi-modality evaluation show that the model trained with multiple data sources not only performs as well as the one trained with only one, but performs significantly better. This is shown by the clear improvements in the DSC and HD metrics, with a 181.237% increase and a 49.954% decrease, respectively. As such, we can say that the Perceiver IO network performs significantly better when exposed to multi-modality data than single-modality.

This could be because the multi-modality model had far more scans to train on - 168 instead of only 140 - which would allow the network to improve the quality of its training; in turn, improving its inference and generalisation capabilities. As such, the improvements in the results are likely due to more training data being available. However, the improvement is still beneficial as it shows that the Perceiver IO model can effectively utilise MRI and CT scan data to perform meaningful inferences on both. This shows that the Perceiver IO network can effectively generalise to multi-modality data through its expressive latent array. In turn, this could allow trained models to effectively infer either CT or MRI data interchangeably. As stated by the AMOS challenge (Ji et al. 2022), this could help spur deployment and real-world usage of such a segmentation solution.

Furthermore, since the Perceiver IO network can effectively train using both modalities, future training jobs for Perceiver-based networks can also be done using multi-modality data. This will allow these networks to have far more data to train using, helping to improve their performances further. This is especially useful for the medical image segmentation field due to the relative scarcity and difficulty in collecting data to train models. As such, Peceiver-based networks that train on multiple data-source may easily outperform non-generalisable, single-modality networks due to their ability to effectively train using more data.

6.7.3 Full-Scan Inferences and Additional Observations

From the full-scan evaluation results, we found that the Perceiver IO network is able to create meaningful, multi-label, multi-modality segmentations. However, compared to state-of-the-art networks, its performance is lacking, coming third/last on the AMOS multi-modality leaderboard and failing to compute any accurate inferences on several organs, namely the gall bladder and bladder.

This could be due to several reasons. However, the most likely is because the Perceiver IO architecture usage described in this paper is far more simplistic than the current state-of-the-art biomedical segmentation networks. This concurs with the original Perceiver paper's findings:

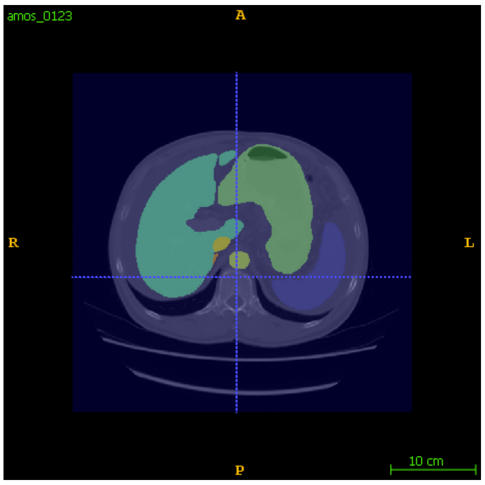
It performs better than generic solutions but rarely better than the state-of-the-art custom solutions (Jaegle, Gimeno, Brock, Vinyals, Zisserman and Carreira 2021). As such, the Perceiver architecture is not directly producing competitive biomedical segmentations in its current state.

However, as shown in Figure 6.3d, it is learning how to predict organs based on very little input information - sometimes without any. As such, we still believe that it is learning an expressive array that is, by its default nature, modelling the human body based on spatial information. As such, the Perceiver network can be seen to tackle the lack of long-range spatial and morphological dependencies seen with other network architectures. From this, we believe that the Perceiver IO architecture does warrant further investigation into its capabilities, further described in Section 7.2.

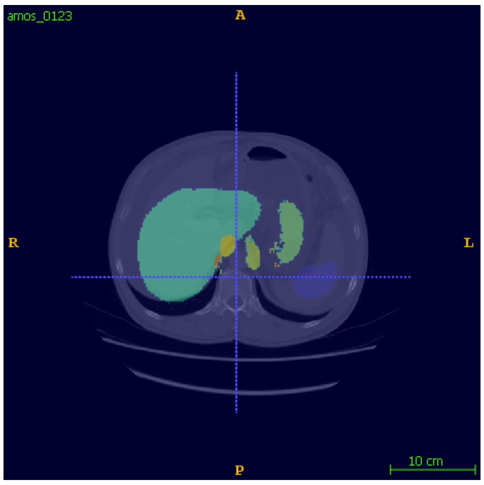
6.7.4 Ablation Studies

The ablation studies performed on the network show that the current implementation of recursive slices significantly improves the Perceiver IO architecture's performance - increasing the DSC metric by up to 176.703%. Recursive slices also decreased the HD metric by 24.917% when used with two overlapping slices. There also does not appear to be any significant increase in inference times when using recursive slices, making their inclusion in the network only a benefit and confirming that they are effective in improving the network's performance. This finding agrees with similar research on recursive inferences using the U-Net architecture.

