TÉCNICO LISBOA

Implementation of a deep-learning based tool for automatic Cardiac MR planning

DeepCardioPlanner

Pedro Louro Costa Osório

Thesis to obtain the Master of Science Degree in

Biomedical Engineering

Supervisors: Prof. Teresa Matias Correia
Prof. Rita Gouveia Nunes

Examination Committee

Chairperson: Prof. João Miguel Raposo Sanches
Supervisor: Prof. Teresa Matias Correia
Member of the Committee: Dr. Thomas Kuestner

July 2023

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.
The work presented in this thesis was performed at the Institute for Systems and Robotics (Lisbon, Portugal), during the period March 2022 - May 2023, under the supervision of Prof. Rita Gouveia Nunes. The thesis was co-supervised by Prof. Teresa Matias Correia from the Centro de Ciências do Mar, Universidade do Algarve (Faro, Porugal) and Markus Henningson from the Division of Cardiovascular Medicine, Department of Medical and Health Sciences, Linköping University (Linköping, Sweden).

Acknowledgments

I am profoundly grateful to my family and friends for their unwavering support and encouragement throughout my thesis. Their presence, understanding, and care have been instrumental in making this project possible, and their contributions have been invaluable.

I would also like to extend my heartfelt appreciation to my thesis supervisors, Prof. Rita Nunes, Prof. Teresa Correia, and Markus Henningsson. Their guidance and expertise have significantly shaped the outcome of this work, and I am especially grateful to Prof. Teresa Correia for her exceptional support not only in my job search but also in preparing me for the interview that secured my current position.

Additionally, I want to express my gratitude to my supervisors for encouraging me to submit an abstract about this work for the International Society for Magnetic Resonance in Medicine (ISMRM) conference this year. Being selected for a digital poster presentation allowed me to gain valuable experience in academia, even during my Masters degree. Their unwavering support and belief in my abilities have played a pivotal role in my academic and professional growth.

Lastly, I would like to acknowledge the contributions of all my friends and colleagues who have aided in my personal growth and provided consistent support during both the successes and challenges of my life.

To each and every one of you, I extend my deepest gratitude for being an integral part of my journey.

Thank you.

Abstract

Cardiac Magnetic Resonance (CMR) is a powerful technique which can be used to perform a comprehensive cardiac examination. However, its adoption is often limited to specialised centres, in part due to the need for highly trained operators to perform the complex procedures of determining the 4 standard cardiac planes: 2-, 3-, 4-chamber and short axis views. Tools for automating this planning process have been proposed (e.g., Cardiac Dot), but still require some user input. Recently, Deep Learning (DL) methods have been proposed to achieve automatic cardiac planning. For example, cardiac anatomic landmark regression from 2D images has been used to prescribe the standard CMR view planes with good results, but it requires extensive manual annotation to build a dataset to train such methods. Manual annotation free methods have also been proposed for computed tomography (CT), but predict each view position and orientation separately. A similar approach based on rapidly acquired volumetric images could be applicable to CMR where automated view planning would be even more valuable. Here, we propose a set of four deep convolutional neural network (CNN) models (*DeepCardioPlanner*), each trained via a multi-task learning (MTL) approach, to predict the orientation and position of each cardiac view plane from a rapidly acquired 3D scan. This work focus on providing a comprehensive overview of the main steps taken while building the final tool, highlighting the various challenges that arose and how they were dealt with. In particular, we cover how we dealt with the particularities of our dataset, we experiment with different network architectures, loss weighting strategies and hyperparameter tuning. We tested the ability of DeepCardioPlanner to automatically plan the four cardiac views on clinically acquired patient CMR data. Test set error metrics revealed to be comparable with the literature, inclusively matching

the range of values for inter-operator variability.

Keywords

Heart; Deep Learning,; View Planning; MRI

Resumo

A Ressonância Magnética Cardíaca (RMC) é uma técnica poderosa que pode ser utilizada para efetuar um exame cardíaco completo. No entanto, a sua adoção é frequentemente limitada a centros especializados, em parte devido à necessidade de operadores altamente treinados para realizar os procedimentos complexos de determinação dos 4 planos cardíacos padrão: 2, 3, 4 câmaras e eixo curto. Foram propostas ferramentas para automatizar este processo de planeamento (p. ex., Cardiac Dot), mas ainda requerem alguma intervenção do utilizador. Recentemente, foram propostos métodos de aprendizagem profunda (DL) para obter um planeamento cardíaco automático. Por exemplo, a regressão de marcos anatómicos cardíacos a partir de imagens 2D tem sido utilizada para prescrever os planos de visualização de RMC padrão com bons resultados, mas requer uma anotação manual extensa para criar um conjunto de dados para treinar esses métodos. Também foram propostos métodos sem anotação manual para a tomografia computorizada (TC), mas que prevêem a posição e a orientação de cada vista separadamente. Uma abordagem semelhante baseada em imagens volumétricas adquiridas rapidamente poderia ser aplicável à RMC, onde o planeamento automatizado de vistas seria ainda mais valioso. Aqui, propomos um conjunto de quatro modelos de redes neuronais convolucionais profundas (CNN) (*DeepCardioPlanner*), cada um treinado através de uma abordagem de aprendizagem multitarefa (MTL), para prever a orientação e a posição de cada plano de visão cardíaca a partir de um exame 3D rapidamente adquirido. Este trabalho centra-se em fornecer uma visão geral abrangente dos principais passos dados durante a construção da ferramenta final, destacando os vários desafios que surgiram e a forma como foram tratados. Em particular, abordamos a forma como lidámos com as particularidades do nosso conjunto de dados, experimentámos diferentes arquitecturas de rede, estratégias de ponderação de perdas e afinação de hiperparâmetros. Testámos a capacidade do DeepCardioPlanner para planear automaticamente as quatro vistas cardíacas em dados de CMR de doentes adquiridos clinicamente. As métricas de erro do conjunto de testes revelaram-se comparáveis às da literatura, incluindo a gama de valores para a

variabilidade entre operadores.

Palavras Chave

Coração; Aprendizagem Profunda; Planeamento de Vistas; Ressonância Magnética

Contents

List of Figures

List of Tables

1

Introduction

Contents

Cardiovascular diseases (CVDs) are the primary cause of premature death and disability in humans world wide and their incidence is on the rise globally [16]. In fact, the prevalence of CVDs in the last decades has suffered an increase from 271 million cases in 1990 to 523 million in 2019 [17], while it is also estimated that 17.9 million deaths per year are caused by them with associated costs of over $800 billion.

Given these alarming statistics, there is a compelling need to develop and enhance technologies that aim to improve the prevention, diagnosis, and treatment of CVDs. Currently, the evaluation and risk assessment of CVDs rely on various imaging techniques. Among these, Cardiac Magnetic Resonance (CMR) has emerged as the single technology which can be used to perform a comprehensive cardiac examination, namely by assessing ventricular function, cardiac morphology, the vasculature, perfusion, viability, and metabolism [18].

Despite its clear usefulness, its wide spread use is still limited by factors like its cost and complexity. Perhaps the most limiting characteristic of CMR is the fact that it relies on a long and complex manual view planning procedure to only acquire the most clinically relevant views. Acquisition is made in this fashion to minimise examination time and the number of breath holds that the patient is required to do. Breath holding in CMR is done to mitigate the artifacts related to the breathing motion. These target view planes are described in the literature as the standard cardiac double-oblique planes: short axis (SAX), 2-chamber (2CH), 3-chamber (3CH), and 4-chamber (4CH) views. The traditional planning protocol is a multi-step process that encompasses the acquisition of multiple auxiliary planes (scouts and localisers) to inform the prescription of subsequent planes until the target ones are found. Inevitably, all these steps lead to significantly larger scan times when compared to other MRI modalities. Not only that, but these standard views are patient specific thus requiring sufficient anatomic knowledge and technical expertise to adapt to the different cardiac morphology and body shape.

However, there has been an effort to design new solutions to overcome the obstacles risen by this planning process. Tools that can assist the operator during planning, or even automatise the whole process simplify the protocol, reducing the burden on the operator, easing his learning curve and reducing scanner time. The latter comes as a major benefit as it would allow to examine more patients within the same time slot while also minimising patient stress during the procedure. Also, as the complexity of the task diminishes, the scarcity of qualified operators is mitigated, thereby ensuring that even clinics without specialised CMR personnel can benefit from this technique's value. Moreover, removing the burden on the operator minimises inter-operator variability which contributes to more reproducible results and with higher quality. All in all, CMR would be able to fully transition from this complex and time-consuming tool used mainly in large tertiary centres by specialised operators to an easier to use and more widespread technique.

Currently, the most useful tool a technologist has to assist him during planning is the Dot system by Siemens [19], especially its variant designed specifically for CMR [20]. This tool relies on a undisclosed

machine learning algorithm to prescribe the standard CMR view planes in a faster way. Nevertheless, it still requires the user to perform some online decisions that at the end of a full day of scanning can amount to concentration-burnout as well as compromise standardisation and the reproducibility of the method.

Over the last years, there has been some work around automatic view plane prescription that suggests the feasibility of escalating from Cardiac Dot's user aid to something that removes the need for user input completely. Particularly, deep learning (DL) based methods have shown impressive results. For instance, DL-based localisation of key cardiac anatomic landmarks from 2D images has been used to sequentially prescribe the standard CMR view planes with good results [13] but needs extensive manual annotation to create the training dataset. To avoid the latter, others have proposed to base their view plane prediction on 3D acquisitions. Classical computer vision network architectures have been used to both segment the heart and predict the standard views' defining vectors from 3D CT volumes [14]. However, this work addresses each view plane prescription task as 2 independent sub-tasks of plane position and plane orientation prediction. By doing so, it not only fails to recognise and leverage the similarities between the two sub-tasks and how sharing hidden layer parameters could help performance [8] as it also requires the training of more than one model for each view plane prescription, which is resourcefully expensive.

The aim of this thesis is to take a step forward from Cardiac Dot system and develop a deep learning based planning tool for CMR able to perform the planning of the four different cardiac planes without user input. Here, we propose a set of four deep convolutional neural network (CNN) models (*DeepCardioPlanner*), each trained via a multi-task learning approach, to simultaneously predict the orientation and position of each cardiac view plane from a rapidly acquired 3D scan. By using a 3D volume to regress the view planes, the model has access to the amount of global context that is usually needed to prescribe these slices in the clinic and also bypasses the extensive landmark annotation that has been unavoidable in many of the previous works.

The rest of this thesis is outlined as follows: Chapter 2 Covers all the theoretical concepts needed to understand this work, as well as it provides a summary of the current state of the art in automatic CMR planning. In Chapter 3 the baseline performance of our proposed tool is presented, some challenges arising from the dataset in use are identified and then resolved. Chapter 4 includes all the experiments aimed at improving the performance and learning behaviour of said tool, using the 2CH view prediction task as a representative example. In Chapter 5, while still using the 2CH view prediction as a representative task, a new architecture is introduced and its impact in performance is evaluated. Chapter 6 studies how the changes applied to the training setting based on the 2CH view prediction task generalise to the remaining view prediction tasks, while also providing a comparison with the literature. Finally, it is in Chapter 7 that we conclude this work with its main findings, the limitations of our final proposed tool and suggested future work.

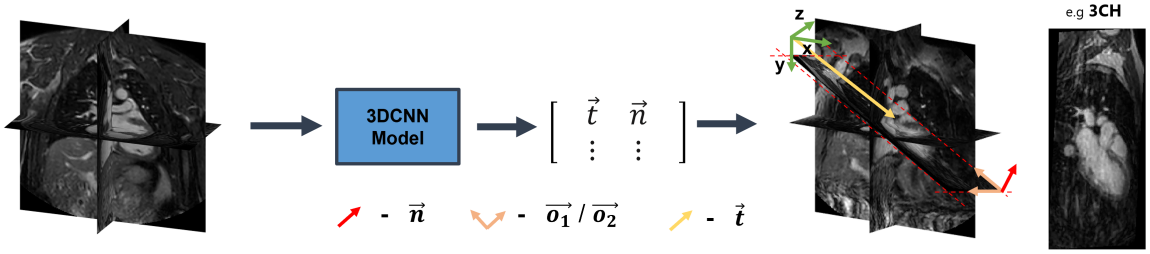


Figure 1.1: Overall scheme of what it is aimed to achieve. The objective is to train a set of 4 3D convolutional neural networks to use a rapidly acquired 3D CMR scout acquisition to predict the vectors that define the position and orientation of each of the standard cardiac view planes: short axis (SAX), 2-chamber (2CH), 3-chamber (3CH, shown in the figure), and 4-chamber (4CH) views. In the training dataset each view plane is defined by two orientation vectors $\overrightarrow{o_1}$ and $\overrightarrow{o_2}$ and position vector $\overrightarrow{t}$, all in the coordinate system of the volume. The model is trained to predict this position vector and $\overrightarrow{n}$, which is the view plane's normal vector, hence perpendicular to both $\overrightarrow{o_1}$ and $\overrightarrow{o_2}$. After model training it is possible to extract these vectors from new never seen volumes with satisfactory accuracy and results comparable to the literature.

1.0.1 Presentations at international conferences

This thesis resulted in a poster and an abstract selected for a power pitch at the 2022 ISMRM Iberian Chapter meeting. Additionally, it also resulted in a digital poster presented at the 2023 ISMRM & ISMRT Annual Meeting & Exhibition, 03-08 June 2023, in Toronto, ON, Canada.

Pedro Osório, Markus Henningsson, Rita G. Nunes, Teresa M. Correia. Implementation of a deep learning based tool for automatic Cardiac MR planning [abstract]. In: 2nd ISMRM Iberian Chapter Meeting 2022, 2022 June 27th-28th, Instituto Superior Técnico, Lisbon, Portugal. Poster P1.9 and Power Pitch.

Pedro Osório, Markus Henningsson, Rita G. Nunes, Teresa M. Correia. DeepCardioPlanner: Implementation of a deep learning based tool for automatic Cardiac MR planning [abstract]. In: 2023 ISMRM & ISMRT Annual Meeting & Exhibition, 03-08 June 2023, in Toronto, ON, Canada. Digital Poster 5134.

2

Background

Contents

This chapter establishes the foundation for our proposed solution by covering essential theoretical concepts and reviewing relevant literature. We begin by highlighting the significance of CMR in cardiac evaluation and examining the physics of CMR, including main modes and principles of magnetic resonance imaging (MRI). Additionally, we discuss CMR planning and its critical aspects. Moving on, we explore machine learning and deep learning, focusing on deep learning fundamentals and the concept of multi-task learning. Lastly, we conduct a comprehensive review of state-of-the-art methods in automatic CMR planning. This groundwork sets the stage for the subsequent chapters, where we present how we have developed our deep learning-based tool for automatic CMR planning.

2.1 Cardiac Magnetic Resonance (CMR)

2.1.1 Why CMR?

Besides CMR, there are various other imaging modalities that can be used perform evaluation and risk assessment of CVDs. In this section we will describe the main ones while also covering their main applications to further understand the added value that CMR has compared to them.

For instance, Computed Tomography (CT) is a non invasive technique with high sensitivity at detecting coronary artery disease (CAD), evaluating pericardial disease, pericardial fluid and neoplasms [21]. In fact, coronary artery calcification is easily detected via this type of imaging and CT angiography has been having an increasing role in CAD detection [22]. Despite CT's wide availability and simple protocol, it does require the use of ionising radiation to obtain the images which is not ideal, thus calling for extra caution to not overexpose individuals to unnecessarily high radiation doses.

Positron Emission Tomography (PET) is another useful technique used detect the uptake of positron-emitting radiotracers in the heart thus allowing the left ventricle volume to be measured in very precise manner as well as the evaluation of myocardial perfusion [21]. Adding to that, PET has been used to predict cardiovascular mortality in individuals with CAD or suspects angina [23]. Notwithstanding, this technique is limited by its high cost, the need for radiotracer administration and the use of ionising radiation. PET is the gold standard for the measurement of myocardial perfusion, but is not ideal for assessing cardiac structures.

The method that is considered first in line examination tool is echocardiography, mostly due to its inexpensiveness and fast acquisition. It provides a reliable, radiation free and non-invasive way to assess pericardial diseases, valve anomalies, and defective ventricular wall motion. Through its Doppler mode, echocardiography can also be used to evaluate valve function, pulmonary pressure, and diastolic left ventricular filling. With Doppler imaging, it is possible to measure myocardial deformations, although, due to the technique's angle dependency, this can only be performed along the ultrasound beam, while the heart deforms in 3D [21]. In addition, its use is limited by a significant operator dependency and the

fact that it is subject to the quality of each patient's acoustic windows.

In this context, cardiac magnetic resonance (CMR) has emerged as the single technology which can be used to perform a comprehensive cardiac examination, namely by assessing ventricular function, cardiac morphology, the vasculature, perfusion, viability, and metabolism [18]. Adding to that, it is able to do all this non-invasively, without using ionising radiation and while providing a high spatial and temporal resolution images. In past years, CMR has started to mature from a tool used exclusively for research in large tertiary centres to what is nowadays considered the gold standard technique for evaluating myocardial function, quantifying myocardial volumes, and detecting myocardial scar [24].

In particular, CMR allows the quantification of left ventricular volume ejection fraction which is a parameter of unmatched importance being used in the assessment of various scenarios from diagnosis of heart failure to determining the need for implantable cardioverters defibrillators and the timing of surgical intervention in patients with valvular heart disease [25]. In comparison with echocardiography, CMR presents itself as a valuable alternative in cases where acoustic windows are poor as it is an magnetic resonance based technology and hence it is not affected by this issue [26]. As we will see in the next sections, instead of relying on sound waves to extract information about the human body, CMR exploits the differences in nuclear magnetic relaxation characteristics of hydrogen nuclei found very widespread around the human body in different chemical environments.

Notwithstanding, there still remain obstacles to the widespread use of CMR. It is an expensive procedure with complex underlying physics and technology which require the tuning of a large number of parameters with an also large number of pulse sequences to choose from. Adding to that, the data generated can be hard to interpret and analyse while also being affected by inherent cardiac and respiratory motion. Perhaps the most relevant hurdle related to CMR is its longer scan times and added complexity stemming from it requiring a manual view planning procedure with an amount of additional planning views and scout acquisitions that is unlike that for any other imaging modality. To further understand the reason behind this planning process, we must first introduce a bit of the physics behind MRI.

2.1.2 Physics of CMR and Main Modes

2.1.2.A Physics of MRI: Magnetisation, Dynamics, Relaxation, Position Encoding and Signal

MRI explores an intrinsic magnetic property of the hydrogen nuclei found all over our bodies. These nuclei possess spin which is an intrinsic quantum mechanical property being described by two discrete set of states, "spin up" ($S = \frac{1}{2}$) or "spin down" ($S = -\frac{1}{2}$). This spin can be described as the angular momentum of a spinning sphere which, when charged, has a magnetic moment (μ) that can be expressed as $\mu = \gamma S$, with γ being the gyromagnetic ratio specific to said nuclei.

In the absence of a strong magnetic field, the magnetic moments of these nuclei are randomly oriented.

However, when a strong magnetic field (B_0) is applied to the system, all the nuclei's magnetic moments will align themselves along the field's direction either on a parallel or anti-parallel fashion, creating a energy gap between the nuclei. This is what is called the Zeeman Effect. Because of the nuclei's spin, their magnetic moment (μ) will not be steadily aligned with the B_0 field but will in fact precess around it. The frequency of this precession is called Larmor Frequnecy, w_L, and will depend linearly on the field's strength:

$$w_L = \gamma B_0 \tag{2.1}$$

When the system is at thermal equilibrium, the amount of nuclei in the spin-up state, meaning parallel to B_0, is slightly larger than the one in the other state. As a result, the net magnetisation ($\vec{M}_0$) will be oriented along it.

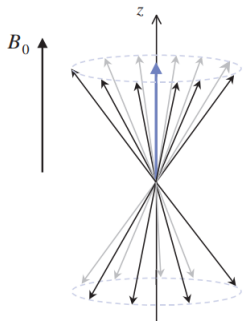


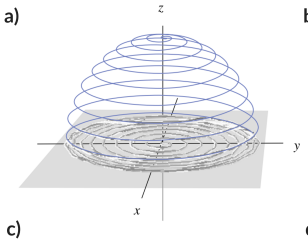
Figure 2.1: Average of many protons produces the net magnetisation $\vec{M}_0$ along the direction of B_0. Extracted from [1].

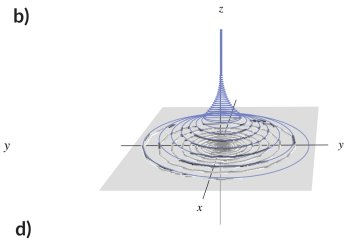
If a resonant magnetic field is applied (B_1), energy is deposited into the spin system and some of the nuclei in the lower energy spin-up state are shifted to the higher energy spin-down state. With this shift, we rotate $\vec{M}$ away from B_0 by an angle that will depend on the strength and duration of said resonant field. For this B_1 field to interfere with the equilibrium in the way explained above, it needs to be applied oscillating at the Larmor frequency through a radio-frequency (RF) pulse.

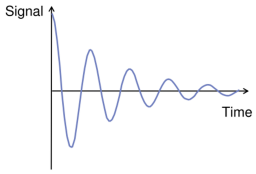
As a convention it is assumed that B_0 is oriented along the z axis, $\vec{M}$ will have components along the z axis called longitudinal magnetisation (M_z) but also along the other directions called transverse magnetisation (M_{xy}). The longitudinal component is reducing during the excitation of the system as the magnetisation is being sent to the transverse plane. At this point, the signal that is acquired is generated by an electrical current induced in a receiver coil by the changing magnetisation of the nuclei in the body during this resonance phenomenon. This coil is sensitive only to magnetisation perpendicular to B_0, i.e. in the transverse plane.

Once the RF pulse is no longer being applied, the perturbed system will relax into the initial thermal equilibrium as nuclei start to shift back to the lower energy state thus dissipating the previously accumu-

lated energy. $\vec{M}$ will rotate back to $\vec{M_0}$ but it will do so at a different pace depending on the component. In fact, the relaxation and regrowth of the longitudinal component is exponential with time constant T_1 while the transverse one is also exponential but with a shorter time constant T_2. This means the decay of the transverse magnetisation is faster than the growth of the longitudinal one. Despite differing, these two relaxation time constants are both affected by the atomic and molecular environment of the nuclei spins, such as type, size, and motion of the particles. As consequence of this, the relaxation time constants will vary from tissue to tissue hence giving us the wanted contrast.







d)

Tissue	T_1(ms) 0.5 T	1.5 T	3.0 T	T_2(ms) 0.5 T	1.5 T	3.0 T
White matter	520[f]	560[a]	832[i]	107[b]	82[c]	110[i]
Grey matter	780[f]	1100[a]	1331[i]	110[b]	92[c]	80[i]
CSF	–	2060[e]	3700	–	–	–
Muscle	560[g]	1075[d]	898[h]	34[g]	33[g]	29[h]
Fat	192[b]	200[b]	382[h]	108[b]	–	68[h]
Liver	395[b]	570[e]	809[h]	96[b]	–	34[h]
Spleen	760[b]	1025[e]	1328[h]	140[b]	–	61[h]

Figure 2.2: (a) During the RF 90°pulse M0 spirals away from the z axis and down towards the transverse plane. (b) After that it relaxes back to its initial state. (c) Signal induced in the receiver coil. (d) Different T1 and T2 relaxation times according to type of tissue and strength of B_0 field. Extracted from [1].

It is now evident what generates the signal in MRI; however, the means by which positional information is obtained from this signal is still yet to explain. This is achieved through various gradients that are applied to modify the amplitude of the resonance frequency linearly as a function of location. The slice selection gradient is applied to vary the field's strength along the direction the slicing is wanted. Assuming a linear gradient, nuclei in different slice planes will be precessing around the local field at a frequency shifted from the central w_L by an amount proportional to their position along the slice selection direction. It is possible to selectively excite the nuclei within only one slice by tuning the RF pulse to the desired precessing frequency interval. After this gradient is turned off and the slice is excited, the nuclei's precessing frequencies return to initial one but their phases are left shifted as they had been precessing at different rates. An important refocusing gradient needs to be employed to synchronise their phases and hence return to the starting situation.

Within a slice, the spatial location of the signal is acquired through a phase encoding step in one direction and an in-plane frequency encoding step in the other one. Similarly to before, during phase encoding, a gradient is temporarily applied shifting the precessing frequencies of the nuclei along one

direction. Once the gradient is turned off, however, no refocusing gradient is employed and the nuclei are left with the phase they have accumulated. It is by analysing these shifts in phase that we are able to retrieve position information. The remaining spatial direction is extracted through the application of a final gradient along it, much like during slice selection but during the read-out, i.e. while the signal in the coil is being captured. This way each frequency component of the acquired signal will be connected to a position along this direction. It is relevant to note that it is the phase encoding step that determines the scanning time since only one phase shift can be accomplished per applied gradient. Consequently, to gather signal from various positions along this axis, the same procedure needs to be repeated multiple times for different gradient intensities. By contrast, one applied frequency encoding gradient during readout is enough to create frequency spectrum. See Figure 2.3 for a visual scheme of acquisition sequence.

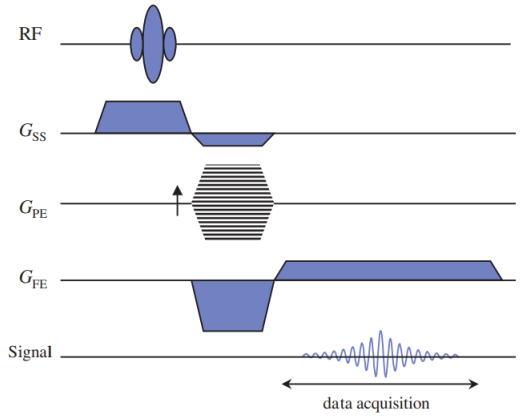


Figure 2.3: Basic MR imaging sequence. G_{ss} is the slice selection gradient, G_{PE} is the phase encoding gradient and G_{FE} is the frequency encoding gradient. Gradient amplitude is shown vertically, time horizontally. Note that for gathering the multiple points along the phase encoding direction this whole sequence must be performed multiple times for different G_{pe} intensities. Adapted from [1].

Considering that the signal results from the sum of the magnetisation of the various nuclei and that the gradients encode positional information in frequency, it is possible to show that the signal in the time domain and the magnetisation in the spatial domain are related through a Fourier Transform (FT). In fact, the MRI signal can be represented in the so called k-space where every point relates to a single spatial frequency and the data in actual image space can be computed via a 2D FT, where the spatial locations are correctly represented (see Figure 2.4).

2.1.2.B Relevant CMR Sequences

By tuning the strength and timing of the gradients and RF pulses we obtain MRI sequences with different speeds of acquisition, type of contrast obtained and other encoded information. In order to understand the main sequences of CMR it is important that some background is provided on the main building blocks

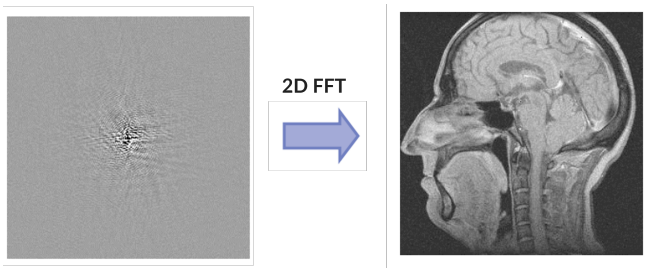


Figure 2.4: The k-space data on the left can be converted into the image domain by 2D FFT. Adapted from [1].

of a MRI sequence.

During relaxation, transverse magnetisation is subject to a extra dephasing phenomenon stemming from inhomogeneity of the main magnetic field which results in a shorter decay time. So the observed decay does not follow the time constant T_2 but a shorter time constant usually denoted as T_2^*. However, we can recover part of this transverse magnetisation through a Spin Echo sequence (SE). Since these B_0 inhomogeneities do not change through time, by applying a refocusing RF pulse we can flip the direction in which the phase is accumulated making it so that after some time all the dephasing is rephased. That is the moment the readout is done, capturing a signal unaffected by this extra dephasing phenomenon.

The Balanced Steady-State Free Precession (bSSFP) sequence is a popular choice in CMR. It is characterised by short intervals between excitations (TR) which never allow the full recovery of the longitudinal magnetisation. At each RF pulse there will be already some residual magnetisation on the transverse plane, contributing for a stronger signal and thus an improved signal to noise ratio (SNR). Just like in SE, this sequence is also not subject to T_2^* decay as the RF pulses are applied in such way that the flip angles induced on $\vec{M}$ are alternated between α and $-\alpha$. RF pulse in bSSFP sequences have then both exciting and refocusing tasks, cancelling out the phase accumulated due to field inhomogeneities. This type of sequence is commonly used in CMR as it provides a good contrast between the myocardium and blood [27]. Adding to that, the fact that TR is very short allows the fast acquisition of slices with good SNR in multiple cardiac phases. In Cine bSSFP sequences, these multiple slices are used to reconstruct movies of the beating heart. To further ensure the quality of these frames, a the acquisition of these images is segmented throughout the different cardiac cycles, thus requiring ECG gating.

It is common to use a gadolinium based contrast agent in CMR imaging as it enhances tissue contrast through T_1 shortening. However, when exploring the contrast between healthy and pathological tissue we must make use of the their difference in contrast wash-out rate rather than T_1 time constant which is shortened in both. Late gadolinium enhancement (LGE) imaging is a tissue nulling technique conceived to null the signal from the healthy tissue devoid of contrast and maximise the one from the contrast retaining diseased tissue. An inversion pulse is applied and the different tissues' longitudinal magnetisation starts recovering at different paces according to their T_1. The shortened T1 in pathological tissue due to the contrast agent allows the magnetisation to recover faster in it than in the healthy tissue. By the time

healthy tissue's M_Z recovers up until 0 (inversion time, IT), the diseased one will already have some magnetisation accumulated. It is in this moment that the excitation RF pulse is applied thereby nulling healthy tissue's contribution to the signal as, contrarily to the diseased tissue, no magnetisation is sent from it to the transverse plane.

2.1.3 CMR Planning

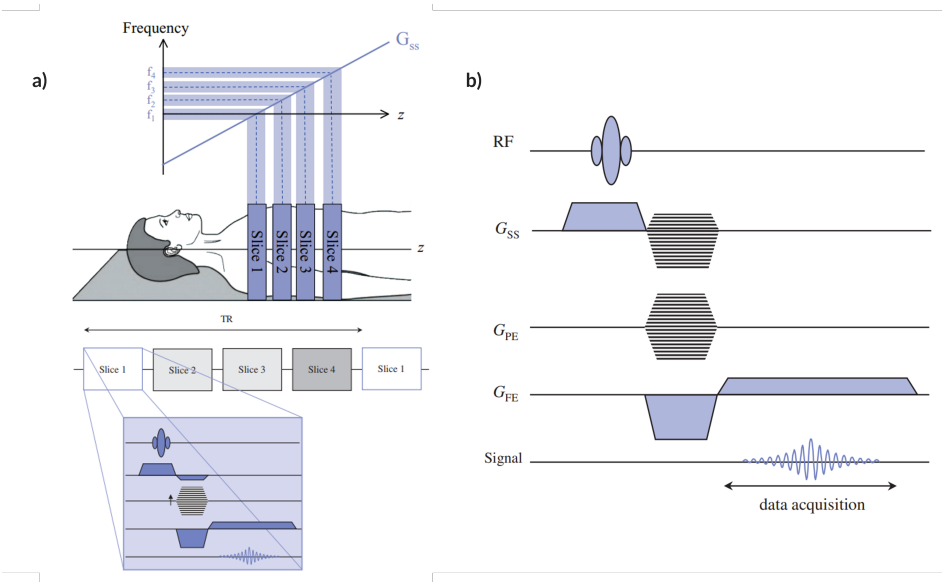


Figure 2.5: Basic 3D MR imaging sequence. (a) 2D multiple slice acquisition (b) True 3D imaging G_{SS} is the slice selection gradient, G_{PE} is the phase encoding gradient and G_{FE} is the frequency encoding gradient. Gradient amplitude is shown vertically, time horizontally. Note that for gathering the multiple points along the phase encoding direction this whole sequence must be performed multiple times for different G_{pe} intensities. Also, for gathering multiple points along the slice selection direction this sequence must run for multiple times for different phase encoding G_{SS} intensities. Extracted from [1].

We have seen how it is possible to acquire the MRI data within a slice, through an initial slice selection step followed by an in-plane phase encoding step and finally a frequency encoding step. Notwithstanding, it is possible to further explore the third dimension through the 2 different ways illustrated in Figure 2.5. One would be to separately acquire multiple contiguous slices along the slice selection axis via the approach explain earlier. Here, the dead time between excitation and read-out could be leveraged to acquire information from multiple different slices simultaneously. Alternatively, an extra phase-encoding step along the slice selection axis can be added to image sequence, yielding a 3D volume with higher SNR, comprised of a larger amount of thinner slices. In fact, it would be ideal to always obtain a 3D volume as they provide much more insight into the patient's body removing the need to meticulously plan and acquire only the most relevant slices for the issue at hand. However, this is not always the best option as increasing the number of slices acquired via either of these methods results in a significant increase of

scan time. Given that MRI protocols are already time-consuming, it is generally advisable to minimise any further increase of examination time. Particularly in the case of CMR, breath-holding techniques are commonly used in 2D acquisitions to hinder the degradation effect of respiratory motion on the image quality and they would simply not be possible to employ in longer 3D acquisitions.

With that in mind, in CMR imaging the approach usually taken is to acquire only the most clinically relevant slices as opposed to a whole volume. The main slices of interest are the standard cardiac double-oblique view planes: short axis (SAX), 2-chamber (2CH), 3-chamber (3CH), and 4-chamber (4CH) views (see Figure 2.7).

These views are traditionally prescribed manually through a multi-step protocol where multiple localiser images are acquired to inform the prescription of subsequent planes. In particular, these localiser images serve as auxiliary acquisitions to provide context about the cardiac morphology of the subject and help guide the operator towards the correct orientation of the target double oblique view planes. Knowledge about the relative spatial location and orientation of the target planes is also leveraged during this process, and some target views can serve as starting points to predict the remaining ones [28].

The SAX view plane should be placed orthogonal to the long axis of the heart, which connects the cardiac apex to the mitral valve. This plane should section the heart at the level of the mid left ventricle. The 4CH view should be perpendicular to the SAX plane while also dissecting the right ventricle (RV) at its maximal lateral dimension and the left ventricle (LV) below the antero-lateral papillary muscle. A well prescribed 4CH view plane will not only show all 4 chambers of the heart but also demonstrate the mitral and tricuspid valves and the right and left atria and ventricles [25]. In turn, the 2CH view plane should be orthogonal to the previous 4CH view plane while simultaneously dissecting the inferior and anterior LV wall parallel to the septum. The left atria and ventricle should be visible in this view. Finally, the 3CH view should dissect the aortic valve and posterior wall while also being orthogonal to either 2CH or 4CH views and to the SAX vies [3]. See Figures 2.6 and 2.7 for visual references.

Due to variability of cardiac morphology and human body shape, the location and orientation of these planes can vary greatly between patients which contributes with added complexity to the task. Consequently, CMR is limited to operators with a sound knowledge of cardiac anatomy and extensive technical expertise. Scan times are typically longer compared to other MRI modalities due to the complexity and multi-step nature of the process, and for the same reason it requires longer training and practise to reach proficiency at [25]. Moreover, since this planning protocol requires the operator to perform multiple sequential decisions, there is potential for some subjectivity to be introduced in the view planning process, leading to operator related variability and hence decrease reproducibility. Naturally, all these factors limit the more wide spread use of this valuable technique.

Scan time, task complexity, inter-operator variability and operator reliance could all be minimised if cardiac view planning in CMR could be performed in a more automatic and self-driven fashion. This

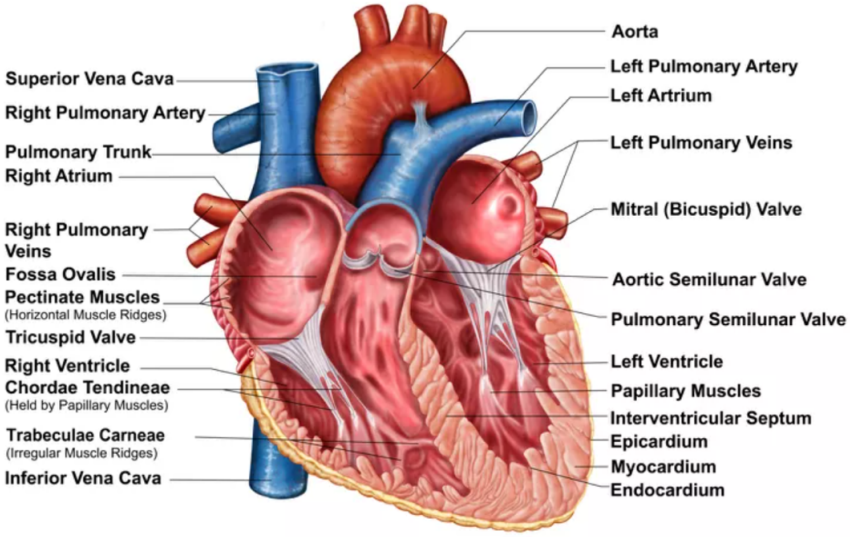


Figure 2.6: Scheme describing the structure of the heart, including the landmarks relevant for cardiac plane prescription. Image extracted from [2].

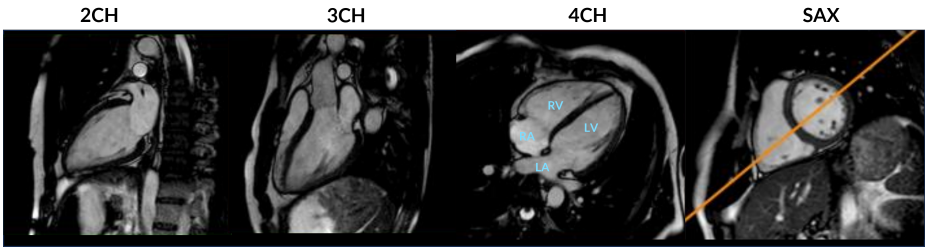


Figure 2.7: 2CH, 3CH, 4CH and SAX view planes. The 4CH view is annotated with location of the cardiac chambers. The tricuspid and mitral valve cannot be very clearly seen in this image, meaning that this 4CH view has not been optimally prescribed. Views extracted from [3].

way the barriers preventing a more wide spread use of CMR would be effectively removed allowing more clinics and patients to benefit from this valuable technique.

