Linköping University | Department of Computer and Information Science
Master's thesis, 30 ECTS | Datateknik
21 | LIU-IDA/STAT-A--21/007--SE

Using Deep Learning to Segment Cardiovascular 4D Flow MRI

– 3D U-Net for cardiovascular 4D flow MRI segmentation and Bayesian 3D U-Net for uncertainty estimation

---

Använda djupinlärning för 4D-flöde kardiovaskulär MR-segmentering

Omkar Bhutra

Supervisor : Johan Alenlöv
Examiner : Per Sidén

External supervisor : Mariana Bustamante

LINKÖPING UNIVERSITY

Linköpings universitet
SE–581 83 Linköping
+46 13 28 10 00 , www.liu.se

Abstract

Deep convolutional neural networks (CNN's) have achieved state-of-the-art accuracies for multi-class segmentation in biomedical image science. In this thesis, A 3D U-Net is used to segment 4D flow Magnetic Resonance Images that include the heart and its large vessels. The 4 dimensional flow MRI dataset has been segmented and validated using a multi-atlas based registration technique. This multi-atlas based technique resulted in high quality segmentations, with the disadvantage of long computation times typically required by three-dimensional registration techniques. The 3D U-Net framework learns to classify voxels by transforming the information about the segmentation into a latent feature space in a contracting path and upsampling them to semantic segmentation in an expanding path. A CNN trained using a sufficiently diverse set of volumes at different time intervals of the diastole and systole should be able to handle more extreme morphological differences between subjects. Evaluation of the results is based on metric for segmentation evaluation such as Dice coefficient. Uncertainty is estimated using a bayesian implementation of the 3D U-Net of similar architecture.

Keywords: Convolutional Neural Networks, 4D flow MRI, 3D U-Net , Cardiovascular image segmentation, Uncertainty estimation, Bayesian 3D U-Net

Acknowledgments

I would like to thank Mariana Bustamante, my external supervisor at the Center for Medical Image science and Visualisation, for the continuous support and advice without which this thesis would not be possible.

I would also like to thank Johan Alenlov, my internal supervisor at Linköping University for the guidance provided which is required for an indepth study of the subject.

The feedback provided by the examiner, Per Siden and the program director, Oleg Sysoev has proved invaluable in the process of the research.

I would like to mention the Center for Medical Image science and Visualisation at University Hospital, Linköping for providing me with the opportunity to perform this thesis for which I am forever grateful.

I am thankful for the continuous support of my parents in all my endeavors.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

MRI (Magnetic Resonance Imaging) has many advantages in biomedical image analysis such as high spatial and temporal resolution, Excellent signal to noise ratio, Free choice of imaging planes and No requirement for geometric assumptions [33] [8].

MRI allows doctors to see the not only organs and tissues but also bones inside the human body without having to surgically operate. MRI's are important tests that can help diagnose a disease and even injuries such as Swelling or blockages in blood vessels, Heart attack damage, Problems related to heart valves, Tissue inflammation that surrounds the heart, Problems with the aorta, Structural problems with the heart walls and chambers, Tumors inside the heart [52].

Whole-heart segmentation's from 3D cardiovascular magnetic resonance images (MRI) play an important role in the diagnosis and treatment planning of cardiovascular disease. Medical doctors can go through cardiac MRI's and find the defects and problems. Oftentimes, analysis is also required where the MRI has to evaluated by quantitative measures for the same. Semantic segmentation allows for such evaluation and can not only be used by medical personnel directly but also be used for automatic diagnosis and prediction of disease.

Computer assisted systems provide an automatic, accurate and quick solution. However, this task is challenging due to varied morphological differences between different patients, ambiguous cardiac boundaries and requirement of qualified personnel for segmentation tasks.

The four chambers of the heart, namely the Right and Left Ventricule and the Right and Left Atrium along with the great vessels, namely the Aorta and Pulmonary artery constitute a majority of the whole heart system.

We propose the use of a three dimensional convolutional neural network architecture called the 3D U-Net and the bayesian implementation of the same architecture be used to segment the 3d images extracted from 4D flow cardiovascular MRI and to estimate uncertainty respectively.

Manual segmentation on every MR slice is tedious and time-consuming, and is subject to inter- and intra-observer variability. An automatic segmentation tool is in high demand in clinical practice.

The proposed methods can not only save the time of the qualified medical personnel to tackle more important issues but also provide a framework for future work based in biomedical image segmentation.

A previously evaluated method based on atlas segmentation has been used to generate labels for the aorta, pulmonary artery, and the four cardiac chambers at each timeframe for all datasets [42] [7].

Using 4D flow Magnetic Resonance Imaging (MRI), the velocity of blood can be measured in the whole thorax in a few minutes. 4D flow MRI is used to diagnose and assess blood flow dynamics in complex diseases. 4D phase-contrast MRI datasets corresponding to 200 subjects have already been acquired and are available and ethically approved to use for this project [42]. Each dataset is composed of a volume of size around 112x112x44x40, which can be sliced into 2D or 3D volumes of different shapes for training and validation purposes.

Semantic segmentation of an image is a task in computer image analysis in which regions of interest are labelled according to what is being presented to the segmentation system [3]. Whole heart segmentation is subject to both inter-observer and intra-observer variability due to a range of factors such as ambiguous ventricule boundaries and morphological differences among subjects. Segmentation tasks routinely requires the annotation of an expert from the field of cardiology. Such a task may either be fully automatic or be semi-automatic with segmentation done by an expert during the process of training.

There is a need for uncertainty estimation in biomedical semantic segmentation. Let us consider a thought experiment; A hospital's medical imagery and visualisation lab is responsible for highly critical patient data such as cardiovascular MRI. Since the cardiovascular part segmentation must be guaranteed to work, the hospital validates each of them painstakingly where they take a 4D MRI scan of the part, annotate the millions of voxels by manually with the help of expert cardiologists, and use the annotated and segmented scan to analyze the part for defects.

Such a process is critical for patient safety but it is not time or cost effective. So the hospital hires a team of statisticians and machine learning engineers to design a deep neural network which automatically segments and validates the cardiovascular parts using the current state-of-the-art volumetric segmentation techniques. The neural-network works and is successful for most of the cases. But, when a patient arrives whose heart is very different from most and the patient has an undiagnosed cardiovascular disease. The neural network in use currently does not recognise this and wrongly segments the patients deformed cardiovascular system.

What was the problem? A deep neural network is known for the accurate segmentation's, but one of major disadvantages is that the uncertainty of its predictions remain unquantifiable. Therefore, the neural network did not have a method to differentiate between segmentations that are definitely correct and the one that just passes.

1.2 Aim

1. Development of a deep-learning based method to segment the cardiac chambers and major vessels in 4D Flow MR images.

2. Evaluation of the results using traditional segmentation measures such as Dice coefficient.

3. Implement a Bayesian Convolutional Neural Network; a Bayesian 3D U-Net which can assist in uncertainty estimation via variational inference.

1.3 Research questions

The data procured for research is 4 dimensional i.e 3D volumes of the MRI scan for each time step. The subjects or patients belong to two large categories of patients with mild and severe

cardiovascular disease [50] [42]. The subjects are injected with Gadolinium contrast media (also known as MRI contrast media, agents or 'dyes'). These are chemical substances used before the MRI scanning. When injected into the patients, the gadolinium contrast medium enhances the quality of the MR images. [15]

1. What type of Neural Network architecture is to be used? There is possibility to slice the data into 2D area and 3D volumes for further use. This also increases the possibility of different network architectures that can be used to train a neural network such as the 3D U-Net, 2D U-Net, Anatomically constrained CNN's, Fully Convolutional Neural Networks for semantic segmentation, V-Net [9] [43] [35] [28] [31].

2. Statistical Methods for performance evaluation? How can we evaluate the segmentation? Uncertainty estimation of the output of the neural network can be done via variational inference. Bayesian methods can be used to produce credible intervals. Commonly used segmentation measures such as the Dice coefficient and variational free energy loss can be used to train the multi-class and bayesian networks respectively [27] [46].

3. What kind of data transformation strategies are to be adopted? The purpose of data augmentation is to increase the variability within the training data, so that the trained network will be able to generalise better on unseen data. The data transformation strategies must be selected so as to replicate what is plausible during the scanning process of a patient. [40] [39] [36]

1.4 Delimitations

Deep learning based approaches require a large amount of computational resources and require training for a long duration. This translates to high computational costs in terms of GPU resources utilised. Deep learning based methods are often criticised to be opaque, where even the designer of the network is specifically unaware of the internal pattern recognition mechanisms by which the network learns to solve the problem [2].

2 Theory

2.1 Related Work

Biomedical 2D image segmentation using Convolutional Neural Networks (CNN) can be done with accuracy comparable to human performance [43]. Owing to the success, serveral attempts have been made to apply 3D CNN's to biomedical segmentation problems. [55] [9] [51]

The original U-Net is entirely a 2D architecture, and likewise some Dense Neural Network architectures for full heart segmentation [43], 3D U-Net was first described in Cicek et al. [9]. A solution to process volumetric data is to take three perpendicular 2D slices as input and fuse the multi-view information abstraction for 3D segmentation [32].

One can directly replace 2D operations in the original U-Net with their 3D counterparts [44], this 3D U-Net has been applied to multi class whole heart segmentation on MR and CT data [55]. However, due to memory constraints of the GPU's these methods have either downsampled the input 3D data which leads to loss of resolution or only predict subvolumes of the data [13] [23] [18].

Similarly other methods that first successfully extract region of interests (ROI) using a localisation network and then apply that to segmentation U-Net [51]. This method makes uses of 2 stages of 3D U-Nets.

Our work is based on the 3D U-Net to segment 3D Volumes of the whole heart extracted from 4D flow MRI that has been previously segmented using multi-atlas based methods [30] [7] [9].

2.2 Feed Forward Neural Network

A neural network can be viewed as a mathematical function that draws inspiration from the structure of neurons in mammalian brains. Similar to how the input is forwarded to the brain for further processing on any of the five human senses being activated. Let $y = f(x)$ be the underlying function that outlines the relation between variable y and explanatory variable x, the neural network is a non-linear mapping $f(x;\theta)$ that approximates this value of y by learning the paramaters θ [2]

When a neuron receives an input satisfying the specific threshold value, it gets activated and forwards the input to the adjacent neurons. If the sum of the inputs that is collected from multiple adjacent neurons exceeds their threshold, they are activated too. Information flows through the network following such a schema. Another way of describing a neuron is the *perceptron*. In statistical terms, the perceptron is a linear and binary classifier. Figure 2.1 shows a perceptron with inputs coming from the left hand side.

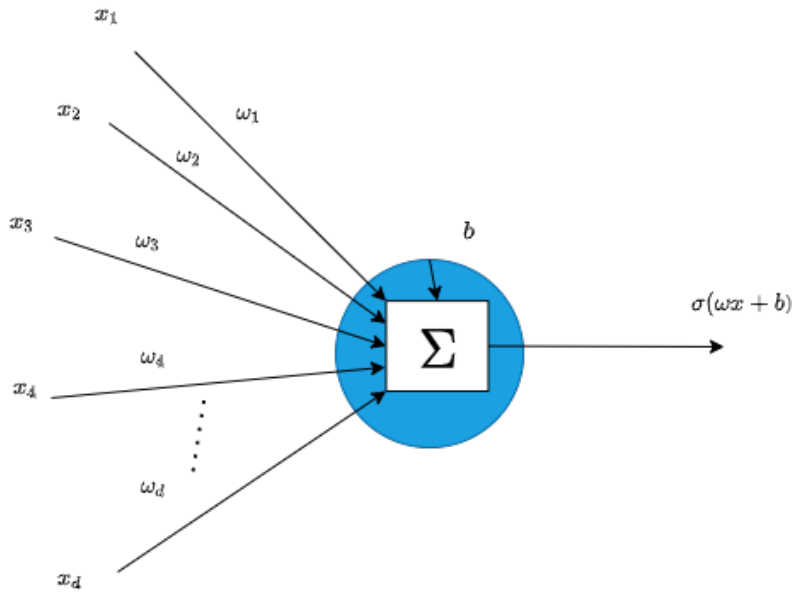

Figure 2.1: **Perceptron** : The input x with d dimensions, weights w and the bias b. The output is a linear function of the weights and the input added to the bias. An activation function, denoted by σ, is applied to the output to limit the range of values (σ is *sigmoid* activation function compressing the output between 0 and 1).

Lets consider an input vector x of size d. The linear function of the perceptron processes the sum of the input and *weights* w with bias added to it. The *intercept* of the linear function is now determined.

A combination of many perceptrons produces a network that resembles a neural network. The resulting model is usually either called a *Feed Forward Network* (FFN) or *Multi-Layer Perceptron* (MLP). Figure 2.2 shows a simple neural network of multiple layers with multiple perceptrons.

The first layer of the neural network is called the *input layer* , the last layer that contains the output is called the *output layer* and layers in between are called *hidden layers*. The computation is vectorised, therefore the variable x is of the actual dimensions $n \times d$, with n denoting the number of observations and d the dimensionality of the input. The *forward pass* is composed of

$$h(x) = \sigma(x^T \cdot w_1 + b) \tag{2.1}$$

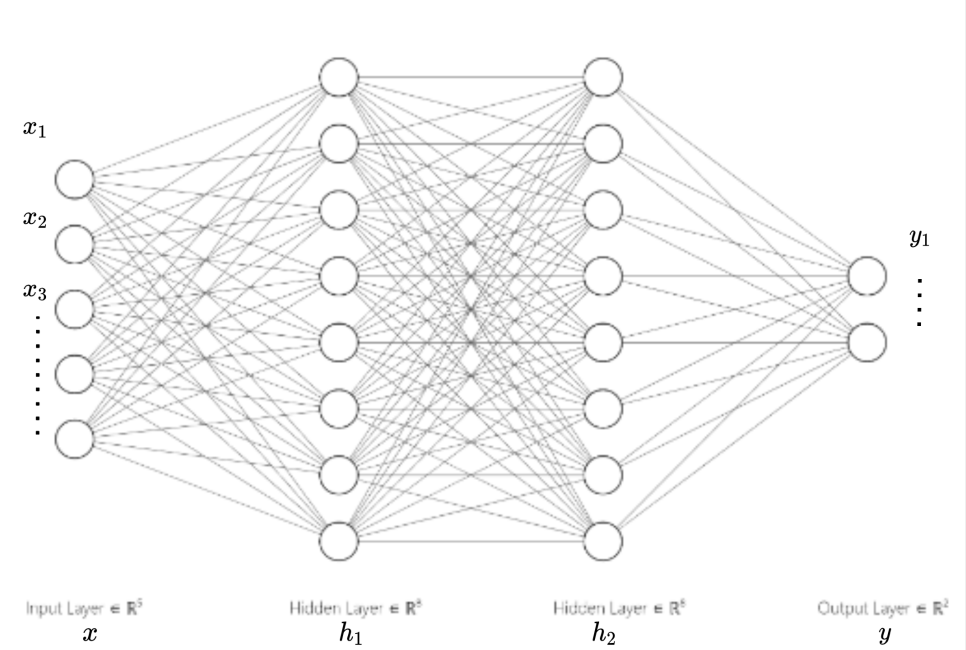

Figure 2.2: An illustrative **feed-forward neural network or Multi-Layer perceptron**.

and

$$y(h) = \sigma(h^T \cdot w_2 + b) \tag{2.2}$$

where, w_1 and w_2 denote the weights and b_1 and b_2 denote the bias in the respective layers. The *Sigmoid* activation function denoted by σ is a mapping of the output between $(0, 1)$ in the equation (2.3). A different activation functions can also be chosen for application.

$$\sigma(x) = \frac{1}{1 + \exp(-x)} = \frac{\exp(x)}{\exp(x) + 1} \tag{2.3}$$

In the subsequent step, we define a *loss* function and apply *backpropagation* to update the weights. The error in the output of the neural network is defined by the loss function. The loss function used depends on the application, for instance, a classification, regression or as in our case, segmentation problem. An assumption can be made that the error is normally distributed about the prediction y. Hence, The *Maximum Likelihood Estimate* (MLE) for the normal distribution can be optimised by maximising the *Mean Squared Error* (MSE), where y denotes the true value and $\hat{y}$ denotes the estimated value.

$$\epsilon(\hat{y}) = \frac{1}{n} \sum_{i=1}^{n} (\hat{y} - y)^2 \tag{2.4}$$

The gradient can now be computed with respect to the corresponding weights and updated. The learning rate α decides the magnitude of change in the weights after each *iteration* or *epoch* i.e one pass of the neural network training over the training dataset. For the purpose of application more complex *optimisers* along with *momentum* are used for the training to converge faster to a solution. The equations and process detailing out the weight update mechanism are referred from [21]. One gradient update can be shown in the gradient of the error function with respect to its weights

$$\nabla \epsilon = \left(\frac{\delta \epsilon}{\delta w_n}, \frac{\delta \epsilon}{\delta w_{n-1}}, \ldots, \frac{\delta \epsilon}{\delta w_1} \right) \tag{2.5}$$

the *chain rule of partial derivatives* can be used to compute the gradient. For the w_2, gradient is

$$\frac{\delta \epsilon}{\delta w_2} = \frac{\delta \epsilon}{\delta \hat{y}} \frac{\delta \hat{y}}{\delta h} \frac{\delta h}{\delta w_2} \tag{2.6}$$

Hence, the weight can be updated according the equation

$$w_2^{'} = w_2 - \alpha \odot \frac{\delta \epsilon}{\delta w_2} \tag{2.7}$$

where, $\odot$ defines the *element-wise product* and α is the *learning rate* which decides the magnitude of change of the weight update.Sometimes, the updates to the structure of the neural network may be required which changes the analytical solution for gradient updates. The application of numerical optimisation techniques can allow for such a change without requiring the derivation of the analytical solution for such a gradient update. Deep neural networks are susceptible to the *vanishing gradient/exploding gradient problem* [38], which is the reason for the application of more complex types of neural networks.

2.3 Convolutional Neural Networks

Inputs to the neural network may have features that are highly correlated to the features adjacent to them. This is generally observed in image classification and segmentation problems. Convolutional Neural Networks (CNNs) can be used for extracting information in such cases where feature extraction from images is required for the later tasks such as regression, classification and segmentation. In theory, a conventional feed forward neural network is also capable of image processing, with the drawback of higher number of parameters required to be learnt as compared to a CNN. Instead of using a weight for each pixel of the image, a CNN uses a *filter* over the inputs i.e a *sliding window* with a certain *stride* over the pixel intensities to create an intermediary output that is a function of the *input* and the *filter*, this process is termed *convolution*.

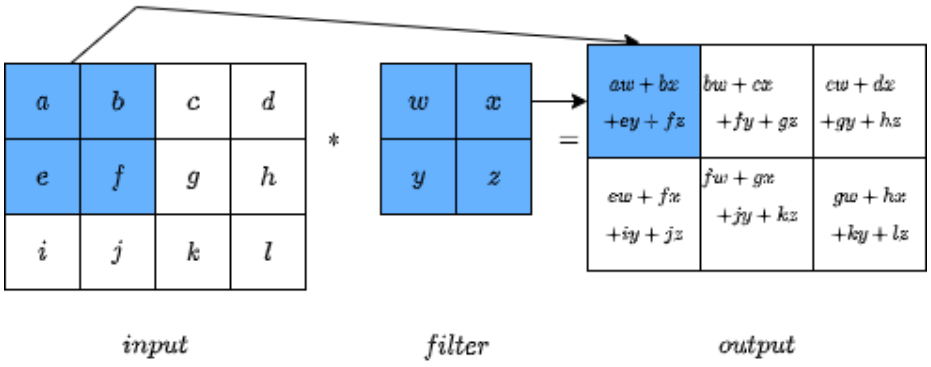

Figure 2.3: Illustrative 2D convolution: The dot product of **input** and **filter** results in the **output**.

The intensity/brightness of each pixel of the image (voxel in 3D case), is generally rescaled and mapped to a range such as $(0, 1)$. This can be seen in figure 2.3 as an example. It is noted that, *zero-padding* may be performed on the input to allow for convolution involving all the input pixel or voxel intensities i.e 0 intensity pixels are added around all the edges and corners. Other forms of padding are also adopted depending on the application. During each convolution step as described above, a product of the input and filter is computed. In the case of the figure 2.3 with 2D inputs or pixels, input shape is 1×4 and 4×1 which results

in an output of shape 1×1. The stride of 1, moves the sliding window of 2×2 to the next position with the output being computed on the right hand side. Increasing the strides will result in a smaller output size, it can be noted that the size of the output is always smaller than the input size.

The convolution operation in the 2D space can be defined as in [21].

$$y_{ij} = \sum_{a=0}^{\lfloor m/s \rfloor} \sum_{b=0}^{\lfloor m/s \rfloor} x_{1+as,j+bs} \mathcal{F}_{a,b} \tag{2.8}$$

where x denotes the input at indices i and j, $\mathcal{F}$ denotes filter, m denotes the size of the filter and s is the stride.

For the purpose of extracting features from images, edges are of interest. Therefore, a difference between adjacent pixels/voxels can determine the edge, as in the case of foreground vs background. Size reduction of the input is also a concern for faster image processing, can be done by a method called *pooling*, an illustrative example of *maxpooling* can be seen in figure 2.4.

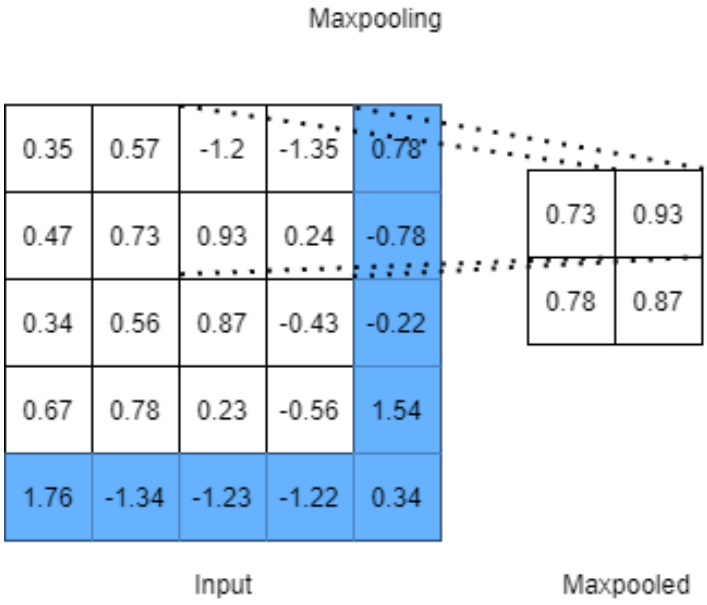

Figure 2.4: Maxpooling is beinng performed on the **input** by a window of size 2×2. The cases with **edges** are either padded with zeros or just ignored.

The concept of *pooling* can also be described as sliding window of mathematical operation such as **max** applied in each case. Generally, no overlap is considered between the region selected. This can result in the edge cases being overlooked, but a padding may also be used. Settings related to pooling and filter size are both considered arbitrarily decided and hence, considered to be a hyperparameter of the neural network.

The filter that is seen in the figure 2.3 is a *high pass filter*. A CNN will be of limited use, if it could only learn with one filter i.e it will be able to extract only one type of features from the input. For instance, an *edge* detection filter works where the intensity changes drastically (example: from black to white). Hence, multiple independently operating filters are used for training. In image processing, one filter is used for each of the RGB channels if available, else the number of filters is decided by the modeller. The construction of a CNN involves multiple layers of filtering and pooling which provide feature set to network resembling a feed forward network for the purpose of the end goal task such as classification, regression or segmentation.

2.4 Upsampling

For a neural network to generate images or image maps such as in the case of semantic segmentation, it generally involves the application of upsampling from a low resolution to a higher resolution. There are many different methods to perform an upsampling operation such as nearest neighbor interpolation, linear interpolation, bilinear interpolation, trilinear interpolation, bicubic interpolation and various others found in literature [49].

Lets go backwards in the figure 2.3 from the section 2.3 , if we want to associate 1 value to 9 other values in a matrix, it will be termed a one-to-many relationship. This is similar to going backwards in a convolution operation.

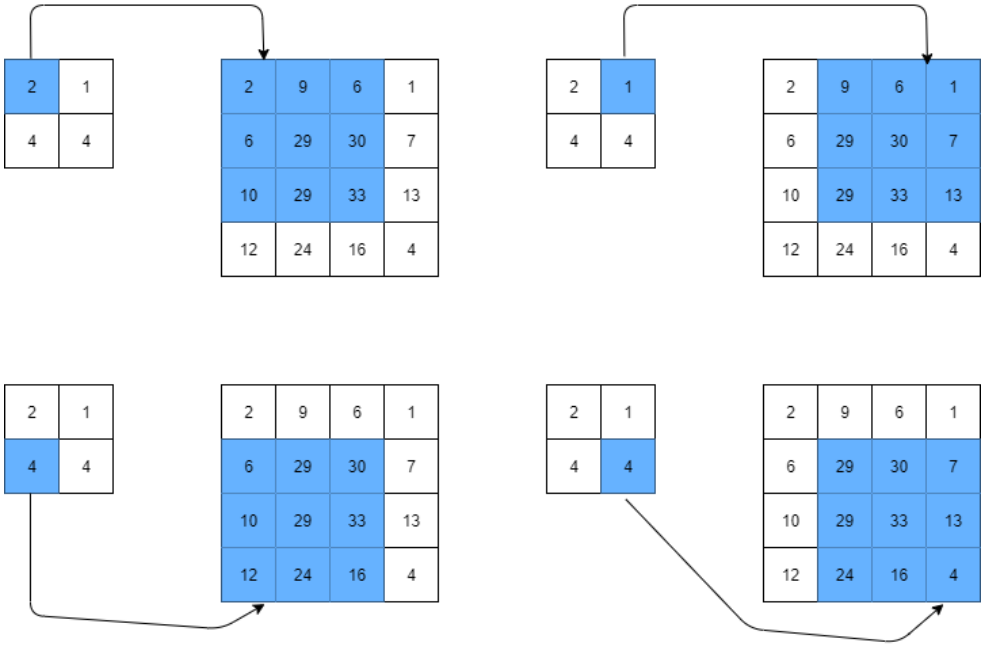

Figure 2.5: Illustrative **Upsampling - Going backward of a convolution**. Refer section 2.4.2

2.4.1 Convolution operation

When a convolution operation is applied on a 4x4 matrix using a 3x3 kernel without padding and a stride of 1, the output will be a 2x2 matrix.

The operation in figure 2.6 computes the sum of element-wise matrix multiplication between the input and kernel. This can be done only 4 times in the illustrative example.