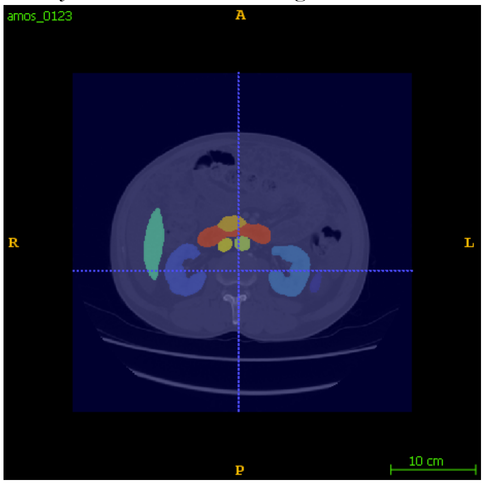
Unlike their recursive counterparts, overlapping slices appear to decrease network performance when viewing the DSC metric. This is particularly interesting as they were initially employed to increase the performance of the Perceiver IO model. Overlapping slices were implemented to help to smooth the edges of the inferred labels, as they were seen to be very rough in earlier implementations of the architecture. As shown in the results, despite the decrease in the DSC, there is also a decrease in the Hausdorff distance in all cases. This indicates that the inclusion of overlapping slices does successfully improve the edges of the segmentations. This is at the cost of the segmentations' overall accuracy. Additionally, overlapping slices do significantly increase the inference times of the models, likely due to the increased number of forward passes that must be completed to infer the same number of scan slices. As such, overlapping slices should not be utilised in future research on this task with the Perceiver architecture. This is with the exception of cases where the network performs poorly at label edges, which may occur with more complex and accurate models. Additionally, we believe it should be noted that this implementation of overlapping scans is one of many forms that overlapping could take. We briefly discuss one such alternative approach that could be explored in future research in Section 7.2.



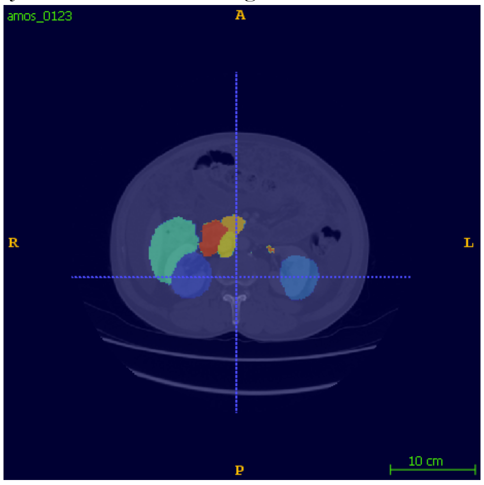
(a) Figure showing the axial ground-truth segmentations of the liver and surrounding area at slice 60



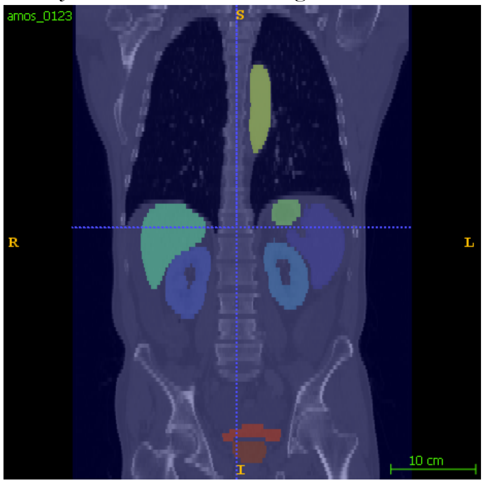
(b) Figure showing the axial inference segmentations of the liver and surrounding area at slice 60