2.2 Machine Learning and Deep Learning

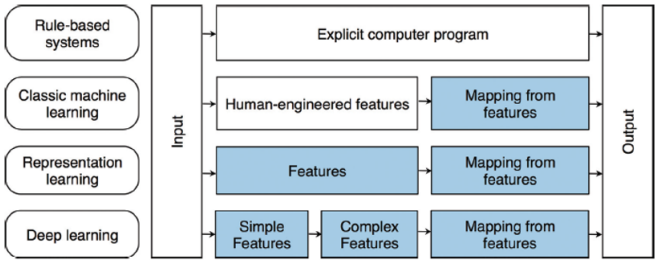


Figure 2.8: Overview of the pipeline to address a certain task according to type of ML approach and compared to explicit task programming. Blue boxes represent components learned by fitting a model to example data. Extracted from [4].

Machine learning is a sub-field of artificial intelligence that focuses on training algorithms to perform tasks by learning patterns from data instead of relying on explicit programming. In traditional machine learning, human experts identify and encode distinctive features in the data, using statistical techniques to organise and analyse the data based on these features. However, for complex computer vision tasks, it is often challenging for experts to determine the optimal image features for machine learning algorithms. In such cases, it is desirable for a computer system not only to learn the mappings of features to desired outputs but also to learn and optimise the features themselves [4].

Representation learning is a type of machine learning in which no feature engineering is used. Rather, these type of algorithms learn features directly from the data, enabling them to automatically capture the complex patterns and dependencies that are most useful for the task at hand. With enough training examples, a system based on representation learning could potentially classify data better than with handcrafted features. The challenge is in how a machine learning system could learn potentially complex features directly from raw data [4].

Deep learning has recently emerged as the most successful representation learning approach with remarkable results in a variety of different tasks from image classification speech recognition, natural language processing, and playing games. It takes advantage of large quantities of data and flexible hierarchical models to learn a composition of features that reflect a hierarchy of structures in the data. These deep learning systems consist of an end-to-end strategy wherein the basic features like signal intensity, edges, and textures are learned as components of more intricate features such as shapes, lesions, or organs. This way, the compositional nature of images is effectively leveraged [4]. The inner workings of such models will be covered in a later section.

First we must delve deeper into what is meant by learning, describing its main ingredients. As defined by [29], "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.". In ML the task, T, is the specific problem that is aimed to address and it can vary widely including classification, denoising, regression, among many others. Classification involves the assignment of input data into one of several predefined categories, while denoising aims to reconstruct clean data from corrupted input samples. Finally, regression is the process of process of predicting a numerical value from a set of input variables.

To effectively assess the capabilities of a machine learning algorithm, it is important to establish a quantitative measure of its performance. This performance measure, denoted as P, is tailored to the specific task T being performed by the system. It is essential for performance measures to accurately quantify the desired behaviour of the system. Of particular interest is the algorithm's performance on previously unseen data, as this indicates its capability in real-world deployment. To gauge this generalisation ability, performance measures are evaluated using a separate test set, distinct from the training data used to train the machine learning system.

As for the type of experiences a model can learn from, we can broadly categorise ML approaches into supervised and unsupervised learning. In supervised learning, the model learns from labelled examples, where input-output pairs are provided during training. This enables the model to make predictions based on known patterns in the data. In contrast, unsupervised learning involves exploring unlabelled data to discover underlying patterns or structures, often through techniques like clustering or dimensionality reduction [7].

In the context of this work, the task we sought to address consists of a regression task where vector coordinates are predicted from input volumetric data. This is achieved via a supervised learning approach where a model is trained on vector labelled input volumes and performance is assessed on multiple hold out sets via performance metrics that we will describe in the next chapter.

2.2.1 Deep Learning Basics

2.2.1.A Neural Networks and Training

The main type of models in DL are neural networks (NN), which are inspired by the structure and functioning of the human brain.

A NN can be defined as a computational model composed of interconnected nodes, referred to as artificial neurons. The structure of a NN is typically organised into layers of neurons. The input layer receives the initial data, which is then propagated through multiple hidden layers before reaching the output layer. The hidden layers serve as intermediate processing stages, enabling the network to extract complex patterns and representations from the input data.

On a closer look, each artificial neuron receives input signals, processes them through a set of weights and biases, applies an activation function, and generates an output signal (y) that is fed to the ones in the next layer. Their operation can be described via the following equation:

$$y = F(\sum_{i=1}^{N} w_i \cdot x_i + b) \tag{2.2}$$

, where w_i and b corresponds to weights and bias of the neuron, x_i is one input feature, N is the number of inputs and F is the activation function.

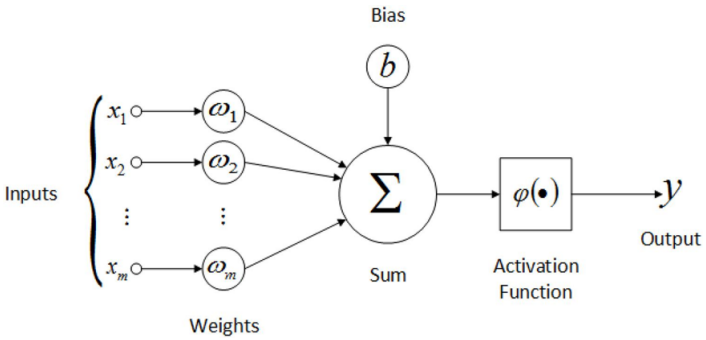


Figure 2.9: Scheme of a single artificial neuron. The weights are depicted w_i and the bias by b. The activation function is here represented as φ and the output by y. Extracted from [5].

The weights and biases determine the strength and relevance of each of the input signals to the final output. Subsequently, this output is input to an activation function that will determine whether the neuron will be activated or not. This last stage is a fundamental step that conveys non-linearity to NNs and thus enables them to model complex non-linear real-world phenomena. There are various types of activation functions that can be used but by far the most common ones are Rectified Linear Unit (ReLU), sigmoid and hyperbolic tangent (tanh) (see Figure 2.10). Sigmoid and Tanh activation functions are usually used at the output layer of the network in situations when the output is expected to be constrained within a specific range, in particular between 0 and 1 for the sigmoid and -1 to 1 for the tanh. Instead, ReLU simply nulls inputs smaller than zero and lets throught positive ones unchanged. It is usually used more extensively throughout the network due to its ability to to speed up convergence and address the vanishing gradient problem which will describe later.

Training a NN involves iteratively adjusting the weights and biases from its every layer to optimise its output predictions with respect to a specific task. It can be described in a loop that has the following steps:

1. **Forward Pass**: The first step in this process consists of passing the input data through the network's layers outputting a prediction via the mechanisms described earlier.

2. **Loss calculation**: This prediction is compared to the target label associated with the current input

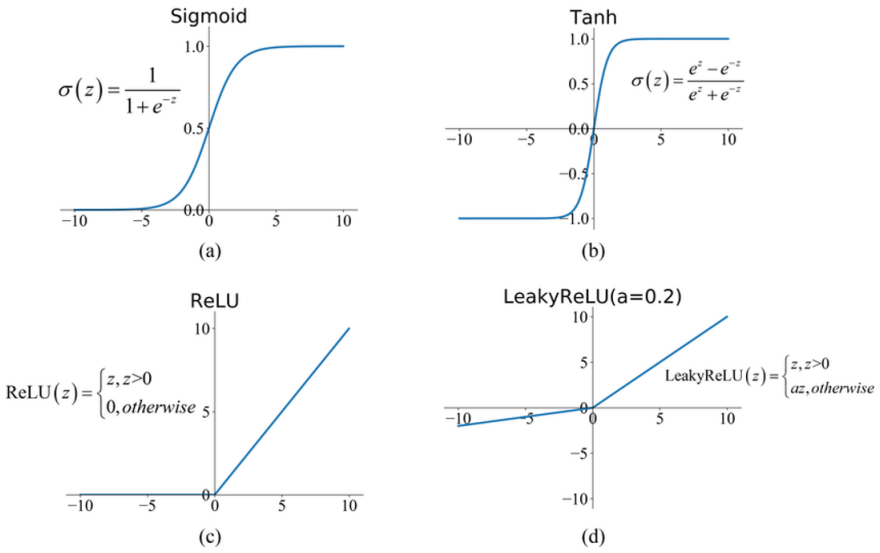


Figure 2.10: Commonly used activation functions: (a) Sigmoid, (b) Tanh, (c) ReLU, and (d) Leaky ReLU. Extracted from [6].

sample using a **loss function**. The loss function measures the discrepancy between the predicted and actual values (loss value) and provides a measure of how well the network is performing.

3. **Backpropagation**: Backpropagation is used to compute the gradients of the loss function with respect to the network's weights and biases. The gradients quantify how much each weight and bias contributes to the loss, allowing the network to adjust its parameters accordingly.

4. **Weight update**: Based on these gradients, optimiser algorithms perform the update of the network parameters (i.e. weights and biases) in the direction that minimises the loss function. The size of each updating step is controlled by the learning rate, which determines the magnitude of the parameter updates. ADAM is a very popular option as an optimiser algorithm and we employ it in this work. It uses adaptive learning rates for each parameter, which are dynamically adjusted based on the history of gradients. Consequently, this optimiser leads to faster convergence and better handling of different scales in the gradients [30].

The forward pass, loss calculation, backpropagation, and label updating steps are repeated iteratively for multiple epochs. Each epoch represents a complete pass through the entire training dataset and as training proceeds the model will gradually start providing better predictions.

Rather than one at the time, input samples are usually fed by batches to the model during training. The weight updates are then based on the averaged gradient within the batch, which reduces noise in the gradient estimates, thereby allowing convergence towards a more reliable and robust solution.

One issue that arises during training and that has great potential for hampering the learning ability of NNs is the vanishing gradient problem. This problem occurs when the gradients computed during

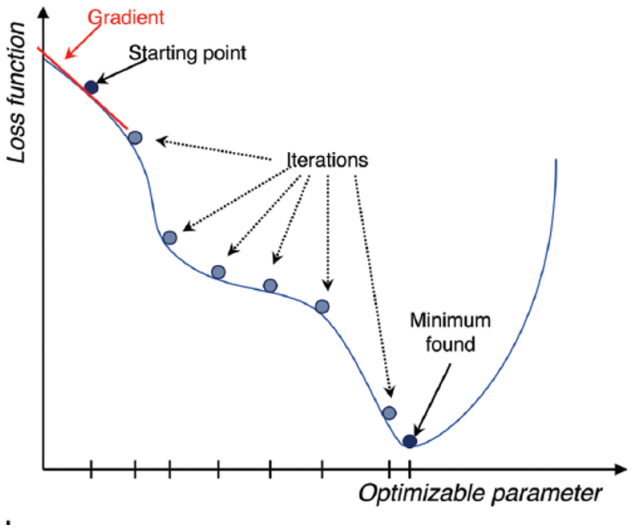


Figure 2.11: Simplified 1D representation of the parameter space of a NN and optimisation path from initial point to optimal solution. Adapted from [4].

backpropagation decay exponentially as they propagate through the network, resulting in increasingly small weight updates that in the end lead to slow convergence or even complete stagnation of the learning process. This effect is particularly visible in deeper networks with a large number of layers. Careful initialisation of the network's weights can minimise this problem, being Xavier and He initialisation strategies the most popular choices to address that.

Additionally, the previously mentioned ReLu activation function also plays an important part in avoiding the vanishing gradient phenomenon. By allowing positive values to pass through unchanged, it allows to maintain strong gradients for positive activations, promoting the flow of gradients during backpropagation. It is worth noting that ReLU is not without its limitations. One potential issue is the "dying ReLU" problem, where neurons can become permanently inactive and output zero for any input. This phenomenon can occur when the weights associated with a neuron are adjusted in such a way that the neuron never activates during training, which is not ideal. For that reason, some variants of the function have been proposed. From among them, we highlight the Leaky ReLU one, which we utilise in our work (see Figure 2.10). It introduces a small slope for negative inputs instead of entirely setting them to zero. By allowing a small, non-zero output for negative inputs, Leaky ReLU ensures that the gradients still flow through these neurons, preventing them from becoming completely dormant.

Data preprocessing is often implemented before training to improve convergence and enable the network to learn more effectively. In particular, scaling the input data to have mean 0 and unitary variance usually helps avoid early saturation.

The choice and design of the loss function play a significant role in the success of training a neural network. Usually in regression tasks the simplest possible loss function is the mean squared error

(MSE). However, it is often beneficial to tailor this function to the specific problem at hand, perhaps even leveraging domain knowledge. A well-designed loss function provides meaningful gradients during backpropagation, enabling the network to learn and adjust its parameters more effectively. We will experiment with that during this work.

While still on the topic of loss functions, it is relevant to note that it is sometimes useful to introduce some constraints to the loss function that help guide the optimisation process towards a particular type of solution. For instance, by adding a parcel to the loss based on the absolute value of the model weights, the minimisation of this new objective will lead to a final solution with lower value weights. This process is called weight regularisation and can be used to achieve better performances. L2 and L1 are the two main approaches to this differing in how the penalty term is computed. L1 relies on the absolute value of the weights, whereas L2 relies on the their squared value. We will revisit this concept in the next section after introducing some new concepts.

Besides the parameters of the NN that are learned during the training process explained above, there are multiple others that are not optimised during said process. These are called hyperparameters and play a big part in multiple aspects of the learning process in NNs and their tuning is of great importance. As examples we can highlight the number of neurons used in the network, number of neurons per layer and optimiser parameters like the learning rate, among others. We will delve into the effect of some of them later on in the next section.

2.2.1.B Overfitting, Underfitting and Model Capacity

The main goal in ML is that our model must perform well not only on the data used for training but also on new, previously unseen inputs. The ability to do so is generally described as the generalisation ability of the model. In order to evaluate it is customary to split our data into 3 different subsets: the training, validation and test subsets. The training subset is used for optimising the model and it should be the largest to maximise the number of examples the model sees. The validation set is used for model evaluation and hyperparameter tuning, providing an unbiased assessment of the model's performance during training. The test set is a separate subset of the data that is kept untouched until the final evaluation stage, acting as realistic simulation of the real-world scenarios the model will encounter. The error on subsets of the dataset other than the training one can be generally addressed by **generalisation error**.

Since the parameters of a model are learned based on a training subset, it is expectable that the model performance on this split of the data will be better than when testing on the others. However, we still expect these other performances to be close to each other as all subsets were sampled from the same underlying data distribution. Given this we can formally define the two factors that determine how well a ML, and hence a DL, model performs: 1. Its ability to make training error small ; 2. Its ability to

minimise the gap between the train and the generalisation error.

This leads us to 2 key concepts that we will keep revisiting throughout this work which is underfitting and overfitting. **Underfitting** happens when the model is not able to obtain low errors neither on the train nor test data. On the other hand, **overfitting** is when the gap between the train and test errors is too large. It is possible to control the likelihood of a model to overfit or underfit by tuning its **capacity**. Here, model capacity refers to the ability of the model to fit multiple functions or, in other words how big its **hypothesis space** is. In general, it can be controlled by the number of parameters of the model in use, or its complexity. In the context of DL, the capacity of a NN model can be determined by its architecture. Architectures with more layers will have more neurons, meaning more learnable parameters and hence larger complexity and capacity.

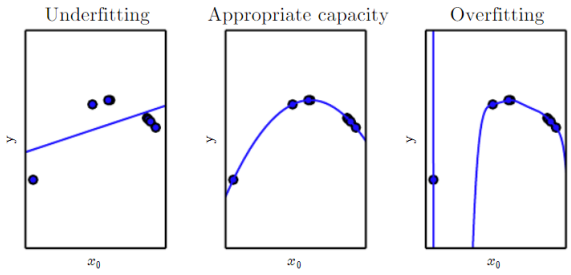


Figure 2.12: We fit three models to this example training set. The training data was generated synthetically, by randomly sampling values and choosing y deterministically by evaluating a quadratic function. (Left) A linear function fit to the data suffers from underfitting, as it cannot capture the curvature that is present in the data. (Centre) A quadratic function fit to the data generalises well to unseen points. It does not suffer from a significant amount of overfitting or underfitting. (Right) A polynomial of degree 9 fit to the data suffers from overfitting. Extracted from [7].

To get a better intuition behind how the model capacity affects the ability of a model to fit a dataset let us look at the example from Figure 2.12. Here, models with different capacity levels are fit to a small subset of points from a quadratic function.

The linear regressor on the left is not able to provide a good fit to the training data and certainly will not do a good job of estimating the values of other unseen points from the quadratic function. Considering the linear regressor is basically fitting a a polynomial of 1st degree to the data, its hypothesis space is limited to linear functions and thus will never succeed in modelling data coming from a 2nd degree polynomial function. Using the vocabulary introduced above, this two parameter regressor does not carry enough complexity, and thus capacity, to appropriately model the data from this distribution. In practise, this model underfits.

The right most subplot depicts the fit of a regressor based on a 9th degree polynomial function with 10 parameters. Contrarily to before, its hypothesis space includes the actual function underlying the training data, however, it is also able to represent many more different and higher degree functions, that also perfectly fit the training data. Consequently, it is harder to navigate through this space and it is

more likely to find suboptimal solutions that fit well the training data but will not generalise for unseen data points. The capacity of this model overshoots the complexity of the distribution underlying the training data and thus it overfits.

This phenomenon of overfitting is also aggravated by the lack of data. In theory, had we enough data to completely represent the full spectrum of values from a quadratic function, even a very high capacity model would be able to fit the data a produce somewhat of a good estimation of the underlying function. This would apply also to the real-world problems that we aim to address with DL. However, unlike in the example problem we have shown, in a real-world scenario there are several constraints limiting the amount of data we have available for training. This is especially true when dealing with tasks within the medical field that require patient data, as this type of data is not only expensive to acquire but it is also usually protected by privacy policies.

Machine learning algorithms will generally perform best when their capacity is appropriate for the true complexity of the task they need to perform and the amount of training data they are provided with [7]. Given this whole analysis, it is possible to describe the training and generalisation errors as a function of model capacity like depicted in Figure 2.13. As capacity increases training error tends to decrease consistently up until its minimum possible value. The generalisation error tends to have more of U-shaped curve, sharing the decreasing behaviour with the training one until the optimal capacity is reached and from then on shooting to larger values. Usually the gap between the train and generalisation errors is called **variance** and the gap from the train error until the minimum theoretical value is called **bias**.

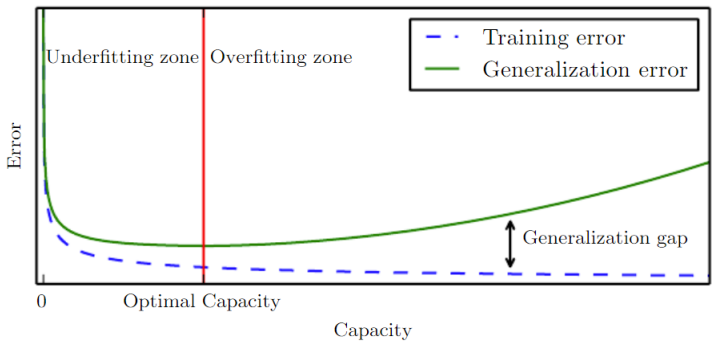


Figure 2.13: Typical relationship between capacity and error. Training and test error behave differently. At the left end of the graph, training error and generalisation error are both high. This is the underfitting regime. As we increase capacity, training error decreases, but the gap between training and generalisation error increases. Eventually,the size of this gap outweighs the decrease in training error, and we enter the overfitting regime, where capacity is too large, above the optimal capacity. Extracted from [7].

It is important to understand that the model capacity affected by the number of parameters/complexity of the model is called **representational capacity** but, in practise, the true capacity of the model is

smaller than that and is denoted by **effective capacity**. Even though a model might have the sufficient representational capacity for the task at hand, finding the optimal function within the hypothesis space is still a difficult optimisation problem. In practise, the learning algorithms in ML, which includes DL, are not build to find the exact best solution but one that significantly and satisfactorily minimises the training error. In fact, the success of the learning algorithm is further conditioned by other factors like the choice of hyperparameters that model its behaviour as well as the regularising constraints of the loss function [7]. We will briefly elaborate on each of them below.

Before, we had introduced weight regularisation and mentioned how it is often beneficial to constraint the values of the parameters weights to smaller values. This effect can be further understood using the concepts introduced in this section. When the representational capacity of a model overshoots the actual complexity of the task at hand, overfitting occurs. In these situations, constraining the value of the trainable parameters can effectively change the preference of some solutions over other within its hypothesis space. By ensuring that a model's weight are kept small, the model is encouraged to find simpler solutions that are less likely to overfit to the training data. The intuition behind this lies on the fact that smaller overall parameters increases the likelihood of some of them being close to zero decreasing the complexity of solution. This way the effective capacity of the model is reduced. Nevertheless, care must be taken to regulate this effect, as too much of an effective capacity reduction can lead to underfitting by providing too simple solutions. The intensity of this type of regularisation can be regulated as an hyperparameter.

The **learning rate** is another hyperparameter whose effect on the effective capacity of the model is also relevant to describe. While increasing weight regularisation intensity leads to a progressively smaller model effective capacity, the learning rate controls the effective capacity in a more complicated fashion. Effective capacity is highest when the learning rate is "correct" for the optimisation problem at hand, rather than when the learning rate is smaller or larger. Consequently, the learning rate will have a U-shaped curve for the training error as depicted in Figure 2.14. Too large learning rates tend to overshoot the optimal solution and fail to converge, while too small can slow down the training process and potentially get stuck in local minima.

In addition to weight regularisation, there are alternative approaches to mitigate overfitting. As previously discussed, expanding the training set can enhance the model's generalisation capability. Nonetheless, in a real life scenario acquiring more data is often limited by various constraints. In these cases we can leverage **data augmentation** for achieving a similar effect. Data augmentation involves applying a variety of transformations and modifications to the existing training data, effectively expanding the dataset without the need for collecting additional samples. However, it is essential to ensure that the augmented data remains realistic and preserves the inherent characteristics of the original data distribution.

In addition to techniques such as weight regularisation and data augmentation, another effective

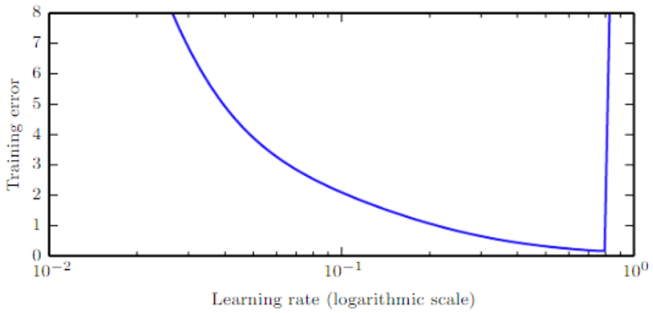


Figure 2.14: Typical relationship between the learning rate and the training error. Notice the sharp rise in error when the learning is above an optimal value. This is for a fixed training time, as a smaller learning rate may sometimes only slow down training by a factor proportional to the learning rate reduction. Generalisation error can follow this curve or be complicated by regularisation effects arising out of having too large or too small learning rates, since poor optimisation can, to some degree, reduce or prevent overfitting,and even points with equivalent training error can have different generalisation error. Extracted from [7].

method to prevent overfitting is **early stopping**. Due to the lack of data, overly complex models tend to become more likely to overfit the training set as the length of the training increases. To avoid that from happening, the performance of the model on the validation subset is continuously tracked and the training is stopped the moment it starts to deteriorate.

2.2.1.C Convolutional Neural Networks (CNNs)

In the last sections we have covered some main aspects about NNs, as well as some key concepts in ML like overfitting, underfitting and model capacity. Now, we can delve into a specific type of neural network that has revolutionised the field of deep learning: Convolutional Neural Networks (CNNs).

CNNs have emerged as a powerful variant of NNs, specifically designed to excel in visual perception tasks such as image classification, object detection, image segmentation, among others. While traditional NNs work well for some tasks, they struggle to effectively capture spatial dependencies in high-dimensional data like images (2D or 3D). The structure of CNNs, particularly the convolutional layers, addresses this limitation by incorporating several key elements.

Convolutional layers are the key building blocks of CNNs. Each convolutional layer consists of multiple filters, also known as kernels, that are applied to the input data via the convolution operation, creating several feature maps. These feature maps will capture relevant spatial information, highlighting certain features or patterns present in the input data.

Mathematically, the 2D convolution operation can be expressed as follows:

$$\mathrm{FM}(i,j) = \sum_{m=1}^{M}\sum_{n=1}^{N} \mathrm{I}(i-m, j-n) \times \mathrm{Kernel}(m,n) \tag{2.3}$$

where $\mathrm{FM}(i,j)$ represents the output feature map value at position (i,j), $\mathrm{I}(i-m, j-n)$ denotes the

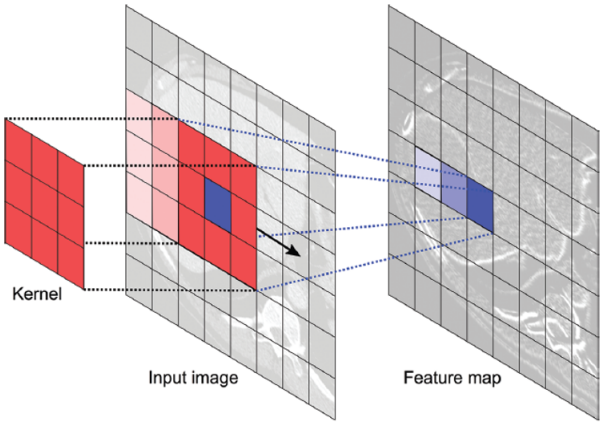


Figure 2.15: Visual depiction of how the 2D convolution works. Extracted from [4].

input image value at position $(i-m, j-n)$, and Kernel(m, n) is the filter coefficient at position (m, n). The size of the feature map is here represented as (M, N).

In a convolutional layer, each (i, j) position of the resulting feature map (FM) is always given by the output signal of a neuron, while each position of the input (I) is given by either the pixel values of the input image, or the output signal of neurons from previous feature maps.

The output signal of a neuron from the resulting map will only depend on the neurons from a specific region of the input data whose size is dictated by the dimension of the kernel. This region is called the receptive field of the neuron. In turn, the filter coefficients will correspond to the weights that will be applied to the signal coming from the neurons representing specific region of the input (see Figure 2.15).

The nature of the filtering operation makes is so that the filter coefficients are kept the same for every (i, j) position of a given feature map, only the subset of input neuron's changes. This weight sharing is employed to enforce the notion that the same feature or pattern can occur at different spatial locations within an input, thereby allowing efficient detection of spatial patterns that can be useful for the task.

Another crucial component of Convolutional Neural Networks (CNNs) is **pooling layers** and they operate on the feature maps generated by the convolutional layers. They divide the feature map into non-overlapping regions, commonly referred to as pooling windows, and apply a pooling operation within each window. This process aggregates information within the window to produce a summarised representation. Their primary purpose is to reduce the spatial dimensions of the input data, thereby decreasing the computational complexity of subsequent layers while retaining essential features. Adding to that, by downscaling the representation, pooling layers facilitate translational invariance, making the model more robust to variations in object position or appearance within the input. There are various types of pooling operations, but the one used in this work is **Max Pooling**, in which the maximum value within each pooling window is selected, capturing only the most salient features.

The processes described above can be generalised to handle 3D data, such as volumetric images in our

case. The convolution operation is extended to three dimensions, with filters sliding over the width, height and depth. Similarly, pooling layers can be applied in 3D by dividing the feature maps into volumetric partitions as opposed to 2D ones.

Convolutional and Pooling layers are usually interleaved and applied as a first stage **feature extraction block** which outputs a lower dimension high-level feature representation of the input. This information is the processed by a **Fully Connected block** that will output the model predictions. This block typically consists of multiple **fully connected layers**, also known as dense layers, wherein, contrarily to convolutional layers, every neuron is connected to every neuron of the previous one. This higher complexity allows them to process the extracted features and establish relationships between them and the desired outputs. See an example of a full CNN architecture at Figure 2.16

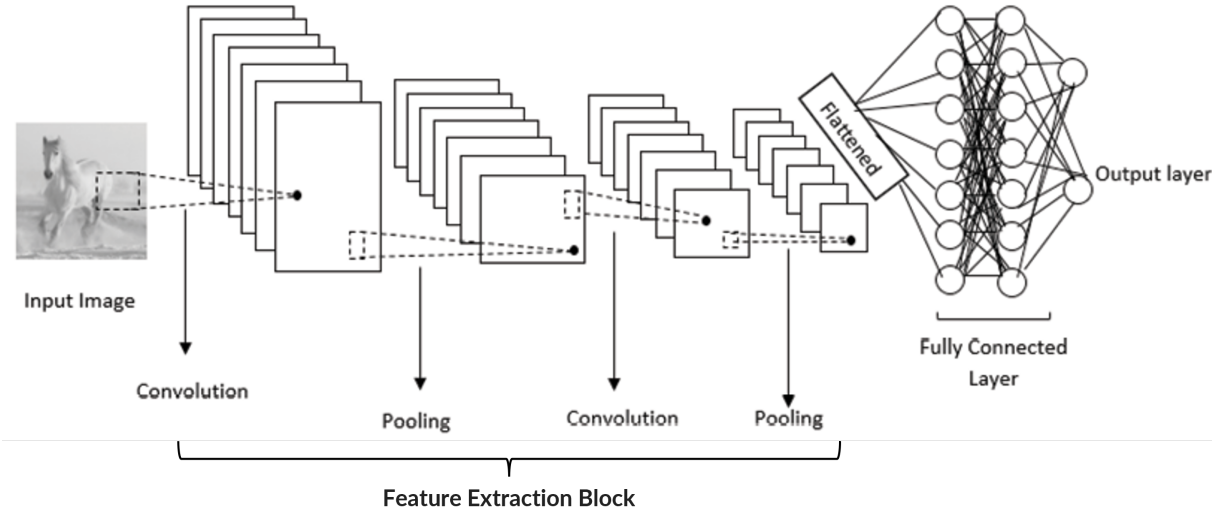


Figure 2.16: Example of a simple 2DCNN with all its main components. Feature extraction block is connected to the fully connected block via a flattening operation that simply reshapes all the features maps from the last step of feature extraction into a 1D vectors. Adapted from [**?**].

Finally, as a side note, it is common to use Batch Normalisation layers in CNN architectures. It improves training stability and convergence by normalising neuron activations within mini-batches, reducing dependence on input scale and distribution.

2.2.2 Multi-Task Learning (MTL)

Having discussed the fundamentals of DL in the previous sections, we now shift our focus to the topic of Multi-Task Learning (MTL).

MTL refers to the paradigm of training a single model to perform multiple related tasks simultaneously, as opposed to training separate models for each task independently. The idea is to leverage training signals from other tasks to guide the training towards to better solutions than the ones found for when training

on a single task.

The motivation behind MTL stems from the observation that some tasks can often share underlying commonalities and can benefit from shared representations and knowledge transfer. By jointly training multiple tasks, a model can learn to extract and leverage shared features, leading to improved generalisation and performance across all tasks.

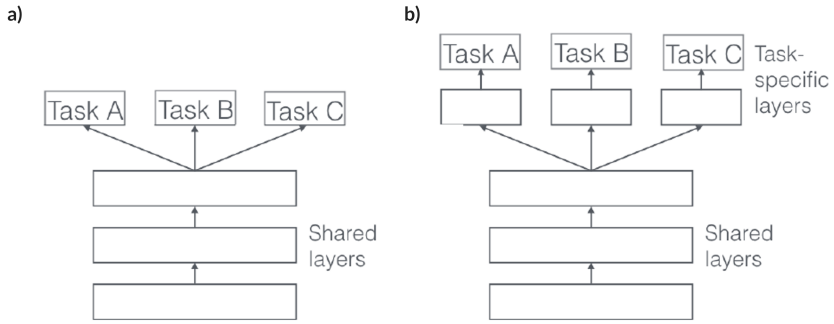


Figure 2.17: Scheme describing the structure of two different hard parameter sharing approaches: (a) fully shared network, where all hidden layers are shared among the target tasks and (b) partially shared network where a common shared trunk of hidden layers exists followed by multiple task specific heads. Adapted from [8].

2.2.2.A MTL Architcetures

In DL, it s possible to highlight two main MTL approaches: **hard** or **soft parameter sharing** of hidden layers. Hard parameter was first introduced by [31] and it generally consists of sharing the hidden layers between all target tasks, while also including (or not) task specific output layers like described in Figure 2.17. In turn, soft parameter sharing does not involve explicit parameter sharing. Rather, distinct models are build for each task but the distance between their parameters is regularised, encouraging them to be similar. This regularisation can be done in several ways but in essence it boils down to an added constraint to the loss function. In this section we will focus more on hard parameter sharing as it has been the approach evaluated in this work.

MTL leads to a reduced risk of overfitting. We had seen before how weigh regularisation alters the effective capacity of the model, by guiding the learning through the hypothesis space towards less complex solutions. Through the same reasoning, by enforcing a model to learn multiple tasks simultaneously, solutions that explain both tasks will be preferred. Consequently, the effective capacity of the model is reduced as the hypothesis space is in practise smaller, thereby minimising the chance of overfitting.

In hard parameter sharing based MTL approaches, the number of parameters shared between the tasks is an important factor to consider. It defines the degree of information sharing between tasks which and it has a significant impact on the model performance. Too extensive information sharing might not be ideal as it can excessively constrain the model's hypothesis space to solutions that somewhat fit both

tasks, but do not adequately fit adequately fit neither. In these situations, improving the performance on one of the tasks will negatively impact the performance on the remaining ones. This phenomenon is called **negative transfer**. On the other hand, not sufficient sharing does not allow the model to effectively leverage information between tasks. The best performing architectures for MTL are those which balance sharing well. We will experiment with different information sharing levels in our work.

2.2.2.B Optimisation in MTL

In order to optimise the weight of the model considering multiple tasks, multiple task specific losses must be designed and combined into a single joint loss function which the model will be trained to minimise. Obviously, the way these sub task losses are combined is a determining factor in model performance.

The simplest approach is merely summing all the losses, however this might be problematic when considering losses with different scales and range of values. In this case, it has been shown that combining the losses via a weighted sum, such that all losses are approximately at the same scale and the gradient align together, can improve overall performance [32] [33].

In this work we test an adaptive loss weighting approach by [34] where the loss weights are learned jointly with rest of the model's parameters. In essence, the authors treat the multi-task network as a probabilistic model, and derive an uncertainty weighted multi-task loss function by maximising the likelihood of the ground truth output. A more in depth description is provided below.

As explained in Cipolla et al [34], homoscedastic uncertainty is an aleatoric uncertainty which is not dependent on the input data. It is not a model output, rather it is a quantity which stays constant for all input examples of the same task and varies between different tasks. In a multi-task setting, this task-dependent uncertainty captures the relative confidence between tasks and can be used as a basis for weighting losses.

The likelihood of the model performing a correct prediction for a single task can be described as the joint probability of observing the target label (y) given a network with output ($f^W(x)$), weights W and input x. Considering the network as a probabilistic model and that the task at hand is regression, this likelihood can be described by a Gaussian distribution with mean given by the output of the model and standard deviation given by a σ noise factor. This noise factor represents the homoscedastic uncertainty for this task.

$$p(y|f^W(x)) = \mathcal{N}(f^W(x), \sigma) \tag{2.4}$$

For a case with N simultaneous regression tasks, the likelihood of the network predicting correctly for all N tasks can be given by $p(y_0, y_1, ..., y_N|f_0^W(x), f_1^W(x), ..., f_N^W(x)) = \prod_{i=0}^{N} p(y_i|f_i^W(x))$.

Maximising this likelihood is equivalent to maximising the corresponding log likelihood. Given that we have modelled it as a Gaussian distribution, it can be shown that maximising the log likelihood is the

same as maximising the following expression:

$$\log(p(y|f^W(x))) \propto -\frac{1}{2\sigma^2} \cdot ||y - f^W(x))||^2 - \log(\sigma) \tag{2.5}$$

Assuming the target labels y_i follow a normal distribution and considering the log likelihood formulation above, it is possible to put everything together to create a minimisation objective as follows:

$$-\log(p(y_0, y_1, ..., y_N|f_0^W(x), f_1^W(x), ..., f_N^W(x))) = -\log(\prod_{i=0}^{N} p(y_i|f_i^W(x))) \tag{2.6}$$

$$= -\sum_{i=0}^{N} \log(p(y_i|f_i^W(x)) \tag{2.7}$$

$$\propto \sum_{i=0}^{N} \frac{1}{2\sigma_i^2} \cdot ||y_i - f_i^W(x))||^2 + \log(\sigma_i) \tag{2.8}$$

To maximise the log likelihood of observing the target labels from the current model is equivalent to minimising the last expression above which, in essence, is to train the network to output our wanted labels. The final loss expression can be reformulated as follows as function of the model's weights W and the task dependent (homeoscedastic) noise factors σ_i.

$$\mathcal{L}(W, \sigma_i) = \sum_{i=0}^{N} \frac{1}{2\sigma_i^2} \cdot \mathcal{L}_i(W) + \log(\sigma_i) \tag{2.9}$$

In this final loss formulation, $\mathcal{L}_i(W)$ represent the sub task specific loss metrics explained earlier and not the squared error presented earlier for simplicity during the derivation.

Considering this loss function, its minimisation can be interpreted as learning not only the best network's weights but also the noise factors σ_i that define the each loss weights in an adaptive manner throughout the training, based on the data. Each loss weight is then given by $\frac{1}{2\sigma^2}$, therefore being larger for smaller noise factors and smaller for larger ones. As a result, when the homoscedastic uncertainty of the task is great, the influence of task i on the network weight update is reduced. This is useful when dealing with noisy annotations because the task-specific weights are automatically reduced for such tasks.

This function also includes a regularisation parcel $\log(\sigma_i)$ that acts to discourage these noise factors from increasing too much which would excessively decrease the loss weights and effectively ignore the data.