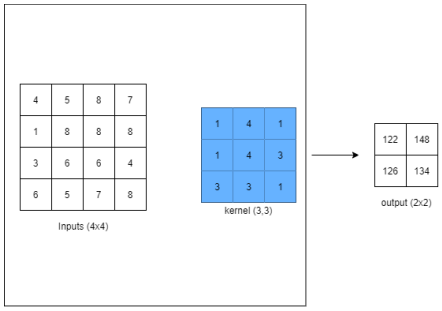

Figure 2.6: Illustrative **Convolution operation**

2.4.2 Transposed Convolution

A convolution operation can be represented as a kernel matrix that is arranged in the form for matrix multiplication to perform convolution operations. Shown in the figure 2.7.

Figure 2.7: Illustrative **Convolution kernel**

This kernel can be arranged as in the figure 2.8

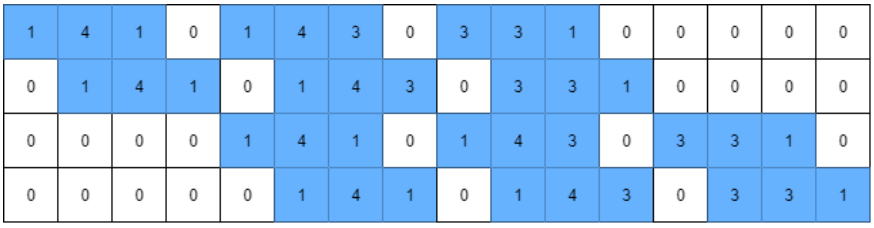

Figure 2.8: Illustrative **Convolution matrix (4,16)**.

Each row in figure 2.8 is defined by one convoution operation. Each row is a rearranged kernel matrix with zero padding at different places.

To use this, the input matrix from figure 2.6 can be flattened from (4x4) to (16x1). The matrix multiplication of 2.8 and the flattened column vector of the input leads to the vector of output (4x1) which can be reshaped so that it is (2x2) as seen in the figure 2.6.

To increase the size from (2x2) to to (4x4), a 16x4 matrix is used. However, the 1 to 9 relationship has to be maintained. The transpose of the convolution matrix in figure 2.8 from (4x16) to (16x4), this can be matrix multiplied with the column vector of the output to generate the output matrix as seen in the figure 2.5.

2.5 Activation Layer

Layers of nodes are present within neural networks that learn mapping of input instances to ouputs

In any given node, the sum of products of the inputs and weights is found which is called the summed activation of the node. Transformation of this value using an activation function is what defines the specific output of the node, also known as 'activation' of the node.

A relatively simple activation function is a linear activation function that involves no transformation. A neural network that consists of only linear activation functions is simple to train but is unable to learn more complicated mapping functions that exist in the real problems. The output layer of the network uses linear activation for the purpose of prediction such as in case of regression problems.

A non-linear activation function can learn complex features within the data. Some examples of non-linear activation functions are: sigmoid, ReLU, PReLU, hyperbolic tangent.

2.5.1 Rectified Linear Unit

For effective application of backpropagation of the errors along with stochastic gradient descent in order to train neural networks, the activation function in use must behave similar to a linear activation function. However, only non-linear activation functions allow for learning of complex structures and relationships within the data [41] [19].

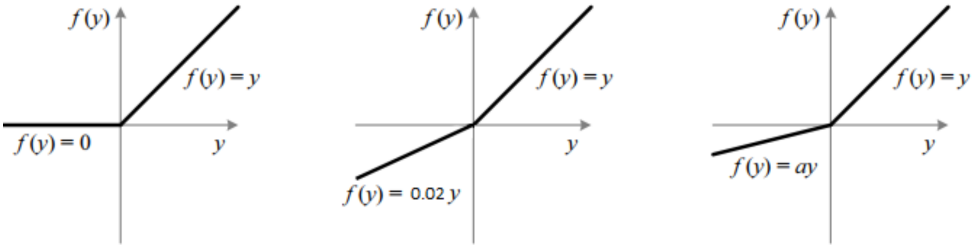

Figure 2.9: **Rectified Linear Activation Function (left), PReLU will be used for multi-class 3D U-Net and ReLU will be used for the bayesian implementation (right) , leaky ReLU (middle)**.

Sensitivity to the activation function's summed input is important to avoid running out of values to output. A rectified linear activation can be used to solve this problem also known as ReL.

A unit or a node that implements ReL activation function is called ReLU or Rectified Linear Unit. Neural networks that use rectifier funcitons are often called rectified networks.

The adoption of ReLU has allowed researchers to implement deep neural networks efficiently.

2.5.2 Parametric Rectified Linear Unit

ReLU has its own limitations such as the problem with large weight updates where the summation of the input to the activation function remains negative, however the input. A node with such a problem will have an output value of 0, this is also known as a 'dying ReLU' Small negative values can be allowed into the function which effectively reduces the nonlinearity of ReLU, such as Leaky ReLU or LReLU which is a modified ReLU function where the input is < 0.

The Parametric ReLU or PReLU can train to learn parameters that control how 'leaky' the activation function will be [41] [19].

$$\text{loss} = \begin{cases} y_i, & \text{if} \quad y_i > 0 \\ a_i y_i, & \text{if} \quad y_i <= 0 \end{cases} \tag{2.9}$$

Above, y_i is any input on the ith channel and a is the negative slope which is a learnable parameter. if a=0, f becomes ReLU if a>0, f becomes leaky ReLU if a_i is a learnable parameter, f becomes PReLU

Above formula can also be written as

$$(f(y_i)) = max(0, y_i) + a_i min(0, y_i) \tag{2.10}$$

For backpropagation, the gradient is

$$(\frac{\partial f(y_i)}{\partial a_i}) = \begin{cases} 0, \text{if} y_i > 0 \\ y_i, \text{if} y_i <= 0 \end{cases} \tag{2.11}$$

		Test segmentation		
		Positive	Negative	Total
Ground truth segmentation	Positive	TP	FN	$TP + FN$
	Negative	FP	TN	$FP + TN$
	Total	$TP + FP$	$FN + TN$	N

Table 2.1: **Illustration of a confusion matrix**

2.6 Loss Functions

2.6.1 Dice Loss

The Dice coefficient is a measure of similarity that is used to estimate the similarity between two samples. Independently developed by botanists, Sorensen and Dice with the latter using it to ascertain the ecologic association among species [12].

Basically, for our application, It is the ratio of two times the area of overlap to the total number of voxels in both the 3D images.

Confusion matrix can be constructed with the count of ground truth segmentation vs test segmentations by voxels to compute the Dice Similarity Coefficient (DSC)

TP or True positive, FP or False positive, and False negative (FN) , with a small non-negative value ϵ in the denominator to avoid indeterminate values.

$$\text{DSC} = \left(\frac{2TP}{2TP + FP + FN + \epsilon} \right) \tag{2.12}$$

$$\text{Dice Loss} = 1 - \left(\frac{2TP}{2TP + FP + FN + \epsilon} \right) \tag{2.13}$$

It is different from the Jaccard index between the predicted segmentation and ground truth, which is the ratio of the area of overlap to the area of the union. DSC is a similarity quotient with its range between 0 and 1. It can be viewed as a similarity measure over sets.

DSC is equivalent to the f1-score which is the harmonic mean between recall and precision, with value ranging from 0 to 1. Higher f1-score signifies better precision and recall.The f1-score is given by

$$\text{precision} = \left(\frac{TP}{TP + FP} \right) \tag{2.14}$$

$$\text{recall} = \left(\frac{TP}{FN + TP} \right) \tag{2.15}$$

$$\text{f1 score} = 2 \times \left(\frac{precision \times recall}{precision + recall} \right) \tag{2.16}$$

2.6.2 Cross Entropy Loss

The cross-entropy loss is defined as based on [10]. Negative log likelihood is equivalant to the binary cross entropy loss, this is applied in the training of the bayesian 3D U-Net.

$$\text{loss}(x, \text{class}) = -\log \left(\frac{\exp x[\text{class}]}{\exp \sum_i x[i]} \right) \tag{2.17}$$

where $x[i]$ denotes the probability of the output of the model to belong to class i. Cross-entropy loss can also be defined in a way that it embeds weights for each class label. In this case, the loss is given by

$$\text{loss}(x, \text{class}) = \text{weight}[\text{class}]\left(-x[\text{class}] + \log\left(\sum_i \exp x[i] \right) \right) \tag{2.18}$$

During training, an energy function is computed by a pixel-wise soft-max over the final feature map combined with the cross entropy loss function.

$$p_k(x) = exp(a_k(x)) / \sum_{k'=1)}^{[K]} exp(a'_k(x)) \tag{2.19}$$

where, the activation in feature channel k at the pixel position $x \in \Omega$ with $\Omega \subset \mathbb{Z}^2$ is denoted by $a_k(x)$. K is the number of classes and $p_k(x)$ is the approximated maximum-function. i.e. $p_k(x) \approx 1$ for the k that has the maximum activation $a_k(x)$ and $p_k(x) \approx 0$ for all other k [11]. The cross entropy then penalizes at each position the deviation of $p_{l(x)}(x)$ from 1 using

$$E = \sum_{x \in \Omega)} \omega(x) log(p_{l(x)}(x)) \tag{2.20}$$

where $l : \Omega \rightarrow [1, ..., K]$ is the true label for each pixel and $\omega : \Omega \rightarrow [\mathbb{R}]$ is a weight map that is introduced to give some pixels more import during the training.

2.7 U-Net

Successful training of deep networks requires many thousands of annotated samples. Ronneberger et. al. [43] first presented the architecture that consists of a contracting path to capture the context and a symmetric expanding path that enables precise localisation. This method was shown to outperform previous best methods and is built upon the fully convolutional network as presented in Long et. al. [28]

A significant modification in the U-Net architecture is the upsampling part where a large number of feature channels are used, this allows the network to propagate context information to higher resolution layers. Consequently, the contracting side is symmetric to the expansive side, thus yielding the U-shaped architecture. There are no fully connected layers in the network but it uses the valid parts of each convolution i.e the pixels for which full context is full context is present in the input image are present in the segmentation map.

2.7.1 Three Dimensional U-Net

A network for efficient Volumetric segmentation was first present by Cicek et. al. [9]. We use the fully automated setup as referenced in the paper where, we assume that a sparsely annotated training set exists. Trained on this dataset, the network segments new volumetric images. This network extends the U-Net as given by Ronneberger et. al. by replacing 2D operations with the correspoding 3D operations.

Volumetric data is available in abundance in the biomedical industry. Annotation of such a data with segmentation labels comes with its cons, since only 2D slices are shown, annotation of large volumes slice by slice is inefficient. Neighboring slices mostly show the same information.

The first U-Net proposed is designed as a 2D architecture [43], while the network proposed by Cicek et. al. [9] takes 3D volumes for input to process them with corresponding 3D operations such as 3D convolutions, 3D max-pooling and also 3D upsampling layers.

Deep Convolutional Neural Networks (CNN's) are are the pinnacle in performance for multi-class segmentation of medical images. However, Most common problem when dealing

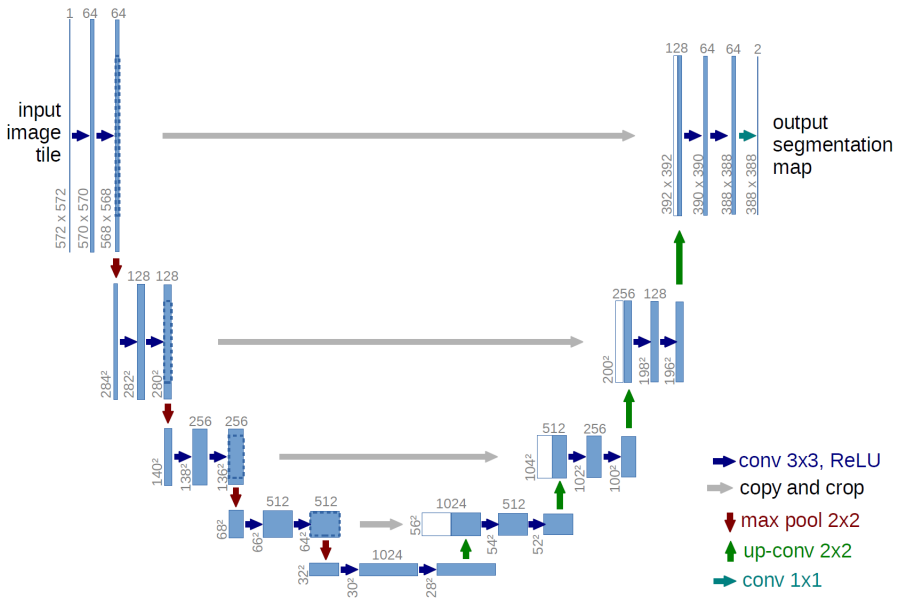

Figure 2.10: U-Net architecture (illustrative for 32x32 pixels at lowest resolution). The blue boxes represent a multi channel feature map with the number of channels present on top of each box. The size of the xy dimensions are at the bottom left of the box's edge. Copied feature maps are in white boxes. Each arrow denotes different operations. **2 Dimensional U-Net [43]**.

with high resolution and large 3D data is the volumes as input into the deep CNNs have to be either cropped or downsampled due to limitations in memory and computation. The above operations lead to loss of resolution and class imbalance in the input data batches, thus downgrade the performances of segmentation algorithms. Wang et. al. propose a two-stage modified U-Net framework applied on a variety of multi-modal 3D cardiac images for full cardiac segmentation [51]. Other efforts such as introducing several auxiliary loss functions [54], anatomically constrained cnn's where some nodes are constrained and shape priors are used [35] [13], Hierarchical 3D fully convolutional networks for multi-organ segmentation [44]. A 3D V-Net has also been proposed where the network architecture is more like a V shape since some convolution steps, copy and crop steps are skipped from the U-Net which can make it faster. A novel cost function is used to optimise the hyperparameter's [31].

2.8 Uncertainty estimation using a Bayesian 3D U-net

It can be assumed that the value of the sigmoid output/ last layer output is a usable measure of uncertainty, but this cannot be done as these values are dependent on the inferred sample being subjectively close to the training distribution, however the activation function squeezed the output. If a training sample is inferred far from the training distribution (i.e deformed/diseased part of the heart), then sigmoid output cannot be an uncertainty estimate [16]. The method to perform variational inference is laid out by Labonte et. al. in [26] [27].

The figure 2.12 illustrates the significance of uncertainty estimation. Measuring uncertainty is not possible in a regular deep neural network, but it is extremely important for the purposes of interpretability and validation.

It is observed that deep neural networks are at their best performance when the test set is similar to the training dataset; So, when the test example is different, since the network has to produce an output it may wrongly segment the input data.

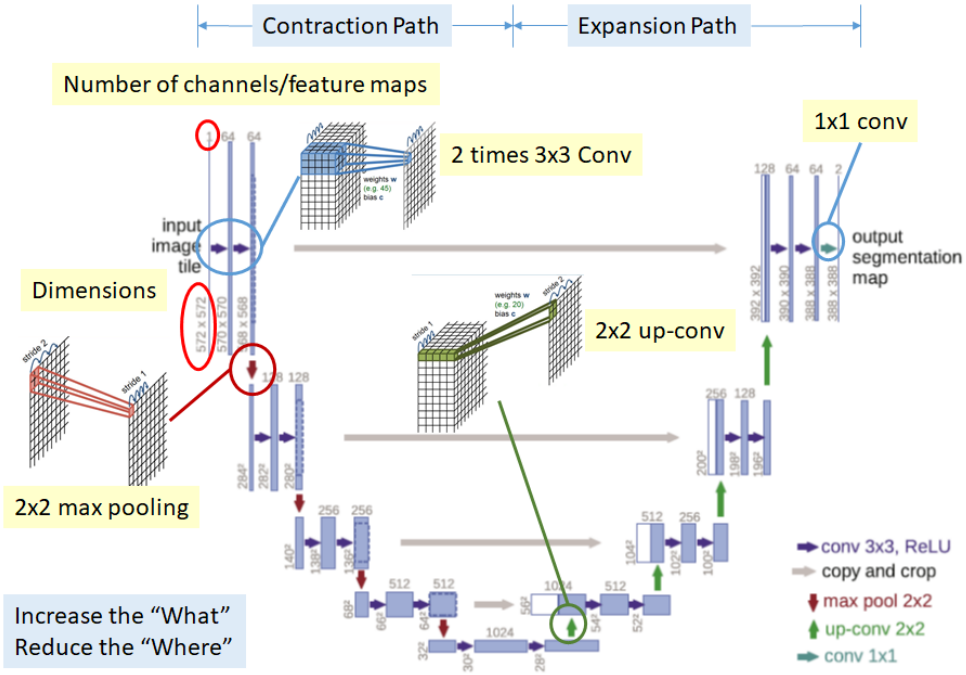

Figure 2.11: Illustrative **3 Dimensional U-Net** [9]

Monte Carlo Dropout Networks drop out layers to roughly estimate deep gaussian processes. The statistical validity has been under scrutiny [37], where it is argued that elementary decision theory proves that the only admissible decision rules are Bayesian and therefore, decision rules which are not Bayesian can be improved or used by an alternative Bayesian method. Hence, we will choose the Bayesian convolutional neural networks (BCNN) model for uncertainty quantification.

Contemporary research illustrates the implementation of neural networks as Bayesian models or probabilistic networks, which are otherwise point estimators. Commonly discussed methods are Monte Carlo Dropout Networks (MCDN) [16] and BCNN with Bayes by Backprop [6].

2.8.1 Bayesian Convolutional Neural Network

Weights are implicitly respresented as multivariate probability distributions since the point estimates do not provide information on the uncertainty inherent in the weight's values. Bayesian neural networks learn probability distributions rather than point estimates, allowing them to measure uncertainty, BCNNs use variational inference to learn the posterior distribution of the weights given the dataset. [27]

Description of the weights of the neural network as probability distributions has a variety of consequences:

1. It makes the neural network non-deterministic; At every computation of a forward pass, a point estimate is obtained by sampling from each weight distribution. Sampling by repeatition such this is called Monte Carlo Sampling and it results in different predictions that help in uncertainty analysis.

2. Backpropagation algorithm is now different because it is not possible to backpropagate through a sampling operation due to the abscence of a gradient. Bayes by Backprop algorithm has been suggested to solve this.[6]

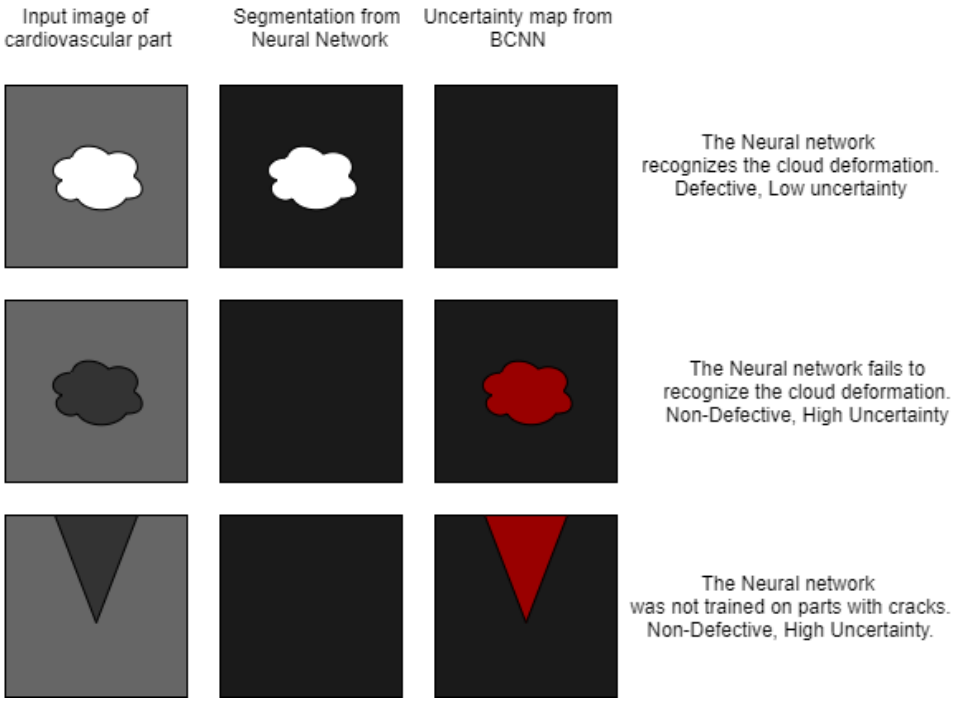

Figure 2.12: Illustrative **uncertainty map from Bayesian convolutional neural network; The second and third columns of images show the illustrative output from the two different neural networks which show that image segmentation from a neural network with point estimates can lead to poor predictions in practice**.

3. It makes the neural network susceptible to vanishing or exploding gradients and hence, difficult to train with reliability. Group normalization is suggested to solve this.

In the recent past, The massive number of weight updates necessary made Bayesian learning not feasible computationally. The solution to this problem was provided by Blundell et al. [6], it was the 'Bayes by Backprop' algorithm. It was the first algorithm to effectively train weights that are described as probability distributions in a neural network. Bayes by Backprop works by updating the posterior distribution with additional computation using the gradients computed during backpropagation to rescale and move the variational parameters of the posterior distribution.

2.8.2 Bayesian Learning

In an ideal setting, we could use Bayes Rule to calculate the distributions over the weights precisely. We begin with the prior distribution over the weights. This is our initial 'belief' of what the weight distribution. Denoted by $p(w)$ is usually a normal distribution. The data is used to calculate the posterior distributions of the weights given the data, denoted $p(w|D)$. It is equivalent to calculating the w at which the maximum likelihood of the dataset is found, given those weights, it is denoted by $p(D|w)$. This calculation is performed using the Bayes' Rule

$$p(w|D) = \frac{(p(D|w)p(w))}{p(D)} = \frac{(p(D|w)p(w))}{\int p(D|w)p(w)dw} \tag{2.21}$$

The integral in the denominator however is usually unmanagable due to the overparameterization in neural networks. Therefore, the posterior distribution is learnt instead of calculating it precisely. Variational learning, also known as variational inference is a suggested

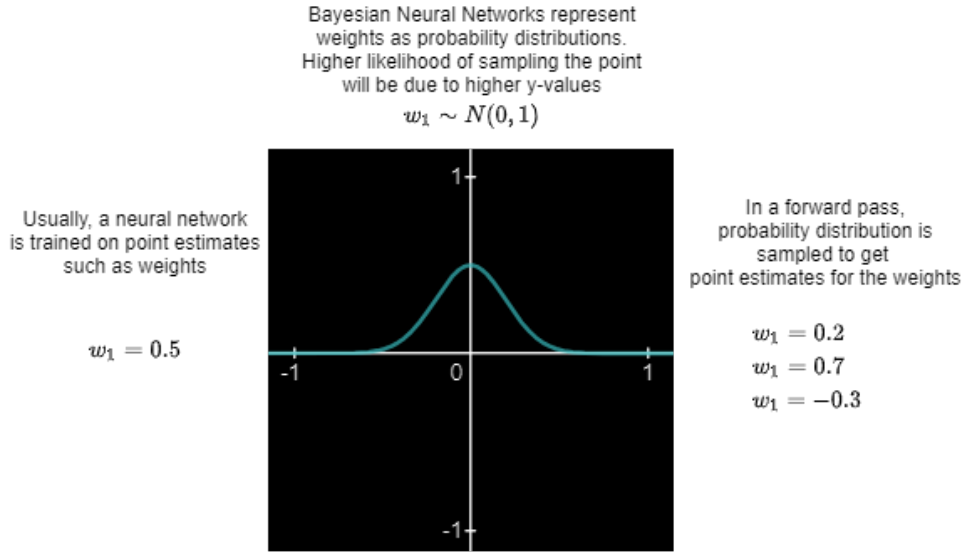

Figure 2.13: Representation of weights in a neural network vs bayesian neural network

method to approximate the posterior distribution [17]. Variational learning can be used to obtain parameters θ of the variational distribution, denoted by $p(w|\theta)$, by minimizing the variational free energy cost function F, which is also known as the expected lower bound (ELBO). The variational free energy is the sum of the Kullback-Leibler (KL) divergence, which is a measure of the distance between the prior distribution, variational distribution and the negative log-likelihood, which is a measure of the goodness-of-fit [26].

$$\mathcal{F}(D,\theta) = KL[(q(w|\theta)||p(w))] - E_{q(w|\theta)}[log(p(D|w))] \tag{2.22}$$

Blundell et al. [6] elucidates the variational free energy loss function as an optimisation problem between the simplicity prior (KL term) and the complexity of the dataset (NLL term). Blundell et al [6] use a gaussian prior which yields L2 regularisation or weight decay. The authors propose various approximations

1. Unbiased Monte Carlo gradients where the cost is approximated such as

$$\mathcal{F}(D,\theta) \approx \sum_{1}^{n} log(q(w^i|\theta)) - log(p(w)) - log(p(D|w^i)) \tag{2.23}$$

   where w^i are i^{th} Monte Carlo samples drawn from the variational posterior of the marginal probability density $q(w|\theta)$

2. Gaussian Variational Posterior can be considered where the weights w are sampled from a unit Gaussian distribution with mean μ and standard deviation σ which can be parameterized such as

$$\sigma = log(1 + \exp\rho) \tag{2.24}$$

   this yields the variational posterior parameters as (μ, ρ)

Limitations are such that exact values of the posterior parameters are not retrieved but with a good approximation can be produced with smaller computational complexity. There is a trade-off between satisfying the complexity of the dataset D and satisfying the simplicity prior $P(w)$. We get closer to the minima of the cost function for as long as we train the neural network. Therefore, the gap between variational distribution and the true posterior distribution reduces. For practical purposes, the KL term serves the purpose of regularization

on the output of the network and thus, avoiding the distribution that is learnt from overfitting at the expense of a lower Negative Log Likelihood term in the training dataset.

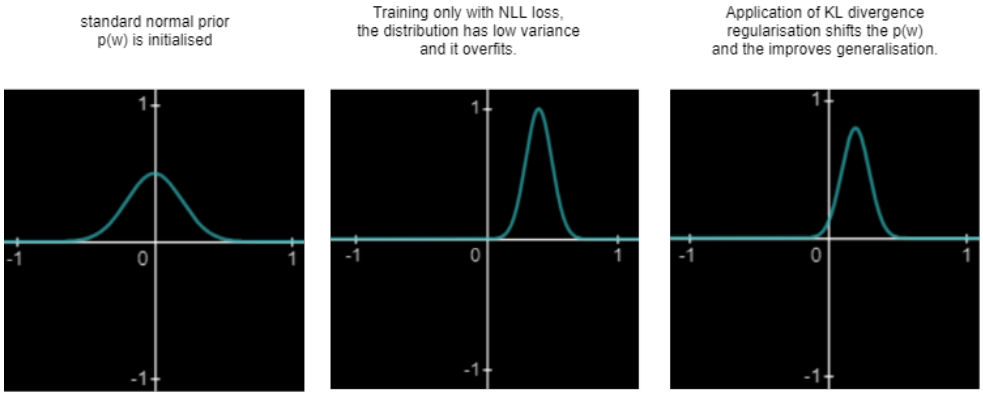

Figure 2.14: Illustration of training a single weight under Variational free energy loss function

2.8.2.1 Evaluating bayesian models

Patch Accuracy vs Patch Uncertainty (PAvPU) was put forward by Mukhuti et al. [34]. The assumption is that when a model is confident about its prediction, it must be accurate on the same. It is also implied that if a model is not accurate on an output, it should be uncertain about the same output.