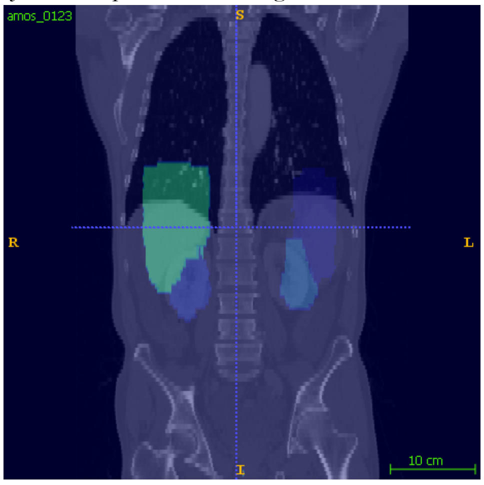
(c) Figure showing the axial ground-truth segmentations of the liver and surrounding area at slice 45



(d) Figure showing the axial inference segmentations of the kidneys and surrounding area at slice 45

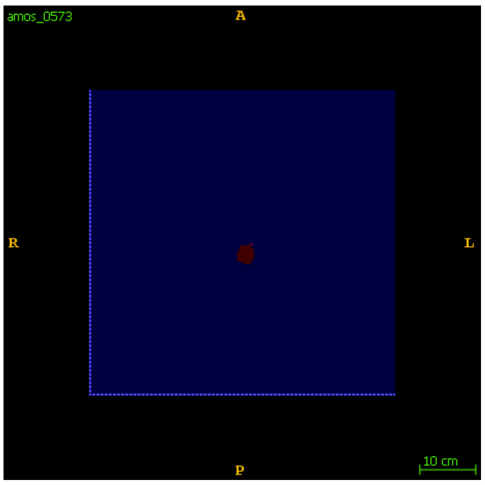


(e) Figure showing the coronal ground-truth segmentations of the liver and kidneys at slice 201

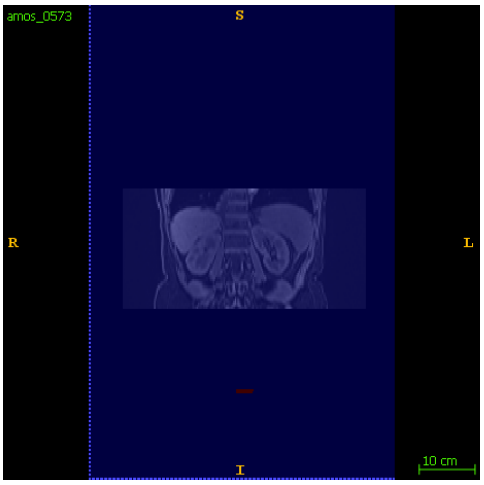


(f) Figure showing the coronal inference segmentations of the liver and kidneys at slice 201

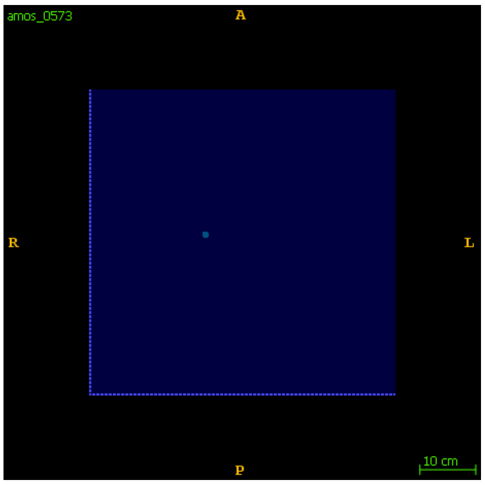
Figure 6.2: *Scan 123 from the AMOS dataset (Ji et al. 2022), visualised using ITK-SNAP (Yushkevich et al. 2006). Screenshots centred on the kidneys and livers across the axial and coronal planes to highlight the differences between the ground truth segmentations and the inference segmentations*



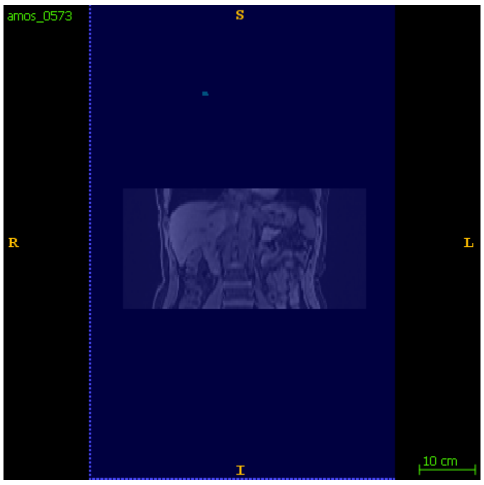
(a) Figure showing mis-inferenced segmentations of the Prostate/Uterus label (the larger) and the Bladder label on the axial plane



