Master of Science
Thesis
HT2021

Using CNNs to Predict Rigid Body Transformation Parameters Which Register Fat Navigators to Apply Prospective Motion Correction in MRI at 7 T

Thomas Olausson

Supervisors

Linda Knutsson, Vincent O. Boer, Jan Ole Pedersen, Esben Thade Petersen, Mads Andersen

This work has been performed at the Danish Research Centre for Magnetic Resonance

Medical Radiation Physics, Lund
Faculty of Science
Lund University
www.msf.lu.se

To the students in the medical physics program at Lund University, who are the ones most likely to read this.

Thomas Olausson, Master of Science dissertation: *Using CNNs to Predict Rigid Body Transformation Parameters Which Register Fat Navigators to Apply Prospective Motion Correction on MRI Sequences at 7 T.* October 2022

Acknowledgments

I wish to show my appreciation to all my supervisors who have guided me throughout the entirety of this project and made this work possible.

Additionally, the following people deserve my gratitude

- Torkil Svensgaard & Ruben Vestergaard – for their great help and tolerance for all my IT related questions.
- Simon Kevin – for the crash-course in C++.
- Simon Yamazaki Jensen & Oula Puonti – for all the interesting discussions.
- Linda Knutsson – for the constant support and faith in me.
- Fang Cao – for the crash-course in networking interface.
- Jan Ole Pedersen – for taking on this project, even though this wasn't assigned to him. A great deal of progress wouldn't have been possible without his efforts.
- Of course, to my parents – for their never-ending support in stuff outside this project.

Lund,
February 13, 2022

Thomas Olausson

Popularized summary in Swedish

Djupinlärningsbaserad bildregistrering för rörelsekorrigering i MRI

MRI är en icke-invasiv diagnostisk procedur som kan ge radiologer väsentlig strukturell eller funktionell information om organ. Ungefär som vid fotografering kan rörelse under förvärvet öka risken för dålig bildkvalitet. Medicinska bildbehandlingsmetoder är inte immuna mot detta problem. Bildtagningstiden varierar beroende på de fysikprinciper som utnyttjas för att ta bilden. MRI är känt för sina relativt långa avbildningstider, vilket kräver att patienter ligger stilla i skannern. Anledningen till detta är att MRI utnyttjar signalen som erhålls från protonerna av vattenmolekyler i patientens kropp via Faradays lag. Olika spatiala frekvenser appliceras på området av intresse. Därför kommer spatiala frekvenser som representerar den volym bäst kommer att producera mest signal. För att få denna signal måste protonerna exciteras till ett högre energitillstånd. Med tiden kommer dessa protoner att återgå till grundtillståndet. En annan faktor till de relativt långa insamlingstiderna är skapandet av de spatiala frekvenser som krävs för att erhålla en acceptabel bild.

Bilder tagna med MR-skanner som är av dålig diagnostisk kvalitet kan kräva att patienter kallas tillbaka till kliniken och undersöks på nytt. Enligt några nyare studier är denna risk för en omundersökning på grund av rörelse under bildinsamling minst 16 %. Denna fråga är framträdande vid pediatriska undersökningar. En vanlig lösning som används för detta problem är att använda lugnande medel eller anestesimedel. Men användningen av dessa typer av droger hos barn är ett omtvistat ämne på grund av dess potentiella negativa hälsoeffekter. Därför finns det en önskan att utveckla MRI-tekniker för att korrigera för rörelse under en skanning.

Ett allmänt sätt att korrigera för rörelse är genom att skaffa information om rörelsen under en undersökning och tillämpa någon sorts korrigering i realtid. Detta har vissa MRI-forskare visat i ett program som de kallade iMOCO. Tanken är att bestämma patientens position i skannern i realtid genom att få en snabb lågupplöst bild av fettet som omger skallen. Två bilder vid olika tidpunkter skjuts igenom en iterativ lösare som försöker få bilderna att helt överlappa med bildtransformationer. Beskrivningen av patienternas rörelser mellan dessa två tidpunkter definieras som dessa transformationer. Sedan används en rörelsepoäng för att bedöma om data som samlats in mellan dessa två tidpunkter är potentiellt värdelösa för god bildkvalitet och behöver krävas, eller så kan avbildningsparametrarna omdefinieras till den nya patientens position.