Given these assumptions, the following conditional probabilities are defined in [34]

1. p(accurate | certain): "The probability that the model is accurate on the output given that it is confident on its output."
2. p(uncertain | inaccurate): "The probability that the model is uncertain about its output given that it has made a mistake in its prediction (i.e, is inaccurate)."

$$p(accurate|certain) = \frac{n_{ac}}{(n_{ac} + n_{ic})} \tag{2.25}$$

$$p(uncertain|inaccurate) = \frac{n_{iu}}{(n_{ic} + n_{iu})} \tag{2.26}$$

Patch Accuracy vs Patch Uncertainty (PAvPU)

$$\text{PAvPU} = \frac{(n_{ac} + n_{iu})}{(n_{ac} + n_{au} + n_{ic} + n_{iu})} \tag{2.27}$$

where n denotes total number of patches, n_{ac} is the number of accurate and certain patches, and n_{iu} is the number of inaccurate and uncertain patches.

$$\text{PAvPU score} = \frac{1}{n} \times (n_{ac} + n_{iu}) \tag{2.28}$$

2.9 Boxplots

A boxplot is a useful visualisation tool that can help summarise data.

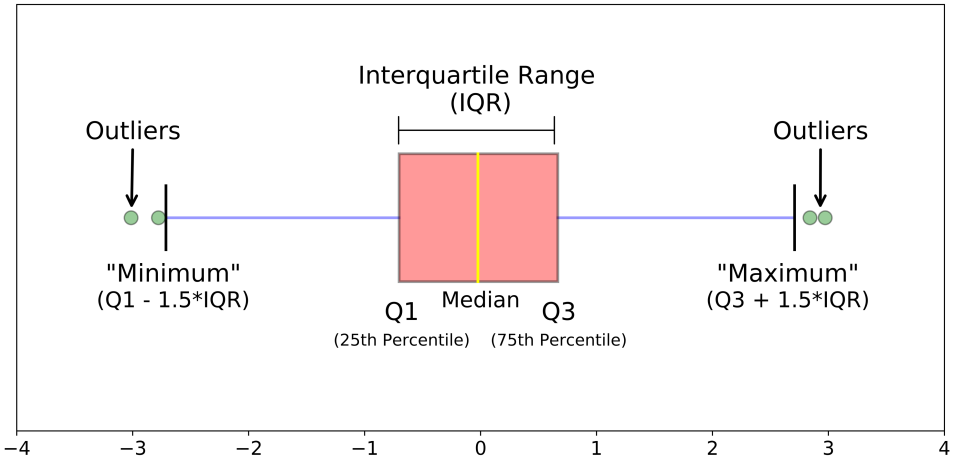

Figure 2.15: Different parts of a boxplot as seen in [22]

3 Method

3.1 Introduction

This project aims to develop and evaluate a deep-learning technique to segment MR images that include the heart and large vessels in four-dimensional image data [30]. The technique will be developed and evaluated on a set of 200 already acquired 4D flow MRI datasets, that have been segmented using a validated multi-atlas based registration technique [7]. A previously evaluated method based on atlas segmentation has been used to generate labels for the aorta, pulmonary artery, and the four cardiac chambers at each timeframe for all datasets. Preliminary evaluation of the results will be done using commonly employed metrics for segmentation evaluation, such as Dice coefficient; and also metrics based on quantitative physiological measures, such as stroke volume of the cardiac chambers and flow volume through the great vessels.

Using 4D flow Magnetic Resonance Imaging (MRI), the velocity of blood can be measured in the whole thorax in a few minutes. 4D flow MRI is used to diagnose and assess blood flow dynamics in complex diseases. 4D phase-contrast MRI datasets corresponding to 200 subjects have already been acquired and are available and ethically approved to use for this project. Each dataset is composed of a volume of size 112x112x44x40, which can be sliced into 2D or 3D volumes of different shapes for training and validation purposes.

Reasoning for the use of dataset and method:

- This multi-atlas based technique resulted in high quality segmentations, with the disadvantage of long computation times typically required by three-dimensional registration techniques.
- In contrast, a deep learning based method would require a large amount of resources and time during training, but would generate results much faster once the tool is assimilated into the clinical process.
- Additionally, a convolutional neural network such as a U-Net trained using a sufficiently diverse set of images should be able to handle more extreme morphological differences between subjects when compared to atlas-based segmentation methods.

- U-Net's have been successfully used in biomedical segmentation problems. It is found from literature review that 3D U-Net performs marginally better than the original 2D Net when the 3D data is available in case of cardiac segmentation.

3.1.1 Problems with previous studies

- Most cardiac segmentation using deep learning is done for highly specific problem sets such as the segmentation of the Left Ventricule, one of the atria or the aorta and other specific parts.
- Due to memory constraints of the GPU's these methods have either downsampled the input 3D data which leads to loss of resolution or only predict sub volumes of the data.
- Other methods that first successfully extract region of interests (ROI) using a localisation network and then apply that to segmentation U-Net. This method makes use of 2 stages of 3D U-Nets.

3.2 Methodology

The first published implementation of the U-Net is in Caffe [14]. The pytorch implementation of the 3D U-Net is used from the following libraries and code repositories: `torchio` [40], The GPU available to train the network is a Nvidia Quadro P6000 with 24GB of dedicated GPU memory. The cardiovascular 4D flow MRI data is available in .nii format (nifti file) for approximately 200 patients. The software ITKsnap is used to perform preliminary visual checks on the data, ITK stands for Insight Segmentation and Registration Toolkit. The data is read in python with a split of 80 percent for training and 10 percent each for validation and testing. Since 3D data is available for each timeframe, we read in 9200 sets of whole 3D volumes, referred to as subjects. Each subject has a 3D MRI data and a 3D segmentation mask data. These 3D volumes have the spatial dimensions of 112X112X44. The 3D data image intensity has to be normalised with either Histogram Standardisation or Z-Normalisation. In image processing, normalisation is a process that changes the range of pixel intensity values. Histogram Normalisation involves calculating a non-linear mapping from the cumulative histogram of the image intensity. It is seen that it can enhance a lot of the fine detail.

Performance in deep learning generally improves with the amount and variability in the data available. Data augmentation is a technique by which new training data can be created using the existing training data itself. This is done by the application of techniques that are domain specific to create variety and abundance within the training data. Data transformation is a type of data augmentation in which transformed versions of the 3D images, in our case, are of the same class as the original 3D image. [20] The purpose of such an augmentation exercise is to increase the set of plausible examples in the training set, for instance, random motion of a 3D Volume may make sense as it may simulate random motion of the patient on the trolley within the scanning bore of the MRI scanner. Multiple data transformation techniques have been computed, visualised individually and combined into different transformation pipelines for comparison of their performances on the validation set.

Data transforms and augmentation are often overlooked in research papers, these are important as they affect the performance of the network and can help in reproducing the study. The CropOrPad function is used but currently only zero padding is being performed. Cropping can be done to reduce the computational burden. Transforms such as Random Flip, Random blur, Random Noise, Random Bias field, Random Ghosting, Random Motion are. These transforms with a low probability parameter of 0.15 to 0.20 can be used to improve the performance of the network.

After the training and validation loaders are set up with batch size of 2 for both training and validation dataset's due to GPU utilisation limitations. The target is the segmentation

mask's which are now being extracted by their labels from the tensors. We have 6 main labels for the classes: Right Ventricule, Left Ventricule, Right Atrium, Left Atrium, Pulmonary artery and the Aorta. Dice score is being used to evaluate the segmentation, it is the ratio of two times the true positives to the sum of false negatives, false positives , 2 times the true positives and a small value epsilon to keep the denominator non-zero. This computation is between each 3D image input against its targets/segmentation mask and can be computed for each part of interest during segmentation as well by splitting the image once into background (Not the heart) and foreground (whole heart) during binary segmentation; background (Not the heart) and multiple foregrounds (Right Ventricule, Left Ventricule, Right Atrium, Left Atrium, Pulmonary artery and the Aorta) during Multi-class segmentation.

The optimisation algorithm in use is AdamW, is a method for stochastic optimisation with decoupled weight decay regularisation [24] [29].

Two different segmentation problems are addressed:

- Binary segmentation has been achieved where the background and foreground are segmented, the whole heart is segmented as one class. The parameters in use in the 3D U-Net for Binary Segmentation, with training and validation batch sized of 8 and 16 respectively. Please refer the appendix for details of parameters
- Multi-class segmentation has been achieved, each of the 6 target parts of the heart. The network is well suited for semantic segmentation and data augmentation with pipeline 2 has been performed. This achieves a dice loss of 8.6 percent on test set, after 25 epochs which takes 30 hours to train. The parameters in use in the 3D U-Net for Multi-Class Segmentation please refer the appendix.

3.2.1 Data preprocessing

Implementation of a 3D U-Net for over 9200 examples is challenging. First, the orientation of the data is standardised.

3.2.1.1 Preprocessing (orientation)

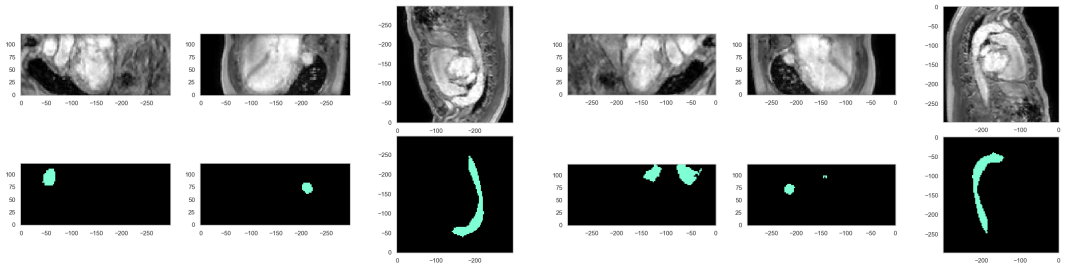

Figure 3.1: **Standardizing orientation: The input image is standardised in orientation to 'RAS' or the right handed coordinate system. In the sample above, The first three image pairs denote one slice in the 'LPS' orientation which are converted to 'RAS' orientation in the next three image pairs. The top images denote intensity and below are segmentation masks. The 3rd and 6th image pairs are in the 'xy' coordinate system.** .

Old orientation: ('L', 'P', 'S') New orientation: ('R', 'A', 'S') ScalarImage(shape: (1, 112, 112, 44); spacing: (2.68, 2.68, 2.80); orientation: RAS+; memory: 2.1 MiB; type: intensity) LabelMap(shape: (1, 112, 112, 44); spacing: (2.68, 2.68, 2.80); orientation: RAS+; memory: 2.1 MiB; type: label)

3.2.1.2 Preprocessing (normalisation)

Histogram normalisation

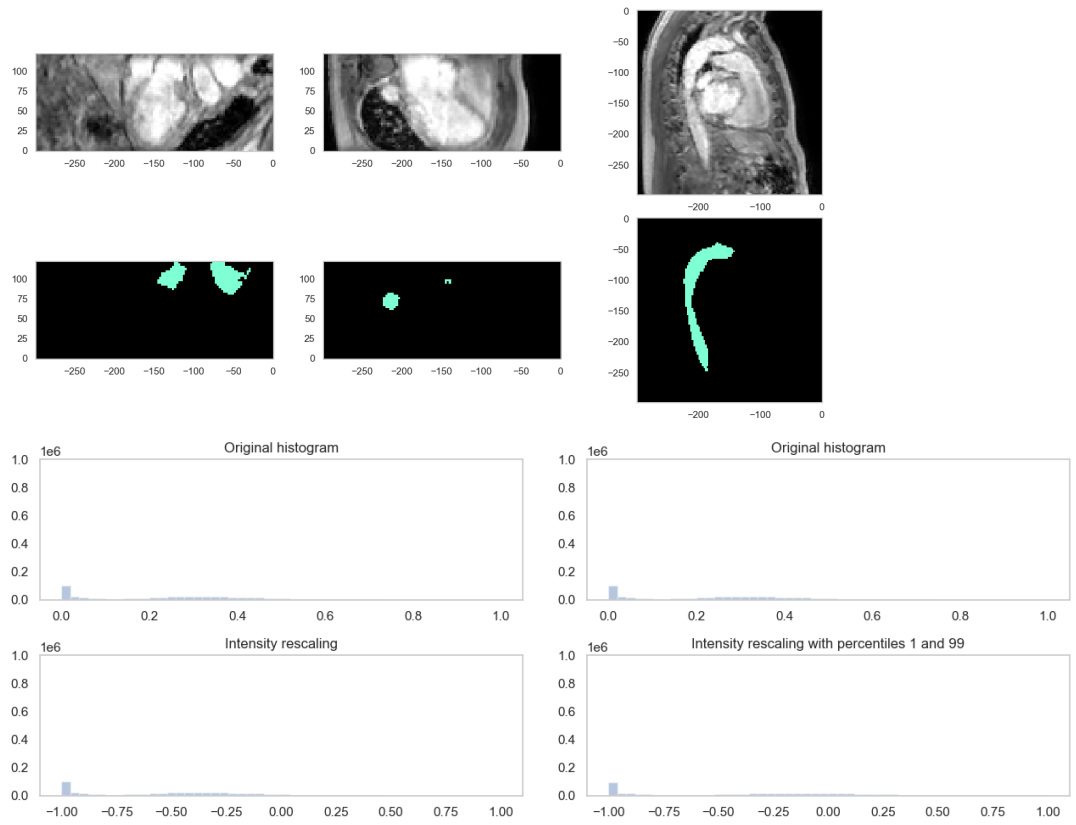

Figure 3.2: **Histogram Standardisation: A non-linear mapping from the cumulative histogram of the intensity. Intensities are rescaled so that it lies within -1 and 1 in the cumulative historgrams in the bottom. The mode is visible over x=0 and x=-1.00, on the original histogram and cumulative historgram respectively corresponds to the background and the 2nd highest mode over x=0.3 and x=-0.5 corresponds to the foreground. Intensity rescaled on the right shows that the foreground is rescaled to x=0 in the cumulative histogram in the bottom right**.

Z-normalisation

Z-Normalisation, also known as Z-score normalisation to Mean=0 and standard deviation=1. All elements of the input vector are transformed into the output vector with mean that is approx. 0 while the standard deviation is in a range close to 1.

$$x'_{\mathrm{i}} = (x_{\mathrm{i}} - \mu)/\sigma \tag{3.1}$$

In this project, Z-Normalisation is used during preprocessing for deep learning for the purpose of normalisation, due to the simplicity in its implementation. The data is generally in the same orientation of 'LPS+' but it has to be same for all the input data. Hence, data is reordered to be closest to canonical (RAS+) orientation. On application of Z-Normalisation, the data points are forced towards zero mean and unit variance.

The second mode in the distribution (top left) in the figure 3.4, corresponding to the foreground, is far from zero mean because the background is significant in the computation of the mean. Any further computations can now be above a threshold value, example: values above the mean to segment the foreground.

The second highest mode is closer to zero after z-normalisation, which correspond to the foreground voxels.

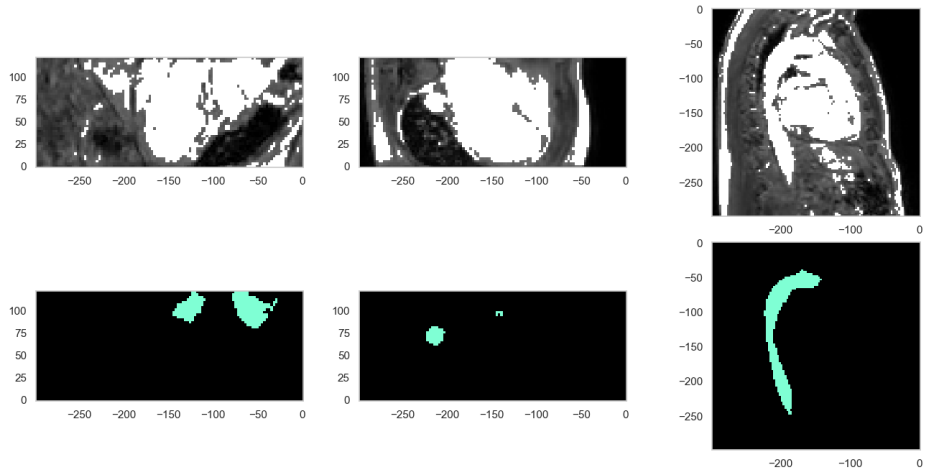

Figure 3.3: **Z-Normalisation, with threshold value: The three image pairs show z-normalised intensity and corresponding segmentation masks of a 3D slice, the parts of the heart i.e foreground and background are visibly distinct.**.

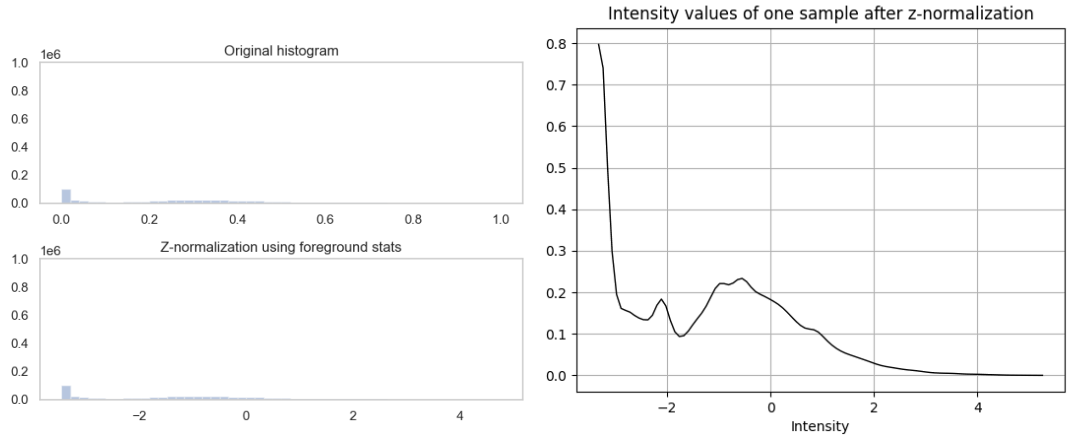

Figure 3.4: **Z-Normalisation, with threshold value: The 2nd highest mode is now close to 0 intensity after z normalisation** .

3.2.2 Data Augmentation

The following data transformation techniques are explored for use in the segmentation task from the research paper and torchio library provided in [40]. All data transformation is done with a low probability p such that $0.15 < p < 0.2$, where p is the probability of application of the said transformation method on the training data.

3.2.2.1 Random affine

In euclidean geometry, affinity or an affine transformation is a geometric transformation that on application can preserve lines and parallelism but not distances and angles [1]. Different positions and size of the patient within the bore of the MRI scanner can be simulated using a

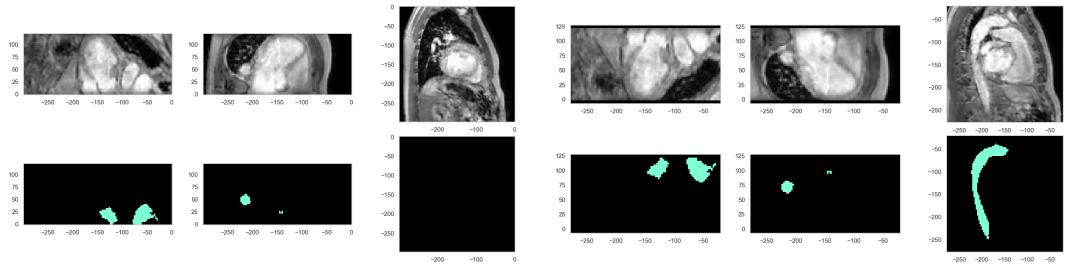

Figure 3.5: **Flip , Cropping and Padding (LHS: Before), (RHS: After)**.

Random Affine transformation with probability values kept low at 1/5. 2D slice of the image along with a grid is used for visualisation of the transform. Random Affine transformation rotates the image with the grid slightly anticlockwise, and with zoom functionality enabled focusing on the heart [39].

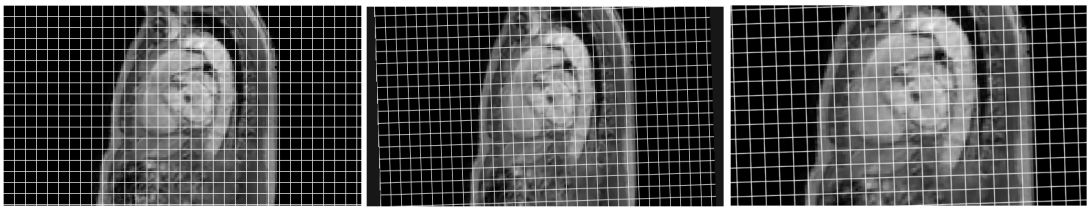

Figure 3.6: **Random Affine: (1: preprocessed intensity with grid), (2: transformation applied), (3: 130 percent zoom).** Some images are rotated with a small angle.

3.2.2.2 Elastic Deformation

Random application of elastic deformation on the intensity can simulate anatomical variations that may exist among patients, the application of a non-linear deformation using the Random Elastic Deformation from [40]. The parameters for this function includes the total grid control points which ascertains the smoothness of deformation. The effects of the application of a low number of control points and large displacement is shown in the figure 3.7

Figure 3.7: **Random Elastic Deformation: (1: preprocessed intensity with grid), (2: more control points, less displacement), (3: less control points, more displacement).** Expansion and contraction of some areas of the image.

Intensity transforms only modify the scalar images of intensities while the segmentation label maps remain as before.

3.2.2.3 Random Blur

Blur is a common distortion that is found in biomedical MRI, it degrades MR images during the acquisition process itself. Blur is different from Noise and Bias fields because it can also be introduced after the acquisition stage, such as during data preprocessing or due to other pathological conditions of the patient [36].

The Random Blur function is used to smoothen or blur the input images. The standard deviations of the Gaussian kernels are computed independantly for each axis. Random blur is explored for application in the segmentation task, it is visualised with the example slice of the 3D image in figure 3.8

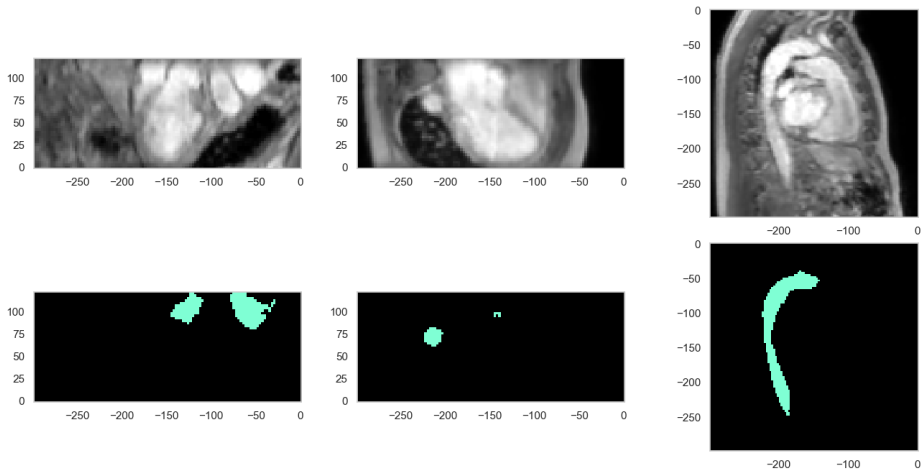

Figure 3.8: **Random Blur: (yz,zx,xy) views of a transformed image slice and corresponding segmentation map.** Blurring and smoothening of the intensities is visible in the above example.

3.2.2.4 Random Noise

Noise in MR images usually belongs to the Rician distribution [36].Since only Gaussian noise is simulated using the Random Noise function, It is usually applied after Z-Normalisation as the zero mean and unit standard deviation of the input scalar is known and the noise mean and variance can be passed as parameters to the Random Noise function provided in *torchio* [40]

3.2.2.5 Random Bias Field

Inhomogeneities of the magnetic field produced by the MR image scanner creates low-frequency distortions in the image intensity which are generally corrected using algorithms such as N4ITK [48] To simulate this artifact, we can use RandomBiasField.

For the example in figure 3.10, the preprocessed data is used an a slice of the 3D image is visualised with added random bias.

K -space transforms: The coils of the MRI scanner produce a signal known as the k-space, it is read in for preprocesssing by the scanners internal software which computes the inverse Fourier transform of the k-space to generate the MR image. When the k-space is altered an artifact is created in the image accidentally, this can be simulated using data transformation such as the following

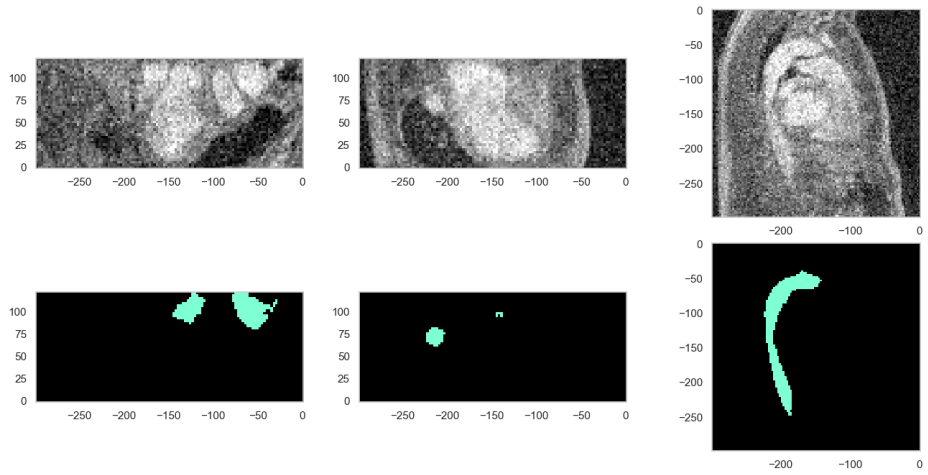

Figure 3.9: **Random Noise: (yz,zx,xy) views of a transformed image slice and corresponding segmentation map.**Random noise adds granularity and is visible in the example above.