(b) Figure showing mis-inferenced segmentations of the Prostate/Uterus label on the coronal plane to show its relation to the scan data



(c) Figure showing a mis-inferenced segmentation of the Gall Bladder on the axial plane



(d) Figure showing a mis-inferenced segmentation of the Gall Bladder on the coronal plane to show its relation to the scan data

Figure 6.3: *Scan 573 from the AMOS dataset (Ji et al. 2022), visualised using ITK-SNAP (Yushkevich et al. 2006) showing mis-inferences of the Prostate/Uterus, Bladder, and Gall Bladder in the presence of no data.*

7 Conclusion

7.1 Summary

In this paper, a model based on the Perceiver IO architecture has been built to create medical image segmentations for multi-label abdominal CT and MRI scans. This model's parameters have then been shown to be optimisable using a derivative-free optimisation algorithm, shown by a decrease in loss of 48.730%. The Perceiver IO model has also been shown to generalise to multiple modalities effectively, producing results 181.237% better when trained on multiple modalities instead of a single modality when compared with a standard metric; the macro-averaged Dice coefficient. Additionally, two extensions to the Perceiver IO architecture - recursive and overlapping slices - have been found to improve the network's performance by up to 176.703% when compared with the same metric.

When comparing the Perceiver IO architecture's results to state-of-the-art models, however, we see that the model does not perform to the same standard despite the parameter optimisation, additional features, and multi-modality training: Our trained model came third/last in a standardised online leaderboard for the abdominal segmentation dataset used for the comparative evaluations with a macro-averaged Dice coefficient of 0.011. As such, in this current state, we do not believe that the Perceiver IO can be used to create competitive biomedical image segmentations.

Despite these undesirable results compared to the current literature, we believe this paper is an effective exploratory work highlighting that the Perceiver architecture should be explored further. Our results show that this architecture is receptive to novel extensions and is incredibly effective when generalising to multi-modality data. As such, we believe that with further refinements to the network architecture and inspiration from existing state-of-the-art systems, such as U-Net, the Perceiver IO architecture could outperform current architectures. From this, we foresee this paper bringing about many opportunities for future research based on these initial evaluations.

7.2 Future Work

These conclusions show that the Perceiver IO network could have a place in biomedical image segmentation tasks. As such, better exploration of its applications is necessary to guarantee that the field continues to grow and innovate. This chapter proposes several ideas for how the Perceiver IO network can be further investigated.

The primary recommendation is to continue the work outlined in this paper, allowing for more resources and time to train the models. This would come in the form of increasing the budget for optimising the network's parameters and training the final full-scan segmentation model until validation loss increases. Additionally, if more computational resources could be utilised to train the model, a deeper and more expressive model should be trained and evaluated. A model such as this may have better segmentation abilities than the models trained in this paper, further solidifying the need to investigate the Perceiver architecture further.

One recommendation for further research builds on what has already been achieved by vision transformers. Firstly, we would recommend research be completed to evaluate if the Perceiver architecture would work effectively as part of a hierarchical sliding-window transformer, like the Swin transformers developed by Liu et al. (2021). From this development, it would be useful to evaluate if these new integrations of the Perceiver architecture can produce higher-quality segmentations than a direct application of the architecture, shown in this paper.

Further, with the success of the U-Net architecture, integrating the Perceiver transformers could yield improved results. As seen when integrating vision transformers with the Swin-UNet architecture, impressive results were gained (Cao et al. 2023). Therefore, integrating the Perceiver IO network into U-Net would hopefully improve results significantly as well. Additionally, due to the Perceiver IO architecture's separated encoder and decoder, there would be no need for the patch merging and embeddings used in the original Swin-UNet. This could allow for the latent expression of the network to be utilised to the fullest extent, possibly yielding far improved and generalisable results.

Once a U-Net style model has been created, further research can be completed to investigate how several existing improvements to the U-Net model affect the Perceiver U-Net model. Touched on this paper, an example could be including recursive information between slabs. In this paper, recursion was implemented as the output of the full network stacked on top of the next set of inputs; however, this definition could be extended to include recursion between each Perceiver model in the encoder and decoder levels. This could further improve the attention and performance of the model across slabs as more of the expressive latent representations can be persisted (Wang, Yu, Hugonot, Fua and Salzmann 2019). On top of recursion, other techniques, such as overlapping voxels from independent sagittal, coronal, and axial segmentations could improve the outputted segmentations, although at the expense of having to train three times as many models for each plane.