2.3 State-of-the-art on Automatic CMR Planning

The Siemens Dot system is a commercially available solution that aids operators during MRI exams [19]. Without removing the autonomy from the operator, it simplifies the workflow and enhances efficiency by

streamlining repetitive or mundane tasks. For instance, this tool is able to automatically position the patient at the centre of the magnet as it knows the common centring points for certain body regions. It also makes it possible to change the pulse sequence strategy with ease and in a faster manner which is key when managing patients which need extra attention. The system also comes with an *AutoAlign* feature that allows the scanner to align the slices as set out in the protocol. A variant of this system, called Cardiac Dot Engine, has been implemented specifically for CMR acquisitions. Being available on Siemens Healthineers MRI scanners, this tool provides guidance reducing the workload of the operator especially during the planning. Its *AutoAlign* feature uses a undisclosed machine learning algorithm to locate five anatomical landmarks - the base of the left atrium, aortic root, acute margin of the right ventricle, base of the left ventricle, and left ventricular apex - and from them extracts the standard views. The *InlineVF* functionality automates the contouring of the endocardium and epicardium of the left ventricle and the detection of the mitral valve basal plane, facilitating the calculation of relevant parameters like the ejection fraction, end-diastolic volume, end-systolic volume, stroke volume, cardiac output, and myocardial mass. It has been seen that this system can alleviate the burden of the CMR planning and interpretation of results while also increasing their overall reproducibility [35].

However, there is still space for further automation of this procedure. Even when using the Cardiac Dot system, the operator still has to perform some online decisions which after a full day of scanning can contribute to concentration-burnout as well as compromise standardisation and the reproducibility of the method.

Over the last years, there has been some work around automatic view plane prescription that suggests the feasibility of escalating from Cardiac Dot's user aid to something that removes the need for user input completely.

Frick et al [9], for instance, relies on fitting an atlas to an input 3D survey CMR scan to extract the cardiac landmarks later used for view prescription (see Figure 2.18a). Similarly, Lu et al. [10] fits a 3D mesh representation of the left ventricle to input 3D survey scans and from there estimates the location of the cardiac landmarks later used for view prescription (see Figure 2.18b). Naturally, these two approaches are limited as they rely on predefined templates for prescribing the view planes and hence are not well suited to handle inter-patient variability.

Later, more refined machine learning based approaches have been proposed. Namely, Alansary et al. [11] presented a reinforcement learning (RL) based approach to automatically prescribe the 4-chamber CMR view from a 3D MRI data. However, due to iterative trial-and-error nature of RL, training takes days thereby lacking computational efficiency.

In the recent years, Deep Learning (DL) methods have become quite popular for a variety of computer vision related tasks in medicine, ranging from disease risk stratification to disease detection and classification from medical images as well as segmentation of anatomical structures and quantification of

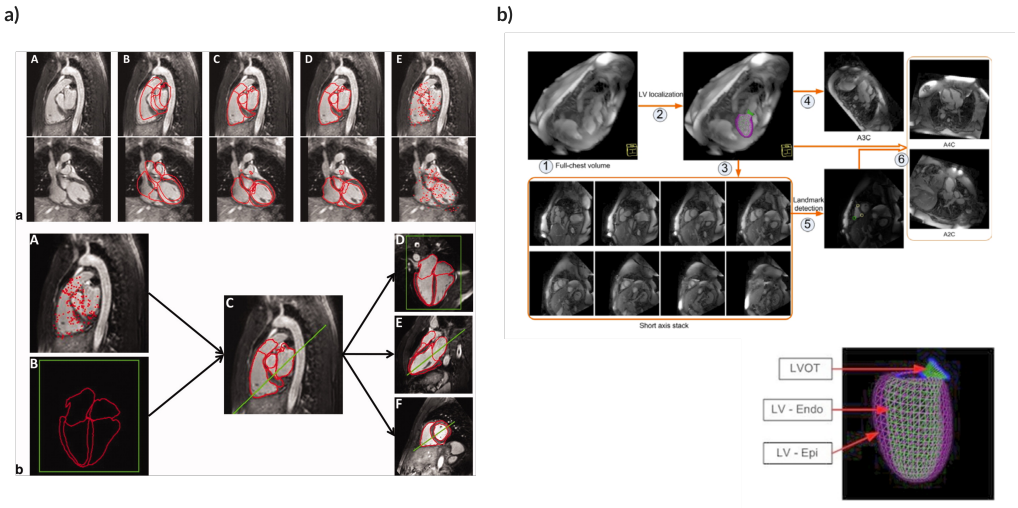


Figure 2.18: Overview of the method proposed by Frick et al [9] (a) and method proposed by Lu et al. [10] (b).

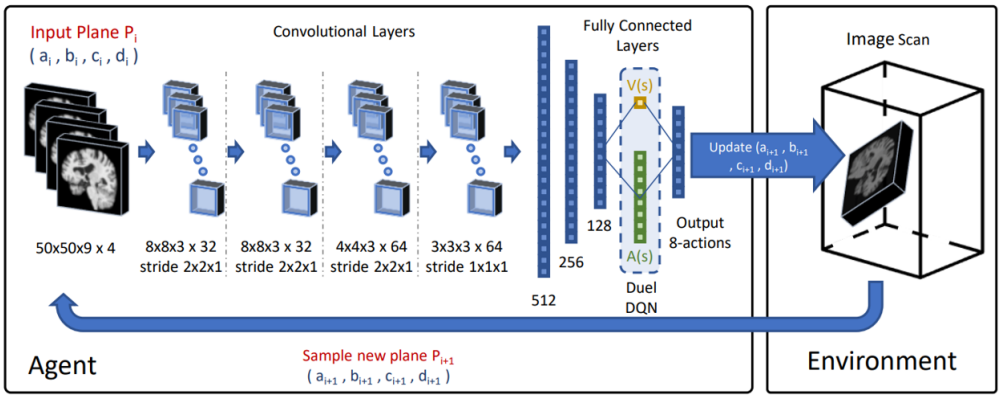


Figure 2.19: Overview of the method proposed by Alansary et al. [11].

imaging features. When it comes to CMR, however, most effort has been seen on the segmentation of cardiac structures and not a lot of papers focus on the automation of the view planning step, which, as previously explained, still remains a significant bottleneck to the widespread use of CMR. Nonetheless, new work with DL addressing this very task has been emerging with remarkable results. From them we can highlight the following.

Le et al. [12] and Blansit et al. [13] both propose a type of approach that, similarly to Frick et al [9] and Lu et al. [10], makes use of cardiac landmarks to prescribe the target cardiac planes. However, instead of relying on a pre-defined template to estimate these points they leverage DL-based heatmap regression. In particular, Le et al. [12] starts by computing the bounding box that tightly crops the heart from the rest of the input 3D scan and then the 6 required landmark locations are regressed via a 3D heatmap. This last step is carried out by means of a 3D extension of the ENet [36]. In turn, Blansit et al. [13] provides an alternative to using 3D volumes as input and adopts a multi-step approach for regressing the required cardiac landmarks. It starts by regressing two LAX landmarks from a 2D localiser slice (VLAX from Figure 2.21). Based on them, multiple SAX slices are prescribed and the heart region is selected by a pre-trained heart bounding box model. Another pre-trained model is used to select the SAX slice containing the Mitral Valve from the stack and that is the slice upon which heat map regression is performed to estimate the location of the other relevant cardiac landmarks. The remaining view planes are prescribed based on the location of all the estimated landmarks. Once again, results are encouraging but both of these two methods rely on extensive manual annotation of the cardiac landmarks that is on itself a time consuming and complex task that poses a constraint to increasing the amount of data available for training. Not only that but basing the view plane prescription on cardiac landmarks alone fails to leverage some global context that might be useful for correctly determining each plane's position and orientation within the patient's body. Furthermore, [13] also requires significant expertise in cardiac anatomy to obtain the 2D localiser slices upon which the whole automatic pipeline is based, consisting in another obstacle to a fully automatic and self-driven pipeline.

Addressing these concerns, other methods not based on cardiac landmarks have been proposed. We had seen how [11] uses RL to approximate the 4CH view plane from a input 3D scan. Chen et al [14] leverages DL to address the same problem using the same type of data but in CT. It makes use of a two step approach to achieve multiple objectives. First, a 3D UNET model is trained to segment the left ventricle and left atria in the input scan. Next, using only the down sampling branch of that UNET initialised with the weights learned for the segmentation task, two different models were trained to predict the orientation and position vectors that define each of the standard view's image planes. Considering the 4 target cardiac view planes, [14] yields 8 view prescription models and 1 segmentation model. Compared to the previous RL approach, this DL method requires much shorter training times of approximately 2h instead of multiple days. By training CNNs to regress the vectors that define each views image plane from

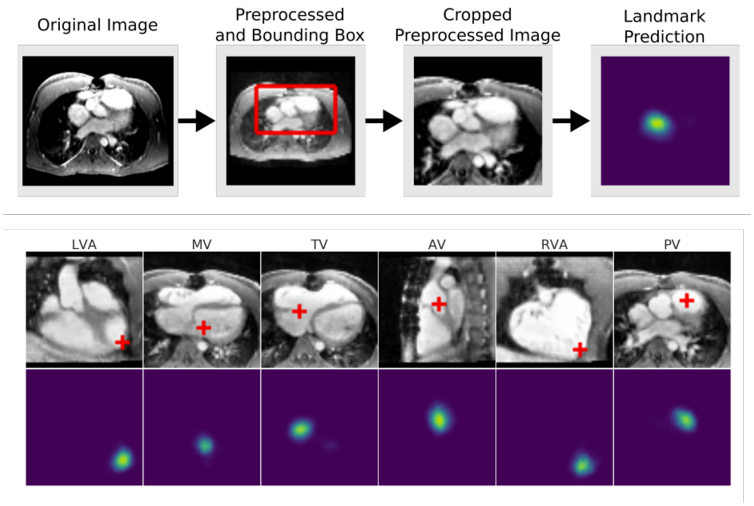


Figure 2.20: Overview of the method proposed by Le et al [12].

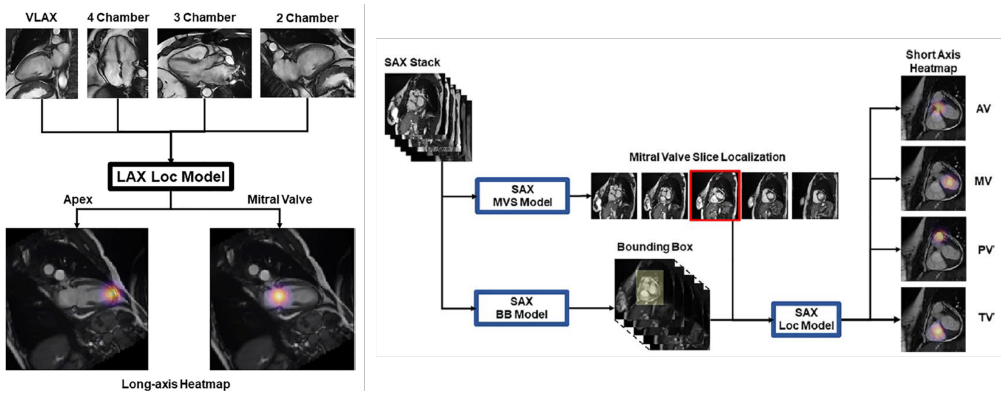


Figure 2.21: Overview of the method proposed by Blansit et al [13].

the input 3D scan, the views are being prescribed taking into account information from the whole scan thereby ensuring that all global context is considered for view prescription. Nevertheless, by predicting each plane's position and orientation separately, this approach requires training more than one model for prescribing a single view which can be expensive in resources. Not only that but, by doing so it fails to leverage similarities between these two tasks, that could potentially exploited to improve performance.

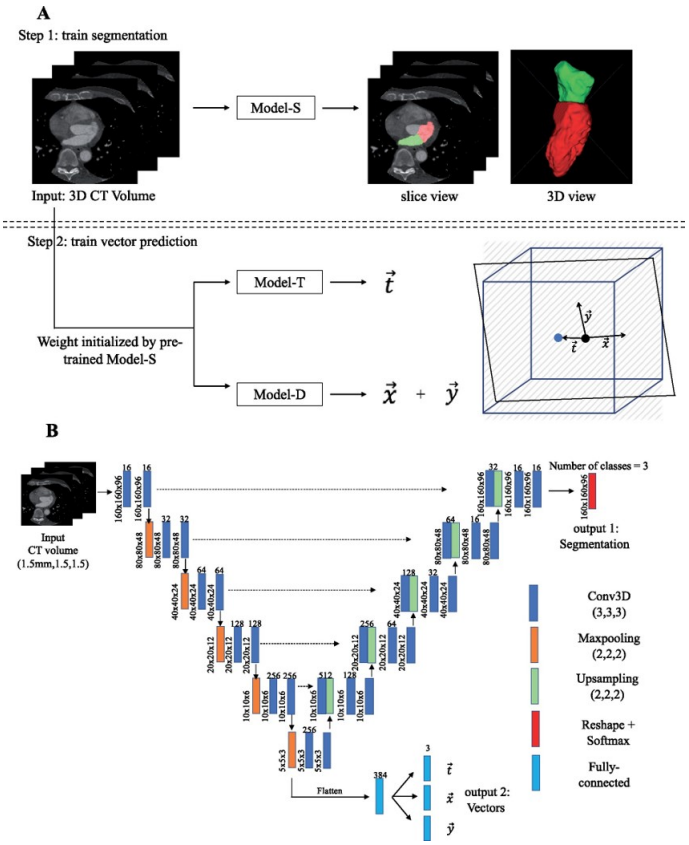


Figure 2.22: Overview of the method proposed by Chen et al [14].

One example of how the position and orientation of a plane can be successfully predicted simultaneously by the same model can be found the work in Corrado et al [15]. Using 4D Flow MRI data, they propose a DL model able to predict the planes for flow measurements in the great vessels using a Residual Learning CNN approach. While the motivation here is not exactly the same as in our use case, it is, in essence, an equivalent scenario as it deals with MRI 3D data to predict plane defining vector labels like before. Here, a fully-shared MTL approach is used to, not only predict the position and orientation of one plane together, but also all the target planes jointly. This allows to take advantage of the similarities between the position and orientation tasks as well as of the relative positioning between the target planes. Note that here the neural network is processing the input 4D Flow MRI data on its 5 channels as opposed to the single channel CT in Chen et al [14]. Also, the input multi channel volume is not the whole volume like before but 32x32x32 patches from the full one to reduce GPU memory requirements and increase the

number of samples.

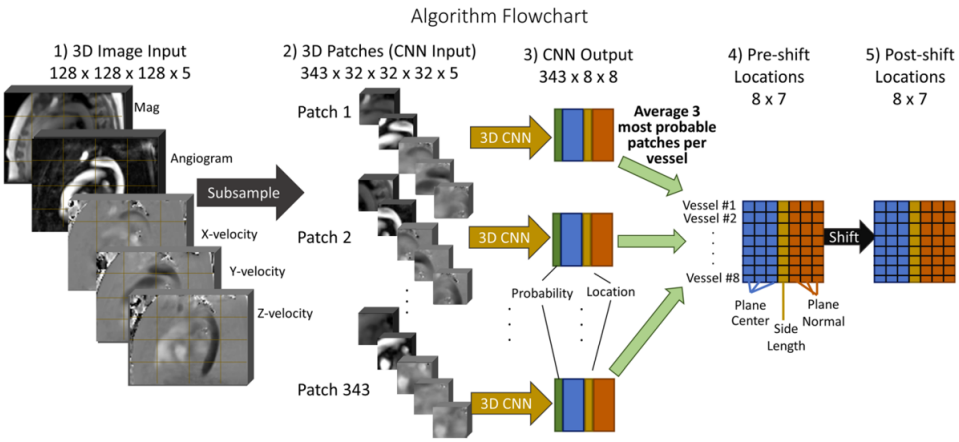


Figure 2.23: Overview of the method proposed by Corrado et al [15].

Considering the existing literature, in this project we address the lack of a non landmark based DL approach for fully automatic standard cardiac view plane prescription in CMR and which is able to estimate each plane's position and orientation prediction jointly via MTL.

3

Dealing with Dataset Challenges

Contents

This chapter presents a detailed overview of the methodology employed in this study. It begins by discussing the baseline performance, followed by the identification and resolution of challenges associated with the dataset. Lastly, a new baseline performance is presented.

3.1 Methodology

3.1.1 Dataset

The dataset used to train our models is comprised of 120 3D CMR scans from different patients labelled with the respective view defining vectors of each of the 4 standard CMR view planes. Each plane is defined by the DICOM image position vector ($\overrightarrow{t}$) [37] and the DICOM image orientation vectors $\overrightarrow{o_1}$ and $\overrightarrow{o_2}$ [38]. Vectors $\overrightarrow{o_1}$ and $\overrightarrow{o_2}$ are unitary, orthogonal to one another and their cross product defines the normal vector ($\overrightarrow{n}$) to the plane, while vector $\overrightarrow{t}$ give us the location of the origin of the plane, meaning the centre of the first voxel of that slice being transmitted. Similarly to the view planes, the location and orientation of the 3D scans also comes specified by an equivalent set of vectors.

Data were acquired on a 1.5T Philips scanner, using standard clinical protocols and an ECG-triggered volumetric bSSFP sequence with field of view of 440x440x150 mm^3 , voxel size of 3x3x3 mm^3 , compressed SENSE acceleration factor six, and scan time of 10 seconds (assuming 60 bpm heart rate). Adding to that, 51% of the scans were acquired via a LGE-Imaging technique thus including the trace of the contrast. The dataset is quite varied as it was obtained from patients from a wide range of ages, with different pathologies and also by multiple operators.

It is important to note that these data were provided by the University of Linköping and its use in this study was approved by that same institution's ethical committee. Adding to that, every patient whose data is included in this dataset is aware of its use and provided their written informed consent.

3.1.2 Data Splitting

In order to be able to evaluate the performance of the models on unseen data the dataset must be split into three different subsets. From the 120 samples, 88 were used for training, 16 were used for the validation set and 16 others for the test set, amounting to a $74\% - 13\% - 13\%$ split. Since the dataset includes scans both with and without contrast, these splits were done in a stratified fashion ensuring that each subset contains a percentage of volumes with contrast as close as possible to that of the entire dataset. This step certifies that no bias is introduced in the evaluation of the various trained models, as all subsets are drawn from the same distribution.

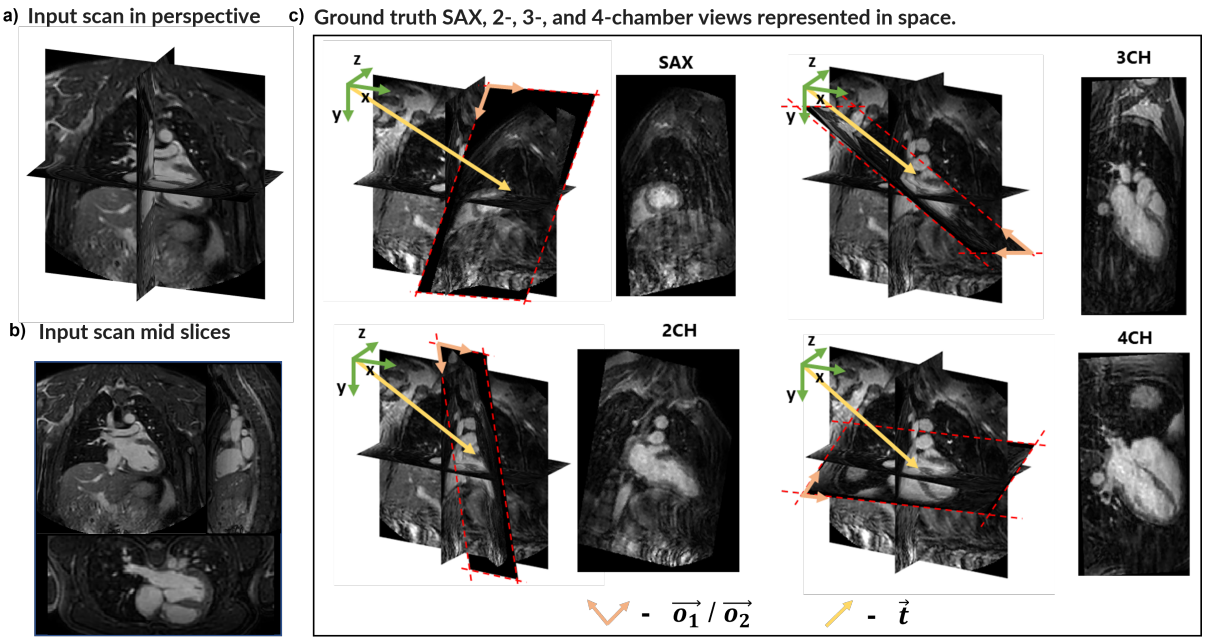


Figure 3.1: a) Input volume for a representative subject, b) 2D mid slices (coronal, sagittal and transverse) and c) corresponding ground truth standard view planes and their view defining vectors: $\overrightarrow{o_1}$ and $\overrightarrow{o_2}$ define orientation and $\overrightarrow{t}$ defines the location of the plane's centre. Includes a scheme of each image plane position and orientation within the volume

3.1.3 Preprocessing

Some data preprocessing steps were done prior to feeding the samples into the network during training.

3.1.3.A Changing Reference Frame

The reference frame with relation to which all the orientations and positions come defined is called DICOM's Reference Coordinate System (RCS), but, since we want our models to be predicting these view planes from the volume, we must ensure that the predictions can be made using a coordinate system centred on the volume. With that in mind, an extra step of transforming each view vector from the aforementioned RCS into a new reference frame centred on the origin of the 3D scan was taken. For that purpose, affine transform matrices were build from the orientation and position vectors of each volume which were then used to perform this transform to the corresponding view vectors. Let us look at the algebra behind this process:

M_v is the matrix that defines the pose of a given volume within the RCS reference frame and is defined as follows:

$$M_v = \begin{bmatrix} & R_v & & \mathbf{t}_v \\ 0 & 0 & 0 & 1 \end{bmatrix}_{(4\times4)} \tag{3.1}$$

where t_v is the position vector of the volume in the RCS and where R_v represents the orientation

of the volume in RCS. The latter is defined by the orientation vectors, o_{1_v}, o_{2_v} and n_v in RCS of the volume, like so:

$$R_v = \begin{bmatrix} o_{1_v} & o_{2_v} & n_v \end{bmatrix}_{(3\times3)} \tag{3.2}$$

The volumes' pose matrix (M_v) in the RCS can be used as an affine transform matrix to bring vectors from the volume's reference frame to the RCS. Consequently, M_v^{-1} can be used to perform the inverse transformation from the RCS to the volume's reference frame.

$$x_{RF_{RCS}} = M_v \cdot x_{RF_v} \Leftrightarrow x_{RF_v} = M_v^{-1} \cdot x_{RF_{RCS}} \tag{3.3}$$

Just like M_v, M_v^{-1} is composed by a rotation and a translation component, R_{RCS} and t_{RCS}, respectively.

$$M_v^{-1} = \begin{bmatrix} & R_{RCS} & & \mathbf{t}_{RCS} \\ 0 & 0 & 0 & 1 \end{bmatrix}_{(4\times4)} \tag{3.4}$$

For a given volume, each view's position vector can then be transformed into the volume's reference frame like described below:

$$t_{RF_v} = M_v^{-1} \cdot t \tag{3.5}$$

The orientation vectors do not define any point in space and, as such, their transformation to the volume's reference frame can be done using solely the rotation component (R_{RCS}) of the affine transform defined by M_v^{-1}.

$$o_{1_{RF_v}} = R_{RCS} \cdot o_1 \tag{3.6}$$

From this point on, every time we mention any view defining vectors ($\overrightarrow{o_1}$, $\overrightarrow{o_2}$, $\overrightarrow{n}$, $\overrightarrow{t}$) we will be referring to their representations in the volume's reference frame.

Additionally, inspired by Chen et al. [14], a tweak was made on the dataset, this time consisting of changing the location the position vector is pointing to from the origin of the slice to the actual centre of it.

After this preprocessing step the dataset comprised of 120 volumes labelled with 12 vectors (3 per view) in the volume's reference frame, depicting both the orientation of said plane as well as the position of its centre. A scheme of the structure of the final dataset can be seen in Figure 3.1.

3.1.3.B Label Variance Adjustment

Upon analysing the distribution of the label vectors' coordinates, some interesting findings were made that motivated an extra preprocessing step to our dataset. The distribution of the x and z coordinates of $\overrightarrow{o_1}$ in the 2CH dataset behave like a bimodal distribution approximately centred at 0. The same could be observed in the 4CH dataset now considering the y and z coordinates of the $\overrightarrow{o_2}$ (Figure 3.2).

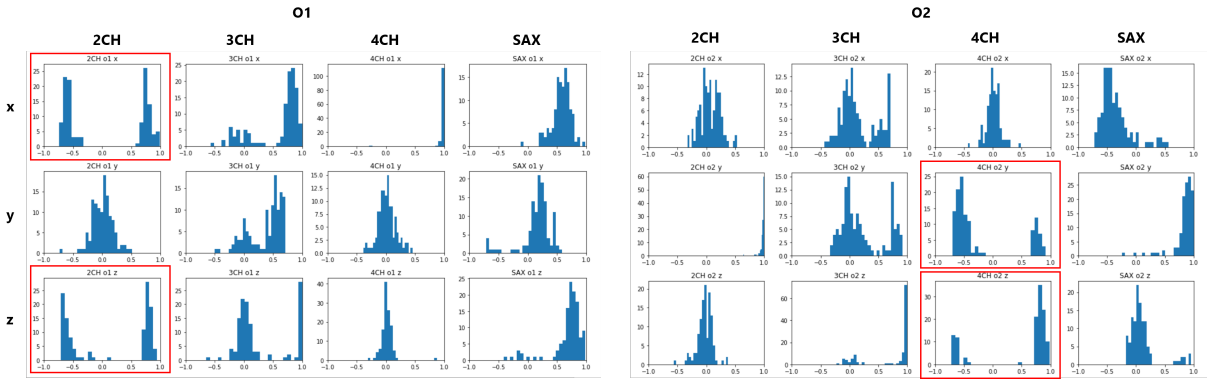


Figure 3.2: Distribution of the orientation vector coordinates across the dataset, in particular vectors $\overrightarrow{o_1}$ on the left and $\overrightarrow{o_2}$ on the right. Plots highlighted in red correspond to the bimodal distributions centred at approximately 0.

When looking into the angles between all the same view orientation vectors across samples (i.e. $\overrightarrow{o_1}_i \measuredangle \overrightarrow{o_1}_j$ and $\overrightarrow{o_2}_i \measuredangle \overrightarrow{o_2}_j$ for every $i, j = 0, \cdots, N_{samples}$), the same bimodal behaviour appears on the 2CH $\overrightarrow{o_1}$ distribution and on the 4CH $\overrightarrow{o_2}$ distribution (Figure 3.3). Unlike the distributions found in other view's datasets (3CH and SAX), these bimodal distributions are centred around approximately 90 degrees. This suggests that the distinct trends observed in the aforementioned plots correspond to nearly symmetric orientation vectors, indicating similar orientations within the volume. The most plausible explanation for these findings is that, given the dataset comprises scans acquired by different technologists, variations in acquisition conventions have likely contributed to the observed variance in orientation vector coordinates. Consequently, this variance is more a result of the lack of a standardised view prescription practice rather than meaningful anatomical or pathological differences visible in the scans.

With that said, to avoid confusing the models during training, this non task related variance was minimised by gathering all the samples with negative $\overrightarrow{o_1}$ x coordinate for the 2CH dataset and transforming each $\overrightarrow{o_1}$ into its symmetric. The same is done for the samples with negative $\overrightarrow{o_2}$ y coordinate for the 4CH dataset. By changing the symmetry of these vectors, the amount of non-meaningful variance within the dataset is greatly reduced while no change is being made to the orientation of the view plane in question. This is possible as any combination of $\overrightarrow{o_1}$ and $\overrightarrow{o_2}$, symmetric or not, always defines the same view plane orientation within the volume (see Figure 3.4A). Figure 3.4B depicts the distribution of the angles between 2CH $\overrightarrow{o_1}$ and 4CH $\overrightarrow{o_2}$ before and after the variance fix.

The impact of this change in model performance will be later analysed in the results section.

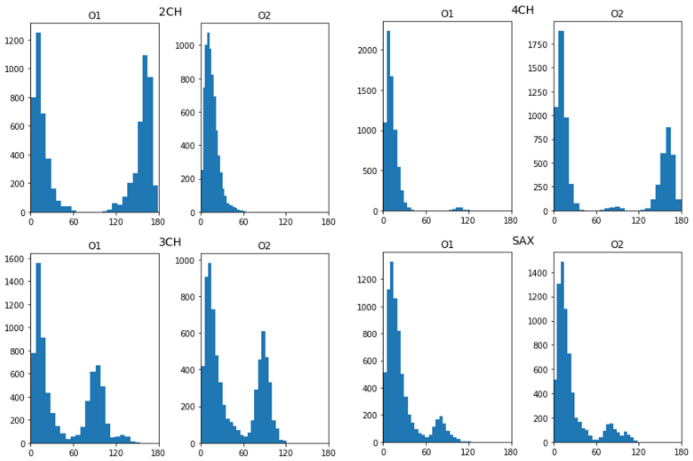


Figure 3.3: Distribution of the angles between all same view $\overrightarrow{o_1}$ and $\overrightarrow{o_2}$ in the the dataset.

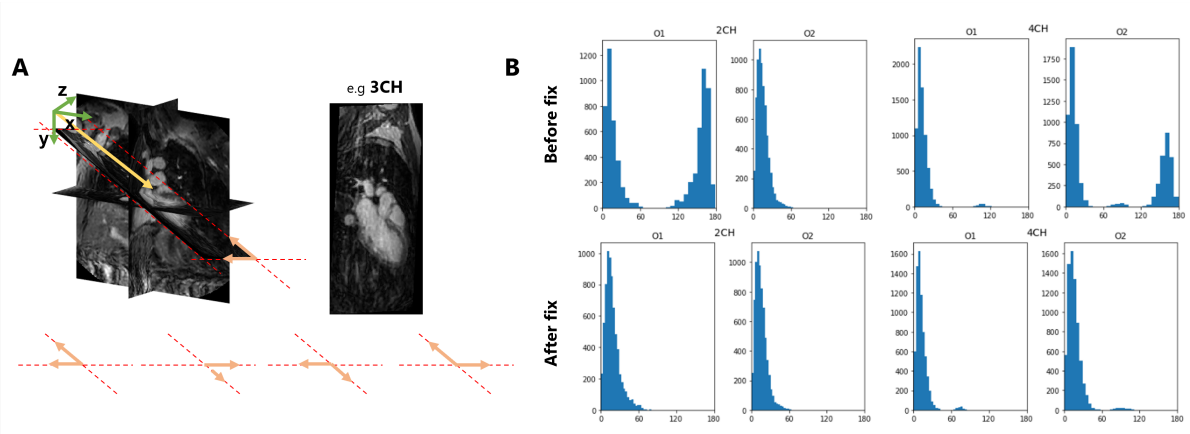


Figure 3.4: **A)** Different orientation vector pairs that depict the same plane orientation. **B)** Distribution of the angles between all 2CH and 4CH $\overrightarrow{o_1}$ and $\overrightarrow{o_2}$ before and after adjusting the dataset.

3.1.3.C Volume Processing

The mean and standard deviation of voxel intensity is computed for each sample in the training set and then the mean of these values are used to centre and standardise the dataset using the following Equation:

$$V_s = \frac{V - \mu}{\sigma} \tag{3.7}$$

, where μ stands for the mean across samples of the average volume pixel intensity and σ is the mean of the standard deviation of pixel intensity across samples. V and V_s stand for the volumes before and after standardisation.

The volumetric data was provided to us in MATLAB binary files (.mat) which undergo an axis swap when being loaded from DICOM format. Consequently, an important preprocessing step was also to swap the x and y of the volumes to ensure the correspondence between the label vectors and volume.

Finally, under the hypothesis that the full resolution is not necessary for good predictions, every volume was resized from an isotropic resolution of 3mm to 4mm. The final input volume dimension was 95x95x42 mm^3. This down sampling was used, among other methods, to help minimise memory usage during training. In the next sections other approaches to address this issue will be described.

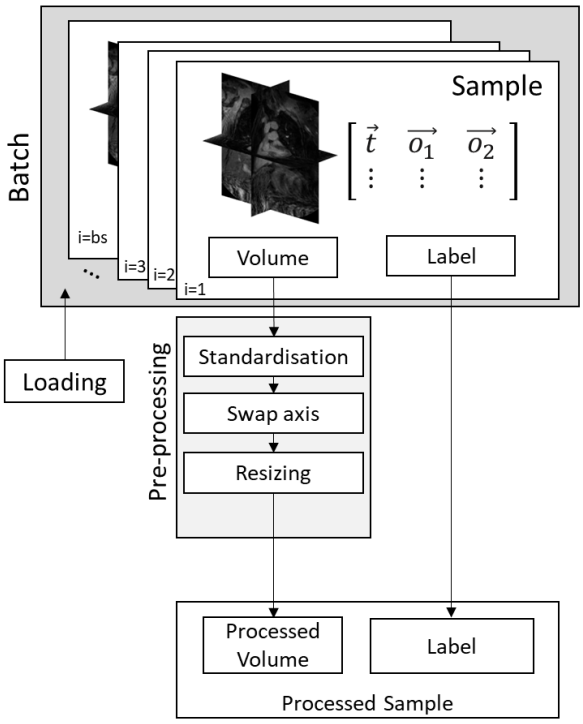


Figure 3.5: Loading and preprocessing pipeline. All the mentioned operation are performed to every sample in a batch during training before feeding them into the network.

3.1.3.D Custom Data Generator

Even though the dataset does not contain a very large number of samples, each one of them includes a large 3D volume and thus, is too large to load all data at once into the GPU memory. To be able to access and process the samples during training, a custom set of functions was implemented to perform this task efficiently without running out of RAM. This implementation is a Custom Data Generator built with Python's high-level package Keras, inheriting the properties of its class *Sequence* [39]. More concretely, this generator is able to load and process the input samples on the fly by batches as they are being needed during training. This way, memory usage is decreased drastically as only the space for one batch needs to be allocated at all times. In Figure 3.5 it is possible to observe a scheme of the full pipeline the batches are put through prior to being fed into the network during training.

Another reason to opt for this custom data generator is that the type of augmentation intended to use includes specific operations that Keras does not provide in the pre-implemented utility functions.

3.1.3.E Volume Reslicing

The dataset used contains the volumes and the vector coordinates that describe each target plane but not the plane images *per se*. As a means of visually evaluating the predicted images, a custom reslicing algorithm was implemented from scratch. This algorithm enables the extraction of any desired view plane within the volume based on its corresponding view-defining vectors (see Figure 3.1c).

3.1.4 Multi-objective Loss Function

Prescribing a plane can be divided into two different sub tasks: position prediction and orientation prediction. In order to train a model to perform both tasks simultaneously a loss function must be computed for each of them and then combined.

In previous works [14], the position loss was set to be the mean squared error between the predicted ($\overrightarrow{t_{pred}}$) and true position vectors of each view's centre pixel ($\overrightarrow{t}$), while the orientation loss was defined as the cosine similarity between the predicted and ground truth orientation vectors ($\overrightarrow{o_1}$ and $\overrightarrow{o_2}$). Despite performing satisfactorily, this approach constraints the model to learn the exact location of the centre of the plane as well as both exact orientation vectors, which fails to leverage some mathematical knowledge of how a plane is be defined. In fact, a plane can be described using solely two vectors: **a position vector of any point within the plane and the plane's normal vector**. Using this information we were able to better design our loss functions in order to not over complicate the prediction tasks and overcome some dataset constraints that we will later describe. Let us look at them in the next two subsections.

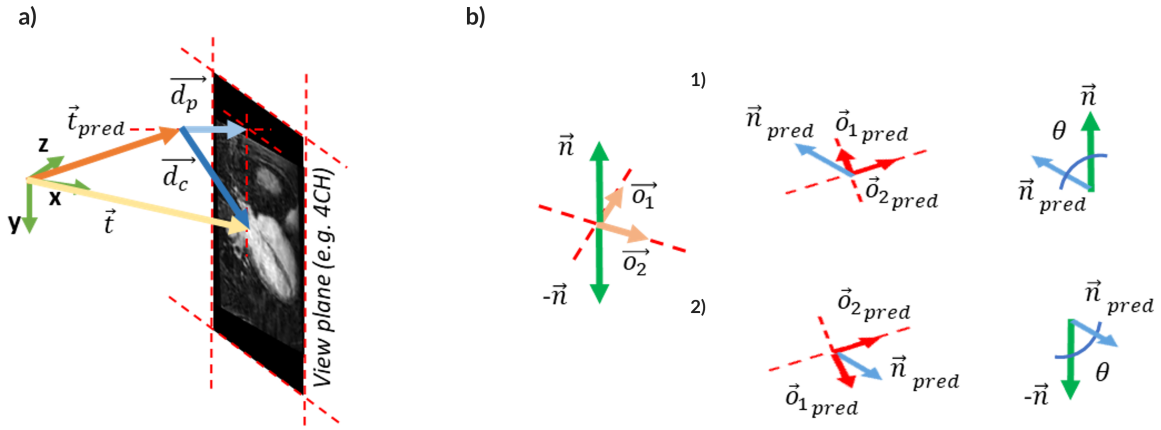


Figure 3.6: a) Visual representation of the vectors on which position prediction metrics are based: $\overrightarrow{t}$ is the previously defined ground truth position vector of the centre of the plane; $\overrightarrow{t_{pred}}$ is the predicted position vector; $\overrightarrow{d_c}$ is the vector whose norm quantifies the distance between the predicted point and the centre of the plane; $\overrightarrow{d_p}$ is the vector whose norm is the distance between the predicted point and plane used both as position loss and performance metric; b) Visual representation of the vectors on which orientation prediction metrics are based: $\overrightarrow{o_1}$ and $\overrightarrow{o_2}$ are the ground truth orientation vectors. $\overrightarrow{n}$ is the ground truth plane's normal vector and $-\overrightarrow{n}$ its symmetric; $\overrightarrow{o_{1_{pred}}}$, $\overrightarrow{o_{2_{pred}}}$ and $\overrightarrow{n_{pred}}$ are the predicted orientation vectors. θ is the angle between the predicted and ground truth planes' normal vectors. (1) and (2) correspond to two equivalent predictions considering the modified cosine similarity loss.

3.1.4.A Plane Position

Granted that we already have a normal vector to the plane, the only element remaining to fully prescribe it is just a point that belongs to it. Considering this, we can set our position prediction task to be the regression of the position vector of any point within the plane. Our proposed position loss function is then defined as the distance between a predicted point and the target plane. By opting for this approach, the regression task is eased as, compared to other works, we are regressing any point within a surface instead of a single specific point.