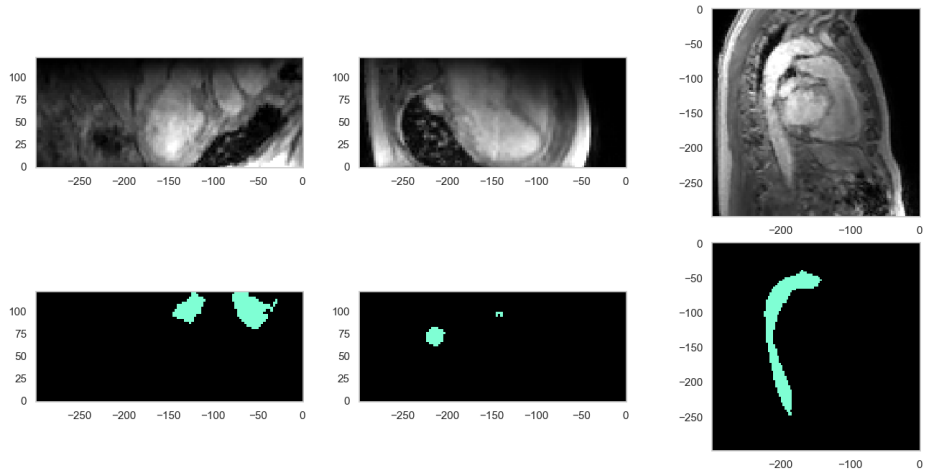

Figure 3.10: **Random Bias Field: (yz,zx,xy) views of a transformed image slice and corresponding segmentation map.**The bias field is visible as darkening of some foreground parts.

3.2.2.6 Random Spike

Spikes may occur in the signal in the k-space due to various reasons. For example, the presence of a high energy component producing a sound of 440 Hz in the audio spectrum registers a tone of that frequency in the time domain. Similar to this, spikes manifest in the k-space that are visible as stripes in the image space.

Random Spike is not chosen for application in our case as spiking activity is not observed or reported in the training data. [42] [7].

3.2.2.7 Random Ghosting

The motion of a patient inside the scanning bore of the MR scanner can produce ghosting artifacts i.e superimposition of adjacent patches of images with neighboring patches in the acquisition of an MRI. This can be simulated by removing each n^{th} plane from the k-space.

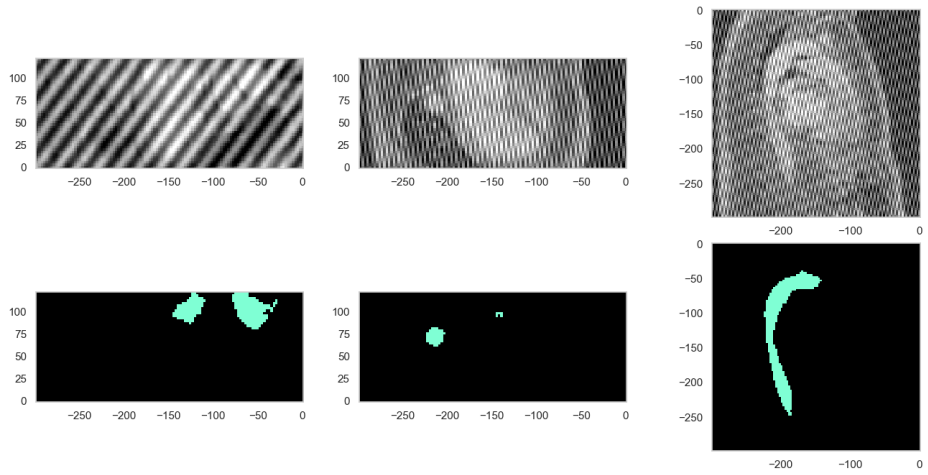

Figure 3.11: **Random Spike: (yz,zx,xy) views of a transformed image slice and corresponding segmentation map.**The random spike from an external source may cause distortion of the MR.

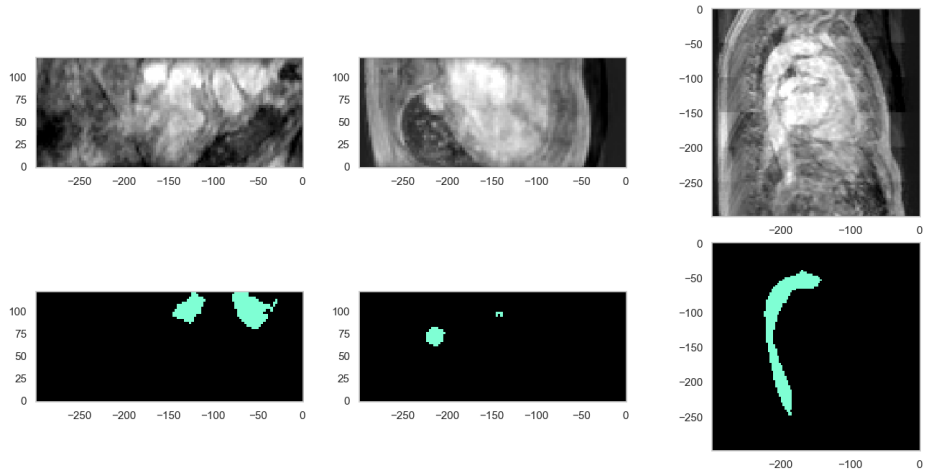

Figure 3.12: **Random Ghosting: (yz,zx,xy) views of a transformed image slice and corresponding segmentation map.**Patients motion artifacts can be simulated with random ghosting transform.

3.2.2.8 Random Motion

The motion of the patient that is voluntary can produce large motion artifacts in the k-space and inturn, the MRI. an implementation of this provided in Shaw et al [45], where the motion artifact is simulated by filling the k-space with random transformed images of the original and computing the inverse Fourier transform of the k-space to produce the MRI. The computation of the direct and inverse Fourier transform is computationally cumbersome, and hence, nearest neighbor interpolation is applied to resample quicker. Small number of random motion transforms can be applied to reduce the runtime to simulate the little motion that may occur involuntarily by large number of patients.

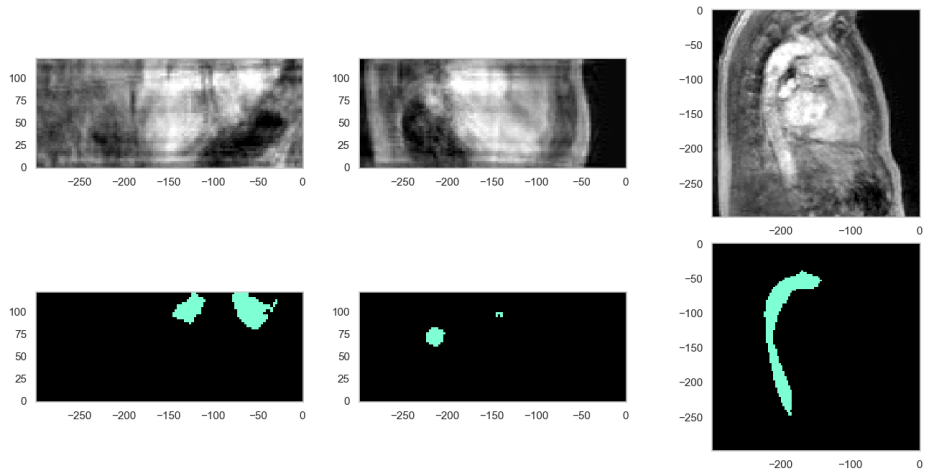

Figure 3.13: **Random Motion: (yz,zx,xy) views of a transformed image slice and corresponding segmentation map.** Larger motion within the MRI scanner can produce motion artifacts that are simulated.

3.3 Implementation

This section deals with the implementation of both the 3D U-Net and the Bayesian 3D U-Net in python using pytorch and tensorflow keras respectively.

3.3.1 Loading Data

The the data is loaded using pytorch, elucidated in torchio [1]: The nifti (.nii) datasets are read into scalar and label maps for the intensities and class labels/segmentation masks. The 4D flow data is discretized into 3D Volumes of size (112 X 112 X 44) for each time step. We now have over 9200 3D volumes

3.3.2 Applying transforms

We explore many different data augmentation and transformation techniques well suited for 4D flow data.

For the binary segmentation 3D U-Net the following transforms were used:

3.3.2.1 Binary segmentation transformation pipeline

- Z-normalisation,
- ToCanonical(), (Orientation)
- CropOrPad((112, 112, 48),zero padding
- RandomMotion(),
- RandomBiasField(),
- RandomNoise(),
- RandomFlip(axes=(0,)),
- RandomAffine(),

[1] Based on `https://github.com/fepegar/torchio`

- RandomElasticDeformation()

For the Multi-Class segmentation 3D U-Net the following two transformation pipelines were used:

3.3.2.2 Multi-Class segmentation transformation Pipeline 1:

Pipeline 1:

- Z-normalisation,
- ToCanonical(), (Orientation)
- CropOrPad((112, 112, 48),zero padding
- RandomMotion(),
- RandomBiasField(),
- RandomFlip(axes=(0,))

3.3.2.3 Multi-Class segmentation transformation Pipeline 2:

- Z-normalisation,
- ToCanonical(), (Orientation)
- CropOrPad((112, 112, 48),zero padding
- RandomMotion(),
- RandomBiasField(),
- RandomBlur()
- RandomFlip(axes=(0,)),
- RandomGhosting(),
- RandomAffine(),
- RandomElasticDeformation()

3.3.3 Implementation of the 3D Unet

Two 3D U-nets are implemented. One with Binary Segmentation to identify the whole heart with all the target classes as one i.e foreground vs the background which includes the rest of the thorax, other parts of the body and outside. The output of the binary segmentation can help identify the necessary volume crop size around the heart which can be used in the subsequent neural network for multi-class segmentation.

The other 3D U-net is for the purpose of multi-class segmentation. The six target parts of the heart or the class labels are : Right Ventricle, Left Ventricle, Left Atrium, Right Atrium, Aorta and the Pulmonary artery.

3.3.4 Training, Validation and Testing

The subjects from the whole dataset are split into training, validation and testing sets with 80:10:10 respectively.

At first, the dice score and dice loss is calculated per batch during training and analysed over training and validation by batch. [2]

Smoothing is applied to determine the trend during training due to the irregularity since the loss by sample batch has high variance.

3.4 Evaluation

One method of evaluation is the analysis of the Dice loss and Dice similarity coefficient which is similar to the f1-score is a commonly used metric to train networks over 3D image data.

This dice loss for each target segmentation class is also computed and analysed. This gives a fair estimation of the model performance on which classes it is not performing well and also on which examples the model is not performing well.

The other method is to use the principles of variational inference and produce uncertainty metrics that can be used to evaluate the segmentation predictions obtained from a network with similar but bayesian architecture.

3.4.1 Variational inference

Variational inference via Bayesian learning has more statistical validity than approximate Bayesian inference via Monte Carlo dropout layers [37]. However, this arrives with its own computational difficulties.

3.4.1.1 Local Reparameterization trick

BCNN's seem incompatible with backpropagation since a random variable that is sampled from a distribution will not have a gradient. However, It was shown by Kingma et al that one can reparametrize the random variable and arrive at a deterministic variable for computational purposes [25]. For example, Let weight w be sampled from a normal distribution with mean equal to μ and variance σ^2. This can be reparameterized to be $w = \mu + \sigma\epsilon$ where ϵ is an additional noise variable which is sampled from a standard normal distribution. In this form the sampled variable can be utilised in the backpropagation algorithm. To scale this computationally, more complex calculations are required. This method is also known as the local reparameterization trick [26], A figure to illustrate this can be found in 3.14 [3].

3.4.1.2 Group normalisation

3D volumes are large and hence, the batch size used during training is limited by the availability of GPU memory. This results in the selection of a small batch size for which batch normalisation cannot be used. Therefore, A technique called group normalisation is used which normalizes channels in groups and provides accurate results independent of the batch size [53]. Due to the problem of vanishing/exploding gradients, group normalisation is significant for the convergence of the model. On experimentation with the number of groups used in group normalisation layers, it is found that the Bayesian 3D U-Net converges reliably with 4 groups. The figure 3.15 from [53].

[2] Dice Loss = 1 - Dice Score

[3] https://stats.stackexchange.com/questions/199605/how-does-the-reparameterization-trick-for-vaes-work-and-why-is-it-important

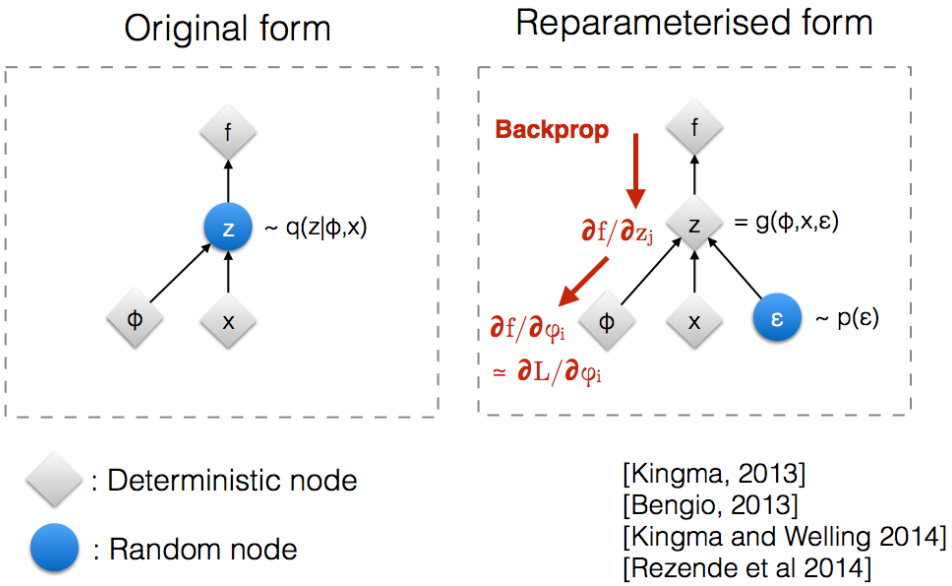

Figure 3.14: **Illustration of local reparameterization trick:** The random node z is approximated by a function of the true posterior $q(z\psi, x)$.Since, backpropogation can not flow through such a random node, this node is changed to a deterministic node that takes a random input with parameters ϵ, this allows for reparameterisation to independent parameters of the activation function at that node and the quantities at each stage can be computed as partial derivatives

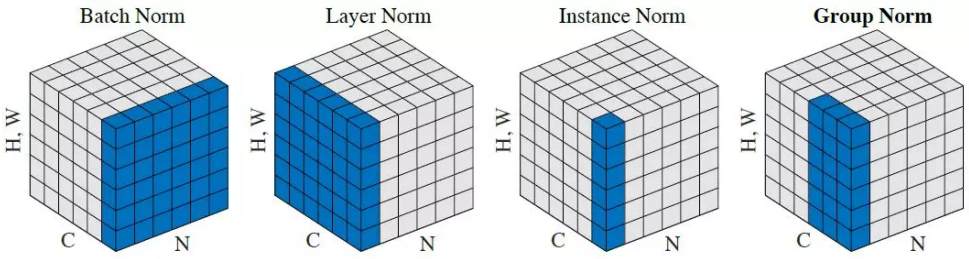

Figure 3.15: **Illustration of Group Normalisation.** For a small batch size where batch normalisation is not possible, channels can be normalised in groups. The figure 3.16 shows where group normalisation is applied within the Bayesian 3D U-Net architecture

3.4.1.3 BCNN architecture

The Bayesian 3D U-Net architecture is based on the encoder-decoder configuration also seen in the 3D U-Net [9]. In the BCNN architecture, the encoder side (contracting path) of the network enables the compression of the input to a latent space the decoder side (expanding path) enables the decompression of the latent space to a segmentation map. The network architecture is illustrated in the figure 3.16

The encoder side of the Bayesian neural network uses usual 3D convolutions to transfer maximum amount of information from the original volume to the latent space while the decoder side of the network uses 3-dimensional Bayesian convolutional layers. Each layer is initialized with a standard normal prior and it approximates the distribution during forward passes. Note the arrows from group normalisation to concatenation, these assist in forwarding features all through the network. This implementation of the BCNN is based on the Bayesian Layers library [47] by TensorFlow Probability, which enables the computation

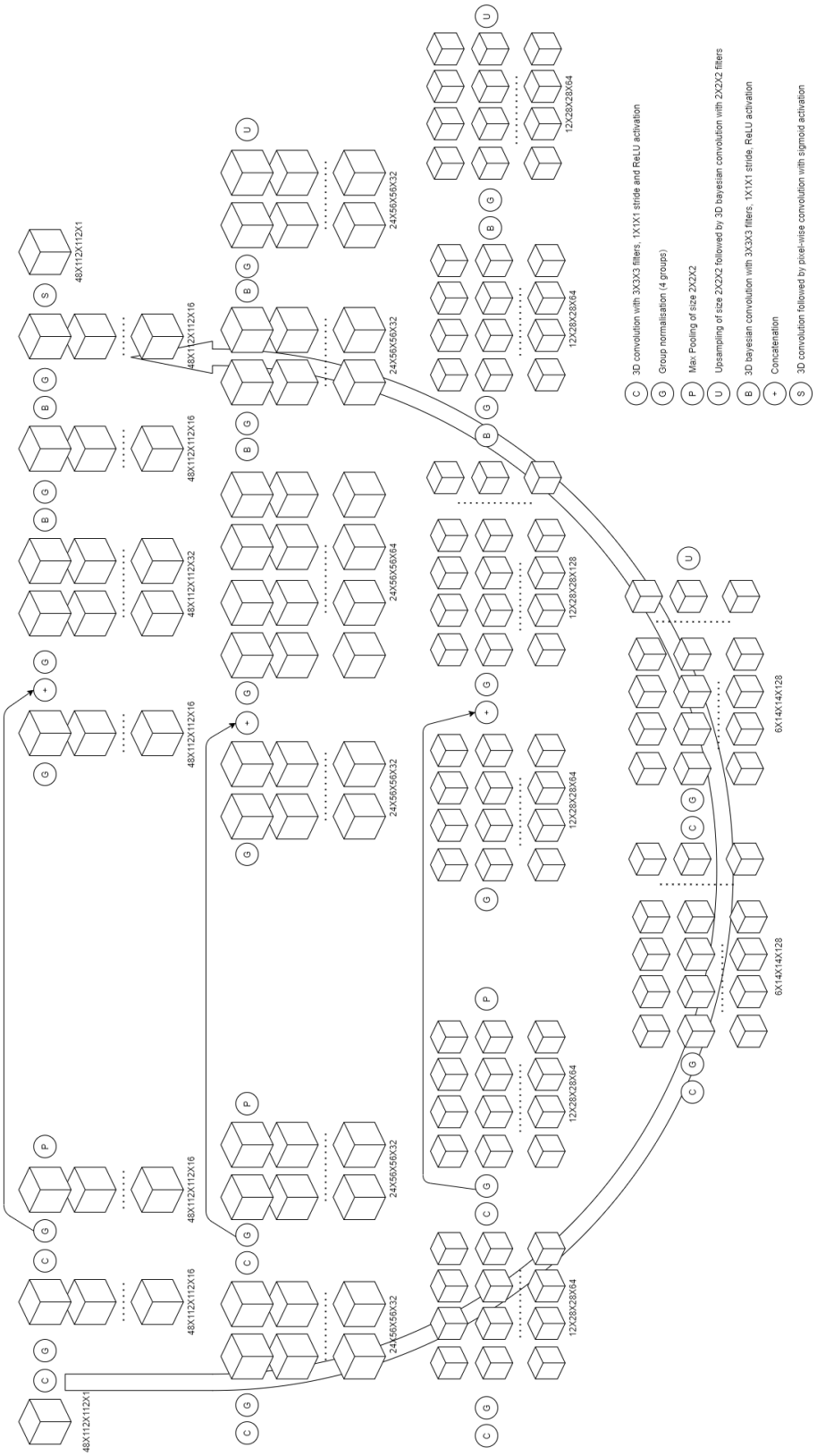

Figure 3.16: **Schematic of the bayesian 3D U-Net implemented.** The encoder part where in each level two convolutions and group normalisation steps followed by a max pooling, the decoder part where two bayesian convolution and group normalisations followed by an upsampling step to finally reach the orignal dimensions.

of the variational free energy loss by keeping a record of the losses that signify KL divergence of the posterior distribution of the layer with respect to the prior distribution [4] [26].

3.4.1.4 Training Considerations

A Bayesian CNN is difficult to implement since the code libraries are not designed for bayesian implementations. It is not possible to save weights and architecture of a trained model with the combination of tensorflow probability layers and usual keras layers. Instead, Only the weights are saved and loaded into an initialised architecture. It also gets more difficult to work with using multi-GPU systems which are usually necessary when working with 3D data. The weights have to be saved as if a single GPU is in use and it is loaded as multi GPU weights in the last but one layer of the network.

Management of 3D data can also be difficult. MRI scans can be in the range of millions of voxels, inferring on all at once can be computationally difficult. A solution to this is 'chunking' where large volumes are separated into overlapping chunks and fed into the Bayesian 3D U-Net. The network architecture and related processes is referred from [4]. The neural network trains and predicts on these chunks which are reconstructed into the complete volume. In the chunking process, a sliding rectangular prism 'window' is passed along the original volume with a preset ratio of overlap which is called 'step size'. The output of this algorithm is a large 5 dimensional numpy array which has all the chunks . This chuking process is referred from [5].

3.4.1.5 Mapping Uncertainty

We choose the Monte Carlo sample size of 48. Sigmoids or the output from the final layer are used to compute 80percent and 20percent credible intervals. The difference between these upper and lower bounds provide an uncertainty measure that is mapped and seen in the figures 4.13.

The PAvPU 2.8.2.1 is evaluated on the inference sample and it gives a measure of confidence in the bayesian network. It is also a measure of uncertainty and can help differentiate the quality of predictions and its inherent uncertainties.

4 Results

The results from both the binary and multi-class segmentation using 3D U-Net and uncertainty estimation using the Bayesian 3D U-Net are presented here.

4.1 Implementation

A table of the parameters used during different training runs of the 3D U-Net are given in the table 4.1.

Table of the hyperparameters used during different training runs								
Training run	Output Classes	Number of encoding blocks	Output channels in the first layer	No. of epochs	Up-sampling type	Activation function	Dice Score (Test)	Dice Score (Validation)
No Data Augmentation	2	3	16	5	Linear	ReLU	0.922	0.867
Data Augmentation Pipeline 1	2	3	16	10	Linear	ReLU	0.93	0.871
No Data Augmentation	7	4	8	10	Trilinear	PReLU	0.88	0.825
Data Augmentation Pipeline 1	7	4	16	10	Trilinear	PReLU	0.889	0.830
Data Augmentation Pipeline 2	7	4	16	10	Trilinear	PReLU	0.896	0.842
Data Augmentation Pipeline 2	7	4	16	25	Trilinear	PReLU	0.914	0.858

Table 4.1: **Different Hyperparameter settings used during training runs of the U-Net**

- Output Classes: Binary segmentation has 2 output classes whereas Multi-Class segmentation has 7 output classes.
- Number of encoding blocks: The number of *levels* in the encoder side of the U-Net i.e number of convolution and max-pooling pairs concatenated. Same number of levels exist on the decoder side as the U-Net is symmetric.
- Number of epochs: the number of times the U-Net passes of the training dataset.
- Upsampling type: Please refer section 2.4
- Activation funciton: Please refer section 2.5
- Dice Similarity coefficient on the test set: It is observed that the Dice score increases with the application of data augmentation, both in binary and multi-class segmentation.

parameters of the U-Net.

4.1.1 Binary Segmentation

This output can be used in a 2 stage U-Net architecture where the required cardiovascular area to be segmented can be identified and cropped to be used in the 2nd stage U-Net for Multi-Class segmentation.

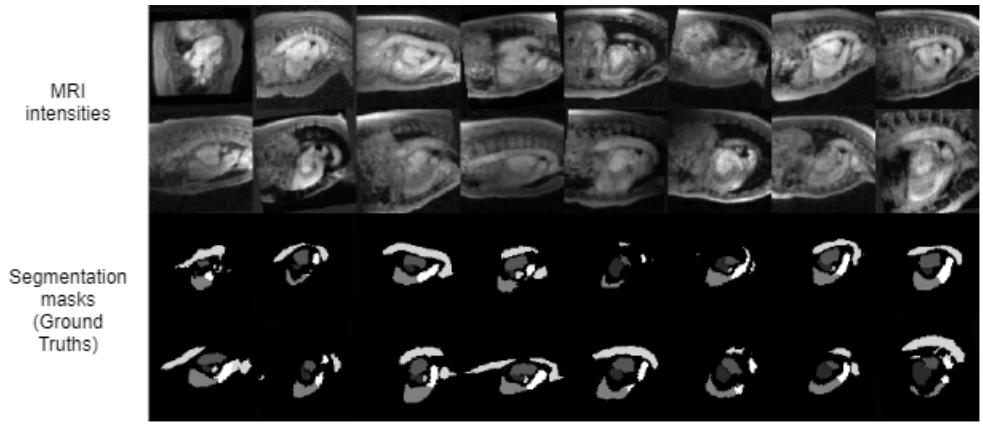

Figure 4.1: **Validation batch of mri intensities (top) and corresponding segmentation masks pairs (bottom) (Ground truth) .** The figure shows the validation data and segmentation masks for a batch of 16 sample slices.

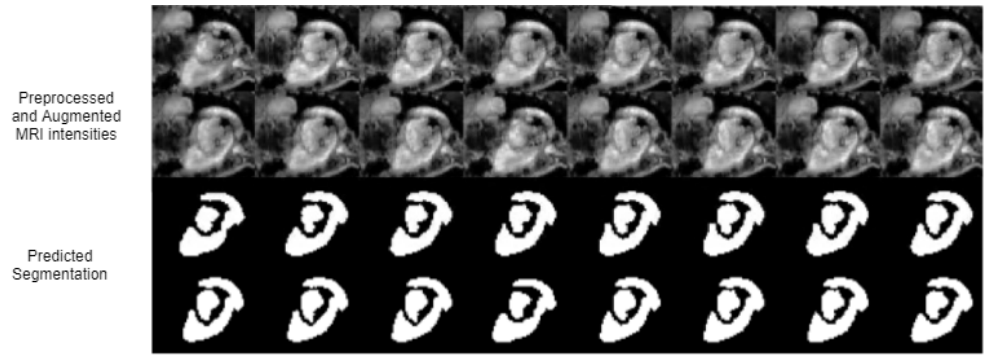

Figure 4.2: **Binary segmentation output at epoch 5,** $DSC = 0.93$ **.** The figure shows the preprocessed and augmented MRI intensities (top) and the segmentation prediction (bottom) of the trained binary segmentation 3D U-Net on the validation data as shown in figure 4.1

The top half consisting of 16 images are preprocessed MRI intensities, 3.3.2.1

It is observed that the trained network can segment the cardiovascular system from the background well. Similar Dice scores are also achieved for the background even in the Multi-class segmentation as seen in table 4.2, due to this the use of binary segmentation in a two-step, two U-Net process as discussed in 3 is skipped. However, this leads to long training duration for converging to a solution in the Multi-class semantic segmentation.