Additionally, a better loss function could improve the quality of the model's segmentations. Borrowing from image and video generation research, a model could be trained in parallel with the segmentation-generating model to determine the authenticity of a segmentation. This second model could be used adversarially to reject low-quality image segmentations, raising the bar for the quality of the segmentations the model must generate. In turn, this should improve the quality of the segmentations created by biomedical segmentation models, although at the cost of even greater computational requirements.

On top of the replication and further research to be completed, several other advances can help to improve biomedical segmentation based on this research. Better preprocessing pipelines could be defined to help improve the training of models due to the lack of biomedical data to train on. Techniques such as random noise, blurring, and random cropping could create multiple possible images to train with from one - allowing for better training. Several Perceiver IO models should be pretrained to allow for more rapid development and testing of their integrations. Additional techniques to improve the Perceiver model's efficiency and effectiveness should also be integrated, such as checkpointing, to allow deeper models to be trained on hardware with less memory. Further, batching could also help stabilise and speed up model training, but may only be possible on systems with large amounts of VRAM.

A | User Manual

Below is the information on how to run the Perceiver IO data processing, training, and inferencing, which is also included as a markdown file with the source code:

A.1 Development

A.1.1 Prerequisites

- Miniconda installed
- Running on Ubuntu 22.04 (development was completed on WSL but should not be necessary for running)
- amos22 dataset: `https://zenodo.org/record/7155725#.Y_7WqHbP1D-` stored somewhere. where the path is important, it will be noted.

A.1.2 Creating environment

This is required for all further steps, which all assume that the conda `perceiver-io` environment is activated already

```
cd perceiver-io
conda env create -f environment.yml
conda activate perceiver-io
python -m pip install poetry -U
poetry install --all-extras
poetry build
```

A.1.3 Preprocessing dataset

This will run automatically when the dataset is loaded, but can be ran individually as well. It assumes that the dataset can be found at `/mnt/d/amos22`, this must be changed inside of `perceiver/scripts/segmentation/preproc.py` if not true:

```
cd perceiver-io
sh scripts/miccai/preproc.sh
```

A.1.4 Running training

```
cd perceiver-io
cp -r /path/to/amos22 /dev/shm # load amos22 dataset into shared memory. not all
    data must be loaded to memory, only, imagesTr_preprocessed/*
    labelsTr_preprocessed/*, dataset.json, and several empty folders: [imagesTr,
    imagesVa, imagesVa_preprocessed, labelsTr, labelsVa, labelsVa_preprocessed]
sh scripts/miccai/train.sh
```

A.1.5 Running inference

```
cd perceiver-io
cp -r /path/to/amos22 /dev/shm # load amos22 dataset into shared memory. not all
    data must be loaded to memory, only, imagesTr_preprocessed/*
    labelsTr_preprocessed/*, dataset.json, and several empty folders: [imagesTr,
    imagesVa, imagesVa_preprocessed, labelsTr, labelsVa, labelsVa_preprocessed]
    This obviously depends on what dataset split is to be inferenced
python -m perceiver.scripts.segmentation.inference # several configuration
    parameters can be changed in the GLOBALS at the start of the corresponding
    file
```

A.1.6 Running automated parameter optimisation

```
cd perceiver-io
cp -r /path/to/amos22 /dev/shm # load amos22 dataset into shared memory. not all
    data must be loaded to memory, only, imagesTr_preprocessed/*
    labelsTr_preprocessed/*, dataset.json, and several empty folders: [imagesTr,
    imagesVa, imagesVa_preprocessed, labelsTr, labelsVa, labelsVa_preprocessed]
sh scripts/miccai/automated_optimise.sh 2>&1 | tee log.txt # optimal parameters
    will be printed to the console, hence recommending using the use of tee
```

A.2 Docker deployment

A.2.1 Prerequisite

- docker installed
- trained model weights
- image used for coregistration (normally found in `amos22/imagesTr_preprocessed/coregistration_image.nii.gz`)

A.2.2 Building image

```
cd perceiver-io
sh scripts/docker_build.sh path/to/model.ckpt path/to/coregistration_image.nii.gz
    tagname
```

A.2.3 Deploying/pushing iamge

```
docker push tagname
```

A.2.4 Using docker image for inference

There is already a prebuilt docker image deployed on ghcr.io. It can be pulled using:

```
docker pull ghcr.io/questiowo/miccai-perceiver-io
```

This may require logging in, information on how to do so can be found in GitHub documentation: `https://docs.github.com/en/packages/working-with-a-github-packages-registry/working-with-the-container-registry#authenticating-with-a-personal-access-token-classic`

Information on how to correctly run the docker image using GPU acceleration can be found in the AMOS22 docker submission guidelines here `https://github.com/JiYuanFeng/AMOS/tree/docker#step-4-run-a-container-from-a-created-docker-image`.