Having Figure 3.6a as reference, the distance of the predicted point to the plane is given by the norm of vector $\overrightarrow{d_p}$, which is the displacement vector between the predicted point and its closest in-plane point in millimetres. This distance can be computed as the projection of the error vector $\overrightarrow{d_c}$ along the plane's normal direction like depicted in the following equation: $\mathcal{L}_{pos} = ||\overrightarrow{d_p}|| = |\overrightarrow{d_c} \cdot \overrightarrow{n}|$, with $\overrightarrow{d_c} = \overrightarrow{t} - \overrightarrow{t}_{pred}$

No available Keras functions are in place for this loss function, so we implemented it from scratch as a custom loss function. The actual backpropagated position loss will be the mean distance to the plane in the training batch.

3.1.4.B Plane Orientation

Given that the position loss already allows us to regress a point within the plane, the last element needed for view prescription is a vector that is normal to the plane and we can regress that directly instead of regressing each orientation vector separately like in [14]. In this work we propose two different approaches for regressing this vector which are described below and further analysed later on.

Considering Figure 3.6b, the first proposed orientation objective function is based on the cosine similarity between the predicted ($\overrightarrow{n}_{pred}$) and ground truth normal vectors ($\overrightarrow{n}$), which essentially is the cosine of the angle between them (θ). This loss (L_{ori}) allows the minimisation of the angle between the vectors with no regard for their norm, which is not meaningful for this sub task. At each optimisation step, each ground truth normal vector is computed by the external product of the two DICOM orientation vectors ($\overrightarrow{o_1}$ and $\overrightarrow{o_2}$) and compared against a prediction provided by the model. We used the Keras built in *CosineSimilarity* loss function [40], which can be defined by the following equation: $\mathcal{L}_{ori} = -cos(\theta)$, with θ angle between $\overrightarrow{n}_{pred}$ and $\overrightarrow{n}$

However, since we are minimising this angle θ, the model will be constrained to approximate the exact normal vector ($\overrightarrow{n}$) which might not always be the best option. In fact, one has to consider that the same plane orientation is described by the normal vector $\overrightarrow{n}$ and its symmetric $-\overrightarrow{n}$, as they are both orthogonal to it. With that in mind, we propose a second orientation loss function consisting of a modified version of the cosine similarity between the real and predicted normal vectors, which is given by: $\mathcal{L}_{ori_{mod}} = 1 - |cos(\theta)|$.

In this approach we use the absolute value of the cosine similarity, which in essence can be described as the absolute value of the cosine of θ, to quantify how close our predictions are with regards to orientation, while effectively allowing both $\overrightarrow{n}$ and $-\overrightarrow{n}$ to be seen as optimal solutions.

For a more visual interpretation of how this loss works we can look at the two predicted normal vectors in Figure 3.6b, scenario 1 and 2. With this new loss function we are basically ensuring that both 1 and 2 are perceived as equally good predictions by the model, as both represent the same predicted plane.

This modified cosine similarity is multiplied by minus one and summed 1 to convert it into a minimisation objective. No available Keras functions are in place for this loss function, so we implemented it from scratch as a custom loss function. The actual backpropagated orientation loss will be the mean loss in the training batch.

3.1.4.C Loss Weighting

The two above mentioned losses need to be combined into a single one that is able to be backpropagated through the shared network.

The first approach taken was a simple sum of the losses, which was later refined to a weighted sum with task specific loss weights, like depicted below:

$$\mathcal{L}(W) = w_{pos} \cdot \mathcal{L}_{pos}(W) + w_{ori} \cdot \mathcal{L}_{ori}(W) \tag{3.8}$$

, where w_{pos} is the position loss weight and w_{ori} the orientation one.

3.1.5 Network Architecture

Throughout this project several architectures were tested, but our main results were obtained using two different hard parameter sharing MTL approaches with distinct levels of information sharing. In this chapter, we will introduce results from a fully shared architecture ($\mathrm{N_A}$), in which every layer is shared between the two tasks. $\mathrm{N_A}$ served as the baseline network upon which we subsequently built and improved upon in the following chapters.

3.1.5.A Fully Shared Architecture ($\mathbf{N_A}$)

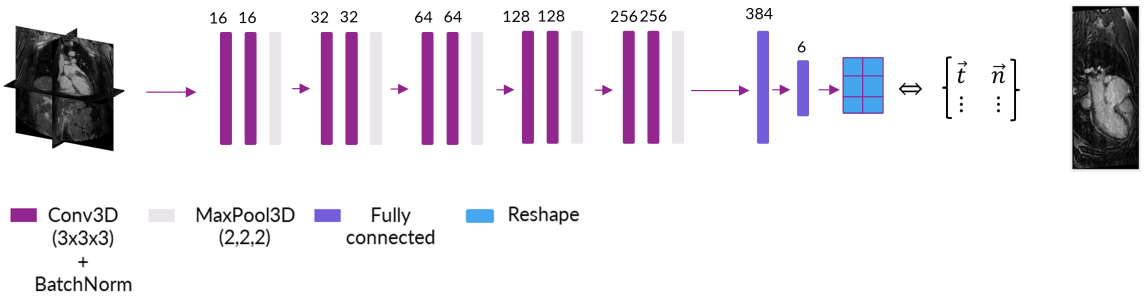


Figure 3.7: Baseline model architecture. Numbers on top of the convolutional and dense layers depict the number of filters and units, respectively.

The chosen baseline architecture is inspired from Chen et al [14] and consists of 5 convolutional blocks with an increasing number of units and a downsampling factor of 2. For a given depth index $n, n = 0, \ldots, 4$, the respective convolutional block comprises of 2 3D convolutional layers with number of filters equal to 16×2^n and an isometric 3 unit kernel, followed by a 2x2x2 Max Pooling layer. After each convolutional layer there is a Batch Normalisation layer that acts as an intermediate standardisation step before applying the non-Gaussian activation function as suggested in the original paper [41]. The activation function employed is a Leaky ReLu used preemptively to avoid the dying neuron issue, while still remaining effective at addressing the vanishing gradient problem.

At the end of the convolutional feature extractor block, the regression of the position and orientation vectors is done through a single dense layer of 384 units connected to 6 final units that are then reshaped into the matrix shown in Figure 3.7 corresponding to the plane position ($\overrightarrow{t}$) and normal vectors $\overrightarrow{n}$. In this first approach, the architecture used was one where every single layer is shared between the two

tasks, taking inspiration on the approach proposed by Corrado et al [15]. Figure 3.7 depicts a visual representation of this baseline network architecture.

L2 weight regularisation [42] is used to avoid the layers' weights from becoming excessively large which leads to overly complex solutions and thus overfitting. Still regarding the layers' weights, the kernel initialiser used was Xavier Uniform, also called Glorot Uniform, [43] which has been known to improve convergence.

3.1.6 Performance Metrics

Performance on the position prediction sub task is assessed by the displacement error (ε_d) which, just like the position loss, is computed as the distance between the predicted point ($\overrightarrow{t}_{pred}$) and the plane in millimetres.

$$\varepsilon_d = ||\overrightarrow{d_p}|| = |\overrightarrow{d_c} \cdot \overrightarrow{n}|, \quad \text{with} \quad \overrightarrow{d_c} = \overrightarrow{t} - \overrightarrow{t}_{pred} \tag{3.9}$$

The orientation sub task performance is assessed through the angulation error (ε_θ), which is the angle between the predicted normal vector to the plane ($\overrightarrow{n}_{pred}$) and the line defined by the ground truth plane normal vector ($\overrightarrow{n}$) in degrees. In practice, this metric can be defined as function of the angle θ between the predicted and true normal vectors (Figure 3.6). Just like the loss, this metric also considers both $\overrightarrow{n}$ and $-\overrightarrow{n}$ as equally correct solutions.

$$\varepsilon_\theta = \begin{cases} \theta & \text{for } \theta \leq 90 \\ 180 - \theta & \text{for } \theta > 90 \end{cases} \tag{3.10}$$

3.1.7 Training Details

As an optimiser we used ADAM with its default parameters as defined in its Keras implementation. The learning rate was set to 10^{-3}. Weight decay was also added to the loss using a weight of 1e-6. Batch size was fixed to 16 samples as more than that would not fit in our GPU's RAM and less would lead to a too unstable training where the gradients estimated are more likely to not accurately represent the overall dataset.

The validation loss was monitored during training with the early stopping callback [44] parameterised with a 50 epoch patience, which ensures the training stops after 50 epochs with no validation loss decrease. Also, to ensure the reproducibility of our findings, a *seeding* function was also implemented and ran before each training.

Given the chosen batch size and preprocessing pipeline, training for one epoch (N = 88 samples) takes approximately 40 seconds. Inference times are under 1 second, excluding volume reslicing.

3.1.8 Experimental Setup

The experiments were conducted using Python version 3.10.12 and Keras (Keras: The Python Deep Learning Library https://keras.io/) with TensorFlow (https://www.tensorflow.org/) backend (version 2.12.0). All models were independently trained on a Google Colab Pro instance used was equipped with a 15.35GB Tesla T4 GPU. The complete code base for the experiments is available open source at the following GitHub repository: [https://github.com/pedr0sorio/DeepCardioPlanner].

3.2 Results and Discussion

3.2.1 Baseline Performance

The baseline model consisted of a network architecture N_A using $\mathcal{L}_{pos}$ and $\mathcal{L}_{ori}$ as position and orientation losses, respectively (see section 3.1.4). The loss weighting approach at this stage was simple summing each sub-task loss which corresponds to using a loss weighting approach from Equation 3.1.4.C with w_{pos} and w_{ori} both equal to one. In addition, there was no data augmentation yet introduced during the training at this point and the learning rate was arbitrarily set to 10^{-3}. Also, at this point no variance adjustment had been done to the dataset.

These settings were used to train 4 models, one per each standard view plane, whose prediction performance on the test set can be observed via box plots in Figure 3.8. The evolution of the two losses during each training can also be analysed in Figure 3.9.

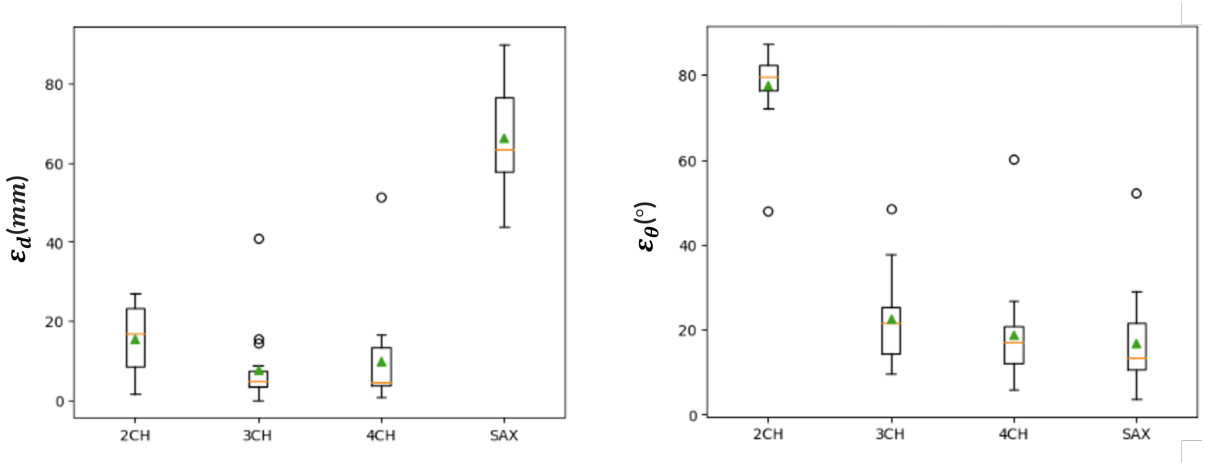


Figure 3.8: Distribution of the displacement (ε_d) and angulation errors (ε_θ) on the test set. Mean error value marked with green triangle and median marked with orange bar.

With the exception of the orientation loss in 2CH, it is observable that all the loss functions demonstrate a decreasing pattern and reach a state of stability over time, albeit with some noise and irregularities most likely due to the heterogeneity of our dataset. This indicates that the models are learning both tasks

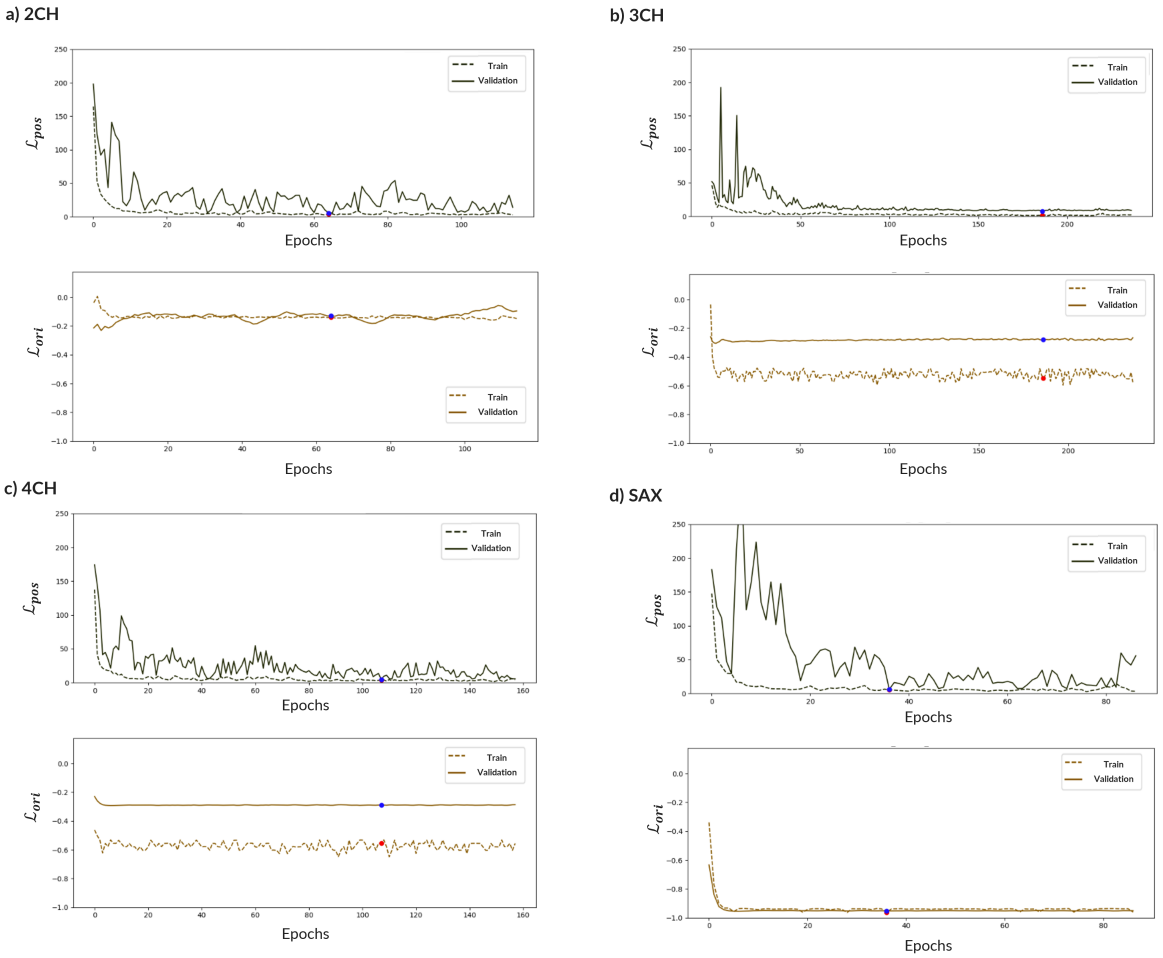


Figure 3.9: Training and validation curves for each task specific loss. Validation curve plotted as a full line and the train curve as a dotted one. Dark brown curves depict the position loss and the lighter brown the orientation one. Best epoch is marked with a blue and red dot on the validation and training curves, respectively.

to some extent. The 4CH and 3CH view models depict a larger gap between the train and validation performance (i.e. variance as defined in 2.2.1.B) on the orientation prediction task, which is clear from the distance between the two curves in the respective loss plots (Figure 3.9).

Also, it is relevant to note that the SAX predicting model appears to have had a shorter training than the others, as it trained for less epochs. Starting from approximately the 30th epoch, the position loss does not seem to further decrease and, since no improvements are seen after 50 epochs, our early stopping callback halts the training process. However, the smaller displacement error obtained for the SAX imaging planes, may indicate that early stopping might have stopped the training too early, not giving a chance for the model to properly model the position of this type of view planes. In fact, early stopping is triggered by a brief dip in the position loss at the highlighted best epoch. This indicates that the model achieved a configuration where it accurately predicts the position for both the validation and training sets. However, this configuration fails to generalise well to the test set, resulting in decreased performance. Again, due to the limited size and heterogeneity of our dataset, the model has the potential to discover suboptimal configurations that accurately predict only specific subsets of the data. As a consequence, this results in more erratic learning curves, which can trigger early stopping prematurely before the model has sufficient time to stabilise. Due to these findings, it is crucial to properly adjust the early stopping patience parameter. Specifically, in the context of SAX, it is recommended to consider increasing it. Note that we can say it was the dip and the unstable position loss that led to the early stopping setting off for two reasons. Firstly, the orientation loss remains relatively stable throughout, without exhibiting notable fluctuations. Secondly, the position loss displays a significantly larger scale compared to the orientation loss. Consequently, since the joint loss is simply the sum of these individual losses, the variations in the position loss primarily determine the behaviour of the overall joint loss.

Looking specifically at the 2CH orientation loss curve, we find that, contrarily to their position counterpart, both train and validation losses do not have a clear decreasing behaviour and, despite stabilising, they do so at a loss value of -0.1, which approximates to about 84°angulation error. We hypothesise that this might be due to the nature of the 2CH view dataset. As we have seen before, the plane orientations for the 2CH dataset follow a bimodal distribution with an approximately 180°difference between the two modes and hence with plane normal vectors approximately symmetric to each other (see figure 3.3).

Considering that this bimodal orientation distribution arises from the absence of a standardised acquisition convention between operators rather than anatomical or physiological factors, the model faces challenges in distinguishing between these two cases based on input volume alone. Consequently, it is reasonable to assume that, during training, the model becomes trapped in an intermediate optimal solution point positioned between the two modes. This point corresponds to predicting normal vectors that minimise the angulation error in relation to both trends, resulting in an approximate angulation

error of 90°, which, considering that the cosine of 90°is 0, it is quite close to where the cosine similarity loss has stabilised ($\mathcal{L}_{ori} \sim -0.1 \Leftrightarrow \varepsilon_\theta \sim 84\,\text{deg}$) for 2CH view model. Probably, the model converges at approximately 84°instead of 90°because one of the modes is slightly more populated than the other and the angulation between the two modes is also not exactly 180°. Through this rationing, we should also expect the performance at the 4CH orientation prediction task to be affected by this phenomenon as the 4CH orientation distribution is also bimodal and centred at approximately 90°(see Figure 3.3), however less intensely as the proportion of samples in the second mode is not as large as the one in the first one.

We have also seen how in practise these two modes actually represent view planes with very similar orientation since rotating a plane 180°does not change the slice it represents. Therefore, it is necessary to ensure that these two modes point to the same general solution. By doing so, we avoid that these 2CH and 4CH bimodal distributions lead to gradients pulling in opposite directions without any corresponding anatomical changes visible in the volume that would justify it.

With that in mind, we tested two different approaches. The first one was the previously explained variance adjustment that turns the 2CH and 4CH bimodal orientation distributions to a single mode and the second one was the usage of a modified cosine similarity orientation loss function ($\mathcal{L}_{ori_{mod}}$) also already described in 3.1.4.

3.2.2 Addressing symmetrical bimodal orientation distributions

3.2.2.A Variance Adjustment

Figures 3.10 and 3.11 provide an overview of the 2CH and 4CH view predicting models' performance before and after adjusting their datasets as described in section 3.1.3.B.

This adjustment translates to flipping the view planes from the minority mode by 180 degrees, which creates as single mode distribution that we should be able to model with the defined cosine similarity orientation loss ($\mathcal{L}_{ori}$).

After applying this adjustment both orientation loss curves start displaying a regular decreasing behaviour and stabilise much closer to the target loss value ($\mathcal{L}_{ori} \sim -1 \Leftrightarrow \varepsilon_\theta \sim 0\,\text{deg}$). It is also relevant to say that the gap between the train and validation loss also decreases significantly for the 4CH model, as the curves appear closer after the adjustment.

The effect of the adjustment is visible not only on the learning curves, but naturally also on the test set performance. The test set angulation error distributions become closer to zero for both view models, being worth noticing a remarkably large decrease in mean angulation error for the 2CH prediction. The displacement error distribution did not show improvement after the fix but it was also not expected to do so. Hyperparameter tuning is still needed to close push these errors down.

Our hypothesis seems to agree with these results as, now, with no bimodal orientation distribution, the 2CH model no longer gets confounded by the variance introduced by the non adoption of a standardised

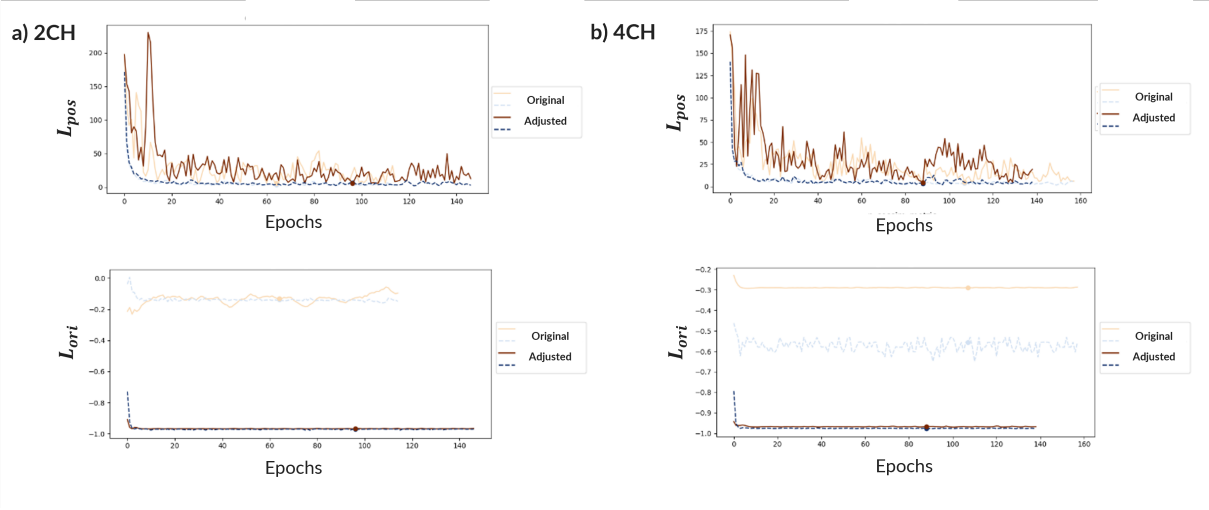


Figure 3.10: Training and validation curves for each task specific loss, for each view model trained with and without variance adjusted datasets. Train and validation curves from the model trained without a variance adjusted dataset are depicted with the lighter shades of blue and orange, respectively, whereas the darker shades are for the models trained with it. Validation curve plotted as a full line and the train curve as a dotted one. Best epoch is marked with a blue and red dot on the validation and training curves, respectively.

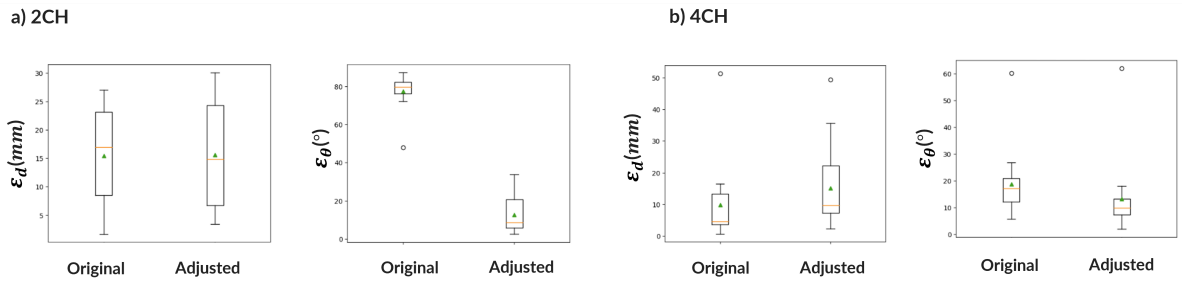


Figure 3.11: Distribution of the displacement (ε_d) and angulation errors (ε_θ) on the test set according to view and use of variance adjusted dataset. Mean error value marked with green triangle and median marked with orange bar.

acquisition convention and can now focus on using actual anatomy to prescribe the view. We could see that our fix also had its positive effect on the 4CH model performance although to a lesser extent, which also agrees with our hypothesis.

3.2.2.B Modified cosine similarity

Figures 3.12 and 3.13 provides an overview of the 2CH and 4CH view predicting models' performance when using the simple cosine similarity for the orientation loss ($\mathcal{L}_{ori}$) and when using its modified counterpart ($\mathcal{L}_{ori_{mod}}$) described in section 3.1.4. No variance adjustment was performed on the datasets for this analysis.

As an addition to the previous performance summary from Figures 3.10 and 3.11, $\mathcal{L}_{ori_{mod}}$ has been tracked as a metric during training even when is was not used as loss. Conversely, $\mathcal{L}_{ori}$ has also been tracked in both settings even when not being used as loss. This will be useful to further understand how this new loss works.

For both views, the new orientation loss has the desired descending behaviour and converges close to zero which in this case is the optimal value. Also, just like before with the variance adjustment, the test set orientation performances improve significantly as the metrics' distribution become closer to zero for both view model, being worth noticing a remarkably large decrease in mean ε_θ for the 2CH prediction. As before, and as expected, the ε_d distribution did not show improvement after training with the new loss. Hyperparameter tuning is still needed to reduce this errors.

Looking at the evolution of the $\mathcal{L}_{ori}$ while training with $\mathcal{L}_{ori_{mod}}$ (Figure 3.12 middle left plot), we can see that the training and validation curves stabilise at different values. For the 2CH model, the training curve stays at roughly 0, whereas the validation one stays at approximately -0.4. For the 4CH model, the train one stays at roughly -0.6 whereas the validation one stays at approximately -0.3. In fact, the model is being trained to approximate the target plane's orientation while also considering its 180°flipped version as a viable target. Consequently, the model predictions will follow any of the modes of the bimodal distribution we saw before, regardless of the mode the ground truth label vector actually belonged to. In practice, this means the model has two viable solutions it can approximate: the $\overrightarrow{n}$ and $-\overrightarrow{n}$, thus it will approximate whichever version of the normal vector provides a stronger gradient. In a batch we can then have predictions that are approximately 180°flipped form their actual ground truth. Since $\mathcal{L}_{ori}$ quantifies the angulation error only with respect to $\overrightarrow{n}$ it will not correctly represent how accurate the predictions actually are. The $\mathcal{L}_{ori}$ metric can then stabilise at different values depending on the proportion of predictions that approximate $-\overrightarrow{n}$ as opposed to $\overrightarrow{n}$ as well as on their overall accuracy. Figure 3.14 illustrates this phenomenon.

Once more, our hypothesis seems to agree with these results as, by redesigning the orientation loss function to take both the ground truth normal vector and its symmetric as optimal solutions, samples

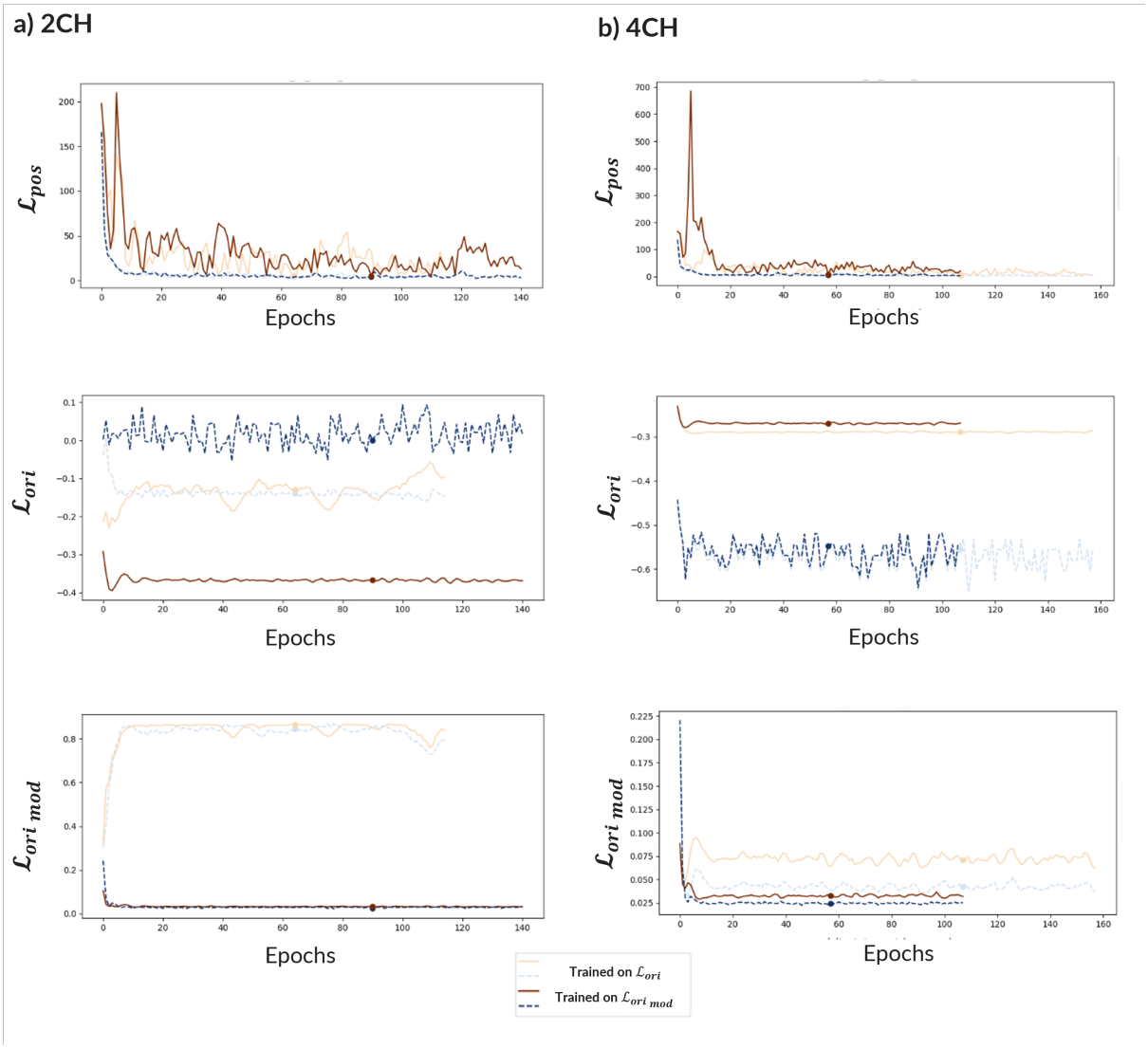


Figure 3.12: Training and validation curves for each task specific loss for each view model when training with $\mathcal{L}_{ori_{mod}}$ or with $\mathcal{L}_{ori}$. Train and validation curves from the model trained with $\mathcal{L}_{ori}$ are depicted with the lighter shades of blue and orange, respectively, whereas the darker shades are for the models trained with $\mathcal{L}_{ori_{mod}}$. Validation curve plotted as a full line and the train curve as a dotted one. Best epoch is marked with a blue and red dot on the validation and training curves, respectively.

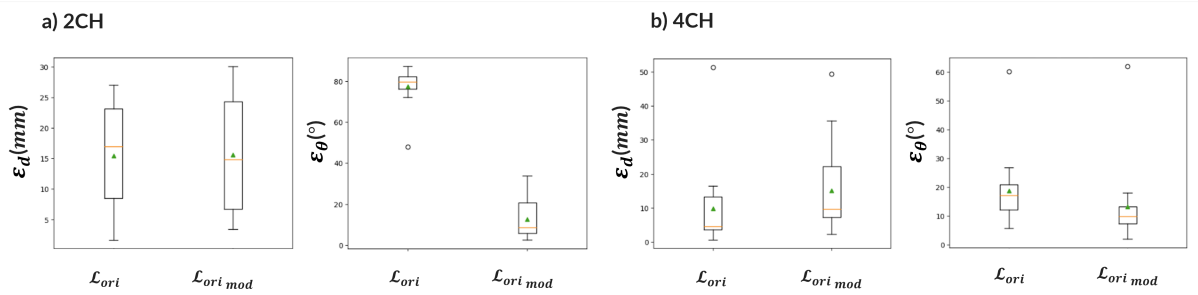


Figure 3.13: Distribution of the displacement (ε_d) and angulation errors (ε_θ) on the test set according to view and the orientation loss function used. Mean error value marked with green triangle and median marked with orange bar.

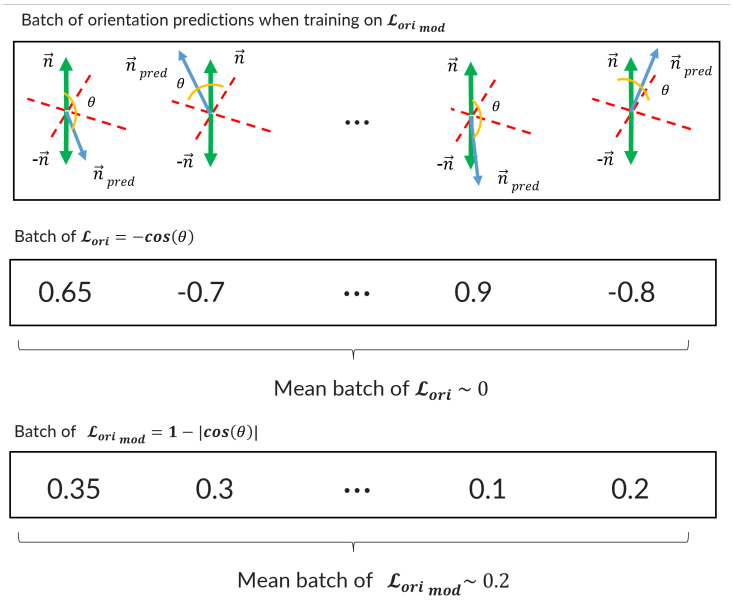


Figure 3.14: Hypothetical batch of orientation predictions with performances quantified based on $\mathcal{L}_{ori}$ and $\mathcal{L}_{ori_{mod}}$ when training on the $\mathcal{L}_{ori_{mod}}$. Distribution of the $\mathcal{L}_{ori}$ metric on a batch of orientation prediction while training on $\mathcal{L}_{ori_{mod}}$ will not represent how accurate the predictions are. Depending on the number of predictions that approximate $-\overrightarrow{n}$ as opposed to $\overrightarrow{n}$ as well as on their overall accuracy the $\mathcal{L}_{ori}$ metric can stabilise at different mean batch values. In this case we try to replicate the scenario observed for the 2CH model.

from opposing modes of the 2CH and 4CH plane orientation distributions will provide gradients towards the same direction, thus removing the issue from before and allowing both models to better learn the orientation task.

Both this approach and the previous aim to address the same issue and managed to solve it successfully. Moreover, from the analysis of Figures 3.11 and 3.13 the performance gain that each approach provides is essentially the same. Since there is no reason to assume that these two approaches would be mutually exclusive, we opted to use the $\mathcal{L}_{ori_{mod}}$ loss for training jointly with the variance adjustment to the dataset for the remaining experiments. Nevertheless, it is relevant to note that the redesign of the loss consists of a better solution to the problem we previously described as it does not require the manipulation of the training data.

Note that this new loss function has the potential to help the models reach convergence even when their orientation distribution is not symmetrically bimodal. The fact that this loss enables two possible solutions, as opposed to one should ease the task for all the view prediction tasks, thus this change was applied to all view models.

3.2.3 Variance Adjusted and Modified Orientation Loss Baseline

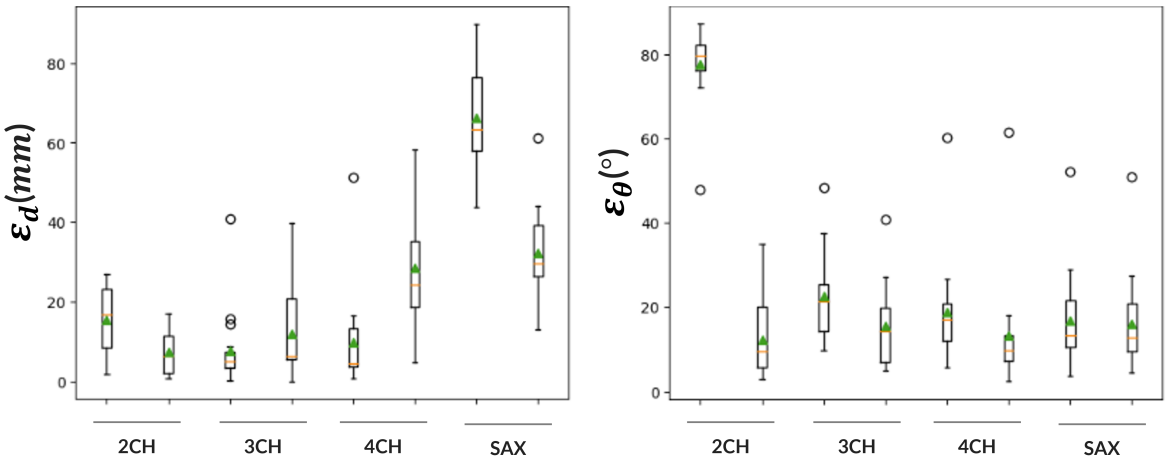


Figure 3.15: Distribution of the performance metrics on the test set for all view models before and after changing orientation loss to $\mathcal{L}_{ori_{mod}}$ and applying the label adjustment.

In this section, we present a new baseline with the aforementioned changes and analyse the results.

Figure 3.15 displays the performance metrics before and after the changes. We observe that the orientation performance improves for every view after implementing the changes. This indicates that the modifications positively affect the models' ability to accurately predict the orientation. However, the position performance only improves for the 2CH and SAX views. Further analysis of the curves will help us better understand the underlying factors that contribute to this behaviour.