4.1.2 Multi-Class Segmentation

The 6 class label's, one for each chamber of the heart, aorta and the pulmonary artery are segmented.

- Left Ventricle (Red),
- Right Ventricle (Blue),
- Left Atria (Green),
- Right Atria (Yellow),
- Aorta (Cyan),
- Pulmonary artery (Pink),

legend for 3D segmentation output of the Multi-class network

4.2 Evaluation

The training batch losses during no-augmentation, augmentation pipeline 1 and augmentation pipeline 2 (three different training runs) are compared in the figure 4.6. It is observed that, when no data augmentation is performed on the training data, the Dice loss is reduces much quicker than in the case of data augmentation.

The validation batch losses of the three data augmentation pipelines are compared in the figure 4.7. It is observed that, the validation data contains some samples where the network does not perform well and this is seen as the spikes in the figure 4.7. The individual Dice losses for each cardiovascular target segmentation can be seen in the figure 4.8, 120 validation samples out of 400 samples are segmented poorly. The data augmentation pipeline 2 (seen as red) in the figure 4.7 shows that the data augmentation that adds computational burden by increasing the variability in the training data, does actually help the U-Net generalise

better on unseen data and this can be seen as the spikes in Dice loss are lower than in the case without data augmentation (seen as green) in the figure. The Dice loss with data tranformation pipeline 2 (red) is 0.369 at epoch 10 while the the Dice loss with data transformation pipeline 1 (blue) is 0.381 at epoch 10.

The validation and test Dice scores for each target segmentation class, at end epoch 25, are shown in the figure 4.8 and 4.10 respectively. A scatter plot is chosen for visual inspection while the legend/color-scheme is kept similar to the colors of the segmentation masks in the figures 4.3 and 4.4. The x-axis represents index of the 3D volume sample in the dataset and y-axis represents the Dice score for the 7 target segmentation classes of each sample data.

The multiple boxplots of the dice losses of the validation and test sets are represented in the figures 4.11 and 4.12 respectively with the summaries in the tables 4.2 and 4.3 respectively. Individual Boxplots for each segmenation class can be found in the Appendix, both for the validation and the test dataset. [1] [2] [3]

Summary of Validation Dice Score in figure 4.11							
Target Class	First Quartile	Median	Third Quartile	Lower Confidence interval	Upper Confidence interval	Mean	Standard Deviation
Background	0.9926	0.9947	0.9964	0.9945	0.9950	0.9914	0.0081
Left Ventricle	0.8581	0.8932	0.9239	0.8884	0.8981	0.8454	0.1351
Right Ventricle	0.8791	0.9216	0.9370	0.9170	0.9259	0.894	0.0645
Left Atrium	0.8525	0.8902	0.9130	0.8858	0.8946	0.8062	0.2015
Right Atrium	0.7657	0.8464	0.8883	0.8374	0.8554	0.8112	0.1074
Aorta	0.9132	0.9410	0.9460	0.9386	0.9434	0.8810	0.1285
Pulmonary Artery	0.8628	0.8970	0.9153	0.8932	0.9009	0.7773	0.2731

Table 4.2: **Summary of the Validation Boxplot.**
Rank ordering of the target classes in decreasing order of Dice score using median: Background, Aorta, Right Ventricle, Pulmonary Artery, Left Ventricle, Left Atria, Right Atria.
Rank ordering of the target classes in decreasing order of Dice score using mean: Background, Right Ventricle, Aorta, Left Ventricle, Right Atria, Left Atria, Pulmonary Artery.

[1] Inter-Quartile Range (IQR) = Third Quartile - First Quartile
[2] minimum = (First Quartile - 1.5*IQR)
[3] maximum = (Third Quartile + 1.5*IQR)

Summary of Test Dice Score in figure 4.12							
Target Class	First Quartile	Median	Third Quartile	Lower Confidence interval	Upper Confidence interval	Mean	Standard Deviation
Background	0.9956	0.9964	0.9968	0.9963	0.9965	0.9959	0.0013
Left Ventricle	0.8986	0.9166	0.9303	0.9141	0.9191	0.9133	0.0216
Right Ventricle	0.9010	0.9276	0.9391	0.9246	0.9306	0.9144	0.0386
Left Atrium	0.8770	0.9009	0.9116	0.8982	0.9036	0.8909	0.0346
Right Atrium	0.8359	0.8798	0.9028	0.8745	0.8851	0.8666	0.0490
Aorta	0.9214303	0.9349434	0.9411	0.9333	0.9364	0.9308	0.0137
Pulmonary Artery	0.8765577	0.8949705	0.9100	0.8923	0.8976	0.8898	0.0315

Table 4.3: **Summary of the Test Boxplot.**
Rank ordering of the target classes in decreasing order of Dice score using both the median and mean result in the same order: Background, Aorta, Right Ventricle, Left Ventricle, Left Atria, Pulmonary Artery, Right Atria.

Summary of Uncertainty maps in figures 4.13						
Sample	Uncertainty mean	Uncertainty variance	Uncertainty minimum	Uncertainty maximum	PAvPU	Prediction Accuracy
Test	0.453	0.00036	0.1135	0.5906	0.8655	0.8704
Validation	0.509	0.0013	0.3273	0.8101	0.6273	0.4121

Table 4.4: **Summary of the the inference samples.**
The difference between the 80th percentile and 20th percentile of the sigmoid output (80-20 Credible interval is mapped to show uncertainty in predictions)
Higher PAvPU in the test shows that it is better than the validation sample in terms of uncertainty quantification. PAvPU is shown for the evaluation of the Bayesian model on the particular sample.

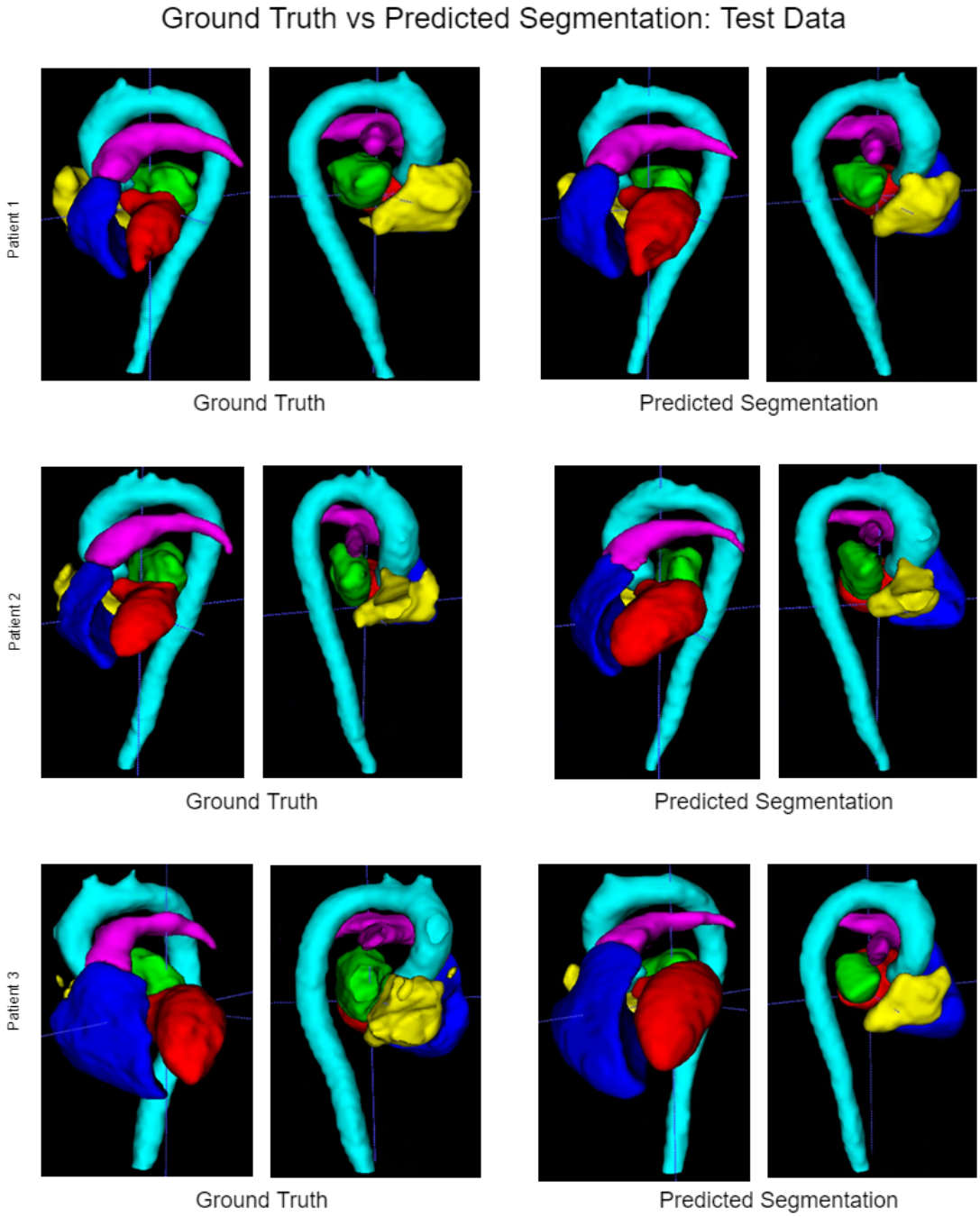

Figure 4.3: **(Left) Validation samples of 3 patients with its segmentation masks (Ground truth).** A validation sample is chosen at random and visualised with the output as NIFTI files in the isometric view at three different angles of rotation.

Figure 4.4: **(Right) Predicted Segmentation on the validation samples of the 3 different patients as shown in fig 4.3**.The predicted segmentation masks from the network that is trained with data augmentation pipeline 2 upto epoch 25. The Dice Similarity coefficients of these samples are good (Above 90percent)

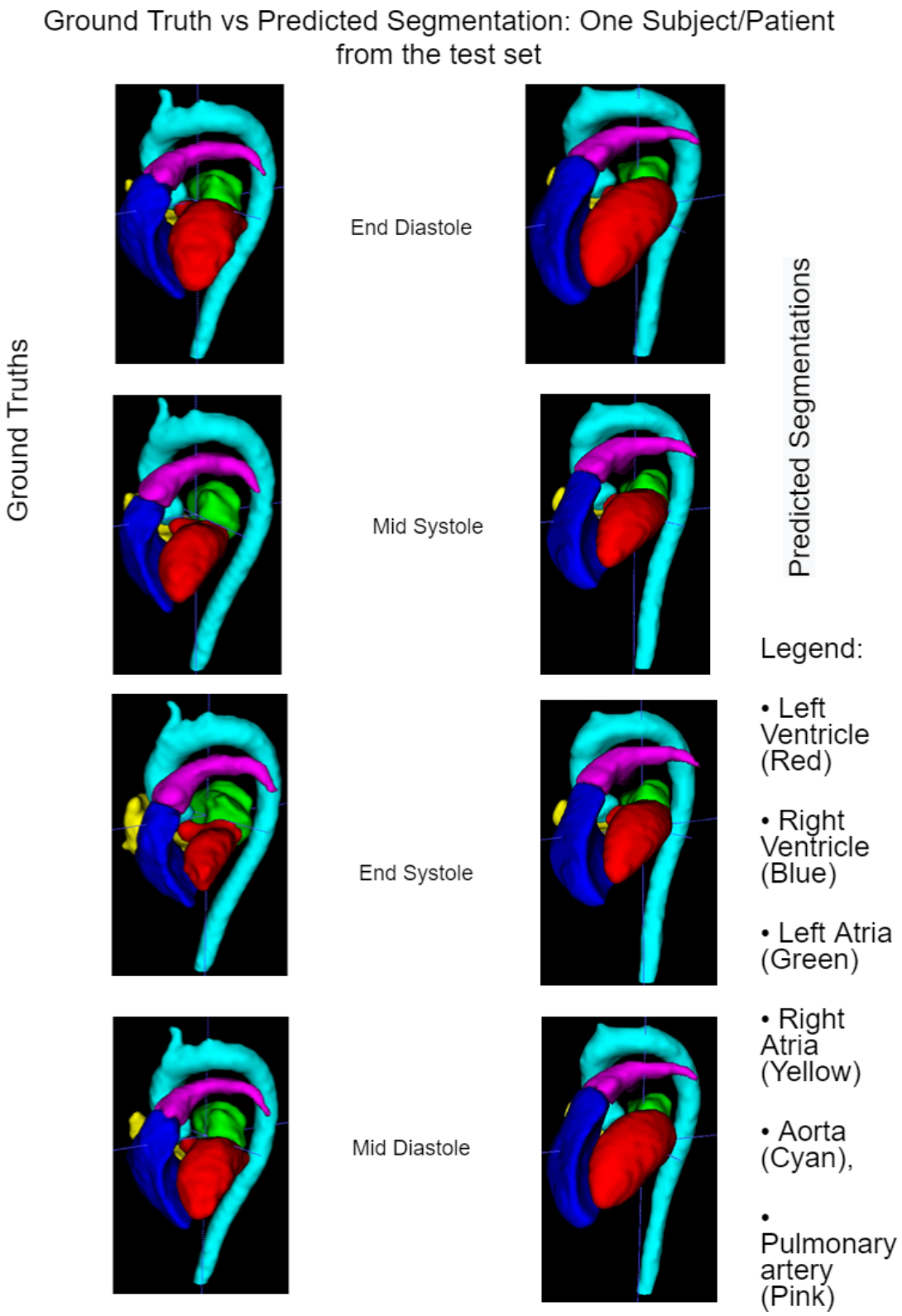

Figure 4.5: **Ground Truths (Left) and Predicted Segmentations (Right) for one subject/patient from the test dataset.** The End Diastole is the phase of the cardiac cycle when the heart is at its largest in volume and the End Systole is the phase of the cardiac cycle when the heart is at its smallest volume.

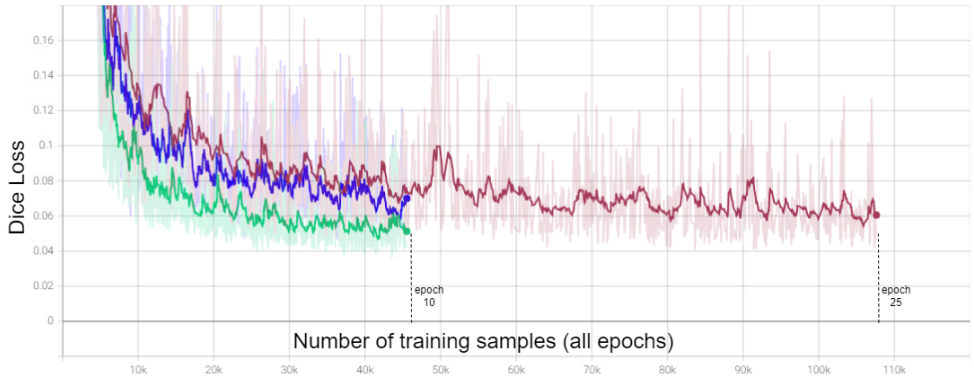

Figure 4.6: **The Training Loss by batch (smoothened); without data transformation (green), with data transformation - pipeline 1 (blue), with data transformation - pipeline 2 (red).** The higher training losses during training with data augmentation are because the CNN overfits more to the training data without any augmentation. However, the additional computational burden added by the transformation of input MRI intensities leads to slightly longer computational duration but allows the model to generalise better and reduce overfitting, as seen in the validation dice losses in figure 4.7

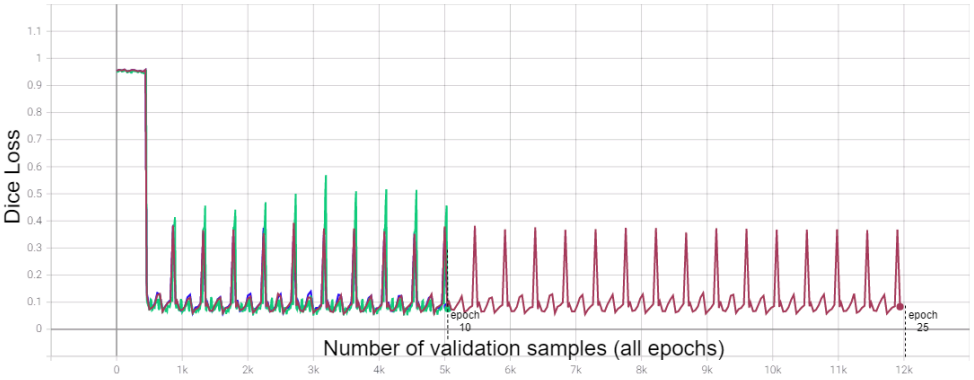

Figure 4.7: **The Validation Loss by batch; without data transformation (green), with data transformation - pipeline 1 (blue), with data transformation - pipeline 2 (red).** The red and blue line seem to be visually overlapping but the red line produces marginally lower loss. The data transformation pipeline 2 is chosen for further training till epoch 25. It is observed that although the training loss reduces, the validation loss does not improve with longer training duration.

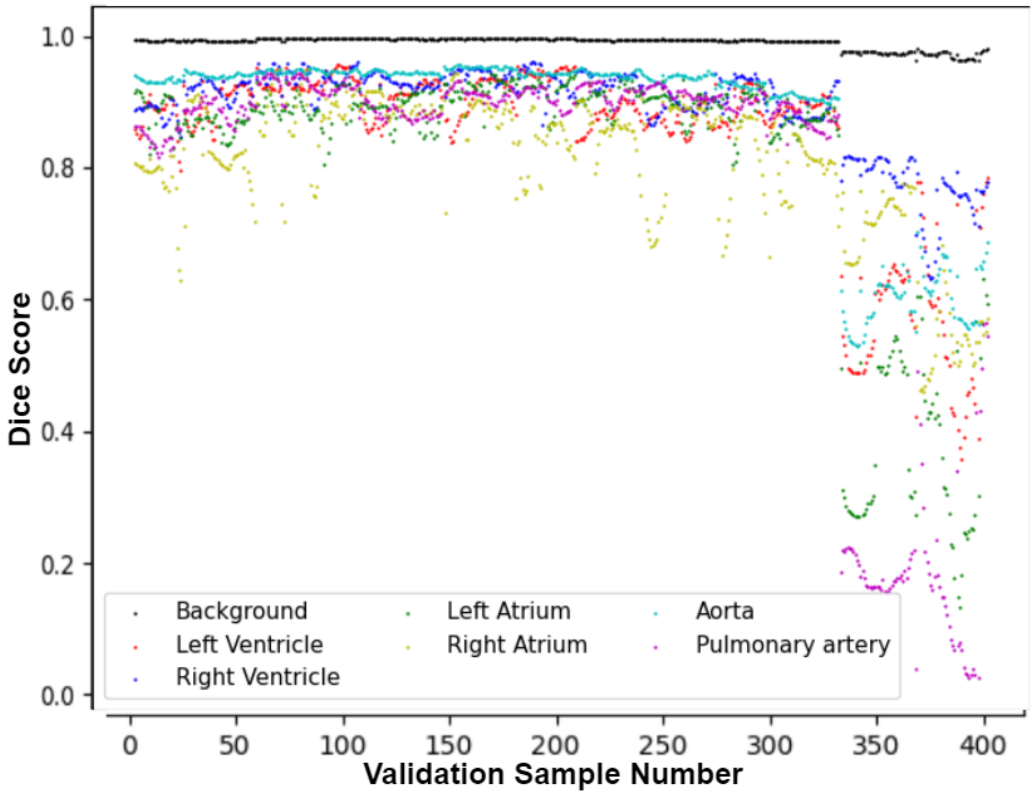

Figure 4.8: **Dice Scores of the validation set for each segmentation class label with the same afforementioned legend at epoch 25.** It is observed that the trained U-Net is not able to segment 120 validation samples well, with the Pulmonary Artery and Left Atrium standing out with very low Dice Scores on these samples. Upon investigation, it is revealed that these validation samples belong to 3 patients (since each patients cardiovascular data is of 40 timeframes that is discretized into 40 3D volumes for use in the 3D U-Net).

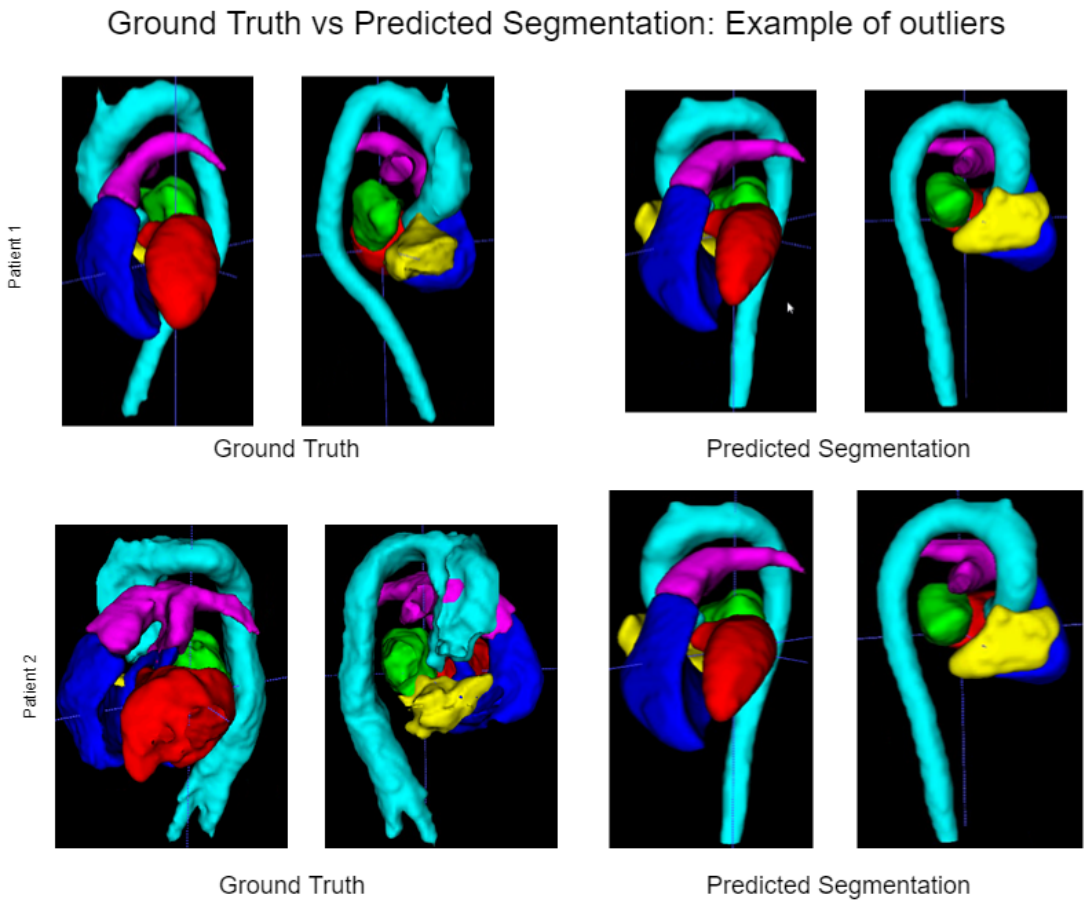

Figure 4.9: **Ground Truths (Left) and Predicted Segmentations (Right) for two of the outliers found present in the validation set.** Patient 1: Heart of a patient with disease and is deformed. Hence, it is very different from other.
Patient 2: Gadolinium contrasting agent was wrongly timed during acquisition, resulting in an incorrect scan that is with motion artefacts during scanning

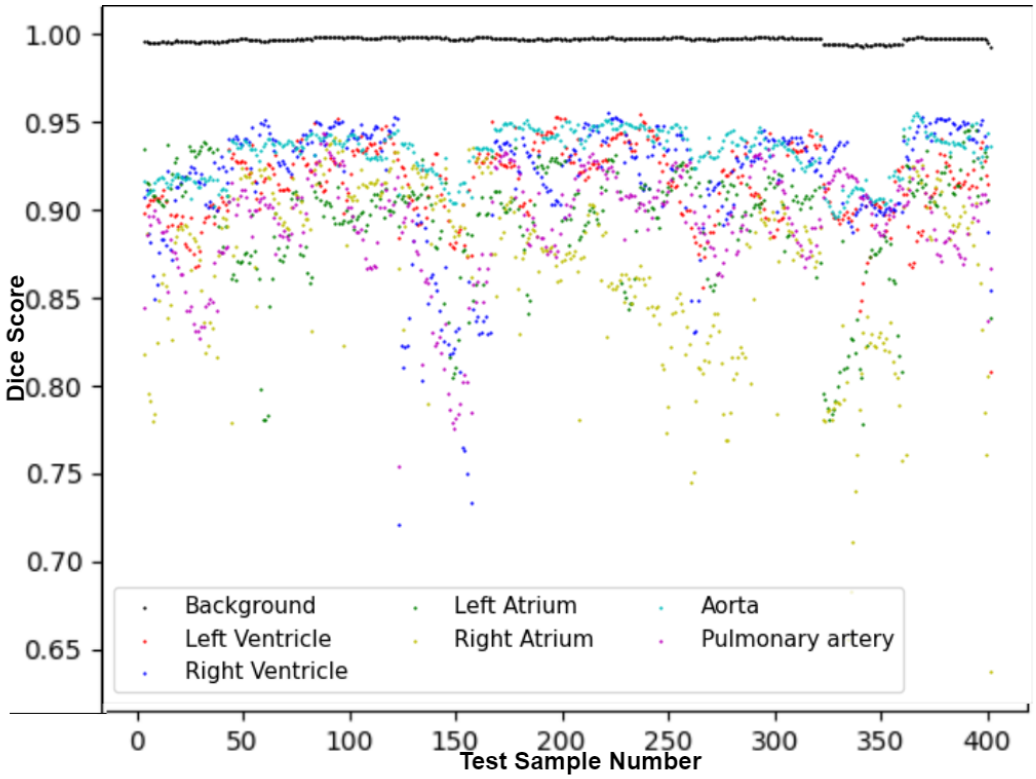

Figure 4.10: **Dice Scores of the test dataset for each segmentation class label with the same afforementioned legend at epoch 25.** It is observed that the trained U-Net performs better on the test/unseen dataset than the validation set. The lowest Dice scores on the test data is on the Right Atrium, Right Ventricle and the Pulmonary artery.