A.2.5 Using OpenShift Compute Cluster

The OpenShift compute cluster GPUs can be used to train on by using the associated `Dockerfile.XXX` docker image. These are currently uploaded, ready for use on hub.docker.com: `https://hub.docker.com/search?q=perceiver-io`.

The configuration yaml files are included in the project as `job-perceiver-io-*.yaml`. The jobs expect the `amos22` dataset to be uploaded to an associated NFS store, which can be attached at `/volume/`. This is configured in the yaml-file and can be adjusted to new users easily.

The docker commands are written into each Dockerfile and are not observant to different numbers of GPGPUs. This means that if a cluster with (not 2) GPUs is available, the docker image has to be rebuilt, repushed, and the job restarted with the correct batch size and devices. Alternatively, the yaml configuration files can be extended to instead have the training job script inside the configuration file instead of the Dockerfile, however, an example is not included of that.

Bibliography

Badrinarayanan, V., Kendall, A. and Cipolla, R. (2017), 'Segnet: A deep convolutional encoder-decoder architecture for image segmentation', *IEEE transactions on pattern analysis and machine intelligence* **39**(12), 2481–2495.

Beare, R., Lowekamp, B. and Yaniv, Z. (2018), 'Image segmentation, registration and characterization in r with simpleitk', *Journal of statistical software* **86**.

Boynton, G. M. (2005), 'Attention and visual perception', *Current opinion in neurobiology* **15**(4), 465–469.

Briot, A., Viswanath, P. and Yogamani, S. (2018), Analysis of efficient cnn design techniques for semantic segmentation, *in* 'Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops', pp. 663–672.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A. et al. (2020), 'Language models are few-shot learners', *Advances in neural information processing systems* **33**, 1877–1901.

Cao, H., Wang, Y., Chen, J., Jiang, D., Zhang, X., Tian, Q. and Wang, M. (2023), Swin-unet: Unet-like pure transformer for medical image segmentation, *in* 'Computer Vision–ECCV 2022 Workshops: Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part III', Springer, pp. 205–218.

Chen, J., Lu, Y., Yu, Q., Luo, X., Adeli, E., Wang, Y., Lu, L., Yuille, A. L. and Zhou, Y. (2021), 'Transunet: Transformers make strong encoders for medical image segmentation', *arXiv preprint arXiv:2102.04306* .

Chen, L., Bentley, P., Mori, K., Misawa, K., Fujiwara, M. and Rueckert, D. (2018), 'Drinet for medical image segmentation', *IEEE transactions on medical imaging* **37**(11), 2453–2462.

Choi, K.-H. and Ha, J.-E. (2022), Semantic segmentation with perceiver io, *in* '2022 22nd International Conference on Control, Automation and Systems (ICCAS)', IEEE, pp. 1607–1610.

Deng, L. (2012), 'The mnist database of handwritten digit images for machine learning research [best of the web]', *IEEE signal processing magazine* **29**(6), 141–142.

Devlin, J., Chang, M.-W., Lee, K. and Toutanova, K. (2018), 'Bert: Pre-training of deep bidirectional transformers for language understanding', *arXiv preprint arXiv:1810.04805* .

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S. et al. (2020), 'An image is worth 16x16 words: Transformers for image recognition at scale', *arXiv preprint arXiv:2010.11929* .

Falcon, W. and The PyTorch Lightning team (2019), 'PyTorch Lightning'.
URL: *https://github.com/Lightning-AI/lightning*

Hatamizadeh, A., Tang, Y., Nath, V., Yang, D., Myronenko, A., Landman, B., Roth, H. R. and Xu, D. (2022), Unetr: Transformers for 3d medical image segmentation, *in* 'Proceedings of the IEEE/CVF winter conference on applications of computer vision', pp. 574–584.