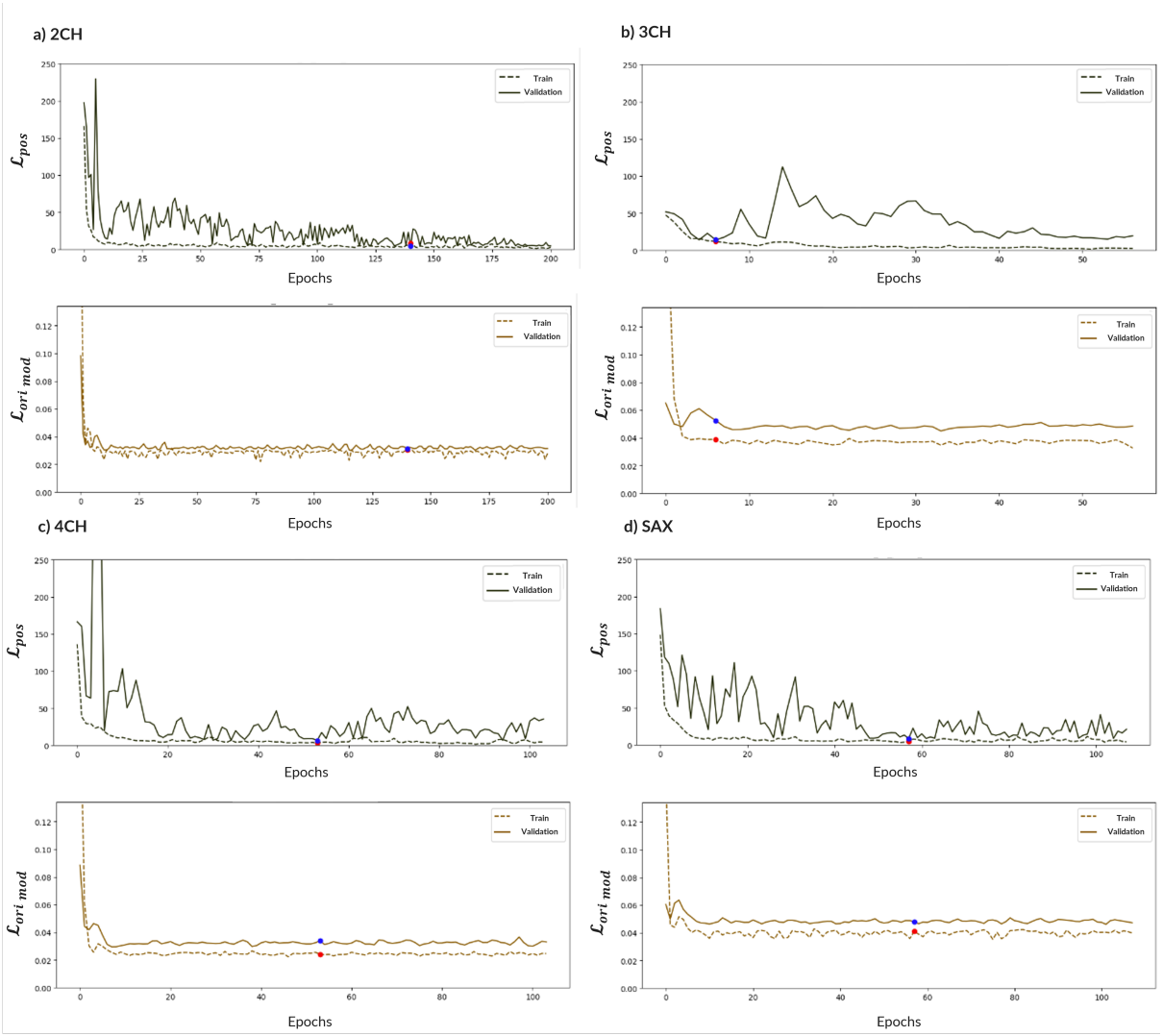


Figure 3.16: Training curves for all view models after changing orientation loss to $\mathcal{L}_{ori_{mod}}$ and applying the label adjustment. Best epoch is marked with a blue and red dot on the validation and training curves, respectively.

Figure 3.16 depicts the evolution of the training and validation losses for each view model during training. Apart from the position loss of the 3CH model, after these changes, all the curves depict the regular descending behaviour. In general, the training performance is better than the validation one which makes sense as the validation subset includes data never seen by the model whereas the training set is the one used for optimising the model and updating its weights.

Noise and irregularities are still noticeable in the curves and affect our results, but this time in a different way. The SAX training that had been cut short in the previous baseline setting is now longer, even without modifying the patience parameter. Its position performance improves because of this. On the other hand, in the case of the 3CH model, the fluctuations in position loss lead to the training being cut short. The best epoch for this model was the 6th, indicating that the model did not have time to fully learn the task. This is evident in the performance metric boxplots shown on Figure 3.15, which show a shift of the 3CH ε_d test distribution towards larger values compared to before. Similarly, the 4CH model also shows a decrease in test set position performance from the baseline setting to this one. The noisy and irregular nature of our curves seem to be, once more, at heart of the problem. Compared to before, the training has been cut short as the best epoch had been found after the 100th epoch while now the best epoch is around the 60th. Increasing the patience parameter may help address this issue, as we will discuss in the next subsection.

3.2.3.A Early Stopping Patience Parameter

Figures 3.17 and 3.18 illustrate the curves and test performance after increasing the patience parameter from 50 to 100 on the 3CH and 4CH view model training, respectively. It is clear that a larger patience increases the performance of the final model as it gives more time for them to stabilise and prevents it from getting stuck in early suboptimal solutions. Indeed this parameter needs to be adjusted to account for the heterogeneity of our data and consequent noisy learning curves.

Despite the sets of curves we see in these figures being for the same model with the exact same set of hyperparameters apart from the early stopping patience, there is still a slight difference between their behaviour in the beginning of the training. This is especially noticeable for the 3CH model (Figure 3.17). Since we have fixed the seed for all random number generators before each training we would expect this not to happen. Nevertheless, some stochastic variability inherent to the GPU remains beyond our control.

3.2.3.B Underfitting and Next Steps

While comparing the behaviour between the position and orientation curves across models, we find that they reach convergence at different points during training. The orientation loss reaches convergence much faster than the position loss. Specifically, the learning of the orientation task seems to occur solely in the

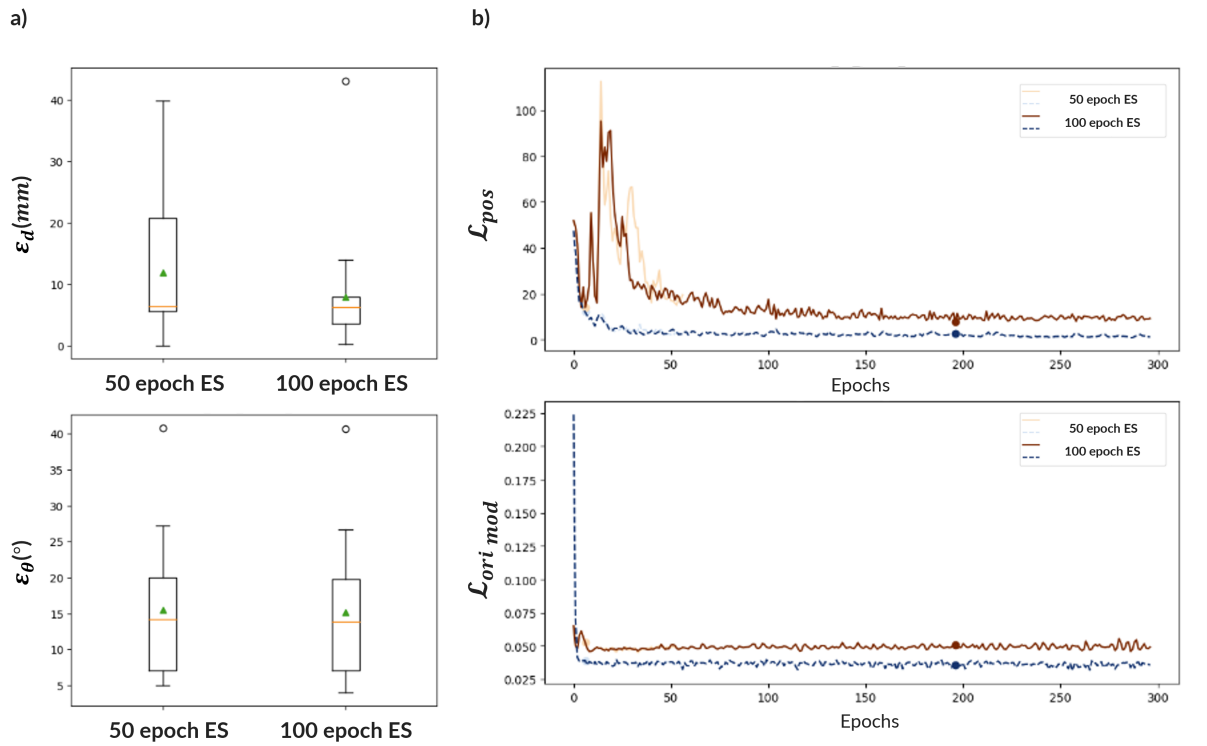


Figure 3.17: 3CH view model test set performance (a) and curves (b) before and after increasing the early stopping patience parameter from 50 epochs to 100. Validation curve plotted as a full line and the train curve as a dotted one. Best epoch is marked with a blue and red dot on the validation and training curves, respectively.

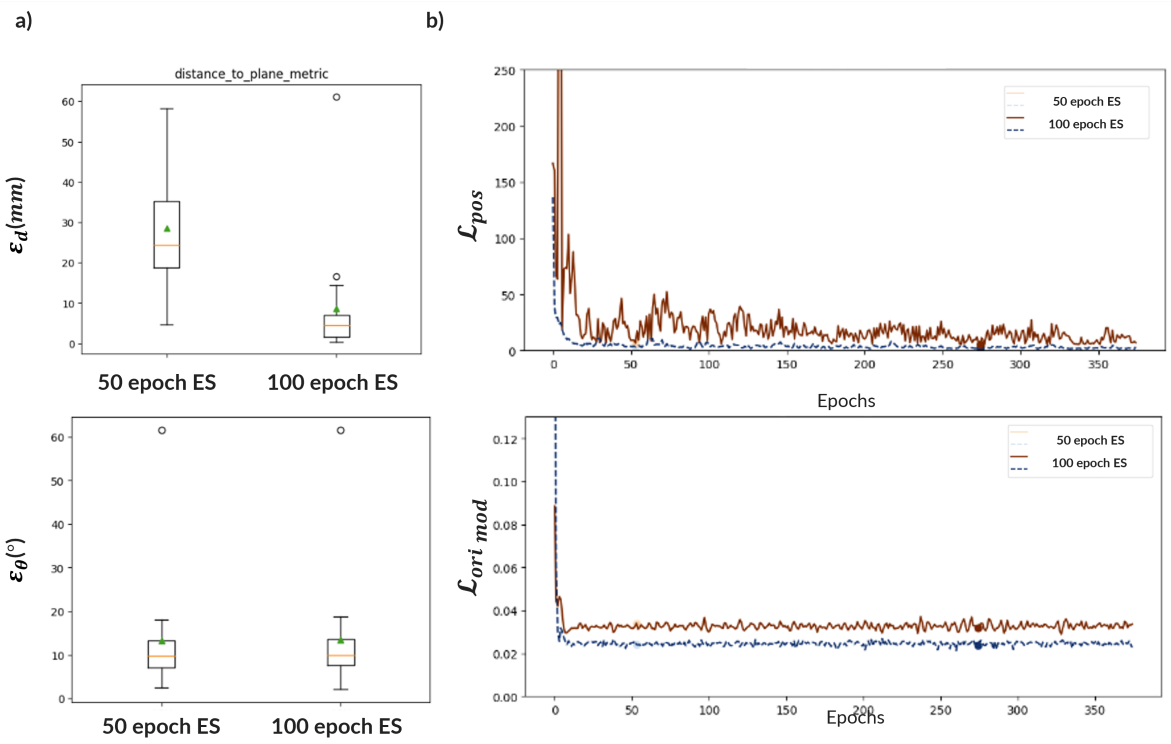


Figure 3.18: 4CH view model test set performance (a) and curves (b) before and after increasing the early stopping patience parameter from 50 epochs to 100. Validation curve plotted as a full line and the train curve as a dotted one. Best epoch is marked with a blue and red dot on the validation and training curves, respectively.

first few epochs, as the loss quickly drops in the beginning and then remains practically unchanged. One would expect that this behaviour would set off early stopping earlier in the training, however, one has to consider that the joint validation loss composed by the simple sum of the two losses is what is being tracked. Consequently, since the scale of the position loss is so much larger than the orientation one, the descending behaviour of the former allows the training to continue.

The distance between the train and validation curves in both tasks is less significant for the 2CH model than for the other models. Also, the general trend across all models is that the variance in the orientation task seems to be very stable all throughout the training and the training loss is also stabilising early in the training at a value much larger than its best theoretical value of 0. These observations might indicate that our models are underfitting the orientation prediction task. There are multiple factors that could be limiting the effective capacity of these models to a degree where they cannot properly model our target tasks. The complexity of the chosen model architecture could be insufficient, the loss weighting strategy could be inadequate or there might be the need for tuning important hyperparameters like the learning rate. These observations pave the way for further experiments, which we will discuss in the subsequent chapters.

4

Single Branch Network - Automated 2CH view prediction optimisation

Contents

In the last chapter, we have addressed some of the challenges that arise from our dataset and defined a new performance baseline. In this one, we act upon the observations we have made regarding model capacity and underfitting. In particular, we will analyse model complexity, experiment with different loss weighting strategies and tune the learning rate.

However, conducting the aforementioned analysis on all 4 view predicting models is unfeasible due to resource limitations and time constraints. Therefore, we employ the 2CH view prediction task as a representative task to investigate and implement changes to the models and training settings. We anticipate that these modifications will generalise to the remaining view prediction tasks as all the models from the last chapter seem to suffer from the same orientation task underfitting issue.

4.1 Methodology

In this chapter, the methodology in use is exactly the same as the one from the previous chapter, however with a few new features. Their description will follow below.

4.1.1 Data Augmentation

At this point, data augmentation is introduced to the preprocessing pipeline previously described in 3.1.3.

The major obstacle to obtaining models with high generalisation ability and high performances is the amount of data available to train the networks. Data augmentation is usually employed to help mitigate this issue. With this method, new data is created through a series of transformations to the input that are aimed to replicate the variance usually seen in MRI imaging. By randomly introducing these changes into the training samples, it is possible to steer the model towards invariance to them. One must take caution, however, to only introduce realistic changes, otherwise generalisation ability is not improved.

Considering the limited size and diverse nature of the dataset, there is a high risk that the trained models may learn features that are overly specific to the training set, resulting in poor generalisation to other subsets. To address this concern, a comprehensive augmentation pipeline was implemented. Yet, since the task being addressed is the regression of vectors that describe the position and orientation of a plane within a volume, one must take into account the possible changes that introducing spatial transformations during augmentation can have on the label vectors. Data augmentation for this problem is then more complex and not as easily implementable as in other tasks, since an extra step of label updating during augmentation must also be ensured.

It is possible to categorise the transformations applied during augmentation into label affecting ones and non label affecting ones. The transformations used are described below and shown in Figure 4.1

Non Label Affecting Transformations

Intensity Scaling: It consists of an element wise multiplication of the 3D scan with a 3D kernel composed of various overlapped Gaussian distributions centred at random locations. The kernel is normalised between 1 and a tunable parameter larger than 1; The larger this parameter, the larger the scaling.

Noise addition: Additive zero mean Gaussian noise is applied to the input scan. The standard deviation of this noise is estimated from the one observed in the background of the scan. The voxels referring to the background are defined as the 10% lowest intensity ones. For each input volume, the standard deviation of the background voxel intensity is computed and then multiplied by a random factor sampled from a half-normal distribution. The width of that distribution is regulated via a positive tunable parameter.

Label Affecting Transformations

Translation, Rotation and Scaling This process requires user the specification of the number of pixels for translation in each direction, number of degrees for rotation about each axis and a single scale factor. The actual parameters used for the transformations are sampled from normal distributions with mean zero and standard deviation equal to the previously mentioned input parameters. This method allows for translations to be in either way of each axis, rotations can be in either the positive or negative direction and scaling to be either expansion or compression. All these geometrical transformations are applied together to the volume through an affine transformation operation, relying on an affine transformation matrix that is built from the input parameters. To ensure that the label-volume correspondence is not lost, the labels are updated by using this same affine transform matrix to generate the new position and orientation vectors that define the same view plane but now in the augmented volume.

4.1.1.A Custom Data Generator

The custom data generator described previously in 3.1.3.D, is now performing an extra set of operations related to data augmentation. Figure 4.2 presents the updated pipeline.

Another reason to opt for this custom data generator is that the type of augmentation intended to use includes specific operations, like label updating, that Keras does not provide in the pre-implemented utility functions.

4.1.2 Loss Weighting

The weighted sum with task specific loss weights has been the approach taken in the previous chapter (see Equation 3.1.4.C). The appropriate choice of weighting between each loss has a significant effect on the model performance, thus requiring appropriate tuning. In this chapter we will experiment different loss weight combinations.

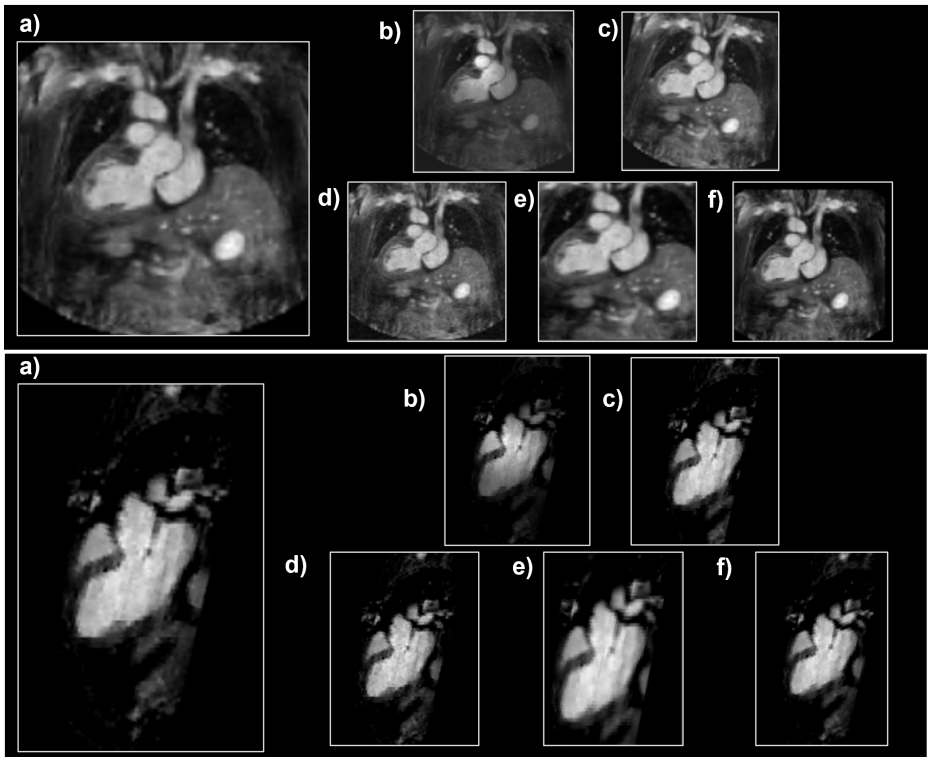


Figure 4.1: Augmentation employed during the training process shown on both the mid coronal slice of the input 3D scan (top) as well as on its 3CH view plane (bottom). Original (a) and augmented slices using intensity scaling (b), rotation (c), noise addition (d), scaling (e) and translation (f).

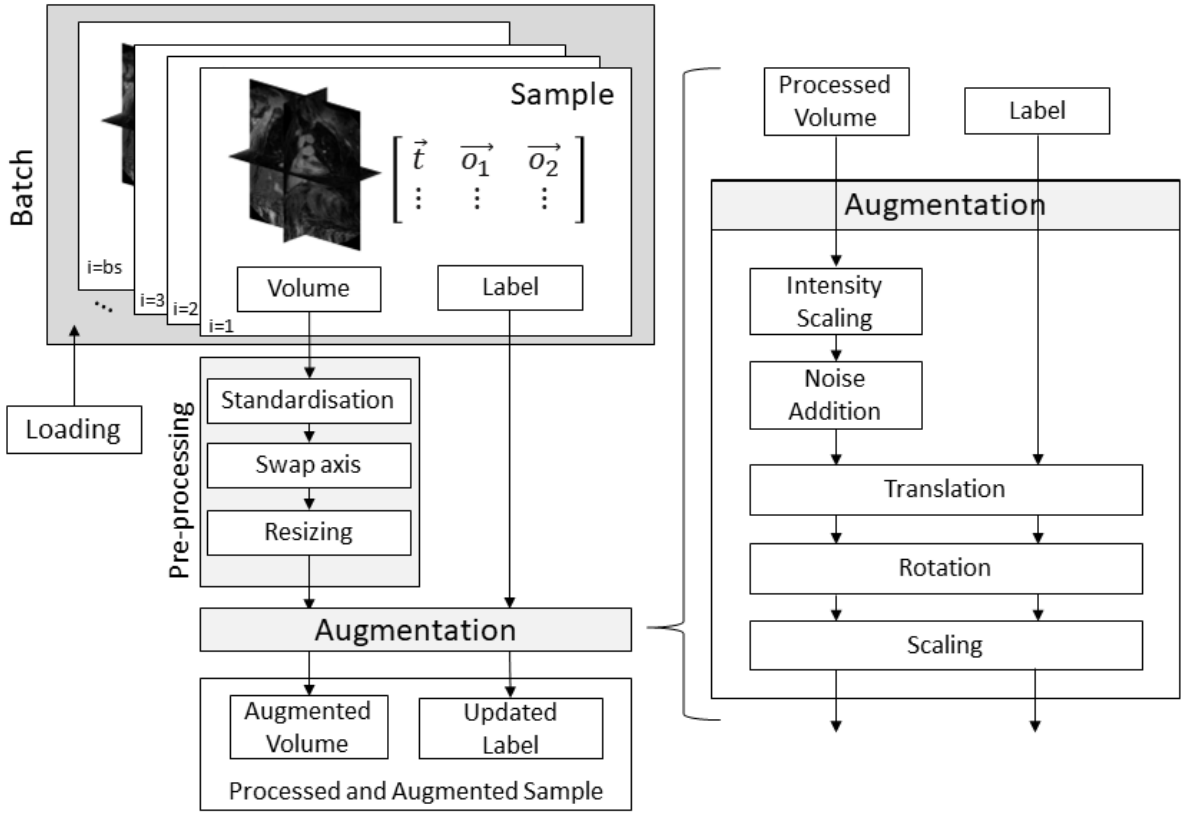


Figure 4.2: Loading, pre-processing and augmentation pipeline. All the mentioned operations, from the loading to augmentation, are performed to every sample in a batch during training before feeding them into the network.

However, this search for the optimal weights is computationally expensive and difficult to resolve with manual tuning, therefore another approach was explored. The loss weights were included as learnable parameters of the network through homoscedastic uncertainty [34].

A detailed derivation and explanation of this method can be found in 2.2.2.B. By generalising from Equation 2.2.2.B to our specific problem we get the following learning objective:

$$\mathcal{L}(W, \sigma_{pos}, \sigma_{ori}) = \frac{1}{2\sigma_{pos}^2} \cdot \mathcal{L}_{pos}(W) + \frac{1}{2\sigma_{ori}^2} \cdot \mathcal{L}_{ori}(W) + \log(\sigma_{pos}\sigma_{ori}) \tag{4.1}$$

Connecting to the initial equation for the joint objective function, the weights from Equation 3.1.4.C can be given by: $w_{pos} = \frac{1}{2\sigma_{pos}^2}$ and $w_{ori} = \frac{1}{2\sigma_{ori}^2}$

In order to use this method for loss weighting in our Keras code, a custom loss layer had to be implemented to allow the weights to be updated during the training via an approach adapted from [45]. To monitor the value of the loss weights and the regularisation factor throughout the training, a custom Keras callback had to be implemented.

4.2 Results and Discussion

4.2.1 Assessing model complexity

The complexity of the model architecture can affect model performance. While very complex architectures tend to make overfitting more likely due to the large amount of tunable parameters, a overly simple architecture can prevent the models from properly learning each task. This effect can be further understood through the concepts of model capacity and hypothesis space introduced in 2.

Given the previous analysis of the learning curves from Figure 3.16 in the previous chapter, it is relevant to assess whether the chosen network architecture carries enough complexity to model our target tasks, especially the orientation one.

The first *sanity check* we can do to evaluate that, is analysing if our chosen architecture is complex enough to overfit the training set in a Single-Task Learning (STL) setting, where each task is learned independently. This is useful because, in case the model does not have the necessary complexity to overfit each task separately, then it will not be able to model both simultaneously and hence a more complex architecture should be employed. In addition, it will provide an STL baseline so that we can analyse the inter dependency between the two defined tasks.

Setting the position prediction task loss weight to zero allows us to update the weights of the model solely based on the gradients coming from the orientation prediction task, and vice versa (Equation 3.1.4.C).

Figures 4.3 and 4.4 provides an overview of the 2CH view predicting model performance in both tasks

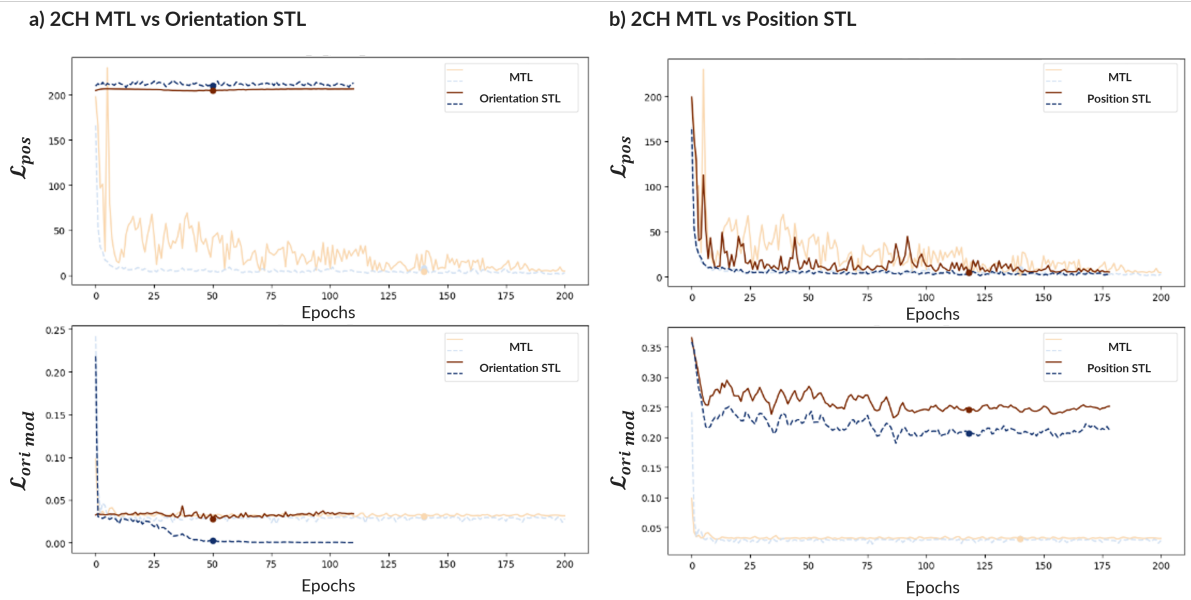


Figure 4.3: Training and validation curves for each task specific loss for the 2CH view model when training on a MTL or on a STL setting. Validation curve plotted as a full line and the train curve as a dotted one. Best epoch is marked with a blue and red dot on the validation and training curves, respectively.

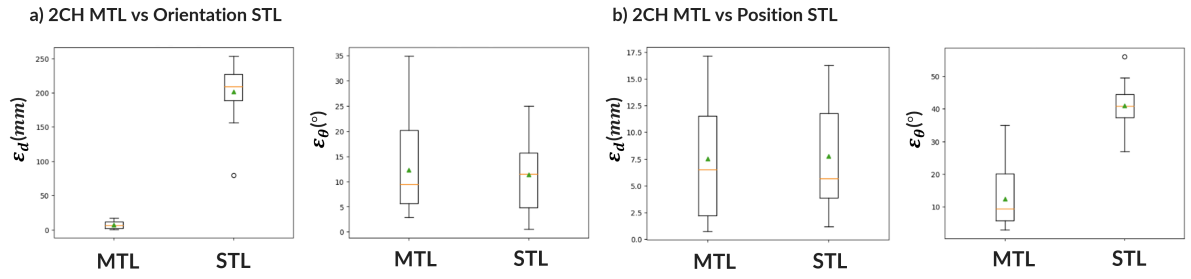


Figure 4.4: Distribution of the displacement (ε_d) and angulation errors (ε_θ) on the test set according to type of learning approach used, MTL or STL, for the 2CH prediction task. Mean error value marked with green triangle and median marked with orange bar.

when training on a Multi Task Learning (MTL) setting or when training on a STL setting.

Firstly, in both STL scenarios, the losses exhibit the desired decreasing behaviour and stabilise during training.

In the position STL setting ((Figure 4.3b)), the position validation loss seems to follow the train one more closely than before for MTL, as the later spikes to larger values than the former throughout the training. This, however, does not translate into the final test performances, as there is no appreciable improvement of the error distributions. These differing behaviours might be due to the high heterogeneity of our dataset, which might have lead to a test split with distinct features unseen in the validation one.

In the orientation STL setting (Figure 4.3a), we see the validation loss curve closely following the one from the MTL setting. On the other hand, the training loss dips to much lower values than what we had seen before in MTL regime, diverging from the validation curve. It is evident that the model, now that there is no information sharing between the tasks, has a more unconstrained hypothesis space, larger effective capacity and thus is able to overfit this task. Since, early stopping is in place, the training is stopped as soon as that happens and the model with best validation performance is saved. In the end, this STL model achieves a test set angulation error distribution that, while not necessarily better than the MTL one, does span over an interval closer to zero.

The main takeaways are that, since both STL models manage to have their training loss converge at values very close to zero, then there is enough complexity in the architecture to model each task separately. Despite this, it could still be argued that the complexity of the model might not be enough to satisfactorily solve both tasks simultaneously. We hypothesise that this is not the case and further experiments were conducted to evaluate other factors that might have contributed to the underfitting seen in the MTL setting.

4.2.1.A Inter-Task Dependency

There are, however, some other remarks one can make regarding these STL models. If we look at the performance of the orientation STL model we can see that it learned only the orientation task, as the predictions it provides for the plane positions are completely random and do not seem to improve as the training continues. This makes sense as the position loss is not being considered for the joint loss in this scenario and hence the network weight update happens only towards minimising the orientation loss.

Interestingly, the position STL model exhibits a distinct behaviour in comparison. Even when the orientation task loss weight is set to 0, there is an observable progressive improvement in $\mathcal{L}_{ori_{mode}}$ metric during the training process. This phenomenon is called knowledge transfer and arises from the intrinsic dependency between the two tasks we aim to address. There exists shared and complementary information between the position and orientation tasks, enabling the model to extract features that benefit both objectives. This observation underscores the potential of multi-task learning in enhancing overall

performance. By simultaneously optimising the model's parameters using both losses, we can potentially leverage these shared patterns to yield positive impacts in performance.

This dependency between the two tasks can be further understood from a physical perspective as the position loss has been set to be the distance between the view plane and a point, forcing the model to approximate a whole surface, instead of a single point, which implicitly requires orientation information as well.

4.2.2 Orientation loss scaling

During the training process of our models, we have been employing a straight forward approach of summing each task-specific loss. However, these losses exhibit significant differences in scale, with the position loss measured in the order of tens of millimetres and the orientation loss ranging from 0 to 1. (see section 3.1.4).

This scale mismatch can very much be at the heart of the different learning behaviours we see for the two tasks. In the current loss weight setting, since the position loss has a much larger relative contribution to the joint loss than the orientation one, then we could expect the learning process to also be dominated by the former. In fact, that is what we see as the train loss for the position task presents a descending behaviour throughout the training and approaches its minimum possible value, whereas the orientation one mostly shows improvement in the first few epochs and then stabilises at a level still distanced from its minimum possible value. To address this issue, we introduce a loss weight tuning step.

By adjusting the weights assigned to each loss, we aim to find a balanced combination that accounts for this difference in scale. This tuning step allows us to effectively account for the relative importance of the two tasks and ensures that neither dominates the overall training process. The priority in this analysis was finding the orientation loss weight (w_{ori} from Equation 3.1.4.C) that simultaneously minimises both training errors, as it quantifies the model's ability to learn the tasks at hand. Regularisation was added later on to close the gap between training and validation performance.

We have employed a grid search approach, training the same MTL model on different values of the orientation task loss: specifically, 1, 10, 100, 1000, and 10000.

Figure 4.5 here depicts the training and validation curves of each loss throughout the training for various models. It is possible to see that for w_{ori} equal to 1 and 10 the models never seem to be able to overfit the orientation task as the training loss curve never seems to diverge from the validation one towards zero as it does for the other larger weights. It is clear that the w_{ori} impacts then the ability of the model to learn this task and we should then choose a weight value larger than 10 to ensure that the model is able to learn that task.

w_{ori} also has some effect on the learning of the position task. As the relative importance of the position loss parcel is decreased due to a larger w_{ori}, we observe a slowing down in the learning pace

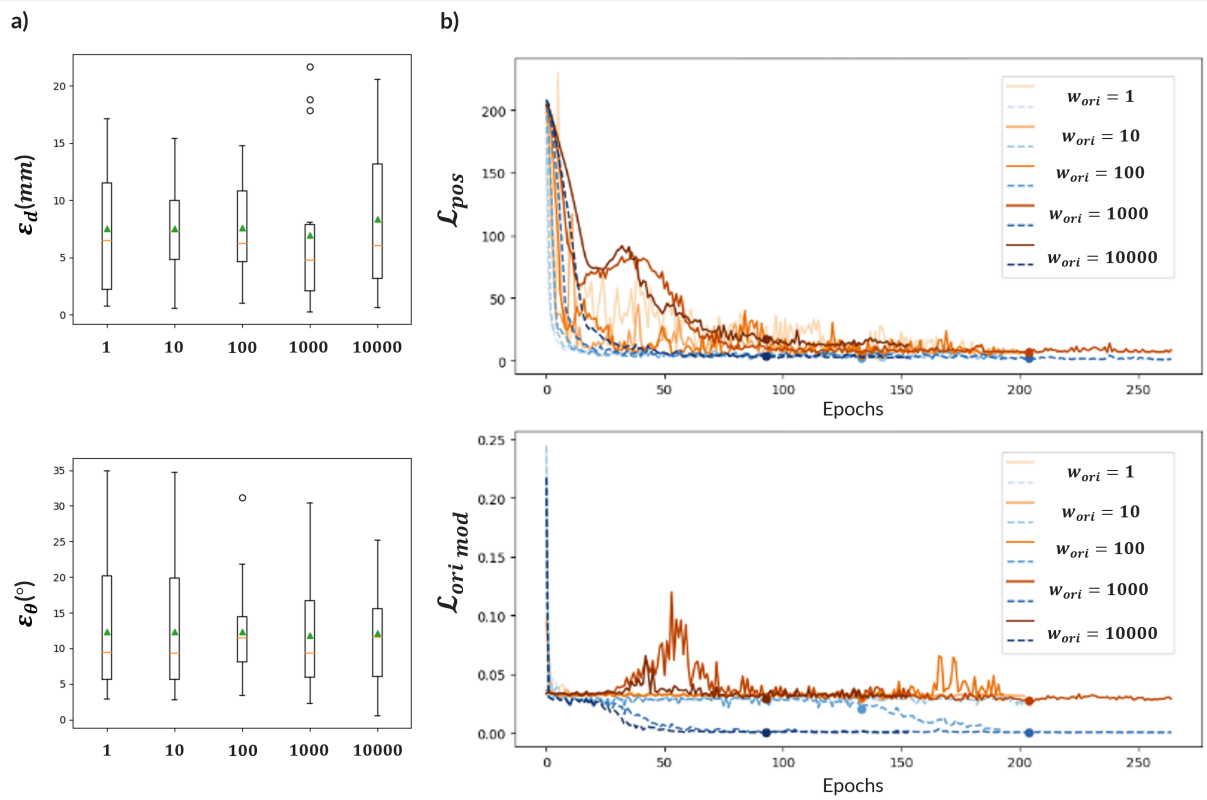


Figure 4.5: (a) 2CH view model test set performance when training on multiple orientation loss weight values. (b) Training and validation curves for each task specific loss for the 2CH view model when training on multiple orientation loss weight values. Validation curve plotted as a full line and the train curve as a dotted one. The hue intensity of the lines depicts the value used for orientation loss weight with larger values described by darker colours. Best epoch is marked with a blue and red dot on the validation and training curves, respectively.

of the position task. This effect is evident through the longer convergence time of both the validation and training curves. For the two largest loss weights we can also see that the model has to overcome a suboptimal minimum of the position loss to approximate the level at which the other models converge at. However, in the case of the largest weight tested (10000), the model's position loss never quite reaches this level, as it stabilises before doing so, leading to the training process being terminated by early stopping. This phenomenon is also evident from the test displacement error distribution that spans until larger values for this specific loss weight setting.

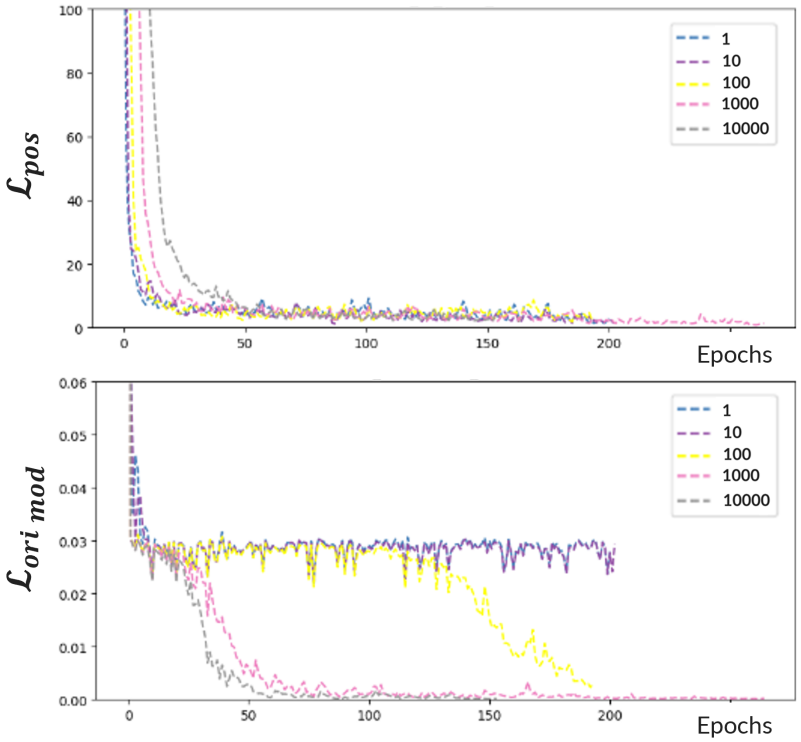


Figure 4.6: Training curves for each task specific loss for the 2CH view model when training on multiple w_{ori} values.