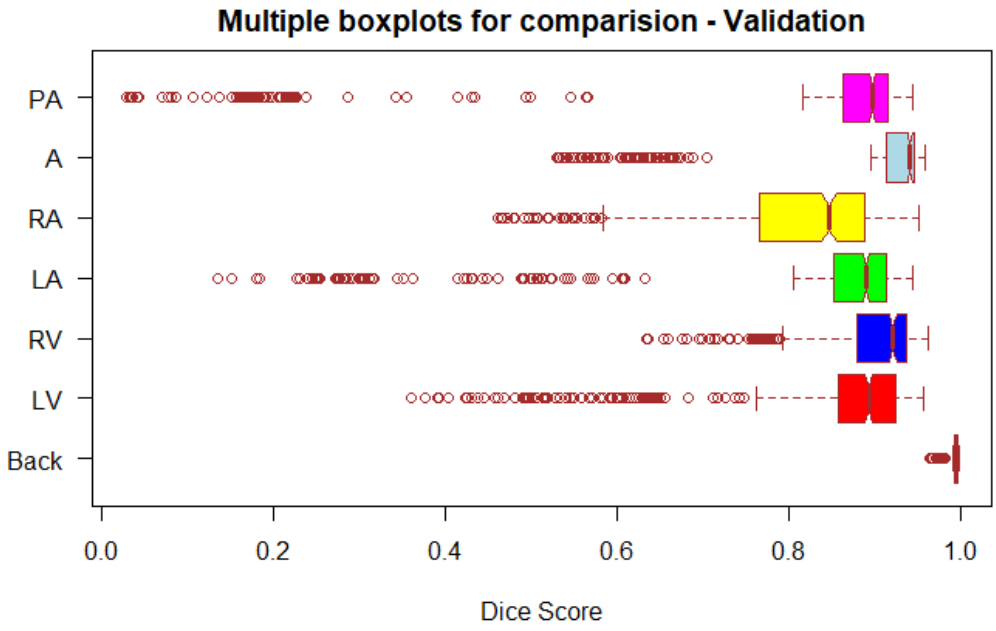

Figure 4.11: **Boxplot of the Dice Scores of the validation dataset for each segmentation class label with the same afforementioned legend.** It is observed that the validation set contains many outliers (represented as 'hollow circles') which is also seen in the figure 4.8 as the very low values. Outliers are the dice scores that lie beyond the minimum (First Quartile - 1.5*IQR) . The Right Atria is observed to have the largest range between the minimum and maximum.

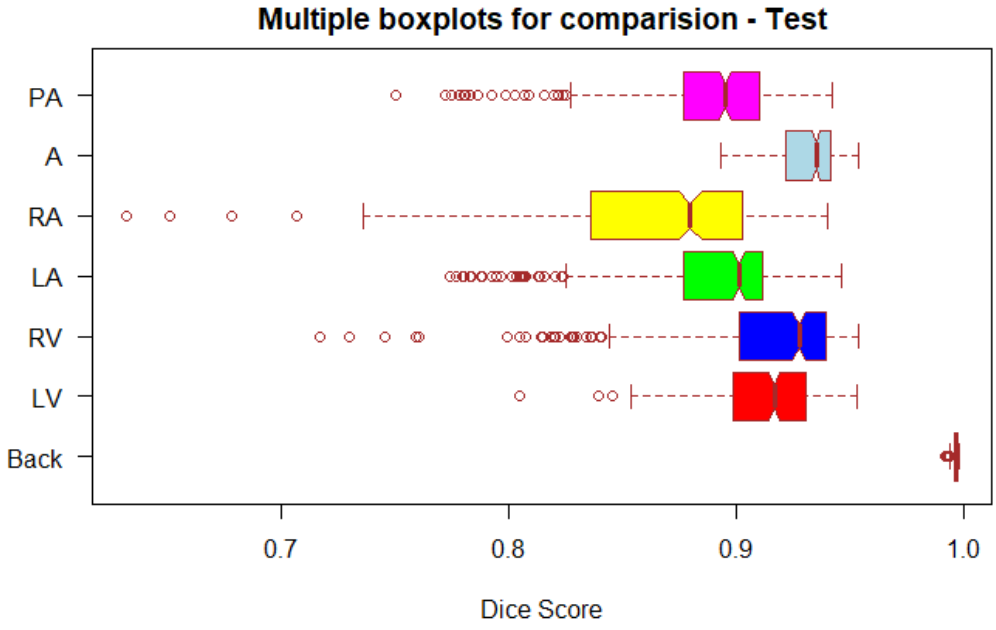

Figure 4.12: **Boxplot of the Dice Scores of the test set for each segmentation class label with the same afforementioned legend.** The test dataset contains far lesser outliers than the validation set while the ranges between the minimum and maximum of the boxplots i.e (excluding outliers) are larger than the validation set. The trained U-Net performs better on this unseen dataset than the validation set.

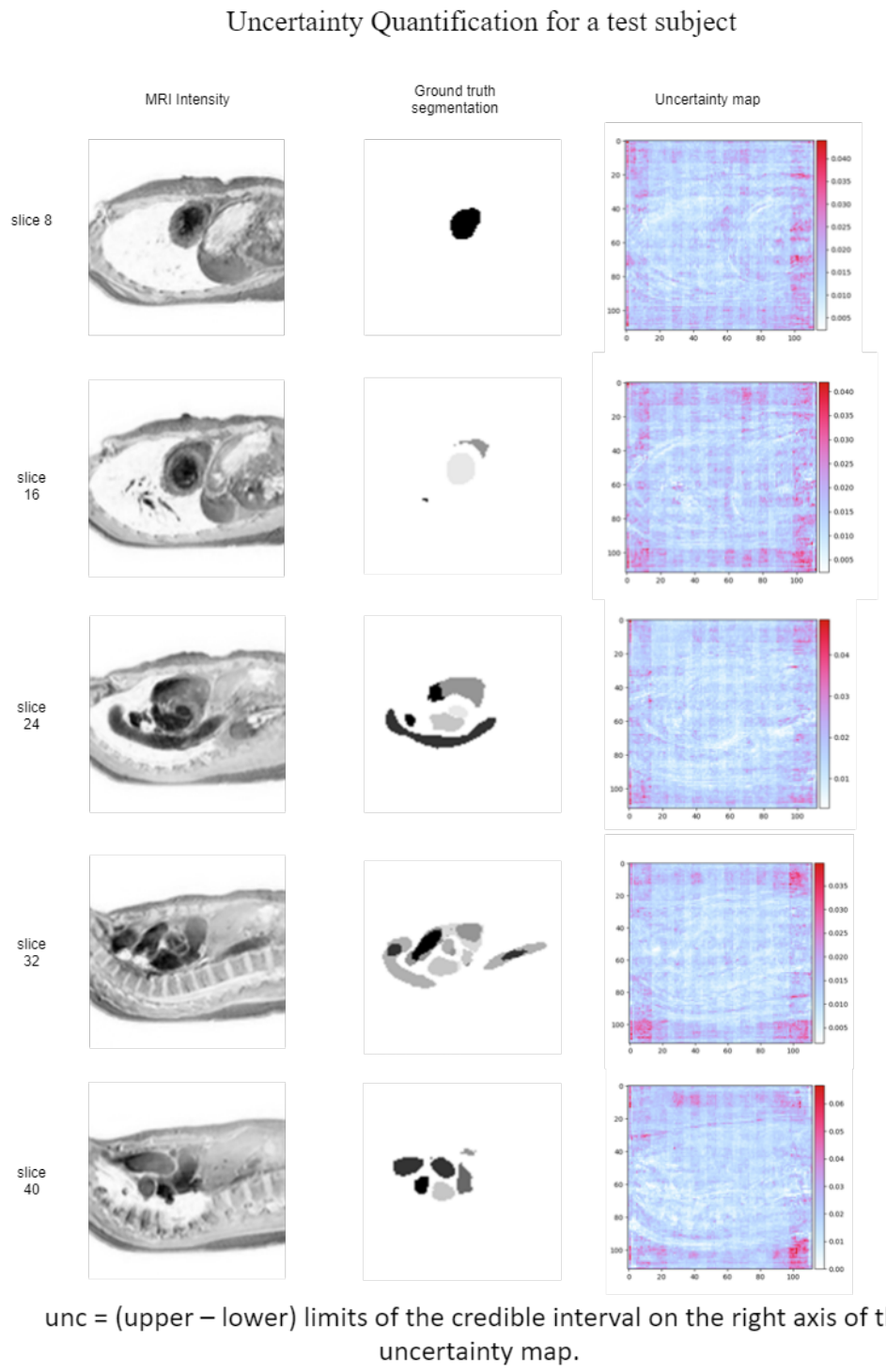

Figure 4.13: **5 equidistant slices of 3D sample - Uncertainty map.** Left: A test sample of 3D cardiac volume MRI intensities, 5 slices at equal intervals. Middle: 3D cardiac segmentation masks (ground truth) and Right: Uncertainty estimation map using variational inference. The MRI intensities are preprocessed and augmented, similar to the previous application. The ground truth segmentation masks are represented in greyscale and each shade corresponds to a target class. The x and y axis represent the original pixel sizes while the scale on the right side is the difference between the 80th percentile and 20th percentile of the sigmoids (80-20 credible interval); the higher it is (more red), higher is the uncertainty

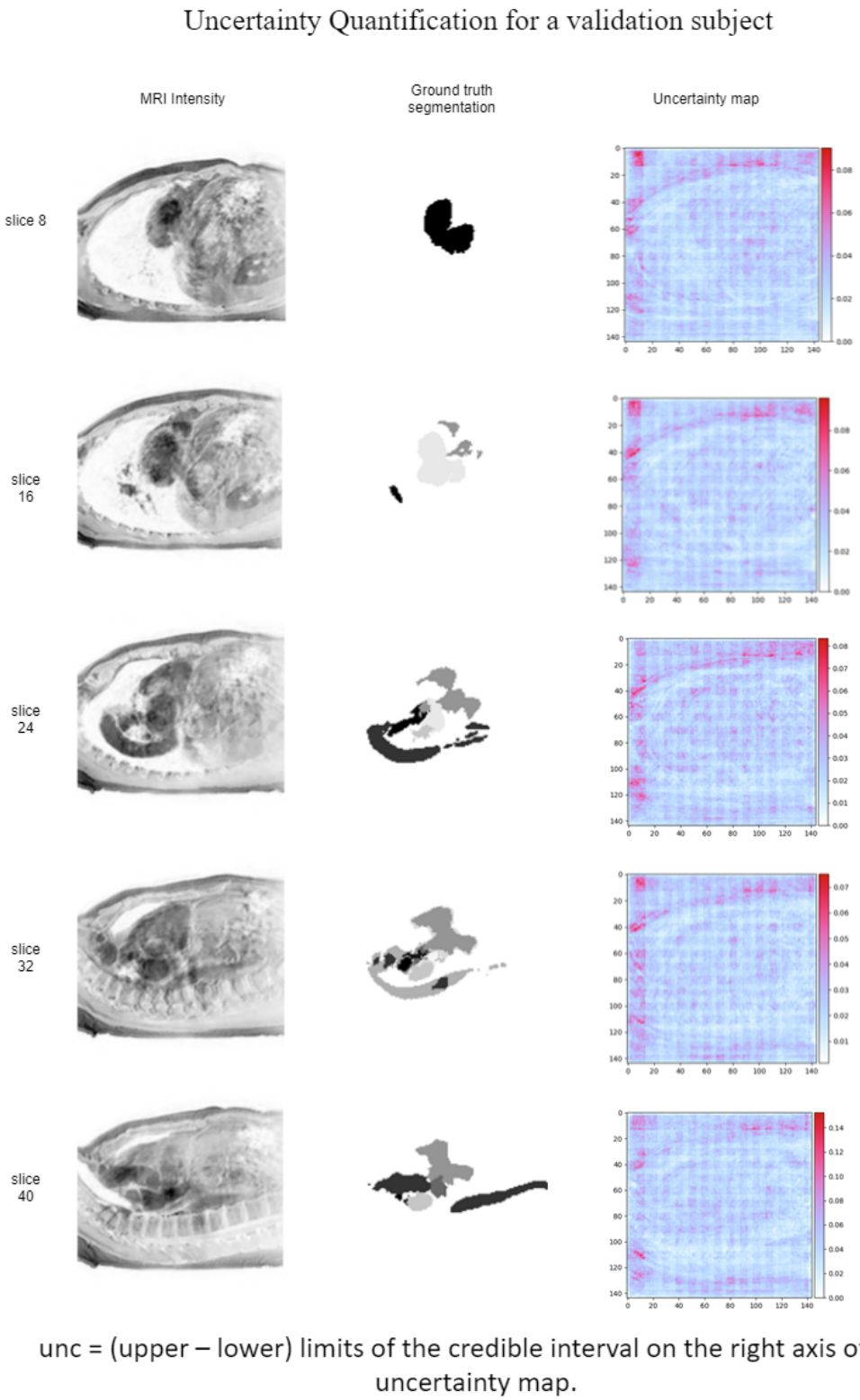

Figure 4.14: **5 equidistant slices of 3D sample - Uncertainty map.** Left: A validation sample of 3D cardiac volume MRI intensities, 5 slices at equal intervals. Middle: 3D cardiac segmentation masks (ground truth) and Right: Uncertainty estimation map using variational inference. The uncertainty is higher in patches within the cardiac areas.

5 Discussion

5.1 Results

5.1.1 Binary Segmentation

The results from the binary segmentation suggest that a 3D slice of 96X96X38 from the full volume of 112X112X48 can be taken from the center of the volume. This sub-volume could be used to train a multi-class model for possibly a better performance due to a valid concern that the neural network will be able to better segment the non-target voxels. However, The Two-step process of training separate networks will lead to extension of training duration. It is also seen in the figures 4.8 and 4.10 that the standalone multi-class segmentation achieves high performance on non-target voxels. Hence, eliminating the need for the Two-step segmentation such as presented in [51].

5.1.2 Multi-Class Segmentation

The figure 4.3 shows the ground truth segmentation of a sample from the validation data which is visualised from a Nifti file (.nii) for a sample of the input data. The figure 4.4 shows the output/predicted segmentation of the trained multi-class 3D U-Net on the same cardiovascular sample as seen in 4.3. The performance of the U-Net is great for semantic segmentation, this particular patients heart is slightly deformed and the network is able to capture even that. The aorta (cyan) is deformed at the top. The model is also able to capture the branches of the Pulmonary Artery which is particularly difficult to do, refer related work 2.1. The ground truth segmentation also has parts of the Superior Vena Cava that is connected to the Right Atrium, our network generalises from its training and does not produce that in the segmentation output which is also a positive.

When considering the mean Dice scores, it is seen in the figure 4.8 and table 4.2 that the model achieves the least performance on the Pulmonary Artery amongst all the classes and the best performance on the Right Ventricle (non-background) class followed by the aorta which is the largest part of the cardiac system and is well segmented even in the binary segmentation case. The table and figure for the test dataset 4.3 and 4.10 show that the model performs well on the unseen dataset with only outliers of all the classes other than the Aorta achieving lower Dice scores than 0.85, the Aorta is best segmented with a mean Dice score of 0.93 and the Right Atrium has the least Dice score with a mean of 0.866. High mean Dice

scores and low standard deviation of the Dice score show that the trained U-Net has a fair degree of certainty in its semantic segmentation prediction.

In the plot of the training loss by batch 4.6, It is seen that the model converges quicker without any data transformation. This is expected since we add additional burden on the model by augmenting and transforming the input data while the segmentation masks remain as is. Data tranformation induces variability within the training data, this makes it slightly difficult for the U-Net to converge to a solution quickly and hence, it seen that the two data transformation pipelines converge slower than without any transformation. The data augmentation pipeline 2 is selected for training for longer durations due to it achieving the least dice loss on the validation dataset as seen in the figure 4.7. It is visibly lower than the green line (no transformation) while the pipeline 2 (red) is also lower than pipeline 1 (blue), at epoch 9 (red - 0.354 vs blue - 0.357) and at epoch 10 (red - 0.369 vs blue - 0.381).

In the plot of the validation loss by batch 4.7, the spikes in the batch loss are observed at regular intervals i.e at the end of every epoch. The detailed breakup of the Dice scores amongst the target classes is seen in figure 4.8. The trained network is not able to predict the segmentation well on the 3D volumes derived from only 3 patients, namely, PX36 and PXXX from the CIP study and PXXX from the SVITMEK study [42]. Left Atria, Right Atria, Left Ventricle and Pulmonary artery are segmented very poorly as seen in the figure 4.8. This is repeated in the subsequent epochs and seen as spikes in the Dice loss in figure 4.7. Upon further investigation of the poorly segmented data, it is seen that the gadolinium contrasting agent that was injected into the patient was not timed properly during scanning. This has resulted in the MRI intensities to be very different from the other data/training dataset.

The caption in the figures 4.11 and 4.12 rank order the Dice scores on both the validation and test data. It is observed that the trained U-Net performs better on the test dataset than the validation set. The validation set has many outliers and the mean Dice scores are brought down due to them. The interquartile ranges are smaller than the test dataset. The U-Net might have even performed much better if not for the 120 bad datasets belonging to the three patients as seen in the median values of the Dice scores. The 95 percent confidence intervals show that the model generally achieves very high Dice scores of 0.84 to 0.94 within the validation set and 0.87 and 0.93 on the test data, excluding background in both cases. The variance of the Dice score is very high on the validation set compared to the test set, 4.3 and 4.2.

5.1.3 Uncertainty estimation using variational inference

The difference in the upper and lower bounds of the credible interval (80percent and 20percent) is plotted after the posteriors are evaluated. The RHS of figure 4.13 show the uncertainty maps at 5 equidistant slices of a 3D volume. It is observed that the bottom boundary of the human body is proving to be highly uncertain, so the semantic segmentation produced by the U-Net of a similar architecture as before will be uncertain at this boundary. The top most rows of pixels seemingly form a box, this may be due to the cropping and padding transform used during preprocessing that is showing up as a box. In the first and second slice (slice at z = 8 and z = 16), it is observed that the uncertainty is slightly higher than the background around the boundaries of the ground truth segmentation while in the third, fourth and fifth slices (slice at z = 24,32,40), the uncertainty is higher in small patches amongst pixels where the ground truth segmentation lies. In general, it can be said that the U-Net is performs fairly well with highest uncertainty reaching 0.17 on a scale of 0 to 1, that too only in the fifth slice where the backbone of the patient is dominating the area of the pixels. This kind of uncertainty map can be produced for all validation and test datasets. The Patch Accuracy vs Patch uncertainty metric is used to evaluate the bayesian model on the inference sample 2.8.2.1. It is found that the PAvPU is higher for the test subject as compared to the outlier subject from the validation set, suggesting that the model is more confident in its predictions on the test sample. Checkerboard patterns are observed in the uncertainty maps. These are

widely attributed as an artefact from the application of transpose convolutions in the upsampling layers, which leave behind checkerboard patches of more/less intensities during upsampling interpolation 2.4.

5.2 Method

There were few limitations in this study that led to the choices made in this study. The ideal setting for training any model is perfect data. The dataset used in this study mainly belongs to few previous studies [50] [42] [7]. The data is not ideal because the research related to the input data of Magnetic Resonance Images deal with patients with mild to severe cardiovascular disease and the research that produces the segmentation masks for training in this study, and is treated as ground truth in this study, is an automated tool that achieves high performance and is only validated by experts/cardiologists on samples of the data since the dataset is quite large and can take up crucial hours of work of the cardiologists. These segmentation masks that we treat as ground truths may not be gold standard and hence will affect the output of our trained model.

Since, working with 3D data comes with its own computational and storage resource challenges. This was also a factor in some decisions that shape the methodology in use; for example, the batch size in use for the multi-class 3D U-Net and the Bayesian implementation was kept low due to the GPU memory issue. In case of a multi-GPU set up, higher batch sizes could be used, higher training epochs could be explored and the 2 step 3D U-Net process could be explored further [51].

In the section 3.3.3, we list the transformation/augmentation applied on our MRI intensity data during training. The histogram standardisation and orientation standardisation are reasonable applied to normalise the data. The cropping is applied accordingly as explained in section 5.1.1 and zero padding is applied given notable literature as mentioned in section 2.1. Many transforms were explored in section 3.2.1 but particularly those were chosen that did not change the segmentation masks of the input data so as to maintain the ground truths of the data; for example: Random Noise was excluded for application for this reason. It was also noted that the data transformation methodology should be performed on a sample that has clearly defined boundaries visible in the MRI intensities i.e a sample where the contrast is highlighted with the gadolinium agent 1.3 so as to increase the replicability and reliability of this study. The use of probability of the application of transformation was kept low to replicate real world scenario's.

The sources used for the data augmentation part was done using the open source library torchio [1] [2]. Proper research is also available such as [45] but is inaccesible in terms of application.

Uncertainty estimation was done with help of both recent research published by Labonte et. al. [27], [34] and open source libraries such as [3] [4]

In the figure 4.13, slices of the samples of the training data input, validation targets (ground truth) are shown. Based on these an uncertainty map is produced in 3D 4.13 on the RHS. This uncertainty is based on the evaluation metric expounded in section 2.8.2.1 and also the credible intervals that are computed from the sigmoids of the model; the difference between the upper and lower bounds of the credible interval provides an uncertainty measure that is mapped for inspection.

[1] https://torchio.readthedocs.io/index.html
[2] https://colab.research.google.com/github/fepegar/torchio-notebooks
[3] https://github.com/sandialabs/bcnn
[4] https://github.com/yuta-hi/bayesian$_u$net

5.3 The work in a wider context

The implementation of such deep neural networks in the healthcare industry will provide much needed help to the doctors and nurses who work in high pressure environments where diagnosis and care are directly a matter of life and death. The work presented in this research not only elucidates methodology for bringing machine learning to production in the field of cardiology but also addresses the critical nature of the cardiovascular diagnosis where uncertainty prevails and the quantification tool for the same may prove to be the saving grace for healthcare professionals.

Critical questions that arise in the implementation of the 3D U-Net deep learning architecture have been answered. This study also lays out a clear path for semantic segmentation in other biomedical applications as well. The Bayesian 3D U-Net for uncertainty estimation can be considered novel in terms of application to cardiac data and the 3D U-Net for semantic multi-class segmentation is also unique as previous deep learning architectures of the similar type are applied on specific areas of the cardiac MR.

Cardiovascular diseases are the leading cause of death in the world over. This work also throws light on the possibility of automatic diagnosis of cardiovascular disease. Cardiac semantic segmentation is the first step in understanding the patients cardiac morphology which can be used to identify and diagnose diseases and anomalies, thereby eliminating the need for cardiologists and radiologists altogether, and allowing for automatic diagnosis of patients at a preliminary stage.

6 Conclusion

A 3D U-Net trained on 3D data is a good choice of convolutional neural network architecture for the purpose of semantic segmentation in this with cardiac data. The segmentation strategy to be adopted is fully automatic because there is a large interest in saving the time of healthcare experts and also pausing during training for manual annotation will only lead to a delay in the segmentation process. Performance evaluation of our implemented models is done through Dice Loss for multi-class 3D U-Net, Implementation of the Bayesian version of a 3D U-Net for uncertainty estimation via the credible intervals mapped and 'patch accuracy vs patch uncertainty' for evaluating the bayesian 3D U-Net. This implementation of the bayesian 3D U-Net for semantic segmentation uncertainty estimation can be considered as a novel contribution to the field of deep learning as this method may not have been applied to such data before. Data augmentation procedures should be chosen according to the plausibility of existence in the real world data, multiple data transformation pipelines are evaluated over the training and validation Dice losses they produce for each segmentation target class.

This research has wide applications in the field of cardiovascular medicine and diagnosis. This study can reduce the time required for segmentation of cardiovascular MRI in a hospital setting, thereby freeing up experts such as doctors, radiologists and nurses for more critical work. Deep learning methods explained here can be applied to other biomedical segmentation applications as well.

6.1 Future Work

The study can be repeated with more data from diverse sources to make the trained network more robust.

The study can be repeated but with the data converted to 2 dimensional so that comparisons can be made between 2D and 3D implementations of multi-class U-Nets and bayesians implementations as well.

Time series analysis can be performed to evaluate the model predictions and if some phases of the cardiac cycle are segmented better than others.

Other models of deep learning can be studied such as the V-Net, anatomically constrained neural networks. Uncertainty estimation can also be done using Monte Carlo Dropout Networks and the results can be compared with the method implemented in this study.

Bibliography

[1] *Affine transformation*. ttps://nipy.org/nibabel/coordinate_systems.html. [Online; accessed 20-Dec-2020]. 2020.

[2] Ian Goodfellow et al. *On the difficulty of training Recurrent Neural Networks*. 2016. eprint: DeepLearning.http://www.deeplearningbook.org..

[3] *An overview of semantic image segmentation*. https://www.jeremyjordan.me/semantic-segmentation/. [Online; accessed 20-Dec-2020]. 2019.

[4] *Bayesian U-Net implementation sandia labs tensorflow*. https://github.com/sandialabs/bcnn. [Online; accessed 20-Dec-2020]. 2020.

[5] *Bayesian U-Net implementation yuta pytorch*. https://github.com/yuta-hi/bayesian_unet. [Online; accessed 20-Dec-2020]. 2020.

[6] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. *Weight Uncertainty in Neural Networks*. 2015. arXiv: 1505.05424 [stat.ML].

[7] Mariana Bustamante, V Gupta, D Forsberg, CJ Carlhäll, J Engvall, and T Ebbers. "Automated multi-atlas segmentation of cardiac 4D flow MRI." In: (2018). URL: https://pubmed.ncbi.nlm.nih.gov/30144652/%20%7Bpubmed%7D.

[8] Johann Christopher. *PET vs MRI for Myocardial Viability*. Indian Journal of Clinical Cardiology. [Online; accessed 20-Dec-2020]. 2020.

[9] Özgün Cicek, Ahmed Abdulkadir, Soeren S. Lienkamp, Thomas Brox, and Olaf Ronneberger. *3D U-Net: Learning Dense Volumetric Segmentation from Sparse Annotation*. 2016. arXiv: 1606.06650 [cs.CV].

[10] *Cross entropy loss*. https://pytorch.org/docs/stable/nn.html?highlight=crossentropyloss#torch.nn.CrossEntropyLoss. [Online; accessed 20-Dec-2020]. 2019.