Huang, X., Deng, Z., Li, D. and Yuan, X. (2021), 'Missformer: An effective medical image segmentation transformer', *arXiv preprint arXiv:2109.07162* .

Jaegle, A., Borgeaud, S., Alayrac, J.-B., Doersch, C., Ionescu, C., Ding, D., Koppula, S., Zoran, D., Brock, A., Shelhamer, E. et al. (2021), 'Perceiver io: A general architecture for structured inputs & outputs', *arXiv preprint arXiv:2107.14795* .

Jaegle, A., Gimeno, F., Brock, A., Vinyals, O., Zisserman, A. and Carreira, J. (2021), Perceiver: General perception with iterative attention, *in* 'International conference on machine learning', PMLR, pp. 4651–4664.

Ji, Y., Bai, H., Yang, J., Ge, C., Zhu, Y., Zhang, R., Li, Z., Zhang, L., Ma, W., Wan, X. et al. (2022), 'Amos: A large-scale abdominal multi-organ benchmark for versatile medical image segmentation', *arXiv preprint arXiv:2206.08023* .

Kaul, C., Manandhar, S. and Pears, N. (2019), Focusnet: An attention-based fully convolutional network for medical image segmentation, *in* '2019 IEEE 16th international symposium on biomedical imaging (ISBI 2019)', IEEE, pp. 455–458.

Kayalibay, B., Jensen, G. and van der Smagt, P. (2017), 'Cnn-based segmentation of medical imaging data'.
URL: *https://arxiv.org/abs/1701.03056*

Krasser, M. and Stumpf, C. (2023), 'A PyTorch implementation of Perceiver, Perceiver IO and Perceiver AR with PyTorch Lightning scripts for distributed training.'.
URL: *https://github.com/krasserm/perceiver-io*

Lee, J.-G., Jun, S., Cho, Y.-W., Lee, H., Kim, G. B., Seo, J. B. and Kim, N. (2017), 'Deep learning in medical imaging: general overview', *Korean journal of radiology* **18**(4), 570–584.

Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S. and Guo, B. (2021), Swin transformer: Hierarchical vision transformer using shifted windows, *in* 'Proceedings of the IEEE/CVF international conference on computer vision', pp. 10012–10022.

Lowekamp, B. C., Chen, D. T., Ibáñez, L. and Blezek, D. (2013), 'The design of simpleitk', *Frontiers in neuroinformatics* **7**, 45.

maintainers, T. and contributors (2016), 'TorchVision: PyTorch's Computer Vision library'.
URL: *https://github.com/pytorch/vision*

Monteiro, M., Newcombe, V. F., Mathieu, F., Adatia, K., Kamnitsas, K., Ferrante, E., Das, T., Whitehouse, D., Rueckert, D., Menon, D. K. et al. (2020), 'Multiclass semantic segmentation and quantification of traumatic brain injury lesions on head ct using deep learning: an algorithm development and multicentre validation study', *The Lancet Digital Health* **2**(6), e314–e322.

Mosinska, A., Marquez-Neila, P., Koziński, M. and Fua, P. (2018), Beyond the pixel-wise loss for topology-aware delineation, *in* 'Proceedings of the IEEE conference on computer vision and pattern recognition', pp. 3136–3145.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J. and Chintala, S. (2019), PyTorch: An Imperative Style, High-Performance Deep Learning Library, *in* H. Wallach, H. Larochelle,

A. Beygelzimer, F. d'Alché Buc, E. Fox and R. Garnett, eds, 'Advances in Neural Information Processing Systems 32', Curran Associates, Inc., pp. 8024–8035.
URL: *http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf*

Patil, D. D. and Deore, S. G. (2013), 'Medical image segmentation: a review', *International Journal of Computer Science and Mobile Computing* 2(1), 22–27.

Pham, D. L., Xu, C. and Prince, J. L. (2000), 'Current methods in medical image segmentation', *Annual review of biomedical engineering* 2(1), 315–337.

Rahman, M. M., Sadique, M. S., Temtam, A. G., Farzana, W., Vidyaratne, L. and Iftekharuddin, K. M. (2022), Brain tumor segmentation using unet-context encoding network, *in* 'Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries: 7th International Workshop, BrainLes 2021, Held in Conjunction with MICCAI 2021, Virtual Event, September 27, 2021, Revised Selected Papers, Part I', Springer, pp. 463–472.