Ultimately, 10000 is a too large of an w_{ori} and hampers the learning of the position task, while 1 and 10 do not guarantee enough orientation loss contribution to the joint loss to allow the model to overfit the training set. As a result, our viable options for the w_{ori} are narrowed down to 100 and 1000. Between these two, we opted to use 1000 for three main reasons. Firstly, considering Figure 4.6 for a deeper look into the training losses' evolution, it is evident that a weight of 1000 leads to a significantly faster stabilisation of the orientation train loss compared to a weight of 100. This advantage comes at the expense of only a slightly slower convergence of the position train loss. Additionally, considering the scales of each task's loss, it also becomes clear that 1000 is the factor that better brings together the scales of the parcels being summed in the joint loss. Since the orientation loss tends to stabilise just

below 0.05 and the position one between 50 and 20 mm, multiplying the former by 1000 before summing the two losses effectively brings the two parcels to the same range of values. Finally, 1000 is the loss weight that offers the best median test set performance on both tasks.

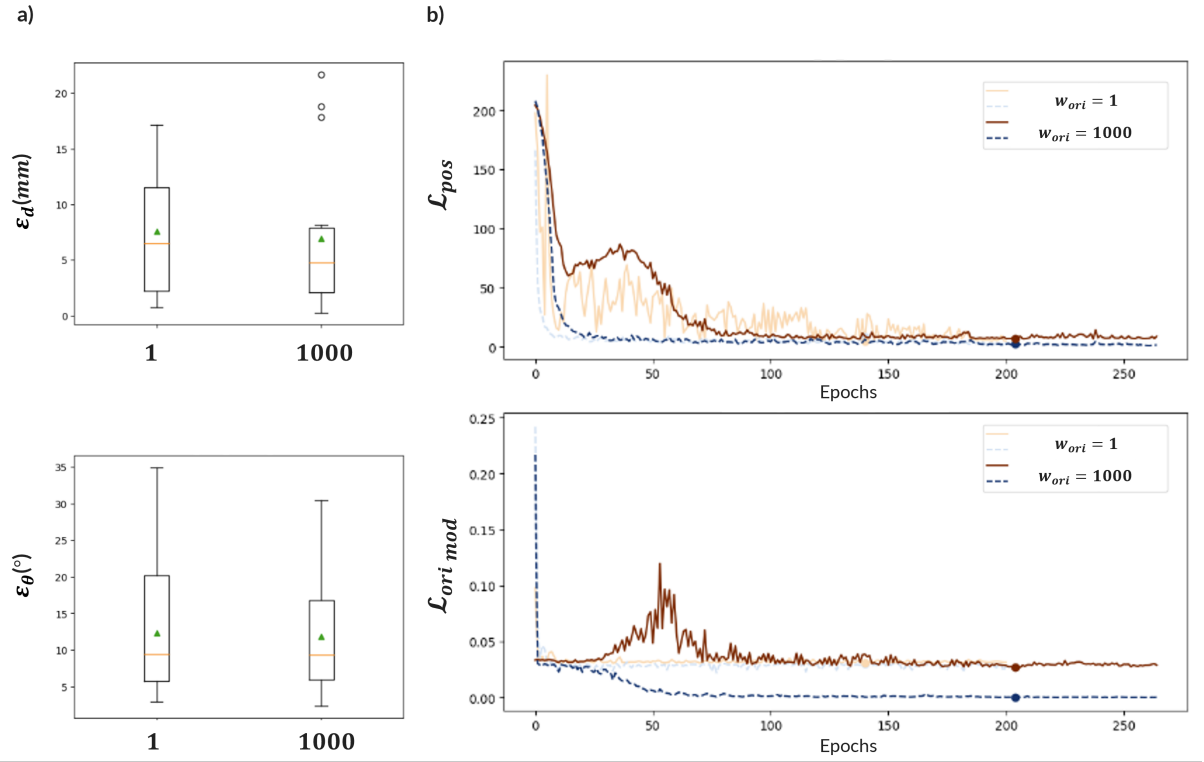


Figure 4.7: 2CH view model test set performance (a) and curves (b) before and after increasing the w_{ori} from 1 epochs to 1000. Validation curve plotted as a full line and the train curve as a dotted one. Best epoch is marked with a blue and red dot on the validation and training curves, respectively.

Figure 4.7 depicts the test set performance and learning curves before and after w_{ori} tuning. The newfound balance between the proportion of the two losses in the joint loss allow the model to reach close to zero train loss values in both tasks, which did not happen before. This weight choice was kept for the subsequent analysis. We can later address this, now greater, gap between training and validation losses with extra regularisation.

4.2.3 Introducing Data Augmentation

Data augmentation is introduced to address the increased train validation gap that arose in the orientation loss after changing its weight. By augmenting the training data with artificially generated variations, data augmentation expands the diversity of the dataset and encourages the model to learn more robust and generalised representations. This regularisation technique may help prevent overfitting by reducing the model's reliance on specific training set patterns, leading to a more balanced convergence state and

improved generalisation performance.

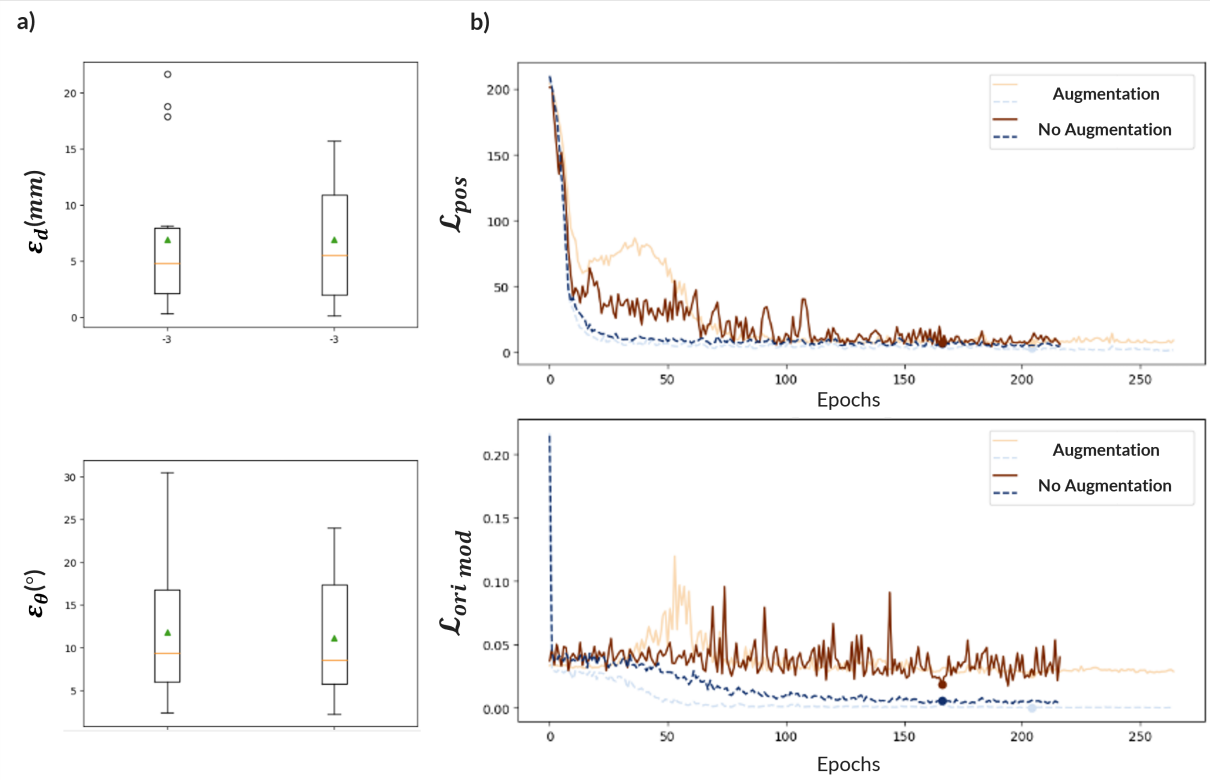


Figure 4.8: 2CH view model test set performance (a) and curves (b) before and after introducing data augmentation. Validation curve plotted as a full line and the train curve as a dotted one. Best epoch is marked with a blue and red dot on the validation and training curves, respectively.

Table 4.1: Parameters for Data Augmentation.

Parameter	Value
Intensity Scaling Multiplier	1.35
Noise Factor	0.05
Scale Factor Standard Deviation	0.1
Translational Shift Standard Deviation (x, y, z)	(12, 12, 12)
Rotation Standard Deviation (x, y, z)	(5, 5, 5)

Figure 4.8 shows the test set performance and learning curves of the 2CH MTL model with and without using data augmentation. Table 4.1 summarises the data augmentation parameters used (see section 4.1.1).

When using data augmentation, the position train loss remains practically unchanged, whereas the orientation one takes longer to converge than before. Also, the validation curves become slightly more noisy than previously. Both these phenomenons can be explained by the increased complexity and diversity in training data after introducing data augmentation. This method introduces various transformations

and perturbations, creating a more diverse and challenging dataset for the model to learn from. This increased complexity leads to a longer convergence time for the training loss as the model needs more iterations to effectively capture and learn from the augmented data. Additionally, the augmented data introduces new variations and patterns that the model must generalise to, resulting in fluctuations and variations in the validation loss curves, hence leading to noisier curves.

We can also see how data augmentation helped reduce overfitting to the training set in both tasks by looking at the shapes of the validation loss curves before and after training with augmentation. The previously observed bump in the position validation loss curve and also the short spike in the orientation one are no longer present. This can be attributed to the regularisation effect of data augmentation. The increased variability and diversity in the augmented data prevent the model from overfitting to specific patterns or features from the train data, resulting in a smoother and more stable validation loss curve that is also closer to the training one.

Even though the effect of augmenting the training data is clear from the learning curves, the test set error distributions does not seem to show great improvement. In fact, a model's performance is influenced by various other factors such as the model architecture and hyperparameter settings. Data augmentation alone may not be sufficient to overcome limitations imposed by these factors if they are contributing significantly to the performance gap. To further enhance the test set performance, it may be necessary to explore other avenues such as tuning the model architecture and adjusting hyperparameters, which leads us to the following sections. We will later once again look at the effect of data augmentation once these other factors are dealt with (see 5.2.2).

Next, we will look at learning rate tuning and we decided to do so over the models trained with data augmentation, even thought the later did not lead to great test set performance improvement. This decision is supported by the fact that, by performing learning rate tuning specifically on models trained with data augmentation, we can fine-tune the learning rate to better match the augmented data distribution and the model's learning dynamics in that context.

4.2.4 Learning rate tuning

Having successfully identified a set of loss weights that lead to minimal train losses for both tasks, we now turn our attention to the learning rate (LR) tuning step. As previously explained in2.2.1.B, the LR controls the effective capacity of the model in a more complicated way than any other hyperparameter and, as such, its tuning should be prioritised.

Once more, the priority in this analysis will be finding the LR setting that simultaneously minimises both training errors. The ability of the model to learn the tasks at hand is given to us by how low of a training error we can achieve in both tasks, while the training to validation performance gap is something that can be addressed later via extra regularisation.

With that in mind, here we adopt a grid search approach over the logarithmic scale and test training the same MTL model on multiple LR values. We experimented with LR $= 10^i$ for i values between -2 and -5 at a step of -0.5. The results can be observed below.

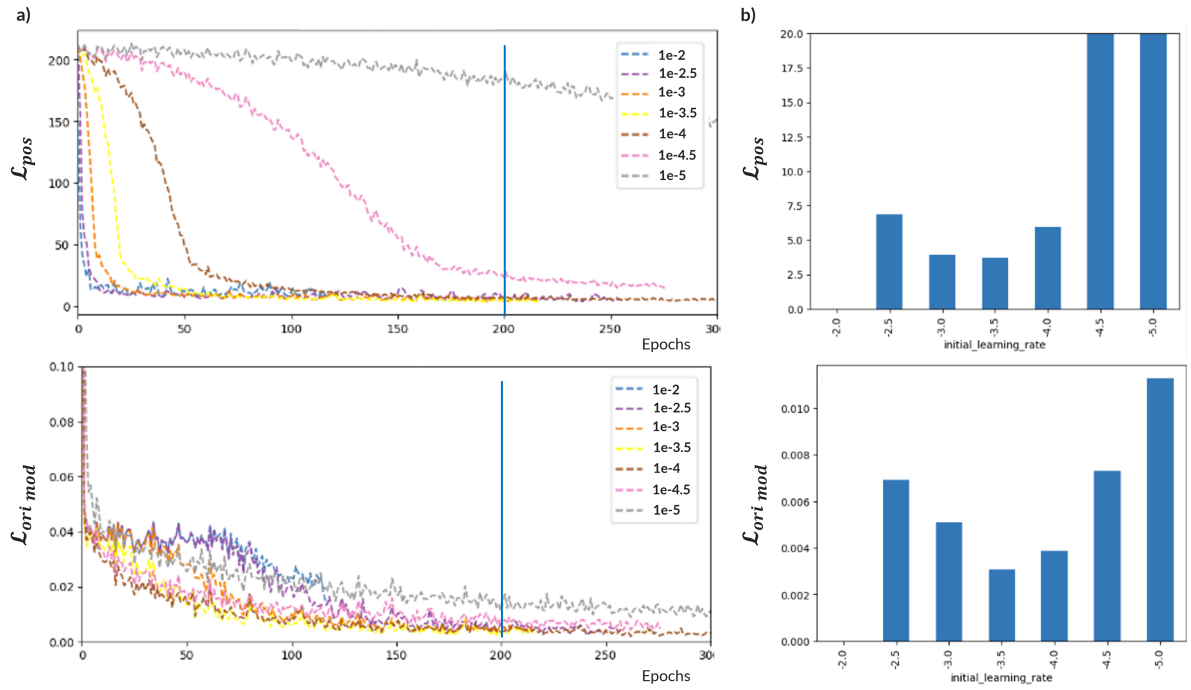


Figure 4.9: Training curves for each task specific loss for the 2CH view model when training on multiple LR values.

4.2.4.A Train Error Analysis

Figure 4.9a depicts the evolution of both training losses when trained on different LR settings. When analysing the effects of LR on both the position and orientation tasks, distinct train loss behaviours can be observed.

In the position task, decreasing the LR very noticeably leads to longer convergence time for the train loss. Specifically, for LR $= 10^{-4}$ and below, the train curve starts depicting a very slow initial loss descent followed by a slower evolution towards zero. For the smallest LR (10^{-5}) the initial descend, slows down so much that it starts resembling a plateau. This behaviour increases training time and risks the model getting stuck at suboptimal training loss level, which is not desirable.

To get a sense of how fast and well the model is able to converge for the larger LR values, it is useful to look at the value of train loss the model reaches for a fixed amount of epochs. Figure 4.9b describes exactly that for 200 epochs on both losses. We find that the position train loss follows the expected quadratic behaviour for the LR (see Figure 2.14 from the Background Chapter) as, for a fixed amount of epochs, we can interpolate a U-shaped curve from train loss values for different LR settings (see Figure

4.9b).

The LR at the inflexion point of the curve is the optimal value for the position task learning, as they allow the mode to reach the smallest loss in the least time. In this case, we find that the optimal LR would be somewhere between 10^{-3} and $10^{-3.5}$, considering our train time constraint of 200 epochs.

Using the same figures can also make a similar analysis for the orientation task. For the two smallest LR values, $10^{-4.5}$ and 10^{-5}, we once more see that the training curves have a much slower decreasing behaviour, being even unable to reach the level achieved by the other models before the learning is terminated. The three largest LR values, 10^{-2}, $10^{-2.5}$ and 10^{-3}, do not also lead to the optimal train loss evolution, as their curves all seem to depict an initial plateau phase and then take some time to reach their minimum level. The two LR values that seem to be better fit for this task are $10^{-3.5}$ and 10^{-4}, which both depict a smooth and fast descend. Figure 4.9b also shows that these are the two rates that lead to the two lowest training losses at epoch 200, meaning that the optimal rate should be somewhere between them. Considering only values we tested, $10^{-3.5}$ would be the best option as not only leads to the best train error at the 200 epoch as it also leads to a train curve that is consistently the lowest of them all throughout the training (See yellow orientation curve at 4.9a).

Considering the train set performance, we find that the two tasks agree in terms of optimal LR values. Both position and orientation train losses seem to be minimised while using a LR of $10^{-3.5}$.

4.2.4.B Test set performance analysis

Now lets consider the test set performances on Figure 4.10. Considering the performance metrics achieved on the test set for various LR values, we can make a similar analysis as we did before with Figure 4.9b with the train loss. This time we are comparing the test set performance of the best validation loss model found for each LR.

Although it is not exactly replicated, the behaviour we saw for the train loss before seems very similar to what we find now using test set performance. Extreme LR values yield worse results, thus, like before, we can see a U-shaped curve, although slightly more clearly for the position task. ε_d seems to be smaller for the range of rates between $10^{-2.5}$ and 10^{-4} which includes the optimal value we saw before. ε_θ seems to be smaller for the range between 10^{-3} and $10^{-4.5}$, like before, however with an outlying larger orientation mean test error for $10^{-3.5}$. Nevertheless, it would also not be expected for the test errors to follow the exact same pattern as the train ones for two reasons. On the one hand, regularisation effects can complicate this U-shaped curve [7].

4.2.4.C Validation performance analysis

We can look at the learning curves shown in Figure 4.11 for a deeper analysis into how the LR affects the performance on the validation and test sets. The same behaviour we saw for the train curves in the

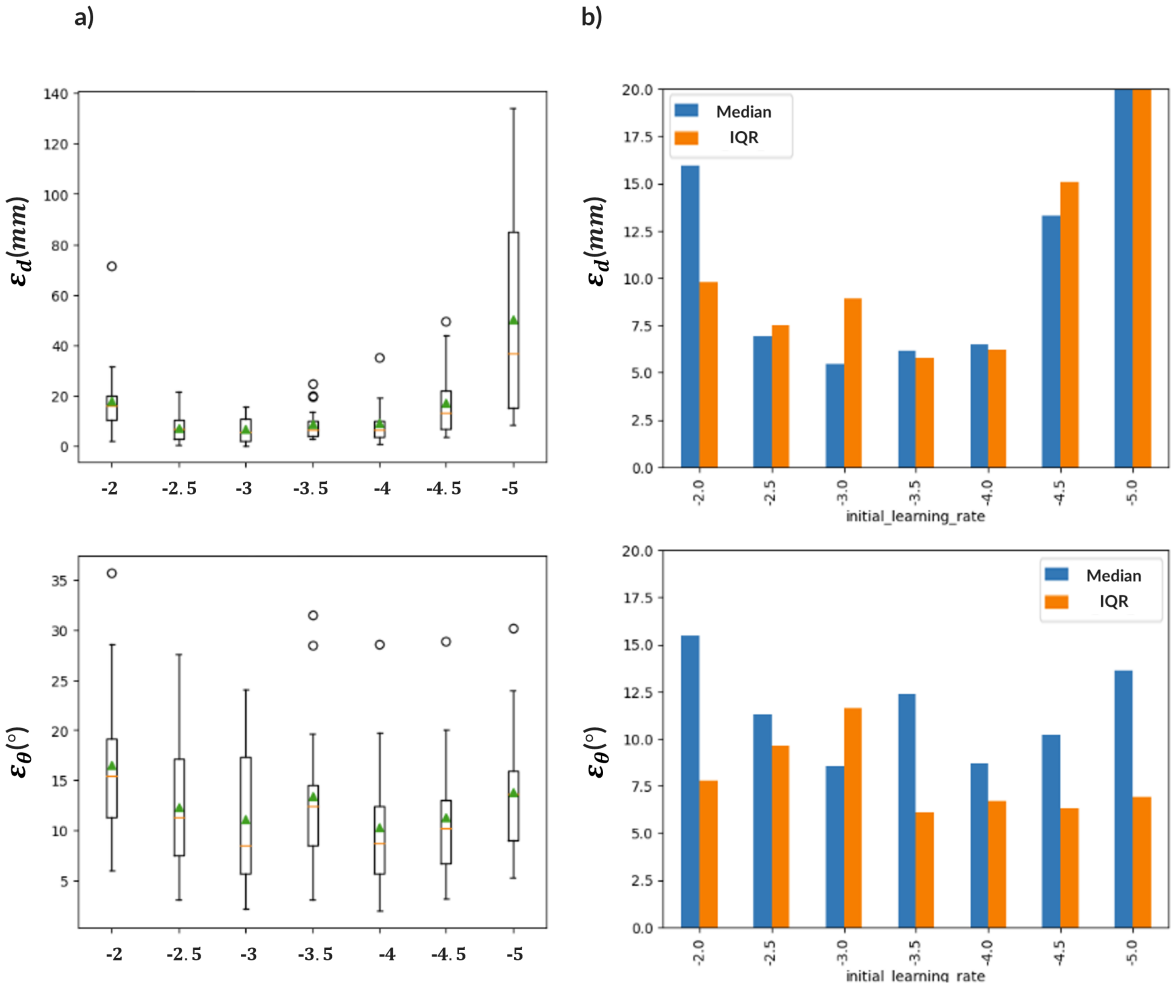


Figure 4.10: (a) Distribution of the displacement (ε_d) and angulation errors (ε_θ) of the 2CH model on the test set according to the LR value used for training. Mean error value marked with green triangle and median marked with orange bar. (b) Median and Inter-quartile range (IQR) values of each test set error distribution according to LR value used for training.

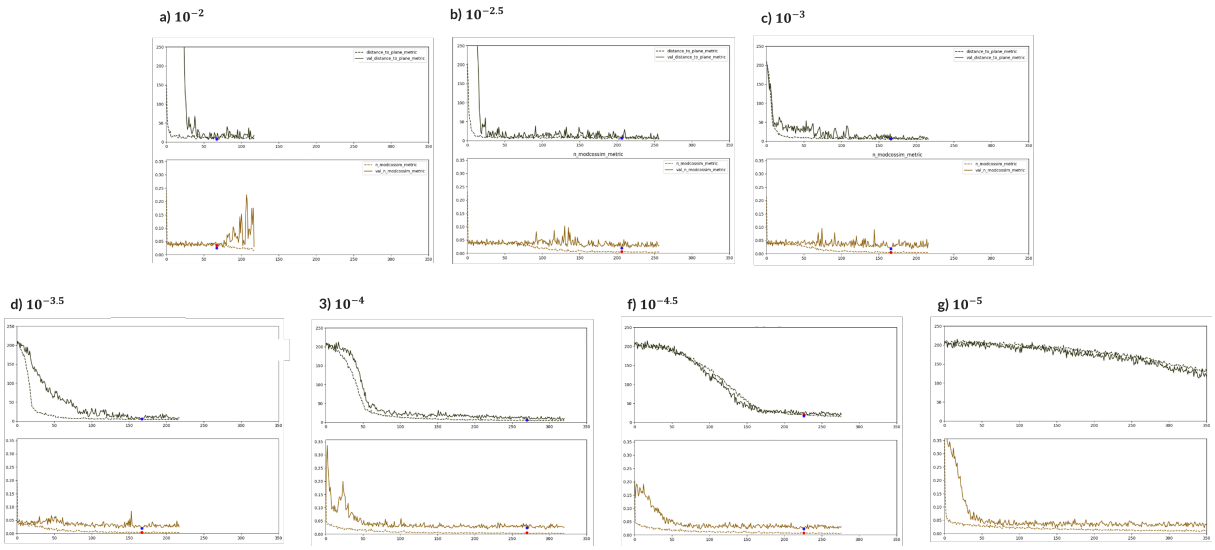


Figure 4.11: Training and validation curves for each task specific loss for the 2CH view model when training on multiple LR values.

position task can now be seen for the validation ones. Smaller LR values stall validation loss convergence. Using the lowest LR tested, 10^{-2}, leads to early termination of the training process compared to the other LR values. This premature end is a result of the model overfitting the training set on the orientation task. From approximately the 75th epoch, the model demonstrates improvement on the training data, while the validation loss spikes to larger values, triggering early stopping. Unsurprisingly, the test set ε_θ distribution exhibits significantly poorer performance for this LR compared to the others. The orientation validation curves until $10^{-3.5}$ have a very instantaneous initial decrease and then stabilise quite early in the training. With smaller LR values this behaviour changes and they start having a more defined and gradual decrease.

With all that said, we will be using a LR of $10^{-3.5}$ for the future experiments as it minimises the train error and consequently maximises the effective capacity of the model for both tasks.

Even though it does not give the best test set performance, it has the potential to do so given that it provides the best train loss for both tasks. This generalisation error is something that we can address in the future with regularisation.

4.2.5 Effect of uncertainty based loss weighting

We have seen before how the loss weights choice are determining factors in the model's ability to learn each task. Previously, increasing the orientation loss weight enabled the model to achieve significantly lower orientation train loss values and thus allowed it to better model that task. It is the importance of the choice of loss weights that motivated our next experiment. In this section we add the loss weights to the set of parameters that can be optimised during the training via the approach proposed by [34] and

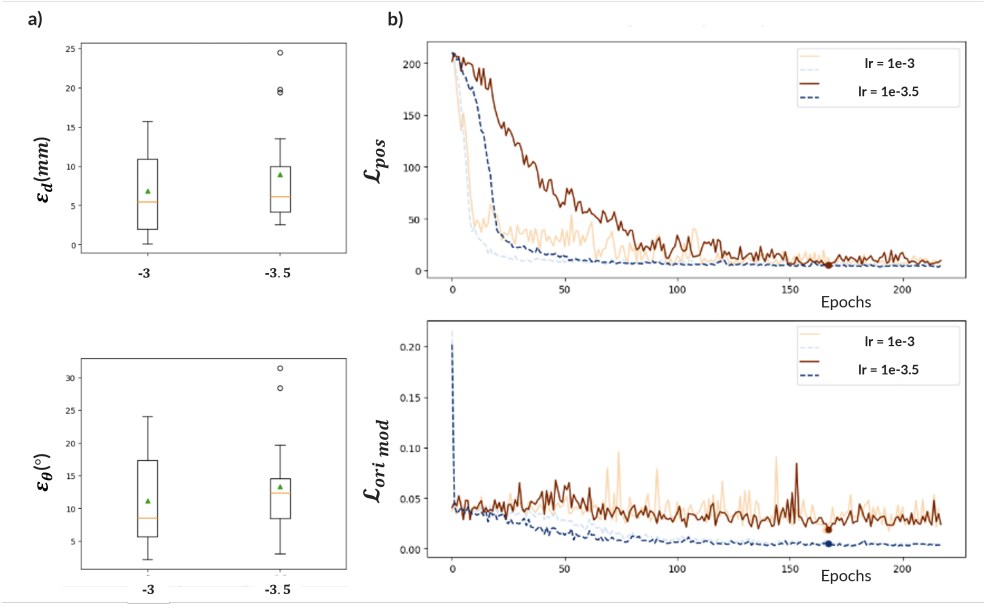


Figure 4.12: 2CH view model test set performance (a) and curves (b) before and after changing the learning rate to the newfound optimal value. Validation curve plotted as a full line and the train curve as a dotted one. Best epoch is marked with a blue and red dot on the validation and training curves, respectively.

already explained in 2.2.2.B.

Figure 4.13 depicts the test set performance as well as the learning curves of the same architecture and training parameters trained with trainable loss weights and without. In both tasks, both train and validation curves do not show any significant differences from one setting to the other. The test set performance degraded slightly on the position task but improved slightly on the orientation one. The evolution of the position and orientation loss weights as well as the regularisation factor from 2.2.2.B can be observed in Figure 4.14. The assumption behind this type of adaptive loss weighting is that it is beneficial for the model to prioritise the learning of the tasks with less homeoscedastic uncertainty or, in other words, the easier tasks where the corresponding annotations are less noisy. From the evolution of the loss weights we can see that up until about the 60th epoch the orientation prediction task has a larger weight and thus a lower uncertainty, meaning is prioritised over the position one. From that point on the priority shifts to the position task. Nevertheless, the evolution of the ratio between the two loss weights during the training tells us that the degree to which a task is being prioritised over the other is never really significant. In fact, throughout the training the largest a loss weight gets with respect to the other is only by a factor of approximately 1.015. With this information it becomes clear why there is not much difference in the training curves between the two settings. The homeoscedastic uncertainty does not vary enough between the two tasks for this method to yield any significant result. For this reason, we move on to the next experiments without using uncertainty based loss weighting.

The training here would probably benefit from a task balancing approach that leverages not the

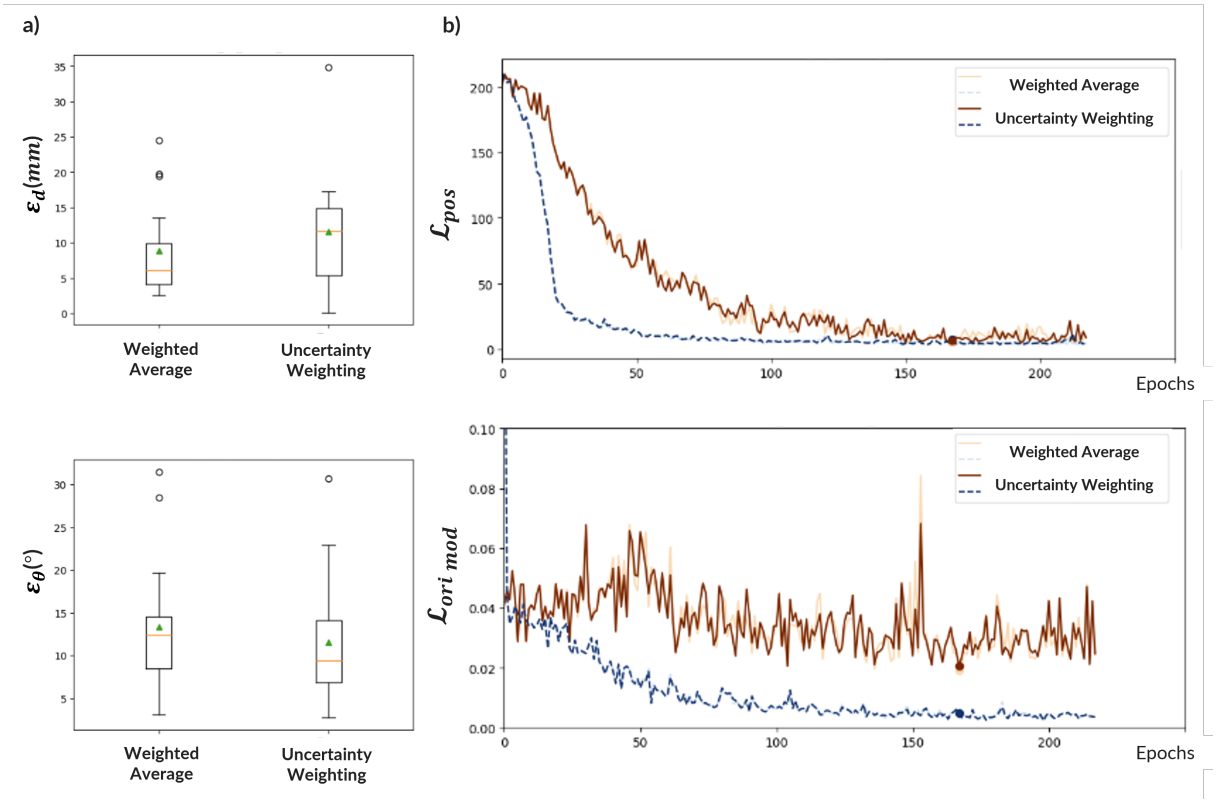


Figure 4.13: 2CH view model test set performance (a) and curves (b) before and after applying a uncertainty based loss weighting strategy. Validation curve plotted as a full line and the train curve as a dotted one. Best epoch is marked with a blue and red dot on the validation and training curves, respectively.

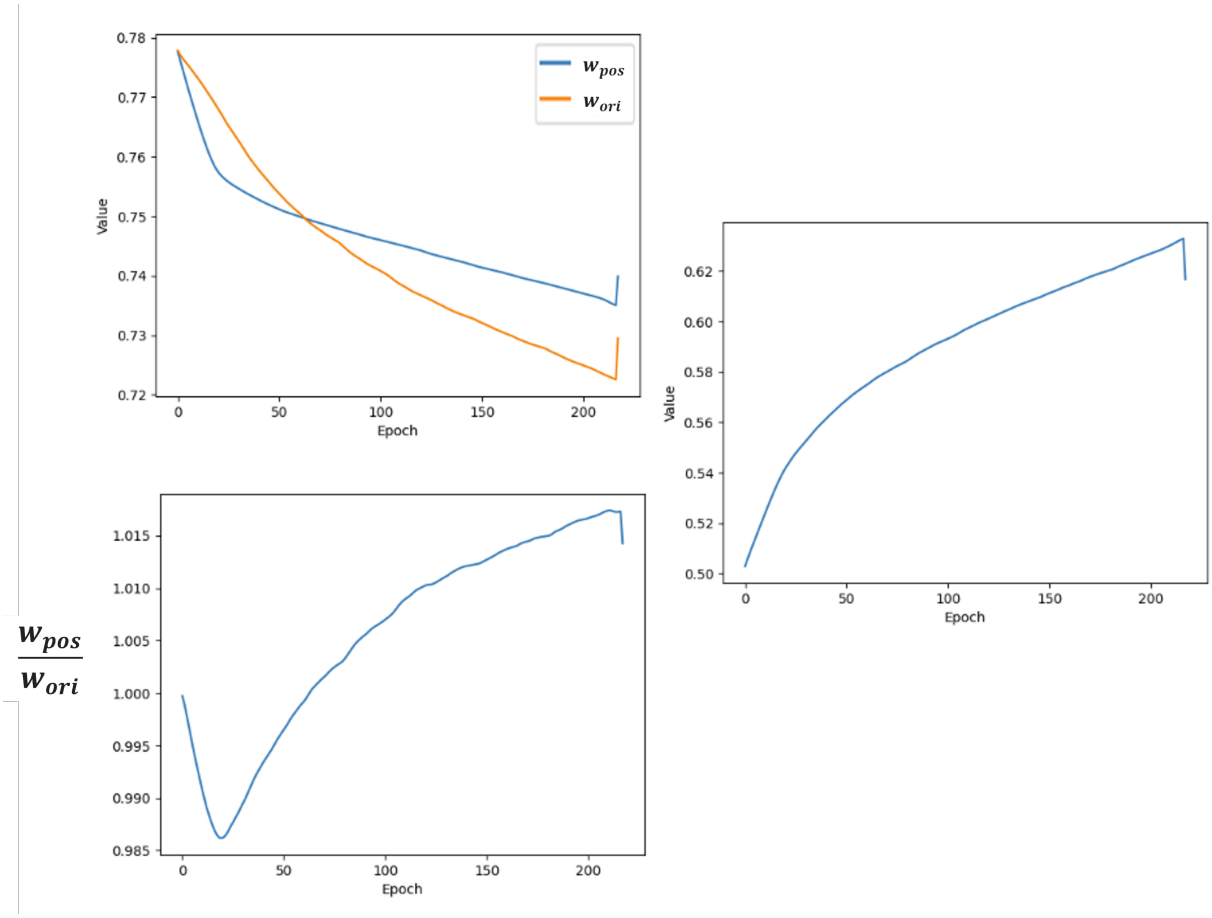


Figure 4.14: (Left) Depiction of the evolution of the uncertainty based loss weights during training both through their individual values (top) as well as through the ratio between them (bottom). (Right) Evolution of the regularising factor during training. The loss weight and the regularising factors are defined in section 2.2.2.B

homeoscedastic uncertainty relative to each task, but rather tries to balance the pace at which tasks are learned, while avoiding gradients of different magnitude. We leave experimenting with the GradNorm task balancing approach [46] as a future work suggestion, which acts exactly that way.

5

Multi-Headed Network - Automated 2CH view prediction

Contents

In the preceding chapter, we focused on mitigating the underfitting issue discussed earlier. To achieve this, we analysed the model's complexity and pursued strategies to enhance its effective capacity for both tasks by tuning the loss weights and learning rate. Additionally, we explored an alternative approach to loss weighting.

In the current chapter, we keep iterating upon the changes applied in the previous one towards the same objective. This time, a new architecture is proposed. We have seen how, in MTL, forcing the a model to learn a hidden representation that captures all tasks imposes a regularising effect in the learning process, hindering overfitting (see section 2.2.2.A from the Background chapter). Notwithstanding, like any other regularisation approach, it should be applied up until an optimal degree, or else it might lead to the model underfitting both tasks. In a hard parameter sharing scenario like the one we have been dealing with so far (N_A), the degree of this regularising effect is determined by the amount of shared hidden layers in place. Consequently, this amount becomes a parameter that should also be adjusted with care. This chapter aims to address exactly that, by assessing whether the current degree of hard parameter sharing is hampering our results. In addition, we further analyse the impact of data augmentation and volume resolution on the results. Finally, one last comparison between MTL and STL is made.

5.1 Methodology

We maintain the same methodology employed in previous chapters, while introducing a new architecture.

5.1.0.A Multi-Headed Architecture (N_B)

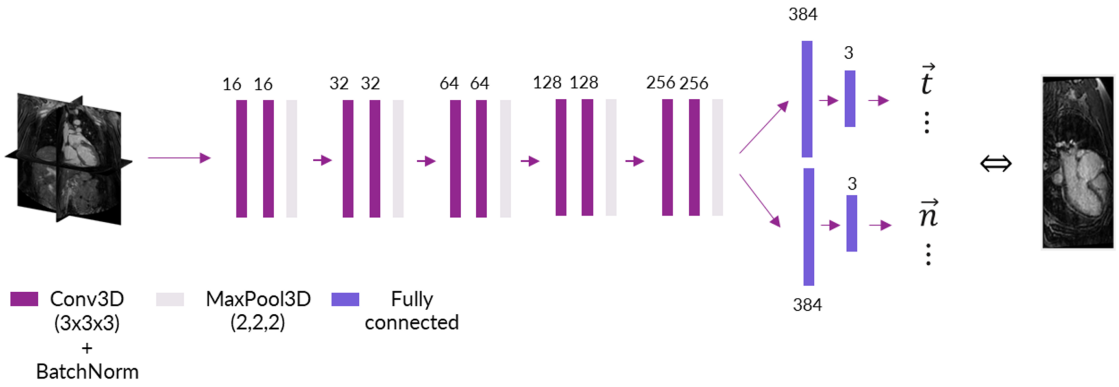


Figure 5.1: Architecture with 2 task specific regression heads.