[11] *Cross entropy losses*. https://www.groundai.com/project/metric-learning-cross-entropy-vs-pairwise-losses/. [Online; accessed 20-Dec-2020]. 2019.

[12] Lee R. Dice. "Measures of the Amount of Ecologic Association Between Species". In: *Ecology* 26.3 (1945), pp. 297–302. DOI: https://doi.org/10.2307/1932409. eprint: https://esajournals.onlinelibrary.wiley.com/doi/pdf/10.2307/1932409. URL: https://esajournals.onlinelibrary.wiley.com/doi/abs/10.2307/1932409.

[13] Yongjie Duan, Jianjiang Feng, Jiwen Lu, and Jie Zhou. "Context Aware 3D Fully Convolutional Networks for Coronary Artery Segmentation". In: *Statistical Atlases and Computational Models of the Heart. Atrial Segmentation and LV Quantification Challenges*. Ed. by Mihaela Pop, Maxime Sermesant, Jichao Zhao, Shuo Li, Kristin McLeod, Alistair Young, Kawal Rhode, and Tommaso Mansi. Cham: Springer International Publishing, 2019, pp. 85–93. ISBN: 978-3-030-12029-0.

[14] T. Falk, D. Mai, R. Bensch, Ö. Çiçek, A. Abdulkadir, Y. Marrakchi, A. Böhm, J. Deubner, Z. Jäckel, K. Seiwald, A. Dovzhenko, O. Tietz, C. Dal Bosco, S. Walsh, D. Saltukoglu, T. L. Tay, M. Prinz, K. Palme, M. Simons, I. Diester, T. Brox, and O. Ronneberger. "U-Net – Deep Learning for Cell Counting, Detection, and Morphometry". In: *Nature Methods* 16 (2019), pp. 67–70. URL: http://lmb.informatik.uni-freiburg.de/Publications/2019/FMBCAMBBR19.

[15] *Gadolinium contrast medium*. https://www.insideradiology.com.au/gadolinium-contrast-medium/. [Online; accessed 20-Dec-2020]. 2015.

[16] Yarin Gal and Zoubin Ghahramani. *Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning*. 2016. arXiv: 1506.02142 [stat.ML].

[17] Alex Graves. "Practical Variational Inference for Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Q. Weinberger. Vol. 24. Curran Associates, Inc., 2011, pp. 2348–2356. URL: https://proceedings.neurips.cc/paper/2011/file/7eb3c8be3d411e8ebfab08eba5f49632-Paper.pdf.

[18] Fumin Guo, Matthew Ng, and Graham Wright. "Cardiac MRI Left Ventricle Segmentation and Quantification: A Framework Combining U-Net and Continuous Max-Flow". In: *Statistical Atlases and Computational Models of the Heart. Atrial Segmentation and LV Quantification Challenges*. Ed. by Mihaela Pop, Maxime Sermesant, Jichao Zhao, Shuo Li, Kristin McLeod, Alistair Young, Kawal Rhode, and Tommaso Mansi. Cham: Springer International Publishing, 2019, pp. 450–458. ISBN: 978-3-030-12029-0.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*. 2015. arXiv: 1502.01852 [cs.CV].

[20] *How to Configure Image Data Augmentation in Keras*. https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks. [Online; accessed 20-Dec-2020]. 2020.

[21] *Human Age Prediction Based on Real and Simulated RR Intervals using Temporal Convolutional Neural Networks and Gaussian Processes*. https://liu.diva-portal.org/smash/get/diva2:1434886/FULLTEXT01.pdf. [Online; accessed 20-Dec-2020]. 2020.

[22] *Interpretation of a boxplot*. https://towardsdatascience.com/understanding-boxplots-5e2df7bcbd51. [Online; accessed 20-Dec-2020]. 2020.

[23] Shuman Jia, Antoine Despinasse, Zihao Wang, Hervé Delingette, Xavier Pennec, Pierre Jaïs, Hubert Cochet, and Maxime Sermesant. *Automatically Segmenting the Left Atrium from Cardiac Images Using Successive 3D U-Nets and a Contour Loss*. 2018. arXiv: 1812.02518 [cs.CV].

[24] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[25] Diederik P. Kingma, Tim Salimans, and Max Welling. *Variational Dropout and the Local Reparameterization Trick*. 2015. arXiv: 1506.02557 [stat.ML].

[26] Tyler Labonte. "Deep Learning Segmentation with Uncertainty via 3D Bayesian Convolutional Neural Networks". In: *Towards DataScience* (2020). eprint: https://towardsdatascience.com/deep-learning-segmentation-with-uncertainty-via-3d-bayesian-convolutional-neural-networks-6b1c7277b078. URL: https://towardsdatascience.com/deep-learning-segmentation-with-uncertainty-via-3d-bayesian-convolutional-neural-networks-6b1c7277b078.

[27] Tyler LaBonte, Carianne Martinez, and Scott A. Roberts. *We Know Where We Don't Know: 3D Bayesian CNNs for Credible Geometric Uncertainty*. 2020. arXiv: 1910.10793 [eess.IV].

[28] Jonathan Long, Evan Shelhamer, and Trevor Darrell. *Fully Convolutional Networks for Semantic Segmentation*. 2015. arXiv: 1411.4038 [cs.CV].

[29] Ilya Loshchilov and Frank Hutter. "Fixing Weight Decay Regularization in Adam". In: *CoRR* abs/1711.05101 (2017). arXiv: 1711.05101. URL: http://arxiv.org/abs/1711.05101.

[30] Michael Markl, Alex Frydrychowicz, Sebastian Kozerke, Mike Hope, and Oliver Wieben. "4D flow MRI". In: *Journal of Magnetic Resonance Imaging* (2012).

[31] Fausto Milletari, Nassir Navab, and Seyed-Ahmad Ahmadi. *V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation*. 2016. arXiv: 1606.04797 [cs.CV].

[32] Aliasghar Mortazi, Jeremy Burt, and Ulas Bagci. *Multi-Planar Deep Segmentation Networks for Cardiac Substructures from MRI and CT*. 2017. arXiv: 1708.00983 [stat.ML].

[33] *MR Fingerprinting Quantitative Imaging*. https://www.ismrm.org/15/program_files/TueEPS02.htm. [Online; accessed 20-Dec-2020]. 2015.

[34] Jishnu Mukhoti and Yarin Gal. *Evaluating Bayesian Deep Learning Methods for Semantic Segmentation*. 2019. arXiv: 1811.12709 [cs.CV].

[35] O. Oktay, E. Ferrante, K. Kamnitsas, M. Heinrich, W. Bai, J. Caballero, S. A. Cook, A. de Marvao, T. Dawes, D. P. O'Regan, B. Kainz, B. Glocker, and D. Rueckert. "Anatomically Constrained Neural Networks (ACNNs): Application to Cardiac Image Enhancement and Segmentation". In: *IEEE Transactions on Medical Imaging* 37.2 (2018), pp. 384–395.

[36] Michael E. Osadebey, Marius Pedersen, Douglas L. Arnold, and Katrina E. Wendel-Mitoraj. "Blind blur assessment of MRI images using parallel multiscale difference of Gaussian filters". eng. In: *Biomedical engineering online* 17.1 (June 2018). PMC6001176[pmcid], pp. 76–76. ISSN: 1475-925X. DOI: 10.1186/s12938-018-0514-4. URL: https://doi.org/10.1186/s12938-018-0514-4.

[37] Ian Osband. "Risk versus Uncertainty in Deep Learning : Bayes , Bootstrap and the Dangers of Dropout". In: 2016.

[38] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. *On the difficulty of training Recurrent Neural Networks*. 2013. arXiv: 1211.5063 [cs.LG].

[39] Fernando Perez-Garcia. *Data preprocessing and augmentation using TorchIO*. https://github.com/fepegar/torchio/tree/master/torchio/transforms. [Online; accessed 20-Dec-2020]. 2020.

[40] Fernando Perez-Garcia. *fepegar/unet: First published version of PyTorch U-Net*. Version v0.6.4. Oct. 2019. DOI: 10.5281/zenodo.3522306. URL: https://doi.org/10.5281/zenodo.3522306.

[41] *PReLU activation*. https://medium.com/@shoray.goel/prelu-activation-e294bb21fefa. [Online; accessed 20-Dec-2020]. 2019.

[42] Frank Rademakers, Jan Engvall, Thor Edvardsen, Mark Monaghan, Rosa Sicari, Eike Nagel, José Zamorano, Heikki Ukkonen, Tino Ebbers, Vitantonio Di Bello, Jens-Uwe Voigt, Lieven Herbots, Piet Claus, and Jan D'hooge. "Determining optimal noninvasive parameters for the prediction of left ventricular remodeling in chronic ischemic patients". In: *Scandinavian Cardiovascular Journal* 47.6 (2013). PMID: 24295289, pp. 329–334. DOI: 10.3109/14017431.2013.857039. eprint: https://doi.org/10.3109/14017431.2013.857039. URL: https://doi.org/10.3109/14017431.2013.857039.

[43] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015* (2015). Ed. by Nassir Navab, Joachim Hornegger, William M. Wells, and Alejandro F. Frangi.

[44] Holger R. Roth, Hirohisa Oda, Yuichiro Hayashi, Masahiro Oda, Natsuki Shimizu, Michitaka Fujiwara, Kazunari Misawa, and Kensaku Mori. *Hierarchical 3D fully convolutional networks for multi-organ segmentation*. 2017. arXiv: 1704.06382 [cs.CV].

[45] Richard Shaw, Carole Sudre, Sebastien Ourselin, and M. Jorge Cardoso. "MRI k-Space Motion Artefact Augmentation: Model Robustness and Task-Specific Uncertainty". In: *Proceedings of The 2nd International Conference on Medical Imaging with Deep Learning*. Ed. by M. Jorge Cardoso, Aasa Feragen, Ben Glocker, Ender Konukoglu, Ipek Oguz, Gozde Unal, and Tom Vercauteren. Vol. 102. Proceedings of Machine Learning Research. London, United Kingdom: PMLR, Aug. 2019, pp. 427–436. URL: http://proceedings.mlr.press/v102/shaw19a.html.

[46] Kumar Shridhar, Felix Laumann, and Marcus Liwicki. *A Comprehensive guide to Bayesian Convolutional Neural Network with Variational Inference*. 2019. arXiv: 1901.02731 [cs.LG].

[47] Dustin Tran, Michael W. Dusenberry, Mark van der Wilk, and Danijar Hafner. *Bayesian Layers: A Module for Neural Network Uncertainty*. 2019. arXiv: 1812.03973 [cs.LG].

[48] N. J. Tustison, B. B. Avants, P. A. Cook, Y. Zheng, A. Egan, P. A. Yushkevich, and J. C. Gee. "N4ITK: improved N3 bias correction". In: *IEEE Trans Med Imaging* 29.6 (June 2010), pp. 1310–1320.

[49] *Up-sampling with Transposed Convolution*. https://naokishibuya.medium.com/up-sampling-with-transposed-convolution-9ae4f2df52d0. [Online; accessed 20-Dec-2020]. 2020.

[50] GEORGIA VARELOGIANNI. *Myokard deformation i vänsterkammare hos hjärtsviktspatienter – metodologiska och patofysiologiska aspekter*. 2013-03-27. arXiv: Registrationnumber:LIO-344591 [eess.IV].

[51] C Wang, T MacGillivray, G Macnaught, G Yang, and D Newby. "A Two-Stage U-Net Model for 3D Multi-class Segmentation on Full-Resolution Cardiac Data". In: *Pop M. et al. (eds) Statistical Atlases and Computational Models of the Heart. Atrial Segmentation and LV Quantification Challenges* (2019).

[52] *When Do I Need an MRI?* https://www.webmd.com/a-to-z-guides/when-do-i-need-an-mri. [Online; accessed 20-Dec-2020]. 2020.

[53] Yuxin Wu and Kaiming He. *Group Normalization*. 2018. arXiv: 1803.08494 [cs.CV].

[54] Xin Yang, Cheng Bian, Lequan Yu, Dong Ni, and Pheng-Ann Heng. "3D Convolutional Networks for Fully Automatic Fine-Grained Whole Heart Partition". In: Jan. 2018, pp. 181–189. ISBN: 978-3-319-75540-3. DOI: 10.1007/978-3-319-75541-0_19.

[55] Lequan Yu, Jie-Zhi Cheng, Qi Dou, Xin Yang, Hao Chen, Jing Qin, and Pheng-Ann Heng. *Automatic 3D Cardiovascular MR Segmentation with Densely-Connected Volumetric ConvNets*. 2017. arXiv: 1708.00573 [cs.CV].

Appendix

Omkar Bhutra

January 25, 2021

Appendix

1 Setting up of data transformation and augmentation:

Old orientation: ('L', 'P', 'S') New orientation: ('R', 'A', 'S') print(fpg$_r as.mri$)... : $print(fpg_r as.heart)... : ScalarImage(shape : (1, 112, 112, 44); spacing : (2.68, 2.68, 2.80); orientation : RAS+; memory : 2.1MiB; type : intensity) LabelMap(shape : (1, 112, 112, 44); spacing : (2.68, 2.68, 2.80); orientation : RAS+; memory : 2.1MiB; type : label)$

Out[11]: Affine array array([[-2.68, 0. , 0. , 0.], [0. , -2.68, 0. , 0.], [0. , 0. , 2.8 , 0.], [0. , 0. , 0. , 1.]])

2 Hyperparameter's sest for the Bayesian 3D U-Net for uncertainty estimation.

"norm": true, "step": 3, "window": [16, 32, 32],

"resume": false, "bayesian": true, "vnet": true, "ensemble": false, "prior$_s td$" : 0.5, "$kernel_s ize$" : 3, "$activation$" : "$relu$", "$padding$" : "$SAME$", "$kl_a lpha$" : 0, "$kl_s tart_e poch$" : 2, "$kl_a lpha_i ncrease_p er_e poch$" : 0.5,

"epochs": 5, "initial$_e poch$" : 0, "$initial_l earning_r ate$" : 0.001, "$lr_d ecay_s tart_e poch$" : 10, "$batch_s ize$" : 2, "$num_g pus$" : 1, "$mc_s amples$" : 48, "$border_t rim$" : 0.1, "$lower_p ercentile$" : 20, "$upper_p ercentile$" : 80

3 NLL Tensorflow probability implementation

A valid Python implementation of the variational free energy loss. Binary cross-entropy is the same as negated NLL.

4 Training the BCNN

TorchIO version: 0.18.1 Dataset size: 9200 subjects Subject(Keys: ('mri', 'heart'); images: 2) ScalarImage(shape: (1, 112, 112, 44); spacing: (2.68, 2.68, 2.80); ori-

entation: LPS+; memory: 2.1 MiB; type: intensity) Dataset size: 9200 subjects Subject(Keys: ('mri', 'heart'); images: 2) ScalarImage(shape: (1, 112, 112, 44); spacing: (2.68, 2.68, 2.80); orientation: LPS+; memory: 2.1 MiB; type: intensity)

Trained landmarks: [0. , 0.333 , 2.208 , 4.331 , 8.027, 19.904, 32.262 41.036 48.527,52.615 , 57.196 , 69.38 , 100.] Training set: 8280 subjects Validation set: 920 subjects

5 Multi Class U-Net Architecture

- (encoder): Encoder((encoding$_b locks$) : $ModuleList((0) : EncodingBlock((conv1) : ConvolutionalBlock((conv_layer) : Conv3d(1, 16, kernel_size = (3, 3, 3), stride = (1, 1, 1), padding = (1, 1, 1))(activation_layer) : PReLU(num_parameters = 1)(block) : Sequential((0) : Conv3d(1, 16, kernel_size = (3, 3, 3), stride = (1, 1, 1), padding = (1, 1, 1))(1) : PReLU(num_parameters = 1))))))$, $(conv2) : ConvolutionalBlock((conv_layer) : Conv3d(16, 32, kernel_size = (3, 3, 3), stride = (1, 1, 1), padding = (1, 1, 1))(norm_layer) : BatchNorm3d(32, eps = 1e − 05, momentum = 0.1, affine = True, track_running_stats = True)(activation_layer) : PReLU(num_parameters = 1)(block) : Sequential((0) : Conv3d(16, 32, kernel_size = (3, 3, 3), stride = (1, 1, 1), padding = (1, 1, 1))(1) : BatchNorm3d(32, eps = 1e−05, momentum = 0.1, affine = True, track_running_stats = True)(2) : PReLU(num_parameters = 1))(downsample) : MaxPool3d(kernel_size = 2, stride = 2, padding = 0, dilation = 1, ceil_mode = False)),$

- (1): EncodingBlock((conv1): ConvolutionalBlock((conv$_layer$) : $Conv3d(32, 32, kernel_size = (3, 3, 3), stride = (1, 1, 1), padding = (1, 1, 1))(norm_layer) : BatchNorm3d(32, eps = 1e−05, momentum = 0.1, affine = True, track_running_stats = True)(activation_layer) : PReLU(num_parameters = 1)(block) : Sequential((0) : Conv3d(32, 32, kernel_size = (3, 3, 3), stride = (1, 1, 1), padding = (1, 1, 1))(1) : BatchNorm3d(32, eps = 1e − 05, momentum = 0.1, affine = True, track_running_stats = True)(2) : PReLU(num_parameters = 1)))), (conv2) : ConvolutionalBlock((conv_layer) : Conv3d(32, 64, kernel_size = (3, 3, 3), stride = (1, 1, 1), padding = (1, 1, 1))(norm_layer) : BatchNorm3d(64, eps = 1e−05, momentum = 0.1, affine = True, track_running_stats = True)(activation_layer) : PReLU(num_parameters = 1)(block) : Sequential((0) : Conv3d(32, 64, kernel_size = (3, 3, 3), stride = (1, 1, 1), padding = (1, 1, 1))(1) : BatchNorm3d(64, eps = 1e−05, momentum = 0.1, affine = True, track_running_stats = True)(2) : PReLU(num_parameters = 1))(downsample) : MaxPool3d(kernel_size = 2, stride = 2, padding = 0, dilation = 1, ceil_mode = False)),),$

- (2): EncodingBlock((conv1): ConvolutionalBlock((conv$_layer$) : $Conv3d(64, 64, kernel_size = (3, 3, 3), stride = (1, 1, 1), padding = (1, 1, 1))(norm_layer) : BatchNorm3d(64, eps = 1e−05, momentum = 0.1, affine = True, track_running_stats = True)(activation_layer) : PReLU(num_parameters = 1)(block) : Sequential((0) : Conv3d(64, 64, kernel_size = (3, 3, 3), stride = (1, 1, 1), padding = (1, 1, 1))(1) : BatchNorm3d(64, eps = 1e − 05, momentum = 0.1, affine = True, track_running_stats = True)(2) : PReLU(num_parameters = 1)))), (conv2) : ConvolutionalBlock((conv_layer) :$

$Conv3d(64, 128, kernel_size = (3, 3, 3), stride = (1, 1, 1), padding = (1, 1, 1))(norm_layer) : BatchNorm3d(128, eps = 1e{-}05, momentum = 0.1, affine = True, track_running_stats = True)(activation_layer) : PReLU(num_parameters = 1)(block) : Sequential((0) : Conv3d(64, 128, kernel_size = (3, 3, 3), stride = (1, 1, 1), padding = (1, 1, 1))(1) : BatchNorm3d(128, eps = 1e{-}05, momentum = 0.1, affine = True, track_running_stats = True)(2) : PReLU(num_parameters = 1))(downsample) : MaxPool3d(kernel_size = 2, stride = 2, padding = 0, dilation = 1, ceil_mode = False))$,

- (bottom_block) : $EncodingBlock((conv1) : ConvolutionalBlock((conv_layer) : Conv3d(128, 128, kernel_size = (3, 3, 3), stride = (1, 1, 1), padding = (1, 1, 1))(norm_layer) : BatchNorm3d(128, eps = 1e{-}05, momentum = 0.1, affine = True, track_running_stats = True)(activation_layer) : PReLU(num_parameters = 1)(block) : Sequential((0) : Conv3d(128, 128, kernel_size = (3, 3, 3), stride = (1, 1, 1), padding = (1, 1, 1))(1) : BatchNorm3d(128, eps = 1e{-}05, momentum = 0.1, affine = True, track_running_stats = True)(2) : PReLU(num_parameters = 1))), (conv2) : ConvolutionalBlock((conv_layer) : Conv3d(128, 256, kernel_size = (3, 3, 3), stride = (1, 1, 1), padding = (1, 1, 1))(norm_layer) : BatchNorm3d(256, eps = 1e{-}05, momentum = 0.1, affine = True, track_running_stats = True)(activation_layer) : PReLU(num_parameters = 1)(block) : Sequential((0) : Conv3d(128, 256, kernel_size = (3, 3, 3), stride = (1, 1, 1), padding = (1, 1, 1))(1) : BatchNorm3d(256, eps = 1e{-}05, momentum = 0.1, affine = True, track_running_stats = True)(2) : PReLU(num_parameters = 1)))),$
- (decoder): Decoder($(\text{decoding}_blocks) : ModuleList((0) : DecodingBlock((upsample) : Upsample(scale_factor = 2.0, mode = trilinear)(conv1) : ConvolutionalBlock((conv_layer) : Conv3d(384, 128, kernel_size = (3, 3, 3), stride = (1, 1, 1), padding = (1, 1, 1))(norm_layer) : BatchNorm3d(128, eps = 1e{-}05, momentum = 0.1, affine = True, track_running_stats = True)(activation_layer) : PReLU(num_parameters = 1)(block) : Sequential((0) : Conv3d(384, 128, kernel_size = (3, 3, 3), stride = (1, 1, 1), padding = (1, 1, 1))(1) : BatchNorm3d(128, eps = 1e{-}05, momentum = 0.1, affine = True, track_running_stats = True)(2) : PReLU(num_parameters = 1))), (conv2) : ConvolutionalBlock((conv_layer) : Conv3d(128, 128, kernel_size = (3, 3, 3), stride = (1, 1, 1), padding = (1, 1, 1))(norm_layer) : BatchNorm3d(128, eps = 1e{-}05, momentum = 0.1, affine = True, track_running_stats = True)(activation_layer) : PReLU(num_parameters = 1)(block) : Sequential((0) : Conv3d(128, 128, kernel_size = (3, 3, 3), stride = (1, 1, 1), padding = (1, 1, 1))(1) : BatchNorm3d(128, eps = 1e{-}05, momentum = 0.1, affine = True, track_running_stats = True)(2) : PReLU(num_parameters = 1)))),$
- (1): DecodingBlock((upsample): Upsample(scale_factor = 2.0, mode = trilinear)(conv1) : ConvolutionalBlock((conv_layer) : Conv3d(192, 64, kernel_size = (3, 3, 3), stride = (1, 1, 1), padding = (1, 1, 1))(norm_layer) : BatchNorm3d(64, eps = 1e{-}05, momentum = 0.1, affine = True, track_running_stats = True)(activation_layer) : PReLU(num_parameters = 1)(block) : Sequential((0) : Conv3d(192, 64, kernel_size = (3, 3, 3), stride = (1, 1, 1), padding = (1, 1, 1))(1) : BatchNorm3d(64, eps = 1e{-}05, momentum = 0.1, affine = True, track_running_stats = True)(2) : PReLU(num_parameters = 1))), (conv2) : ConvolutionalBlock((conv_layer) : Conv3d(64, 64, kernel_size = (3, 3, 3), stride = (1, 1, 1), padding = (1, 1, 1))(norm_layer) : BatchNorm3d(64, eps = 1e{-}05, momentum = 0.1, affine = True, track_running_stats = True)(activation_layer) :

$PReLU(num_parameters = 1)(block) : Sequential((0) : Conv3d(64, 64, kernel_size = (3, 3, 3), stride = (1, 1, 1), padding = (1, 1, 1))(1) : BatchNorm3d(64, eps = 1e - 05, momentum = 0.1, affine = True, track_running_stats = True)(2) : PReLU(num_parameters = 1))))$,

- (2): DecodingBlock((upsample): Upsample(scale_factor = 2.0, mode = trilinear)(conv1) : $ConvolutionalBlock((conv_layer) : Conv3d(96, 32, kernel_size = (3, 3, 3), stride = (1, 1, 1), padding = (1, 1, 1))(norm_layer) : BatchNorm3d(32, eps = 1e−05, momentum = 0.1, affine = True, track_running_stats = True)(activation_layer) : PReLU(num_parameters = 1)(block) : Sequential((0) : Conv3d(96, 32, kernel_size = (3, 3, 3), stride = (1, 1, 1), padding = (1, 1, 1))(1) : BatchNorm3d(32, eps = 1e - 05, momentum = 0.1, affine = True, track_running_stats = True)(2) : PReLU(num_parameters = 1)))(conv2) : ConvolutionalBlock((conv_layer) : Conv3d(32, 32, kernel_size = (3, 3, 3), stride = (1, 1, 1), padding = (1, 1, 1))(norm_layer) : BatchNorm3d(32, eps = 1e−05, momentum = 0.1, affine = True, track_running_stats = True)(activation_layer) : PReLU(num_parameters = 1)(block) : Sequential((0) : Conv3d(32, 32, kernel_size = (3, 3, 3), stride = (1, 1, 1), padding = (1, 1, 1))(1) : BatchNorm3d(32, eps = 1e - 05, momentum = 0.1, affine = True, track_running_stats = True)(2) : PReLU(num_parameters = 1))))))$,

- (classifier): ConvolutionalBlock($(\text{conv}_layer) : Conv3d(32, 7, kernel_size = (1, 1, 1), stride = (1, 1, 1))(block) : Sequential((0) : Conv3d(32, 7, kernel_size = (1, 1, 1), stride = (1, 1, 1)))))$,