Rapin, J., Gallagher, M., Kerschke, P., Preuss, M. and Teytaud, O. (2019), Exploring the mlda benchmark on the nevergrad platform, *in* 'Proceedings of the Genetic and Evolutionary Computation Conference Companion', pp. 1888–1896.

Rapin, J. and Teytaud, O. (2018), 'Nevergrad - A gradient-free optimization platform', `https://GitHub.com/FacebookResearch/Nevergrad`.

Rogowska, J. (2000), 'Overview and fundamentals of medical image segmentation', *Handbook of medical imaging, processing and analysis* pp. 69–85.

Ronneberger, O., Fischer, P. and Brox, T. (2015), 'U-net: Convolutional networks for biomedical image segmentation'.
URL: *https://arxiv.org/abs/1505.04597*

Sharma, N., Aggarwal, L. M. et al. (2010), 'Automated medical image segmentation techniques', *Journal of medical physics* 35(1), 3.

Sinha, A. and Dolz, J. (2020), 'Multi-scale self-guided attention for medical image segmentation', *IEEE journal of biomedical and health informatics* 25(1), 121–130.

Tan, J., Wang, Y., Wu, G. and Wang, L. (2022), 'Temporal perceiver: A general architecture for arbitrary boundary detection', *arXiv preprint arXiv:2203.00307* .

Tian, J., Liu, L., Shi, Z. and Xu, F. (2019), Automatic couinaud segmentation from ct volumes on liver using glc-unet, *in* 'Machine Learning in Medical Imaging: 10th International Workshop, MLMI 2019, Held in Conjunction with MICCAI 2019, Shenzhen, China, October 13, 2019, Proceedings', Springer, pp. 274–282.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł. and Polosukhin, I. (2017), 'Attention is all you need', *Advances in neural information processing systems* **30**.

Wang, C.-Y., Bochkovskiy, A. and Liao, H.-Y. M. (2022), 'Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors', *arXiv preprint arXiv:2207.02696* .

Wang, W., Yu, K., Hugonot, J., Fua, P. and Salzmann, M. (2019), Recurrent u-net for resource-constrained segmentation, *in* 'Proceedings of the IEEE/CVF international conference on computer vision', pp. 2142–2151.

Wang, Y., Zhou, Y., Shen, W., Park, S., Fishman, E. K. and Yuille, A. L. (2019), 'Abdominal multi-organ segmentation with organ-attention networks and statistical fusion', *Medical image analysis* **55**, 88–102.

Yaniv, Z., Lowekamp, B. C., Johnson, H. J. and Beare, R. (2018), 'Simpleitk image-analysis notebooks: a collaborative environment for education and reproducible research', *Journal of digital imaging* **31**(3), 290–303.

You, Y., Li, J., Hseu, J., Song, X., Demmel, J. and Hsieh, C.-J. (2019), 'Reducing bert pre-training time from 3 days to 76 minutes', *arXiv preprint arXiv:1904.00962* .

Yushkevich, P. A., Piven, J., Cody Hazlett, H., Gimpel Smith, R., Ho, S., Gee, J. C. and Gerig, G. (2006), 'User-guided 3D active contour segmentation of anatomical structures: Significantly improved efficiency and reliability', *Neuroimage* **31**(3), 1116–1128.

Zhang, J., Ji, B., Jiang, Z. and Qin, J. (2021), Cr-unet: Context-rich unet for liver segmentation from ct volumes, *in* '2021 International Conference on Electronic Information Engineering and Computer Science (EIECS)', IEEE, pp. 282–285.

Zhang, S., Fu, H., Yan, Y., Zhang, Y., Wu, Q., Yang, M., Tan, M. and Xu, Y. (2019), Attention guided network for retinal image segmentation, *in* 'Medical Image Computing and Computer Assisted Intervention–MICCAI 2019: 22nd International Conference, Shenzhen, China, October 13–17, 2019, Proceedings, Part I 22', Springer, pp. 797–805.

Zhao, T., Cao, K., Yao, J., Nogues, I., Lu, L., Huang, L., Xiao, J., Yin, Z. and Zhang, L. (2021), 3d graph anatomy geometry-integrated network for pancreatic mass segmentation, diagnosis, and quantitative patient management, *in* 'Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)', pp. 13743–13752.

Zhou, S., Nie, D., Adeli, E., Yin, J., Lian, J. and Shen, D. (2019), 'High-resolution encoder–decoder networks for low-contrast medical image segmentation', *IEEE Transactions on Image Processing* **29**, 461–475.