So far we have been using the fully shared architecture, N_A, in which every single layer is shared between the two sub tasks. In this chapter, a new design is tested in which the hard parameter sharing between the tasks is limited to the feature extraction block, from which two task specific heads perform

the predictions separately, N_B. Each head is composed of one dense layer with 384 units connected to another one with 3 units matching the number of vector coordinates. This multi-headed architecture allows the model to interpret the same set of extracted features differently for each task, which can prove beneficial in some cases. See Figure 5.1 for a visual depiction of the architecture.

The same L2 weight regularisation approach and kernel initialisers are used in this architecture and in N_A.

5.2 Results and Discussion

5.2.1 Introducing Separate Regression Heads

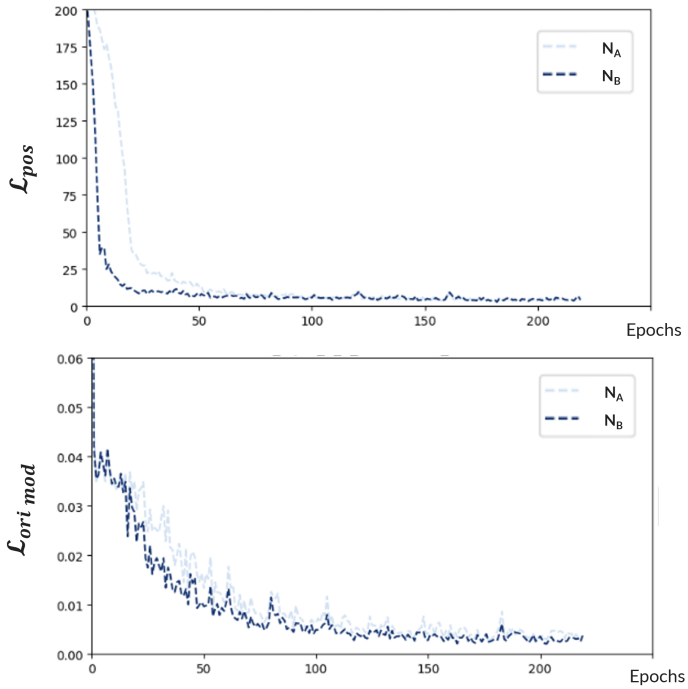


Figure 5.2: Training curves for each task specific loss for the 2CH view model when training on a fully shared architecture (N_A) or a multi-headed one (N_B).

Here, we compare the performance and learning curves between the best performing 2CH model found so far using N_A architecture and a new model trained with the same hyperparameters but with this new multi-headed architecture, N_B.

Figure 5.2 depicts the evolution of the two training losses during the training. In comparison with N_A, N_B allows a significantly faster convergence of both tasks' losses. In general, with this new setting the models are able to reach lower values earlier in the training. In particular, for the orientation task, we can also observe a train loss that is lower than the one for N_A all throughout the training. The behaviour

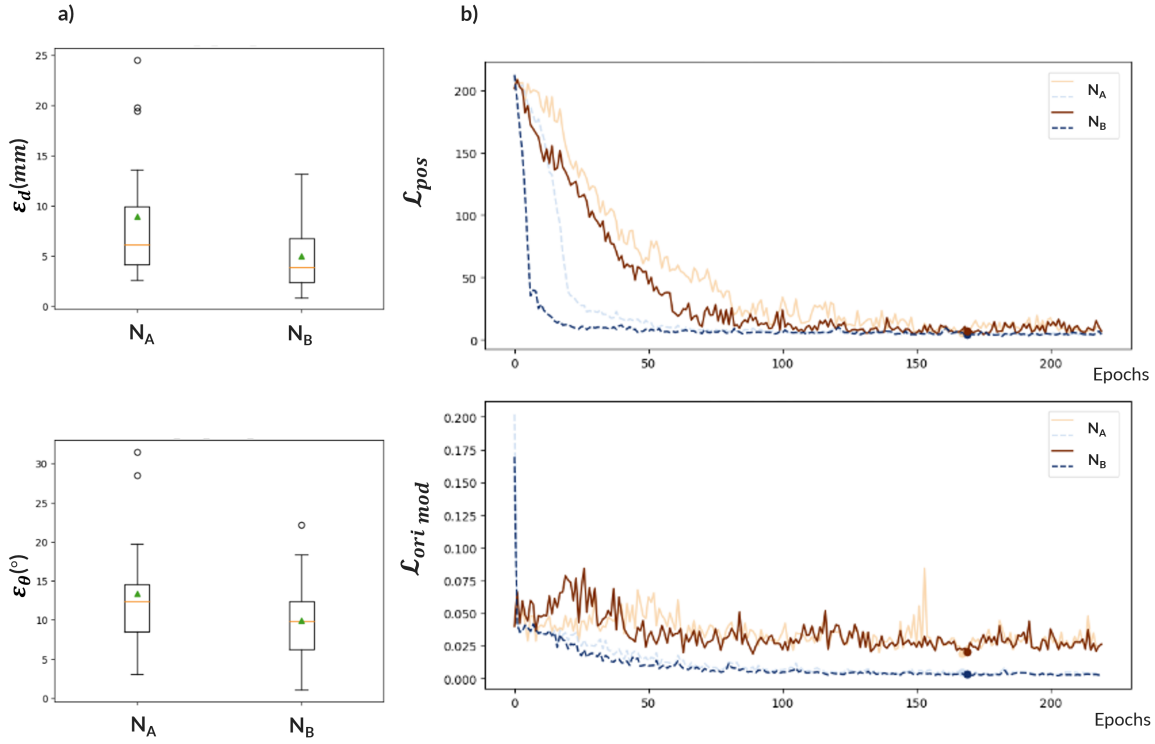


Figure 5.3: 2CH view model test set performance (a) and curves (b) when training on a fully shared architecture (N_A) or a multi-headed one (N_B). Validation curve plotted as a full line and the train curve as a dotted one. Best epoch is marked with a blue and red dot on the validation and training curves, respectively.

of these training losses indicates that the effective capacity of the model increases when moving from the baseline architecture to the new one.

Furthermore, from Figure 5.3 we can tell that this change, not only increased the effective capacity of the model, but also did it while not sacrificing generalisation ability. The position validation loss curve, just like its training counter part, has a faster convergence, reaching lower loss values earlier in the training. It ends up stabilising at the same level as before although with a slightly less noisy behaviour. Similarly, the orientation validation loss curve also stabilises around the same level as before but reaches that state earlier in the training. In fact, the generalisation ability of the model increased with this new architecture as the test set error distributions shift towards lower values after the change.

Since both the training losses and the test set performance improve with this new change, it is possible to say that reducing the level of information sharing between the tasks lead to an increase of effective model capacity towards the optimal capacity level for this problem. Figure 5.4 depicts a version of Figure 2.13 from the background chapter where this performance boost is illustrated.

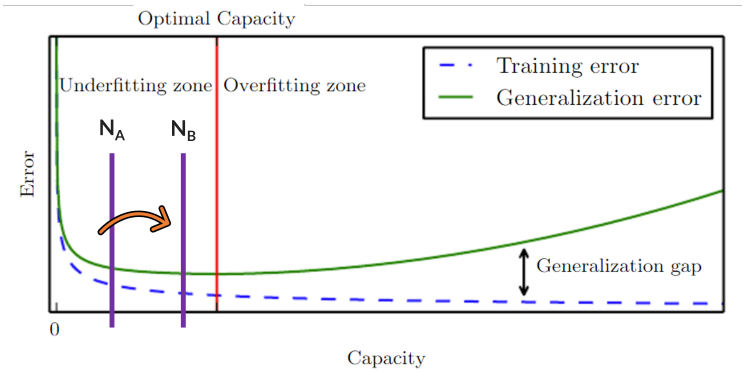


Figure 5.4: Illustration of the impact of transitioning from architecture N_A to N_B on the model's effective capacity, showing its increase towards its optimal value. Based on previous Figure 2.13. Adapted from [7].

These observations indicate that sharing the whole architecture between the tasks was excessively constraining the learning and hampering the model's ability to learn each task. The learning benefits from a lower level of parameter sharing, as having a separate head dedicated to each task outperforms the baseline fully shared design. Thus, the multi-head architecture was used for all subsequent experiments.

5.2.2 Effect of Data Augmentation

Data augmentation has already been introduced previously at section 4.2.3. However, at that point, despite some regularisation effect being visible on the learning curves, it did not translate into the test set performances. This could be attributed to the suboptimal learning rate and the high degree of parameter sharing between the two tasks, which constrained the model's effective capacity. With these

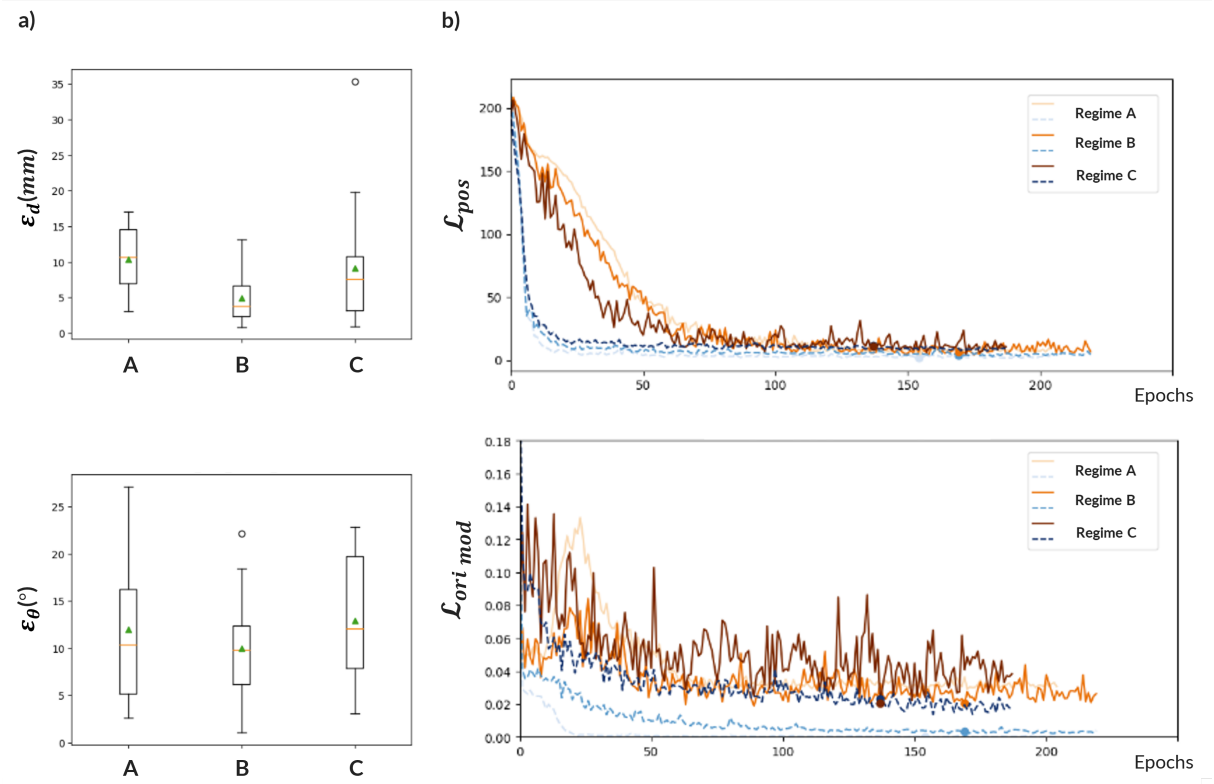


Figure 5.5: (a) 2CH view model test set performance when training on 3 different data augmentation regimes. (b) Training and validation curves for each task specific loss for the 2CH view model when training on 3 different data augmentation regimes. Validation curve plotted as a full line and the train curve as a dotted one. The hue intensity of the lines depicts the intensity of the data augmentation regime used, with larger intensity described by darker colours. Best epoch is marked with a blue and red dot on the validation and training curves, respectively.

confounding factors now addressed, through the optimisation of the learning rate and adjustment of parameter sharing, we can delve into a more insightful analysis of how data augmentation truly influences the model's performance.

With that in mind, we train the same model on 3 different data augmentation regimes: Regime A corresponds to not using any data augmentation; Regime B corresponds to a moderately intense data augmentation setting, which has been the one used since section 4.2.3; Regime C corresponds to an intense data augmentation scenario. Table 5.1 presents the augmentation parameters used for each regime. The intensity of a data augmentation setting is regulated by the scale of each transformation's parameters. Larger parameter values increase the standard deviations of the normal distributions from which the shifting, rotation, scaling, noise addition and intensity scaling values are sampled from. Thus, the larger the parameter values the larger the extra variance introduced to the training data.

Table 5.1: Data Augmentation Regimes tested.

Parameter	**Regime**		
	A	**B**	**C**
Intensity Scaling Multiplier	1	1.35	1.4
Noise Factor	0	0.05	0.1
Scale Factor Standard Deviation	0	0.1	0.25
Translational Shift Standard Deviation (x, y, z)	(0, 0, 0)	(12, 12, 12)	(20, 20, 20)
Rotation Standard Deviation (x, y, z)	(0, 0, 0)	(5, 5, 5)	(15, 15, 15)

Figure 5.5 depicts the test set performances and the learning curves achieved by the same model when trained on each data augmentation regime. Just like we had seen before, when using data augmentation, the train losses show a slower decreasing behaviour, never reaching values as low as the ones achieved without any augmentation introduced. This effect also scales with the intensity of the data augmentation used as the minimum train loss achieved by the model increases from regime B to C in both tasks. This comes as a direct result of the increased complexity and diversity of the training data when using this approach, which results in a more challenging dataset to learn from.

It is also clear from the validation curves that, as the intensity of data augmentation increases, the learning process becomes increasingly noisier. The augmented data introduces new variations and patterns that the model must generalise to, resulting in the fluctuations and variations of validation loss that we can see in the figure. As the variance introduced into the training data increases, we must also expect to witness a scaling up of these fluctuations and noise. This effect is evident in both tasks, although it is particularly pronounced in the orientation task.

Up until this point we have been using the data augmentation regime B. Based on the analysis of Figure 5.5, it is apparent that it allows to bring the training and validation curves together in comparison to not using augmentation (regime A). This happens not only because the train loss is now larger, but also because the validation one now reaches a lower level than before. Additionally, from regime A to B

we can also see a notable improvement of both task's performance, as both error distributions shift to lower values while also having even smaller IQR. These two observations make it clear that the model benefits from using data augmentation regime B, as its generalisation ability is improved.

Notwithstanding, further increasing the intensity of data augmentation does not yield further improvements to the models generalisation ability. Despite regime C still providing an improvement of the test set position performance in comparison to regime A, it still leads to worse test set performances then in regime B in both tasks. The high intensity data augmentation shows to be excessive for the orientation task and sub optimal for the position one. This can be further understood from the learning curves. The orientation train loss curve for regime C is notably larger than for the other two regimes, struggling even to reach values below the ones seen for the validation one in regime B. This indicates that the training data has been injected with excessively large variance, making it especially challenging for the model to learn the task at hand. In addition, the corresponding validation loss curve shows a highly noisy trend which could be a evidence of how the augmentation process can be introducing unrealistically distorted samples. On the other hand, the position validation curve for regime C, actually shows the fastest decrease and convergence out of the regimes tested. Nevertheless, this task's training loss never reaches very low values which acts as a constrain to how low the validation loss can get, thereby leading to the suboptimal test set performance we have described earlier.

The distinct effect that regime C has on the two tasks in comparison to regime B might also be explained by the evolution of the augmentation parameters from one scenario to the other. In fact, the intensity of the rotation transforms increases by a factor of 3 while the transnational transforms increase by factor of only approximately 2. This justifies the more erratic behaviour of the losses in the orientation task than in the position one.

In essence, the most intense data augmentation scenario (regime C) does not contribute to a better performance than the one we have been using until now (regime B). These results allow us to infer that, at least for the orientation task, augmentation regime C is excessively intense and, thus, does not realistically replicate the usual variance seen in MRI. In the end, it actually bias the model and hampers its ability to generalise for unseen samples. Given its test set performance, it was regime B that proved to achieve that.

5.2.3 Effect of volume resolution

Even though we have implemented a custom data loader that helps reducing memory utilisation (see section 3.1.3.D), the size of the volumes limits how big our batch sizes can be. Because of this, we have been resizing them from an isotropic resolution of 3mm to 4mm.

We have been assuming that a 4mm isotropic resolution carries enough information from the volumes for the model to extract meaningful features for view plane prediction as well as 3mm. In this section,

this hypothesis is put to the test. We train the same model twice on the exact same settings, differing only in how much the volumes are resized during the preprocessing. The first scenario is the current setting of resizing to a 4mm isotropic resolution whereas in the second scenario no resizing is applied and the input scans have a 3mm isotropic resolution.

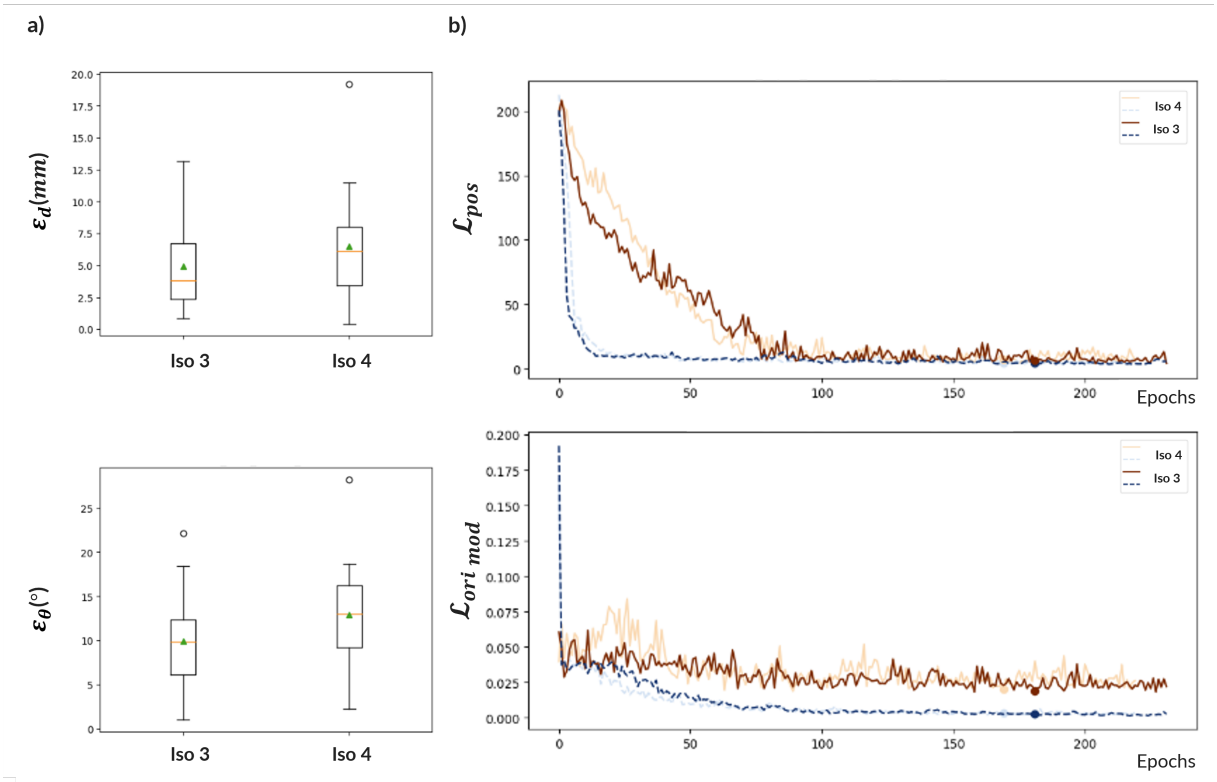


Figure 5.6: 2CH view model test set performance (a) and curves (b) when training with volume resizing to 4mm isotropic resolution (Iso 4) compared to training with 3mm isotropic resolution (Iso 3). Validation curve plotted as a full line and the train curve as a dotted one. Best epoch is marked with a blue and red dot on the validation and training curves, respectively.

Table 5.2: Impact of Volume Resolution on Training Duration

Volume Resolution	Duration (Epochs = 100)	Seconds per Epoch
4mm isotropic	64.0 min	38.4 s
3mm isotropic	101.8 min	61.1 s

Figure 5.6 shows the test set performances and learning curves in both aforementioned scenarios.

We find that training on scans with a higher resolution (3 mm) does not bring any improvement to the model's performance in either of the tasks. The test set error distributions do not differ greatly between the two scenarios but shift to slightly larger values when training on 3 mm isotropic resolution scans. The learning curves behaviour also does not change greatly from one setting to the other. If anything, the orientation train loss seems to have a slight slower convergence and its validation counterpart a less noisy

trend. Regarding the position curves, although both the train and validation losses converge slightly earlier in the 3mm isotropic setting, the difference to the 4mm isotropic scenario is neglectable.

The added detail and information provided in the scans (see Figure 5.7) did not prove to be beneficial for either tasks' learning. Perhaps, increasing the resolution leads to the introduction of fine variations of pixel intensity in the scans that are also not intuitively useful for view prescription. In fact, this agrees with the traditional CMR view prescription protocol which relies more on higher level and lesser fine features of the scans, like over all location and orientation of the heart.

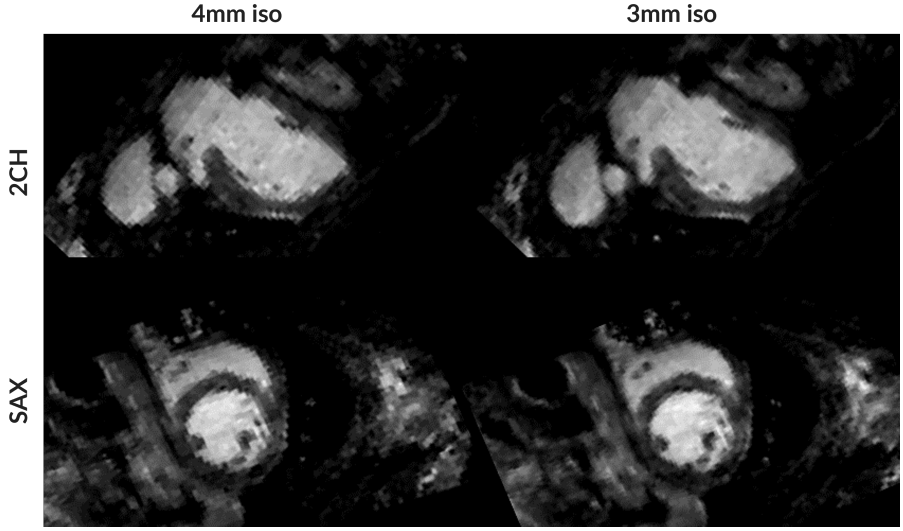


Figure 5.7: 2ch, sax slices on the 2 resolution settings

Additionally, using higher resolution scans takes up much more GPU RAM, as the input volumes go from a 95x95x42 mm^3 dimension to 127x127x57 mm^3. This unavoidably leads to a slower training process. Table 5.2 depicts the time it takes for each of the models to reach 100 epochs and the time per epoch, depending on the volume resizing setting. Increasing 1mm the resolution in every direction amounts to increasing the epoch duration by about a factor of 2. Given our limited access to Google Colab's GPUs, this is not desirable.

Given the above stated reasons, our initial hypothesis of 4mm isotropic resolution being enough to model both tasks holds, saving us about 50% training time and improving performance.

5.2.4 Comparison with STL

In this section we compare the best performing 2CH model trained on a MTL setting with the 2 equivalent STL models trained on the same hyperparameters. Earlier in section 4.2.1, an initial comparison was conducted between these training approaches. However, the potential advantages of employing a MTL approach as opposed to a STL one may have been hindered due to the lack a proper loss weighting setting, the LR tuning and the right MTL architecture.

Figures 5.8 and 5.9 depict the comparison between the test set performances as well as learning curve behaviours between MTL, Position STL and Orientation STL, respectively.

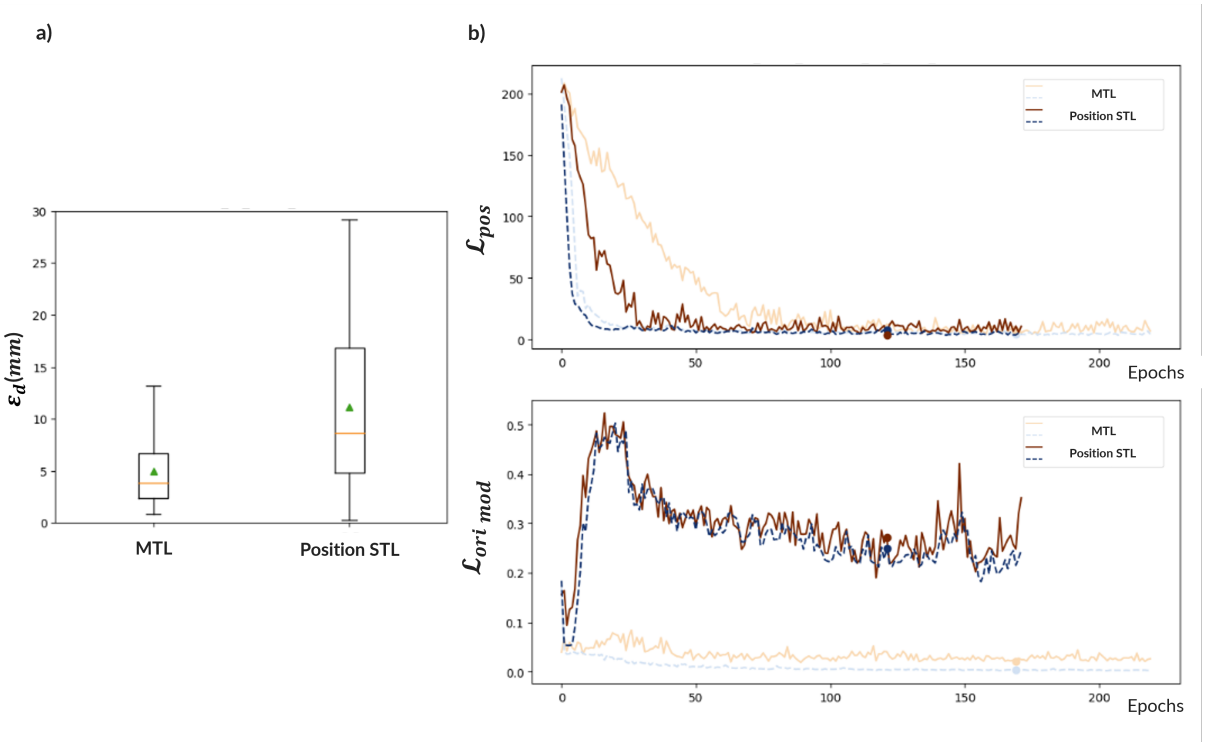


Figure 5.8: 2CH view model test set performance (a) and curves (b) when training on a MTL or on a position STL setting. Validation curve plotted as a full line and the train curve as a dotted one. Best epoch is marked with a blue and red dot on the validation and training curves, respectively.

Similarly to before (see section 4.2.1), we can see that learning the position task leads to the model also learning the orientation task, as even in a position STL setting the orientation loss depicts a somewhat consistent decreasing behaviour (Figure 5.8). The observed knowledge transfer stems from the inherent correlation between the position and orientation tasks. Shared and complementary information between these tasks enables the model to extract features that benefit both objectives simultaneously.

The previous observation underscores the potential of MTL in enhancing overall performance. Indeed, this potential is confirmed once we consider the test set performances on Figures 5.8a and 5.9a. In comparison to an STL setting, the MTL approach leads to a notable increase of test set performance on the position prediction task, while also providing a slight improvement in orientation performance. These results highlight the ability of this MTL model to leverage the shared and complementary information between the two tasks.

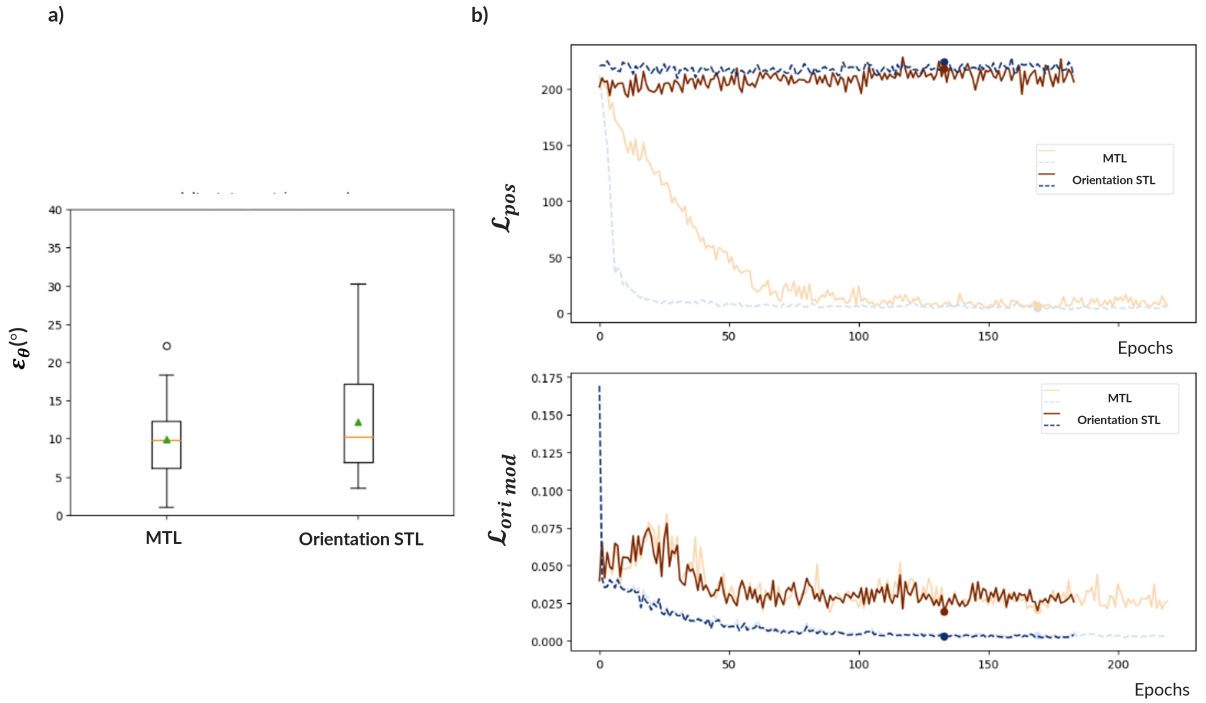


Figure 5.9: 2CH view model test set performance (a) and curves (b) when training on a MTL or on a orientation STL setting. Validation curve plotted as a full line and the train curve as a dotted one. Best epoch is marked with a blue and red dot on the validation and training curves, respectively.

6

DeepCardioPlanner - Automated 2CH, 4CH, 3CH and SAX view prediction

Contents

Ever since the baseline performances provided in the end of Chapter 3, we have come to optimise the training setting for the 2CH view predicting model. In this chapter, we investigate the generalisation of these changes to the remaining view predicting models. Finally, we present a comparative analysis between our results and those reported in the literature.

6.1 Methodology

In this chapter, we will use the best training setting found for the 2CH view prediction to train one model for each one of the remaining views.

We have seen before that due to the high heterogeneity of the dataset the learning progress is noisy. Therefore, we have also experimented increasing the early stopping patience from 50 epochs to 100 for the 3CH, 4CH and SAX model trainings.

6.2 Results and Discussion

6.2.1 2CH

Figure 6.1 compares between what was the 2CH view prediction performance and learning behaviour before and after optimising its training setting.

After optimisation of the training setting, we find that the orientation training loss is effectively minimised during training and approaches values closer to zero which did not happen before. Adding to that, this happens while the corresponding validation loss is also able to stabilise at a lower value. All in all, the applied changes act not only improve the effective capacity of the model but also its generalisation ability. The underfitting issue observed in the baseline is successfully mitigated.

Moreover, all this is achieved while also attaining an improvement in both orientation and position test set performance.

6.2.2 Testing generalisability to other view prediction tasks

6.2.2.A 3CH

Both orientation train and validation losses converge at lower values after applying the changes. Test set angulation errors shift towards lower values. However, the improvement in final validation loss between the two scenarios is of approximately 0.025 which translates to around 12°(see section 3.1.4). Clearly, this improvement in validation performance does not match the one seen for the test set. Once again, the heterogeneity of our dataset might have led to test and validation splits with very distinct features and hence some performance improvements may not generalise between subsets.

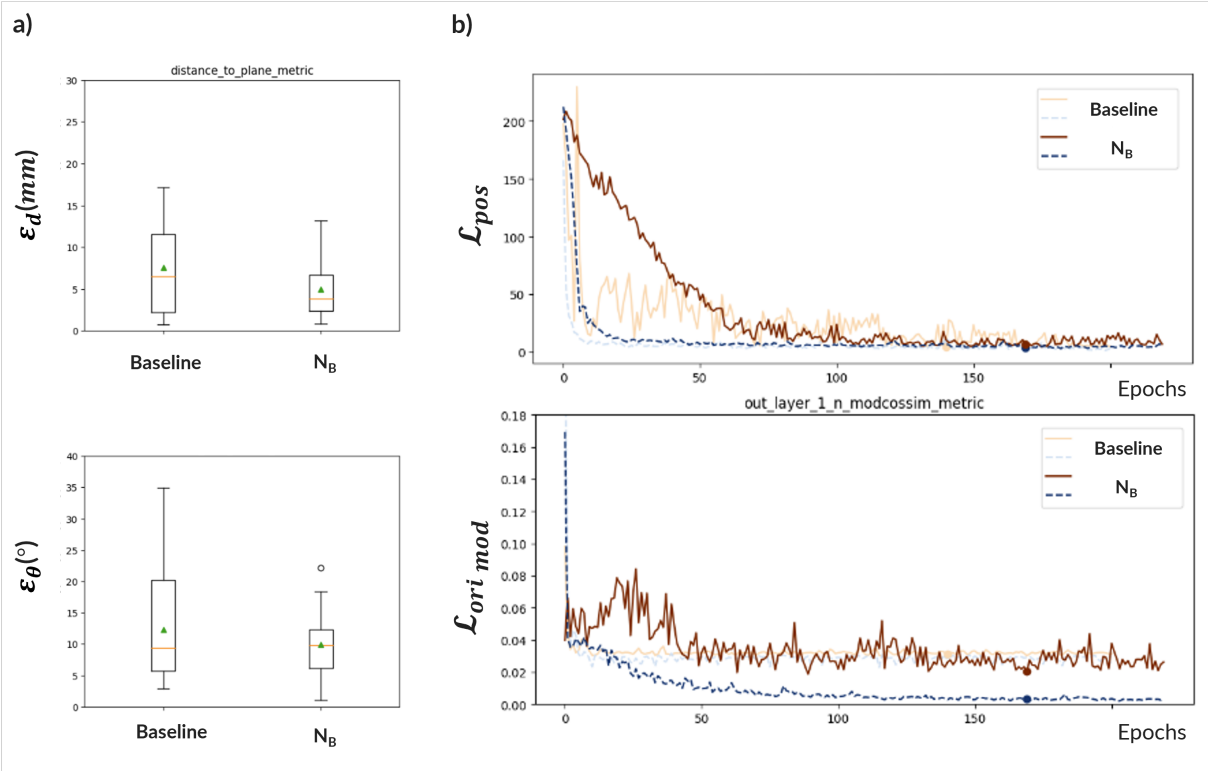


Figure 6.1: 2CH view model test set performance (a) and curves (b) from last baseline (variance adjusted + $\mathcal{L}_{ori_{mod}}$ from section 3.2.3) compared to best training setting found (N_B) . Validation curve plotted as a full line and the train curve as a dotted one. Best epoch is marked with a blue and red dot on the validation and training curves, respectively.

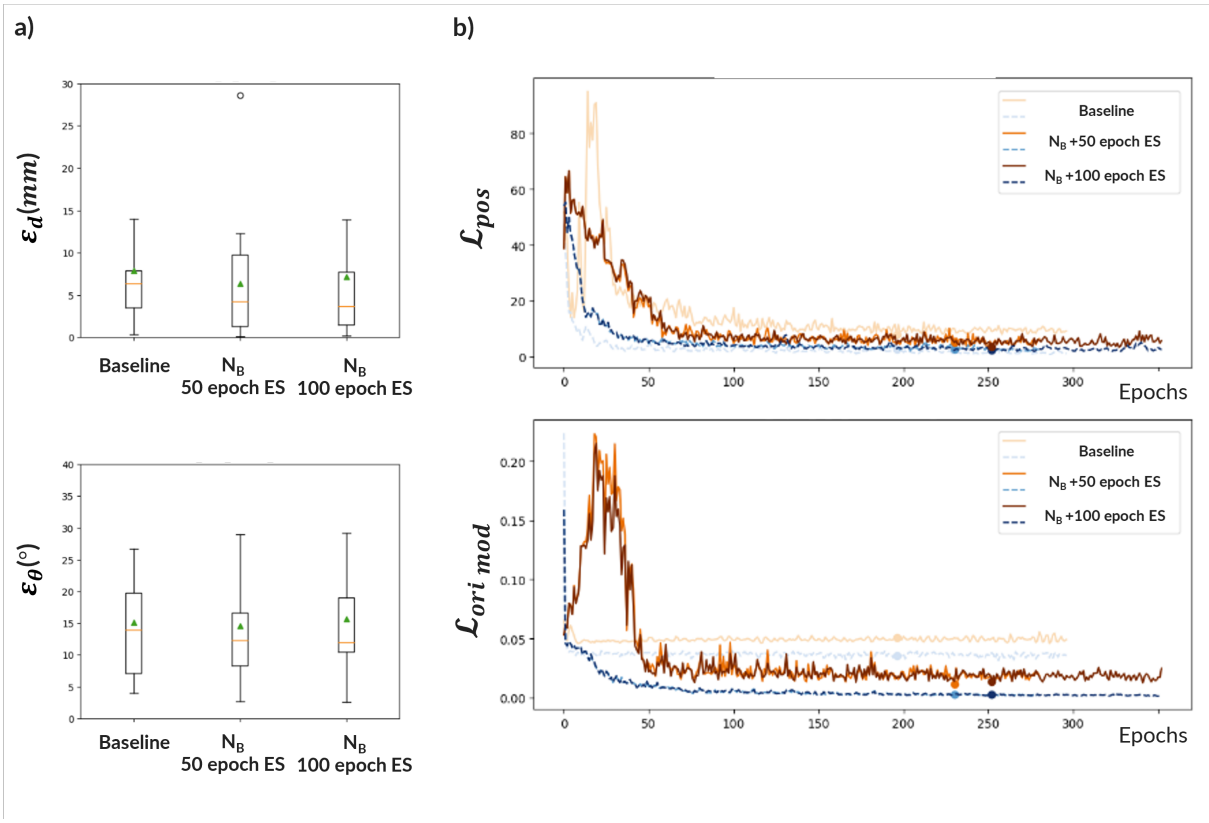


Figure 6.2: 3CH view model test set performance (a) and curves (b) from last baseline (variance adjusted + $\mathcal{L}_{ori_{mod}}$) compared to best training setting found for the 2CH view model (N_B) and the later with increased early stopping patience from 50 to 100. Validation curve plotted as a full line and the train curve as a dotted one. Best epoch is marked with a blue and red dot on the validation and training curves, respectively.