6 Bayesian 3D U-Net architecture

BCNN Architecture		
Layer (type)- Connected to	Output Shape	Param
input$_1$($InputLayer$)	[(None, 16, 32, 32, 1)]	0
conv3d (Conv3D) input$_1$[0][0]	(None, 16, 32, 32, 16)	448
(GroupNormalization) conv3d[0][0]	(None, 16, 32, 32, 16)	32
conv3d$_1$($Conv3D$)$group_normalization$[0][0]	(None, 16, 32, 32, 16)	6928
group$_normalization_1conv3d_1$[0][0]	(None, 16, 32, 32, 16)	32
max$_pooling3d$($MaxPooling3D$) − $group_normalization_1$[0][0]	(None, 8, 16, 16, 16)	0
conv3d$_2$($Conv3D$) − $max_pooling3d$[0][0]	(None, 8, 16, 16, 32)	13856
group$_normalization_2 - conv3d_2$[0][0]	(GroupNormalization) (None, 8, 16, 16, 32)	64
conv3d$_3$($Conv3D$) − $group_normalization_2$[0][0]	(None, 8, 16, 16, 32)	27680
group$_normalization_3conv3d_3$[0][0]	(GroupNormalizatio (None, 8, 16, 16, 32)	64
max$_pooling3d_1$($MaxPooling3D$)	(None, 4, 8, 8, 32)	0
conv3d$_4$($Conv3D$)$max_pooling3d_1$[0][0]	(None, 4, 8, 8, 64)	55360
group$_normalization_4conv3d_4$[0][0]	(GroupNormalizatio (None, 4, 8, 8, 64)	128
conv3d$_5$($Conv3D$)$group_normalization_4$[0][0]	(None, 4, 8, 8, 64)	110656
group$_normalization_5conv3d_5$[0][0]	(GroupNormalizatio (None, 4, 8, 8, 64)	128
max$_pooling3d_2$($MaxPooling3D$)	(None, 2, 4, 4, 64)	0
conv3d$_6$($Conv3D$)$max_pooling3d_2$[0][0]	(None, 2, 4, 4, 128)	221312
group$_normalization_6conv3d_6$[0][0]	(GroupNormalizatio (None, 2, 4, 4, 128)	256
conv3d$_7$($Conv3D$)$group_normalization_6$[0][0]	(None, 2, 4, 4, 128)	442496
group$_normalization_7conv3d_7$[0][0]	(None, 2, 4, 4, 128)	256
up$_sampling3d$($UpSampling3D$)$groupnorm$	(None, 4, 8, 8, 128)	0
conv3d$_flipout$($Conv3DFlipout$)$upsampling$	(None, 4, 8, 8, 64)	131136
group$_normalization_8conv3d_flipout$[0][0]	(None, 4, 8, 8, 64)	128
concatenate (Concatenate) group norm	(None, 4, 8, 8, 128)	0
group$_normalization_9concatenate$[0][0]	(None, 4, 8, 8, 128)	256
conv3d$_flipout_1$($Conv3DFlipout$)$groupnorm$	(None, 4, 8, 8, 64)	442432
group$_normalization_10conv3d_flipout_1$[0][0]	(None, 4, 8, 8, 64)	128
conv3d$_flipout_2$($Conv3DFlipout$)$groupnorm$	(None, 4, 8, 8, 64)	221248
group$_normalization_11conv3d_flipout_2$[0][0]	(None, 4, 8, 8, 64)	128
up$_sampling3d_1$($UpSampling3D$)$groupnorm$	(None, 8, 16, 16, 64)	0

7 Dice loss implementation

- define get dice score(output, target, epsilon=1e-9):
- p0 = output
- g0 = target
- p1 = 1 - p0
- g1 = 1 - g0
- true positives = (p0 * g0).sum(dim=SPATIAL DIMENSIONS)
- false positives = (p0 * g1).sum(dim=SPATIAL DIMENSIONS)
- false negatives = (p1 * g0).sum(dim=SPATIAL DIMENSIONS)
- num = 2 * tp
- denom = 2 * tp + fp + fn + epsilon
- dice score = num / denom
- return dice score
- define get dice loss(output, target):
- return 1 - get dice score(output, target)

[1]

8 Dice Loss Boxplots Test summary

'stats'- extreme of the lower whisker ,lower quartile, median, upper quartile, extreme of the u pperwhisker [,1] [,2] [,3] [,4] [,5] [,6] [,7] [1,] 0.9939415 0.8534654 0.8440270 0.8254216 0.7361740 0.8931254 0.8271929 [2,] 0.9956477 0.8986921 0.9010764 0.8770568 0.8359140 0.9214303 0.8765577 [3,] 0.9964101 0.9166637 0.9276562 0.9009634 0.8798500 0.9349434 0.8949705 [4,] 0.9968503 0.9303132 0.9391153 0.9116476 0.9028163 0.9411073 0.9100782 [5,] 0.9974957 0.9530730 0.9536969 0.9461945 0.9398804 0.9534671 0.9417938

n- number of observations used in the boxplot [1] 400 400 400 400 400 400 400

conf - 95percent confidence intervals [,1] [,2] [,3] [,4] [,5] [,6] [,7] [1,] 0.9963151 0.9141656 0.9246512 0.8982307 0.8745647 0.9333889 0.8923224 [2,] 0.9965051 0.9191617 0.9306613 0.9036961 0.8851353 0.9364979 0.8976186

outliers in the multiple boxplots [1] 0.9938015 0.9936790 0.9933020 0.9930769 0.9929569 0.9928613 0.9927654 0.9928135 [9] 0.9928683 0.9929227 0.9929967 0.9930399 0.9931296 0.9932365 0.9919433 0.9923072 [17] 0.9925842 0.9928495

[1] https://github.com/fepegar/

0.9929066 0.9921712 0.9920868 0.9918633 0.9931229 0.9922061 [25] 0.9924964 0.9926924 0.9927431 0.9927744 0.9928004 0.9927978 0.9929570 0.9927340 [33] 0.9926295 0.9930643 0.9926038 0.9931678 0.9935325 0.9934789 0.9933372 0.9931152 [41] 0.9917659 0.8395156 0.8456121 0.8048573 0.7172236 0.8194574 0.8076037 0.8186080 [49] 0.8196184 0.8360349 0.7995931 0.8402430 0.8288110 0.8223668 0.8203493 0.8144485 [57] 0.8152423 0.8184811 0.8278652 0.8299409 0.8045140 0.7609734 0.7595184 0.7457486 [65] 0.7298200 0.8406089 0.8337197 0.8365657 0.8269088 0.8271983 0.8337197 0.8365657 [73] 0.8269088 0.8271983 0.8273903 0.8282000 0.7943953 0.7768679 0.7768268 0.7793692 [81] 0.8128873 0.8011124 0.8035258 0.8072216 0.8229240 0.7922801 0.7886497 0.7834506 [89] 0.7767867 0.7804247 0.7828981 0.7881153 0.7967876 0.8045679 0.8065026 0.8069217 [97] 0.8136882 0.8236363 0.8231802 0.8158684 0.8066357 0.8014424 0.7742880 0.8201734 [105] 0.8139590 0.8051679 0.6510267 0.6784387 0.7067615 0.6321207 0.8238150 0.7502786 [113] 0.8090258 0.8195207 0.8217804 0.8250898 0.8156914 0.8067126 0.7922823 0.7828073 [121] 0.7752523 0.7722586 0.7781014 0.7804672 0.7863967 0.7985867 0.8029987 0.7986248 [129] 0.7812890

names [1] "background" "left ventricle" "right ventricle" "left atria" "right atria" "aorta" [7] "pulmonary artery"

9 Dice Loss Boxplots Validation summary

'stats'- extreme of the lower whisker ,lower quartile, median, upper quartile, extreme of the u pperwhisker [,1] [,2] [,3] [,4] [,5] [,6] [,7] [1,] 0.9918898 0.7618353 0.7937404 0.8063110 0.5843010 0.8957418 0.8170857 [2,] 0.9926245 0.8581038 0.8791423 0.8525897 0.7657989 0.9132705 0.8628541 [3,] 0.9947880 0.8932779 0.9216709 0.8902412 0.8464700 0.9410959 0.8970913 [4,] 0.9964923 0.9239588 0.9370760 0.9130630 0.8883794 0.9460282 0.9153634 [5,] 0.9976907 0.9569599 0.9617341 0.9442934 0.9511064 0.9582951 0.9436430

n- number of observations used in the boxplot [1] 465 465 465 465 465 465 465

conf - 95percent confidence intervals [,1] [,2] [,3] [,4] [,5] [,6] [,7] [1,] 0.9945046 0.8884526 0.9174260 0.8858103 0.8374884 0.9386957 0.8932439 [2,] 0.9950714 0.8981031 0.9259157 0.8946721 0.8554516 0.9434961 0.9009387

names [1] "background" "left ventricle" "right ventricle" "left atria" "right atria" "aorta" [7] "pulmonary artery"

10 Dice Loss Boxplots

11 Multi Class 3D U-Net training distributions: examples

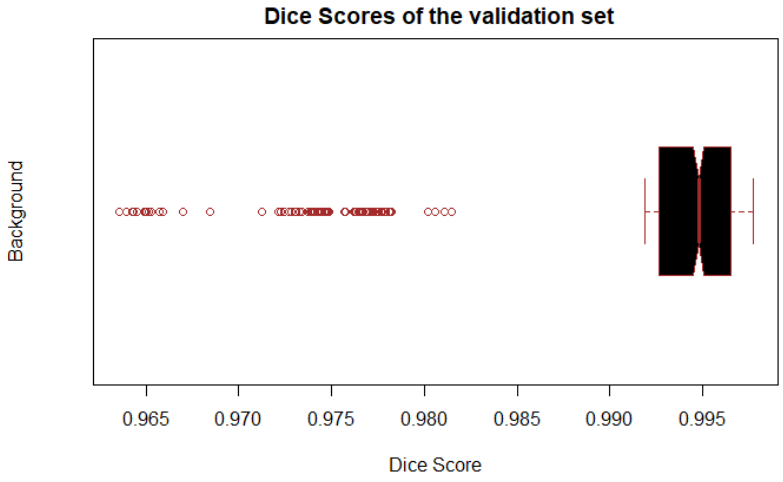

Figure 1: **Boxplot of the Dice Scores of the validation set for each segmentation class label with the same afforementioned legend.**

Figure 2: **Boxplot of the Dice Scores of the test dataset for each segmentation class label with the same afforementioned legend.**

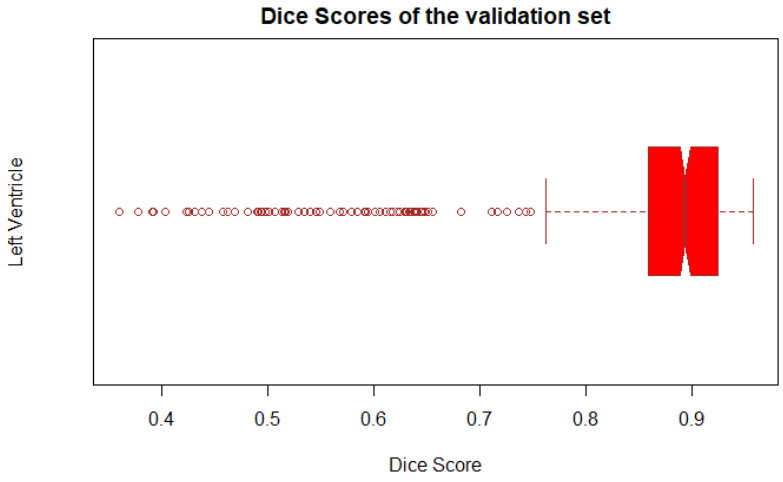

Figure 3: **Boxplot of the Dice Scores of the validation set for each segmentation class label with the same afforementioned legend.**

Figure 4: **Boxplot of the Dice Scores of the test dataset for each segmentation class label with the same afforementioned legend.**

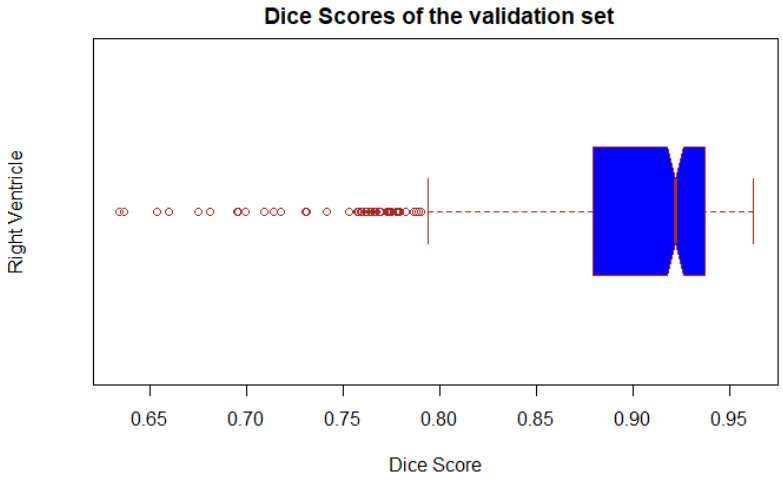

Figure 5: **Boxplot of the Dice Scores of the validation set for each segmentation class label with the same afforementioned legend.**

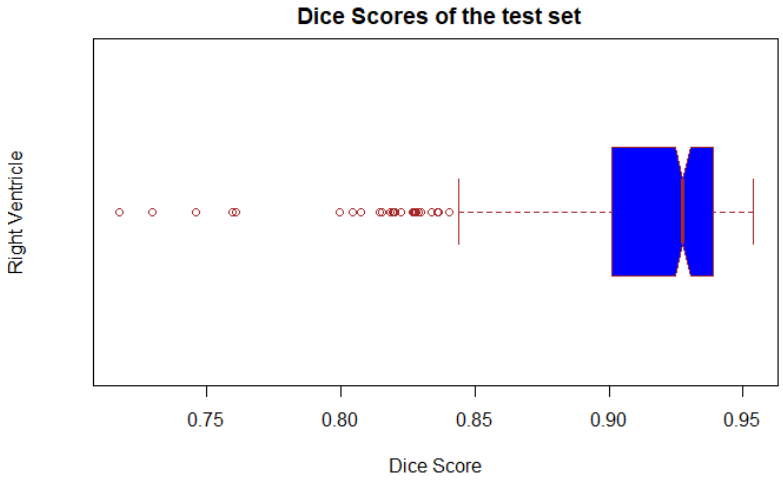

Figure 6: **Boxplot of the Dice Scores of the test dataset for each segmentation class label with the same afforementioned legend.**

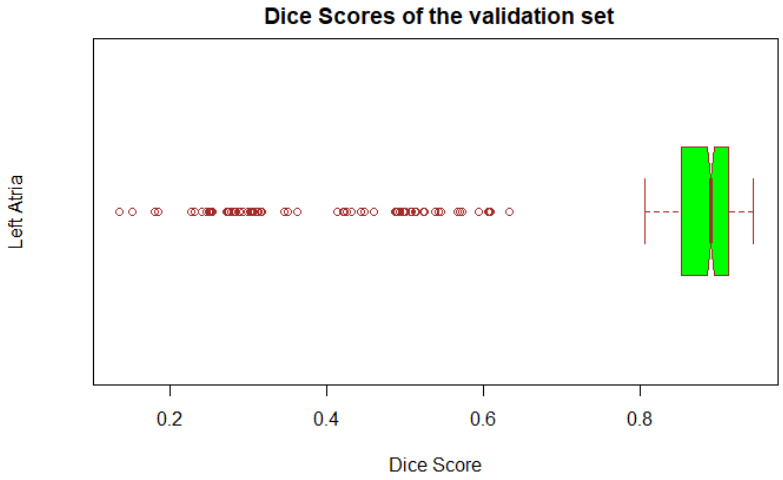

Figure 7: **Boxplot of the Dice Scores of the validation set for each segmentation class label with the same afforementioned legend.**

Figure 8: **Boxplot of the Dice Scores of the test dataset for each segmentation class label with the same afforementioned legend.**

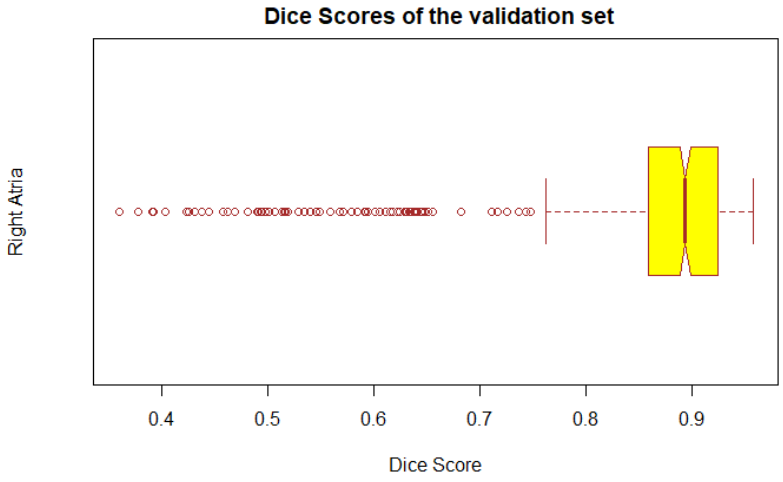

Figure 9: **Boxplot of the Dice Scores of the validation set for each segmentation class label with the same afforementioned legend.**

Figure 10: **Boxplot of the Dice Scores of the test dataset for each segmentation class label with the same afforementioned legend.**

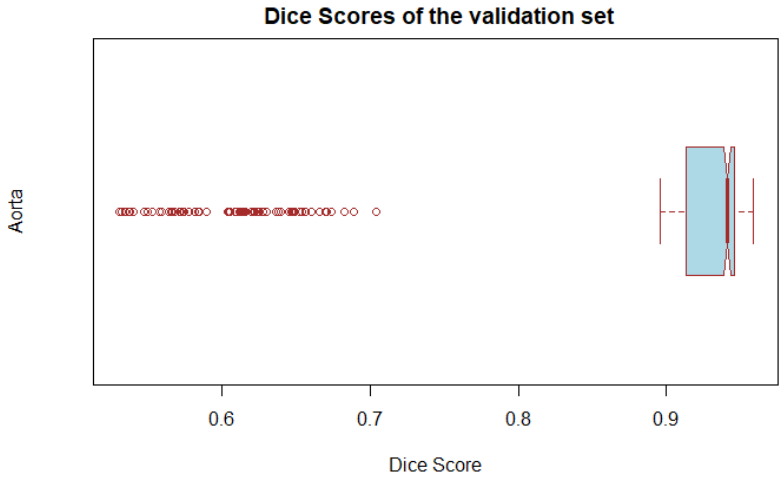

Figure 11: **Boxplot of the Dice Scores of the validation set for each segmentation class label with the same afforementioned legend.**

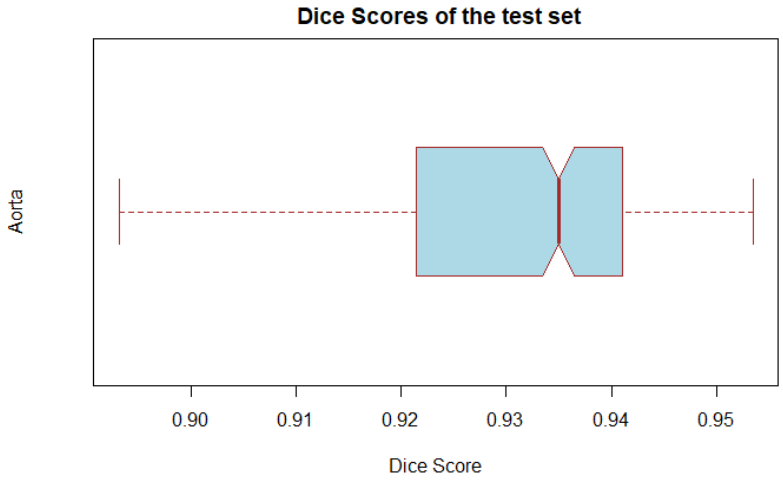

Figure 12: **Boxplot of the Dice Scores of the test dataset for each segmentation class label with the same afforementioned legend.**

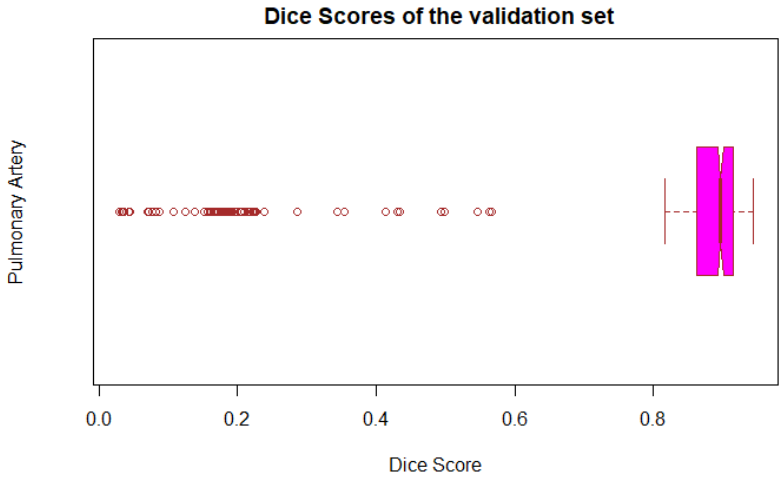

Figure 13: **Boxplot of the Dice Scores of the validation set for each segmentation class label with the same afforementioned legend.**

Figure 14: **Boxplot of the Dice Scores of the test dataset for each segmentation class label with the same afforementioned legend.**

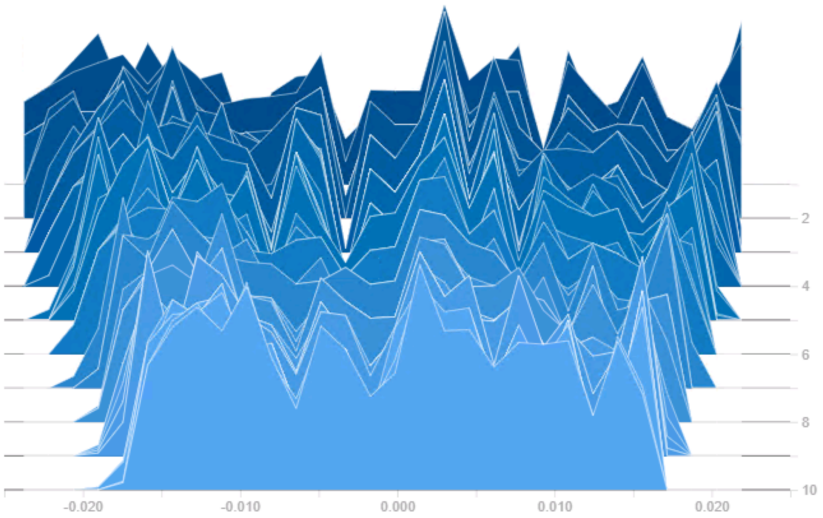

Figure 15: **Bias in the encoder block 2 , final deconvolution layer** .

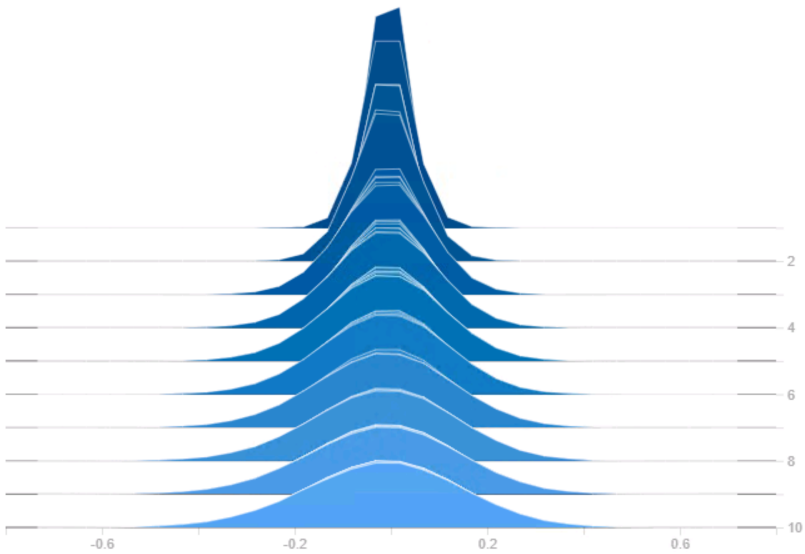

Figure 16: **Weight in the encoder block 2 , final deconvolution layer** .

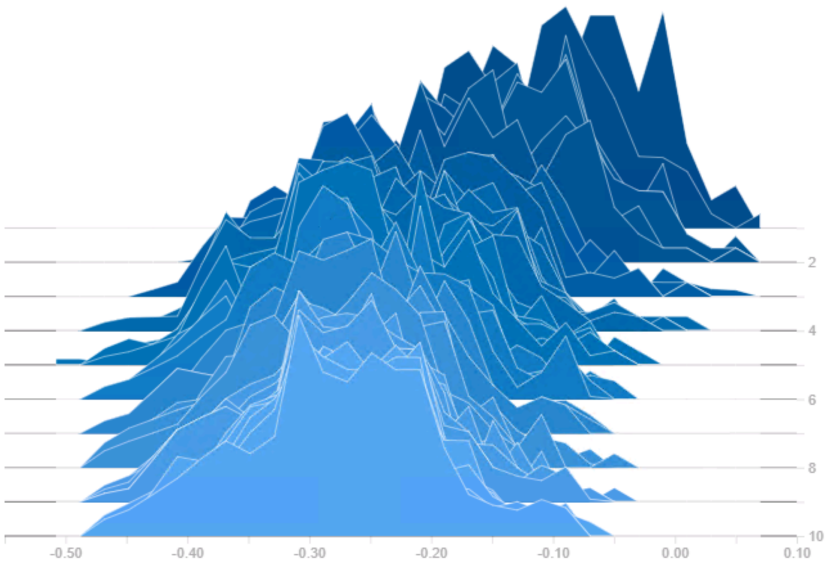

Figure 17: **Bias in the encoder block 2 , final normalisation layer** .

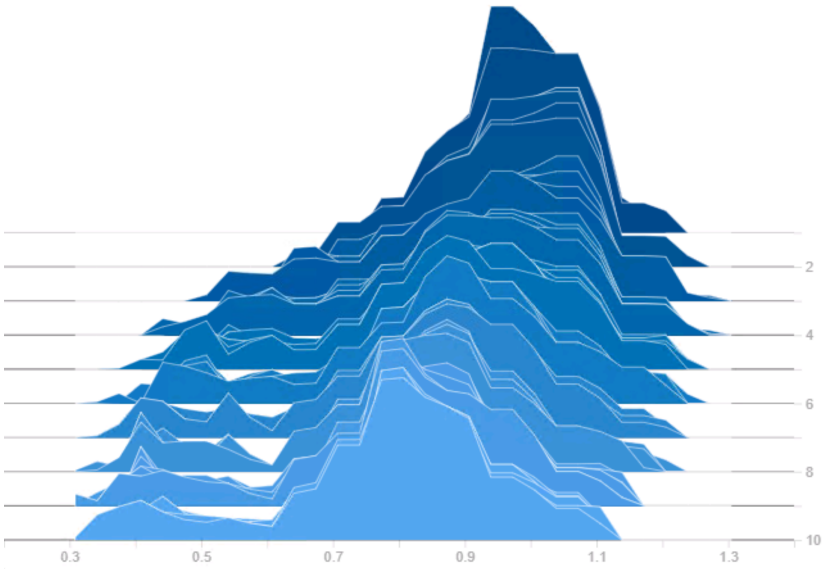

Figure 18: **Weight in the encoder block 2 , final normalisation layer** .