Test set displacement errors also shift to lower values (lower median), which can be explained by also lower validation position loss after applying the changes (see Figure 6.2).

Increasing the early stopping patience does not lead to a significant difference in the learning behaviour. The best model is achieved for patience of 50 epochs.

6.2.2.B 4CH

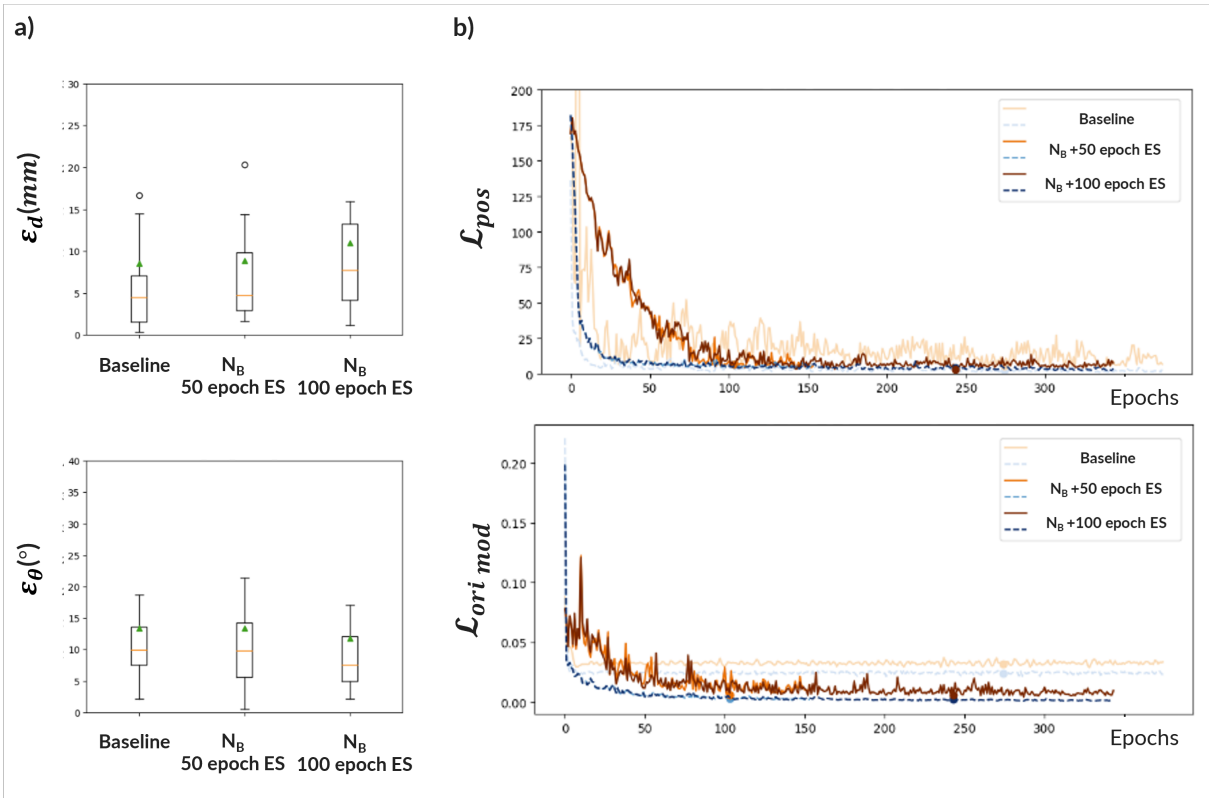


Figure 6.3: 4CH view model test set performance (a) and curves (b) from last baseline (variance adjusted + $\mathcal{L}_{ori_{mod}}$) compared to best training setting found for the 2CH view model (N_B) and the later with increased early stopping patience from 50 to 100. Validation curve plotted as a full line and the train curve as a dotted one. Best epoch is marked with a blue and red dot on the validation and training curves, respectively.

Contrarily to what seen for the 2CH and 3CH views, the applied changes only benefit the orientation prediction task. Both orientation train and validation losses converge at lower values after applying the changes and lead to a test set angulation error spanning to lower values shown by a lower median value. On the other hand, despite the position validation loss converging at lower values than before, the test set displacement errors actually show a slight shift towards larger values. Once more, these conflicting behaviours between the position validation and test set performance when compared to the previous baseline are likely to be caused by the heterogeneity of our dataset, which might have lead to a test split with distinct features unseen in the validation one.

The increase of the early stopping patience parameter further increases the gap between the position and orientation prediction performance.

The best model is considered to be achieved when applying the changes from the 2CH experiments and while using a patience of 50 epochs. The significantly lower convergence state of the orientation train and validation losses allied to the slight improvement in orientation test set performance compensate for the slight decrease in position prediction performance.

6.2.2.C SAX

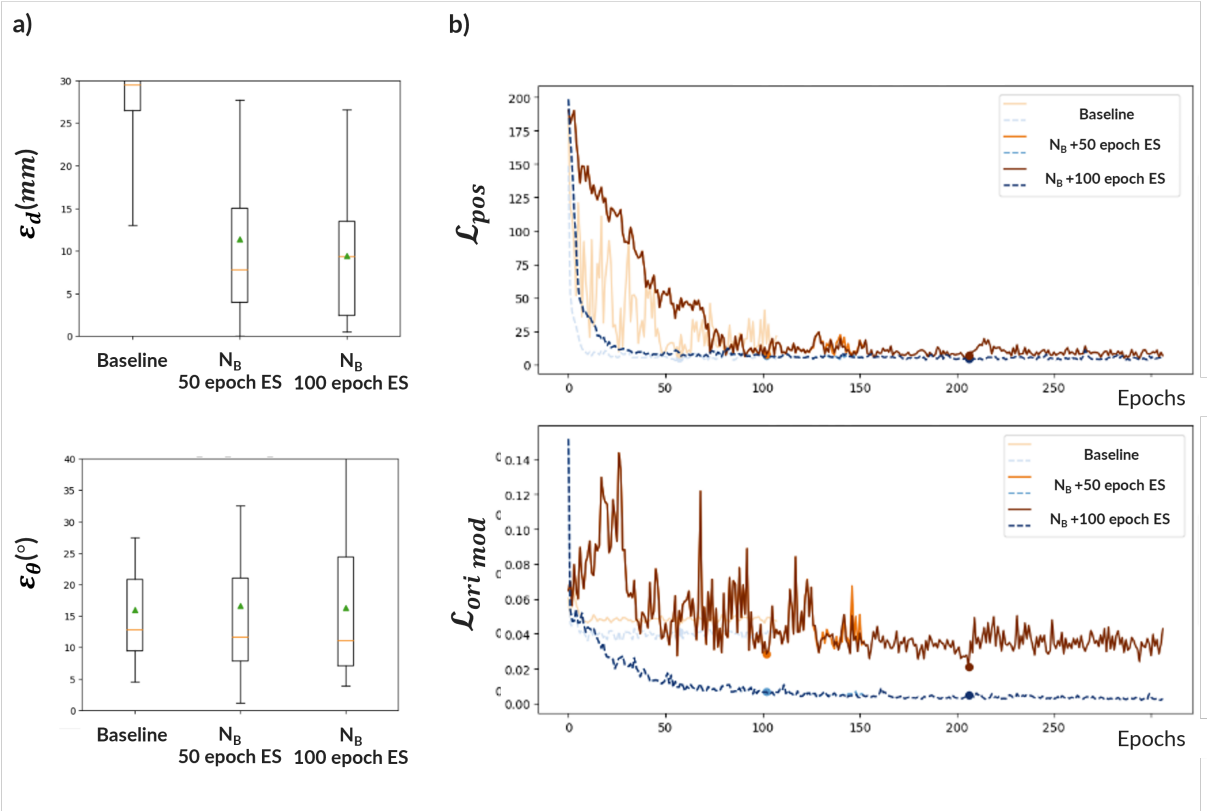


Figure 6.4: SAX view model test set performance (a) and curves (b) from last baseline (variance adjusted + $\mathcal{L}_{ori_{mod}}$) compared to best training setting found for the 2CH view model (N_B) and the later with increased early stopping patience from 50 to 100. Validation curve plotted as a full line and the train curve as a dotted one. Best epoch is marked with a blue and red dot on the validation and training curves, respectively.

Just like in 2CH and 3CH, both orientation train and validation losses converge at lower values after applying the changes, leading to lower test set angulation errors as depicted by the median value. Likewise, test set displacement errors also shift to lower values (lower median), which can be explained by also lower validation position loss after applying the changes. Training on this new setting enabled the loss curves to be less noisy during the early training, which in turn allowed the training to run for longer even when patience is still fixed to 50 epochs. After increasing it to 100 epochs, the positive effect

of these changes is enhanced as both task's losses reach lower values and, consequently, lower test set errors. This last improvement with the patience parameter is more clearly understood for the orientation task as both median and mean error values decrease. In the position prediction task this effect is more subtle as, although the distribution is slightly shifted down, only the mean displacement error decreases.

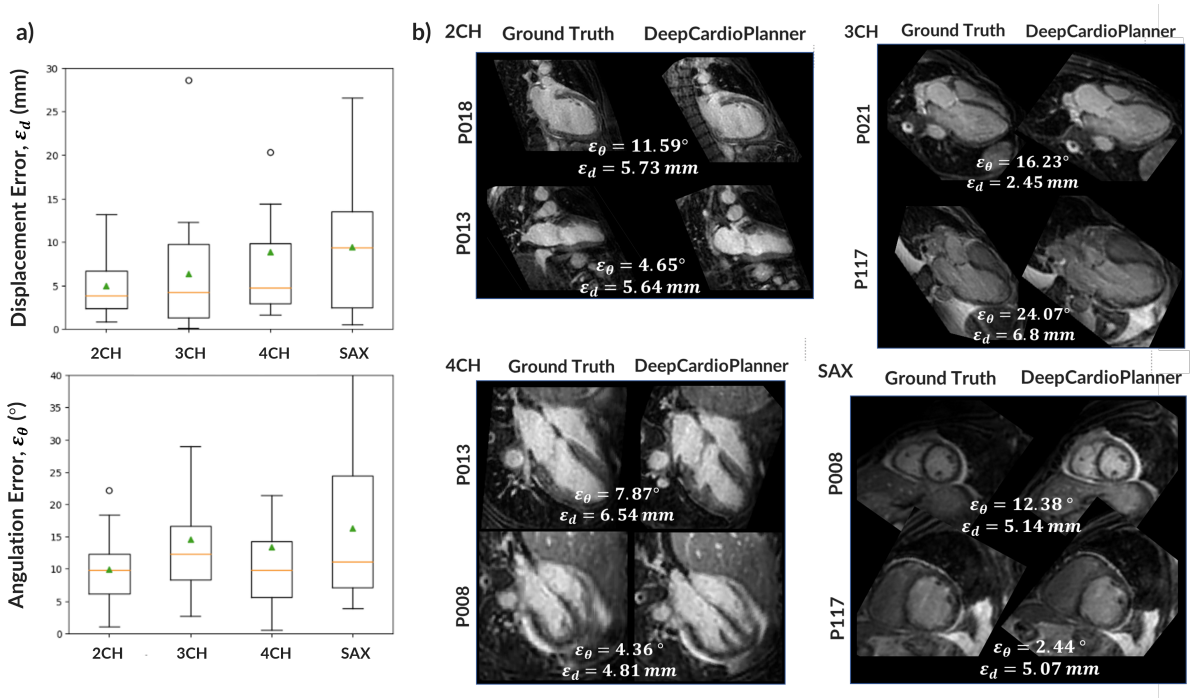


Figure 6.5: a) Box plot of the test set displacement (ε_d) and angulation error (ε_θ) of the best view models found for each view plane prediction task. Median values indicated by an orange dash and the mean values by a green triangle. b) Example images of ground truth and predicted slices by the best model tuned for each view.

In summary, despite existing some effects due to the heterogeneity of the dataset, it is possible to say that the changes prescribed based on the 2CH view prediction task generalise to the 3CH, 4CH and SAX prediction tasks. However, one must note that future work should include the hyperparameter tuning for each specific view plane model, to maximise the performance.

DeepCardioPlanner comprises the top-performing models, one for each view, which collectively enable the automatic view prescription for all four standard CMR view planes. Figure 6.5 provides an overview of the test set performances of *DeepCardioPlanner* while also depicting some example test predictions compared to their respective ground truths. Note that often what appear to be larger errors do not always relate to a great loss of plane prediction accuracy from a visual assessment.

6.2.3 Comparison with literature

In this section we compare our results with the ones obtained by Chen et al [14] with CT imaging and a STL approach. Let us analyse Figure 6.6 whose corresponding median, 1st and 3rd quartile values can be seen in Table 6.1.

With regards to the orientation prediction, our MTL approach leads to angulation errors that not only are comparable to that of [14] but are also consistently within the range of the inter-reader differences. This observation applies for all view prediction tasks apart from SAX as the inter-reader differences were not provided for this case.

These results serve as proof of concept of how MTL can be leveraged to achieve results comparable to STL, while only requiring one model per view plane.

In general we can see larger IQRs for our model performances, which is likely due to the larger heterogeneity of our dataset when compared to [14]. In fact, [14] makes use of strict inclusion criteria to maximise the quality of the training and test data, which is something that we could not afford to do due to the limited size of our dataset. In addition, our dataset incorporates both CINE and LGE images, introducing an increased level of complexity to the modeling task due to the presence of contrast in some of the images. This is something not accounted for in [14].

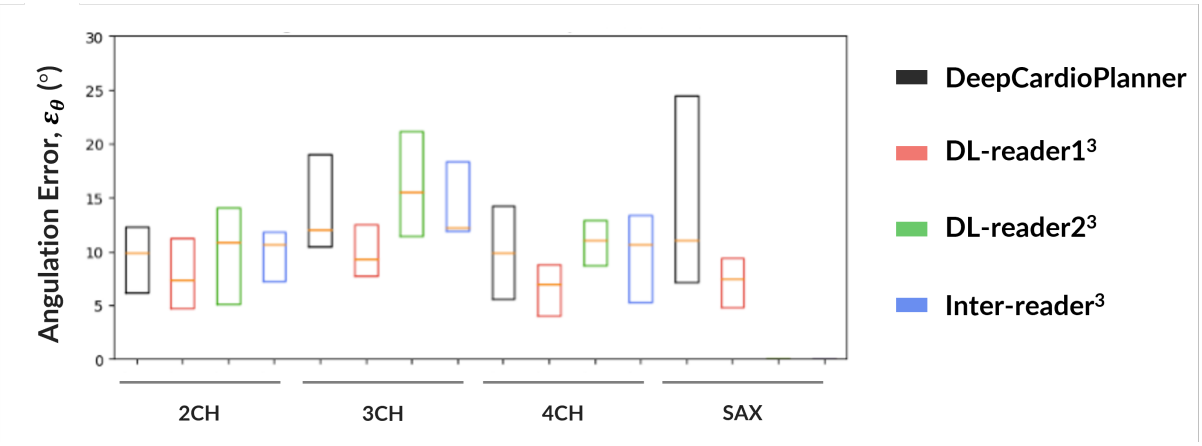


Figure 6.6: Comparison of test set angulation error distribution between our tool and the method by Chen et al [14]. DeepCardioPlanner: Our performances; DL-reader1: Their performances on samples with prescription by reader1; DL-reader2: Their performances on samples with prescription by reader2; Inter-reader: Differences in plane orientation between the two operators.

Considering only MR imaging, Table 6.2 compares the angulation prediction performance by means of mean and standard deviation of angulation error distribution between DeepCardioPlanner and two other ML based automated view prescription strategies.

The *DeepCardioPlanner* orientation prediction performance did not quite reach the level attained by the cardiac landmark based approach from [13], only the 2CH prescription accuracy comes close to their values. Considering that the refining of the training setting was based solely on 2CH view prediction

Table 6.1: Test set angulation error compared between our method and the equivalent STL method in CT from Chen et al [14]. DeepCardioPlanner: Our performances; DL-reader1: Their performances on samples from the same operator that generated their training set; DL-reader2: Their performances on samples from a never seen operator; Inter-reader: Differences in plane orientation between the two operators. Differences were reported as median (IQR)

Comparison	**Cardiac View**			
	2CH	**3CH**	**4CH**	**SAX**
DeepCardioPlanner	9.8 (6.2,12.3)	12.0 (10.4,19.0)	9.8 (5.6,14.2)	11.1 (7.1,24.5)
DL-reader1	7.3 (4.7,11.2)	9.3 (7.7,12.5)	7.0 (4.0,8.8)	7.5 (4.8,9.4)
DL-reader2	10.9 (5.1,14.2)	15.5 (11.4,21.2)	11.1 (8.7,12.9)	-
Inter reader	10.6 (7.2,11.8)	12.2 (11.9,18.4)	10.6 (5.3,13.4)	-

task, it is fair to assume that individually tuning the training setting for each view predicting model will close the gap between the performances of these two approaches.

We must also take into account that [13] base their approach on 482 same sequence cardiac MRI studies, while we had available 120 studies, including both CINE and LGE sequences. This is also a factor contributing to our lower test set performance. However, we can easily overcome this challenge by leveraging the nature of our research problem. It only adds 10 extra seconds to the regular CMR protocol to acquire a new sample. In contrast, a landmark based approach like [13] would require additional domain expert cardiac landmark annotations which are costly.

Additionally, it is relevant to note that, unlike *DeepCardioPlanner*, the approach proposed by [13] does not provide fully automatic view prescriptions as it assumes an initial localiser slice that can still be hard to prescribe.

With all these factors considered, the approach taken in *DeepCardioPlanner* has the potential to become a viable alternative to landmark based automated view prescription in CMR.

Finally, *DeepCardioPlanner* also struggles to reach the performance attained by [11] using RL. However, our trainings are only approximately 2 hours long, while in theirs are between 2 to 4 days. Given this, our approach is preferred.

Table 6.2: Test set angulation error compared between our method, the landmark-based method from [13], and the RL-based approach from [11].

Comparison	**Cardiac View**			
	2CH	**3CH**	**4CH**	**SAX**
DeepCardioPlanner	9.98 ± 5.54	15.65 ± 10.88	13.38 ± 14.63	16.35 ± 11.59
Blansit et al [13] (Landmark based)	6.53 ± 6.28	9.02 ± 8.83	5.16 ± 3.80	4.93 ± 4.86
Alansary et al [11] (RL)	-	-	8.72 ± 7.44	-

7

Final Remarks

Contents

7.1 Summary of main findings

In this work we have explored how it is possible to leverage a MTL approach to train one deep learning model to predict each one of the standard SAX, 2CH, 3CH and 4CH CMR view planes, from a rapidly acquired 3D scan (∼10sec).

The nature of our problem allows two equally correct solutions for each plane's orientation, i.e. the one actually prescribed and its flipped version. Consequently, each view's dataset plane orientation distribution can appear bimodal, with a 180°angle between the two modes. To ensure that both modes are deemed as correct solutions we not only propose a specific label preprocessing step as well as a re-design of our orientation loss function. Both approaches successfully address the problem leading to improved convergence and better performance metrics.

Orientation loss scaling is introduced to bring the two losses to the same range of values and ensure that neither one of them dominates the training. This helps increasing the model's ability to model the orientation task as its train loss starts converging at much lower values, increasing the model's effective capacity to learn. Through the same reasoning, the learning rate is also tuned to the value that allows the lowest train loss in both tasks.

We explore the use of uncertainty based loss weighting to force the model to also learn the best loss weight values during training. This approach does not lead to any significant impact on the model's ability to learn the tasks as the weights barely vary during training. This suggests that the homeoscedastic uncertainty, upon which this loss weighting is based, does not vary enough between the two tasks for it to yield any effect.

Here, two different network architectures with different levels of parameter sharing between the two tasks are compared, finding that having a separate head dedicated to each task outperforms the baseline fully shared design. Forcing every layer of the network to be shared between both tasks as excessively constraining the learning and hampering the model's ability to learn.

A particular data augmentation pipeline is also proposed, finding that a moderate balance of data augmentation intensity benefits model test performance which shows its improved generalisation ability.

We have also studied how using higher resolution, more detailed scans impact model performance. It was found that the finer details given by a 3mm isotropic resolution volume did not benefit model performance nor training time, when in comparison to a 4mm isotropic one. This agrees with the traditional CMR view prescription protocol which relies more on higher level features of the scans, rather than fine variations of pixel intensity which could be here acting as confounding factors and hence hampering learning.

Furthermore, given the losses we have designed and training setting, using a MTL approach outperforms having a separate model for each task (STL). Training on solely the position loss leads to a gradual improvement of orientation loss during training which highlights the inter-dependency of the two tasks

and explains the better performance achieved when training on both objectives simultaneously.

We have also seen how all the changes we have made to the training setting based on the 2CH prediction task generalise quite well to the other tasks, highlighting how all the view predicting models suffered from the same type of underfitting problems we aimed to address for the 2CH model.

DeepCardioPlanner comprises the top-performing models, one for each view, which collectively enable the automatic view prescription for all four standard CMR view planes from a rapidly acquired scan.

DeepCardioPlanner yields performances metrics within the literature ranges for the same task with CT and wherein STL approach was taken [14]. Not only that but the attained error distributions are much alike the inter operator differences from the same work [14]. The resemblance in the error distributions further supports the validity and reliability of our model's predictions, as they align with the inherent variability observed in human assessments. In addition, these results serve as proof of how MTL can be leveraged to achieve results comparable to STL, while only requiring one model per view plane.

We further improve upon previous work as we manage to obtain these satisfactorily accurate results without the use of cardiac landmark annotations for model training. Despite landmark based approaches having been shown to provide good results, they require expert operators to manually annotate the location of several cardiac landmarks which is a lengthy and complex task that poses a constraint to increasing the amount of data available. An approach like ours relies on a dataset acquired solely based on an extra 10 seconds added to the protocol per sample while adding no extra complexity to the task, which facilitates gathering more data for model training. This is quite an important factor when considering addressing the usual constraint of deep learning projects which is the lack of data.

This automatic method has the potential to greatly reduce examination time and complexity as it is based on 10sec scans and 1sec prescription time, which may increase efficiency and clinical utility of CMR.

7.2 Limitations

It is important to acknowledge some other limitations of our work inherent to our research design.

Firstly, the sample size in this study was relatively small (120 patients), which may limit the generalisability of our findings to larger populations. Additionally, due to resource constraints, our data comes from a single site in Sweden and also from a single scanner, meaning that these results might also not generalise to other demographics or scanner models.

Also, some of the analysis could have benefit from k-fold cross validation to mitigate the GPU inherent variance and guarantee extra robustness to our results. This did not happen due to the lengthy nature of our trainings and the fact that our access to a GPU was very limited.

Despite the changes decided based on the 2CH model having, for the most part, generalised to the remaining views, it is likely that a finer and individual tuning of the learning rate, loss scale factors and

data augmentation parameters for each of the view predicting models would improve results. Given that each view plane dataset exhibits distinct characteristics, it is reasonable to anticipate that the models would benefit from different training settings. Again, the access to the GPU is once more a constraint.

7.3 Future Work

In light of the findings presented in this study, there are several promising avenues for future research that could build upon our current work.

Firstly, expanding the sample size to include data from more centres with different scanners in different locations of the world would strengthen the generalisability of our results.

Next, it is likely that a more refined MTL approach will benefit performance. Particularly, we leave as future work experimenting with other less constraining approaches of parameter sharing like soft parameter sharing [8] and other loss weighting approaches, like GradNorm [46], which focus more in balancing the pace at which tasks are learned, while avoiding gradients of different magnitude rather than just relying on uncertainty differences between the tasks.

Moreover, besides the previously mentioned finer tuning of the hyperparameters, other augmentation techniques like distortion and k-space corruption would also be worth experimenting with.

For further works, it would be recommended to perform a statistical analysis in order to assess if the performance error metrics are statistically different from zero. Also, designing and conducting a clinical reading where domain experts would classify the quality and clinical relevance of the predicted planes would contribute with extra robustness to any future results.

Finally, one other interesting avenue to explore is whether we can extend this multi-task learning approach from single view prediction per model to a multi view prediction per model. Training the model to prescribe all view planes jointly might produce good results by leveraging the relative position and orientation between views.

Bibliography

[1] D. W. McRobbie, E. A. Moore, M. J. Graves, and M. R. Prince, *MRI from Picture to Proton*. Cambridge university press, 2017.

[2] ThoughtCo, "The Heart Wall: Anatomy, Function, and Conditions," https://www.thoughtco.com/the-heart-wall-4022792, [Accessed: 28th May 2023].

[3] B. Herzog, J. Greenwood, S. Plein, P. Garg, P. Haaf, and S. Onciul, "Cardiovascular magnetic resonance pocket guide," *Eur Soc Cardiol*, 2017.

[4] G. Chartrand, P. M. Cheng, E. Vorontsov, M. Drozdzal, S. Turcotte, C. J. Pal, S. Kadoury, and A. Tang, "Deep learning: a primer for radiologists," *Radiographics*, vol. 37, no. 7, pp. 2113–2131, 2017.

[5] Unknown, "Neural networks," Online, 2023, accessed on June 29, 2023. [Online]. Available: https://belvederef.github.io/cv-notebook/computer-vision-theory/neural-networks.html

[6] J. Feng, X. He, Q. Teng, C. Ren, H. Chen, and Y. Li, "Reconstruction of porous media from extremely limited information using conditional generative adversarial networks," *Physical Review E*, vol. 100, 09 2019.

[7] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[8] S. Ruder, "An overview of multi-task learning in deep neural networks," 6 2017. [Online]. Available: http://arxiv.org/abs/1706.05098

[9] M. Frick, I. Paetsch, C. D. Harder, M. Kouwenhoven, H. Heese, S. Dries, B. Schnackenburg, W. D. Kok, R. Gebker, E. Fleck, R. Manka, and C. Jahnke, "Fully automatic geometry planning for cardiac mr imaging and reproducibility of functional cardiac parameters," *J. Magn. Reson. Imaging*, vol. 34, pp. 457–467, 2011.

[10] X. Lu, M.-P. Jolly, B. Georgescu, C. Hayes, P. Speier, M. Schmidt, X. Bi, R. Kroeker, D. Comaniciu, P. Kellman, E. Mueller, and J. Guehring, "Lncs 6893 - automatic view planning for cardiac mri acquisition," 2011.

[11] A. Alansary, L. L. Folgoc, G. Vaillant, O. Oktay, Y. Li, W. Bai, J. Passerat-Palmbach, R. Guerrero, K. Kamnitsas, B. Hou, S. McDonagh, B. Glocker, B. Kainz, and D. Rueckert, "Automatic view planning with multi-scale deep reinforcement learning agents," 6 2018. [Online]. Available: https://arxiv.org/abs/1806.03228

[12] M. Le, J. Lieman-Sifry, F. Lau, S. Sall, A. Hsiao, and D. Golden, "Computationally efficient cardiac views projection using 3d convolutional neural networks," 11 2017. [Online]. Available: http://arxiv.org/abs/1711.01345

[13] K. Blansit, T. Retson, E. Masutani, N. Bahrami, and A. Hsiao, "Deep learning–based prescription of cardiac MRI planes," *Radiology: Artificial Intelligence*, vol. 1, no. 6, p. e180069, Nov. 2019. [Online]. Available: https://doi.org/10.1148/ryai.2019180069

[14] Z. Chen, M. Rigolli, D. M. Vigneault, S. Kligerman, L. Hahn, A. Narezkina, A. Craine, K. Lowe, and F. Contijoch, "Automated cardiac volume assessment and cardiac long- and short-axis imaging plane prediction from electrocardiogram-gated computed tomography volumes enabled by deep learning," *European Heart Journal - Digital Health*, vol. 2, no. 2, pp. 311–322, Mar. 2021. [Online]. Available: https://doi.org/10.1093/ehjdh/ztab033

[15] P. A. Corrado, D. P. Seiter, and O. Wieben, "Automatic measurement plane placement for 4d flow MRI of the great vessels using deep learning," *International Journal of Computer Assisted Radiology and Surgery*, vol. 17, no. 1, pp. 199–210, Aug. 2021. [Online]. Available: https://doi.org/10.1007/s11548-021-02475-1

[16] G. D. Flora and M. K. Nayak, "A brief review of cardiovascular diseases, associated risk factors and current treatment regimes," *Current pharmaceutical design*, vol. 25, no. 38, pp. 4063–4084, 2019.

[17] C. S. Punla, https://orcid.org/ 0000-0002-1094-0018, cspunla@bpsu.edu.ph, R. C. Farro, https://orcid.org/0000-0002-3571-2716, rcfarro@bpsu.edu.ph, and Bataan Peninsula State University Dinalupihan, Bataan, Philippines, "Are we there yet?: An analysis of the competencies of BEED graduates of BPSU-DC," *International Multidisciplinary Research Journal*, vol. 4, no. 3, pp. 50–59, Sep. 2022.

[18] G. M. Pohost, "The history of cardiovascular magnetic resonance," *JACC: Cardiovascular Imaging*, vol. 1, no. 5, pp. 672–678, Sep. 2008. [Online]. Available: https://doi.org/10.1016/j.jcmg.2008.07.009

[19] "Dot at magnetom world." [Online]. Available: https://www.magnetomworld.siemens-healthineers.com/clinical-corner/protocols/dot-engines

[20] e. a. Pueyo J., "Cardiac dot engine: Significant time reduction at cardiac magnetic resonance imaging," *MAGNETOM Flash 5*, 2014.

[21] F. Atzeni, M. Corda, L. Gianturco, M. Porcu, P. Sarzi-Puttini, and M. Turiel, "Cardiovascular imaging techniques in systemic rheumatic diseases," 2 2018.

[22] W. A. AlJaroudi and F. G. Hage, "Review of cardiovascular imaging in the journal of nuclear cardiology 2020: positron emission tomography, computed tomography, and magnetic resonance," pp. 2100–2111, 10 2021.

[23] V. L. Murthy, M. Naya, C. R. Foster, J. Hainer, M. Gaber, G. D. Carli, R. Blankstein, S. Dorbala, A. Sitek, M. J. Pencina, and M. F. D. Carli, "Improved cardiac risk assessment with noninvasive measures of coronary flow reserve," *Circulation*, vol. 124, pp. 2215–2224, 11 2011.

[24] M. Salerno, B. Sharif, H. Arheden, A. Kumar, L. Axel, D. Li, and S. Neubauer, "Recent advances in cardiovascular magnetic resonance," *Circulation: Cardiovascular Imaging*, vol. 10, no. 6, Jun. 2017. [Online]. Available: https://doi.org/10.1161/circimaging.116.003951

[25] T. F. Ismail, W. Strugnell, C. Coletti, M. Božić-Iven, S. Weingärtner, K. Hammernik, T. Correia, and T. Küstner, "Cardiac MR: From theory to practice," *Frontiers in Cardiovascular Medicine*, vol. 9, Mar. 2022. [Online]. Available: https://doi.org/10.3389/fcvm.2022.826283

[26] T. A. McDonagh, M. Metra, M. Adamo, R. S. Gardner, A. Baumbach, M. Böhm, H. Burri, J. Butler, J. Čelutkienė, O. Chioncel, J. G. F. Cleland, A. J. S. Coats, M. G. Crespo-Leiro, D. Farmakis, M. Gilard, S. Heymans, A. W. Hoes, T. Jaarsma, E. A. Jankowska, M. Lainscak, C. S. P. Lam, A. R. Lyon, J. J. V. McMurray, A. Mebazaa, R. Mindham, C. Muneretto, M. F. Piepoli, S. Price, G. M. C. Rosano, F. Ruschitzka, A. K. Skibelund, R. A. de Boer, P. C. Schulze, M. Abdelhamid, V. Aboyans, S. Adamopoulos, S. D. Anker, E. Arbelo, R. Asteggiano, J. Bauersachs, A. Bayes-Genis, M. A. Borger, W. Budts, M. Cikes, K. Damman, V. Delgado, P. Dendale, P. Dilaveris, H. Drexel, J. Ezekowitz, V. Falk, L. Fauchier, G. Filippatos, A. Fraser, N. Frey, C. P. Gale, F. Gustafsson, J. Harris, B. Iung, S. Janssens, M. Jessup, A. Konradi, D. Kotecha, E. Lambrinou, P. Lancellotti, U. Landmesser, C. Leclercq, B. S. Lewis, F. Leyva, A. Linhart, M.-L. Løchen, L. H. Lund, D. Mancini, J. Masip, D. Milicic, C. Mueller, H. Nef, J.-C. Nielsen, L. Neubeck, M. Noutsias, S. E. Petersen, A. S. Petronio, P. Ponikowski, E. Prescott, A. Rakisheva, D. J. Richter, E. Schlyakhto, P. Seferovic, M. Senni, M. Sitges, M. Sousa-Uva, C. G. Tocchetti, R. M. Touyz, C. Tschoepe, J. Waltenberger, M. Adamo, A. Baumbach, M. Böhm, H. Burri, J. Čelutkienė, O. Chioncel, J. G. F. Cleland, A. J. S. Coats, M. G.

Crespo-Leiro, D. Farmakis, R. S. Gardner, M. Gilard, S. Heymans, A. W. Hoes, T. Jaarsma, E. A. Jankowska, M. Lainscak, C. S. P. Lam, A. R. Lyon, J. J. V. McMurray, A. Mebazaa, R. Mindham, C. Muneretto, M. F. Piepoli, S. Price, G. M. C. Rosano, F. Ruschitzka, and A. K. S. and, "2021 ESC guidelines for the diagnosis and treatment of acute and chronic heart failure," *European Heart Journal*, vol. 42, no. 36, pp. 3599–3726, Aug. 2021. [Online]. Available: https://doi.org/10.1093/eurheartj/ehab368

[27] J. C. Carr, O. Simonetti, J. Bundy, D. Li, S. Pereles, and J. P. Finn, "Cine MR angiography of the heart with segmented true fast imaging with steady-state precession," *Radiology*, vol. 219, no. 3, pp. 828–834, Jun. 2001. [Online]. Available: https://doi.org/10.1148/radiology.219.3.r01jn44828

[28] C. M. Kramer, J. Barkhausen, C. Bucciarelli-Ducci, S. D. Flamm, R. J. Kim, and E. Nagel, "Standardized cardiovascular magnetic resonance imaging (CMR) protocols: 2020 update," *Journal of Cardiovascular Magnetic Resonance*, vol. 22, no. 1, Feb. 2020. [Online]. Available: https://doi.org/10.1186/s12968-020-00607-1

[29] T. M. Mitchell, *Machine Learning: An Artificial Intelligence Approach*. McGraw-Hill, 1997.

[30] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[31] R. Caruana, "Multitask learning: A knowledge-based source of inductive bias," in *International Conference on Machine Learning*, 1993.

[32] O. Sener and V. Koltun, "Multi-task learning as multi-objective optimization," *Advances in neural information processing systems*, vol. 31, 2018.

[33] A. K. Lampinen and S. Ganguli, "An analytic theory of generalization dynamics and transfer learning in deep linear networks," *arXiv preprint arXiv:1809.10374*, 2018.

[34] R. Cipolla, Y. Gal, and A. Kendall, "Multi-task learning using uncertainty to weigh losses for scene geometry and semantics." IEEE Computer Society, 12 2018, pp. 7482–7491, works under the assumption that the label vectors follow a gaussian distribution¡br/¿.

[35] P. Kelvin Chow, "16 months of exercise – a case study of automated cmr with cardiac dot engine," *MAGNETOM Flash SCMR Edition 2020*, 2020. [Online]. Available: https://cdn0.scrvt.com/39b415fb07de4d9656c7b516d8e2d907/1800000007049336/ba98d934c789/Chow_Automated_CMR_MAGNETOM_Flash_SCMR_2020_1800000007049336.pdf

[36] A. Paszke, A. Chaurasia, S. Kim, and E. Culurciello, "Enet: A deep neural network architecture for real-time semantic segmentation."

[37] L. Innolitics, "Dicom standard browser - image plane position." [Online]. Available: https://dicom.innolitics.com/ciods/mr-image/image-plane/00200032

[38] ——, "Dicom standard browser - image plane orientation." [Online]. Available: https://dicom.innolitics.com/ciods/mr-image/image-plane/00200037

[39] "Tf.keras.utils.sequence :   tensorflow v2.10.0." [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/utils/Sequence

[40] "tf.keras.losses.CosineSimilarity," TensorFlow API Documentation, accessed on June 30, 2023. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/losses/CosineSimilarity

[41] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," 2 2015. [Online]. Available: https://arxiv.org/abs/1502.03167

[42] TensorFlow, "L2," https://www.tensorflow.org/api_docs/python/tf/keras/regularizers/L2, 2023, accessed May 28, 2023.

[43] ——, "Glorotuniform," https://www.tensorflow.org/api_docs/python/tf/keras/initializers/GlorotUniform, 2023, accessed May 28, 2023.

[44] "Earlystopping," https://keras.io/api/callbacks/early_stopping/, 2023, accessed May 28, 2023.

[45] Y. Gal, "Multi-task learning example," GitHub repository, accessed on June 30, 2023. [Online]. Available: https://github.com/yaringal/multi-task-learning-example

[46] Z. Chen, V. Badrinarayanan, C.-Y. Lee, and A. Rabinovich, "Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks," in *International conference on machine learning*. PMLR, 2018, pp. 794–803.