För att ett sådant system ska fungera i en MRI-pipeline måste det vara snabbt. Specifikt i storleksordningen 100-tals millisekunder. Ett exploderande forskningsfält är djupinlärning. Potentialen med att noggrant lära in mönster av problem för att tilldela vikter till neuroner som kan utföra en tidvis beräkning har visat sig vara möjlig med djupinlärning. Deep learning-forskare har skapat flera olika typer av nätverk som har kunnat överlappa medicinska bilder. En potentiell fördel för iMOCO-programmet skulle vara att ersätta den iterativa lösaren med ett nätverk för djupinlärning. Man tror att detta nätverk skulle kunna utföra bildregistreringen inom en fast tidsperiod, vilket är en fördel jämfört med den iterativa lösaren eftersom det tar en obestämd tid att nå en acceptabel lösning för ett specifikt bildpar. En annan potentiell fördel är att detta nätverk kan utföra en korrekt registrering på kortare tid.

Huvudsyftet med detta examensarbete var att utveckla ett sådant nätverk och implementera det på en mjukvara för MR-skannrar för att testa det in vivo. Båda målen uppnåddes framgångsrikt, men nätverkets noggrannhet vid registrering av bilder var dålig i jämförelse med iterativa lösare. Olika tillvägagångssätt för att träna ett nätverk undersöktes, och resultaten visade att indata spelade en stor roll för uppgifternas noggrannhet. Ej övervakade tillvägagångssätt för träning överträffade de övervakade träningsmetoderna, troligen på grund av utbildningen på verkliga data i den ej övervakade metoden. Den här avhandlingen visade att det krävs mer tänkande för att träna ett nätverk för att utveckla ett genomförbart nätverk som kan ersätta den iterativa lösaren i iMOCO-programmet.

Abstract

The framework for a fat navigator-based prospective motion correction (PMC) is already embedded on the 7 T scanner at Hvidovre hospital. This framework is dubbed iMOCO, and it tackles the motion artifact problem by registering fat navigators after each k-space readout train to some reference fat navigator taken at the start of the scan with rigid body transformation parameters. A grade of motion is determined by the magnitude of the registration. Based on the severity of the grade of motion, reacquisition of potentially motion corrupted lines in k-space can be performed or updates to the FOV can be made. This will reduce the risk for motion artifacts in the image for diagnosis. The registration method in iMOCO uses an iterative approach, which reaches acceptable solutions around 250 ms. Recently, deep learning (DL) has been used for several image registration tasks. The feasibility of replacing the current registration method in iMOCO with a DL network was investigated in this thesis.

The fundamental type of DL network used was convolution neural networks (CNN). The inputs into the network would be the image pair sent to the iMOCO program, which would be a reference fat navigator and a motion fat navigator. The output will be six values representing the rigid body transformations to register the two navigators. Two main design choices of CNNs were compared. These designs differed from how the input data was handled. One design extracted features of each image independently in different branches of a network. The other design choice had the image pair subtracted before inputted into a single branch in a network. The design choices were compared for interference speed and accuracy. The single branch design was trained once in a supervised manner and once in an unsupervised manner. The two-branch design was trained only in a supervised manner. The data set used for the supervised training was 18000 fat navigators from 100 subjects with synthetic motion fat navigators were created using rigid body transformations. The data set used for the unsupervised training was 2200 fat navigator pairs from 2 subjects. Embedding a DL network to the iMOCO program was done by network interface using a TCP communication between the host computer of the scanner and a computer with a GPU. This connection costs 15 ms for sending the necessary data for registration. The networks trained in a supervised manner were tested on data collected in vivo, and a comparison between the iMOCO registration and DL network registration was made by a simple difference between the two. The networks trained in an unsupervised manner were tested on the same data but compared with a normalized cross-correlation (NCC) metric before and after different registration methods.

With the initial data augmentations, the two-branch network and the one branch network reached a similar mean square error (MSE) on the validation data set after training. Using the one branch network at least halved the interference time to around 100 ms, then when using the two-branch network. After reducing the magnitude of transformations in the synthetic motion data, the difference between the registrations with DL and iMOCO decreased with the larger magnitudes of transformations. The unsupervised trained network did not outperform FreeSurfer's rigid body registration tool on unseen data. The NCC median FreeSurfer reached was while the DL network reached (without any applied registration). This was confirmed visually as well.

This thesis shows the importance of data in training neural networks. More realistic transformations in the generation of synthetic motion showed to have halved differences between more robust registration methods. There are practical limitations with the generation of synthetic motion data as well. An unsupervised trained network that used in vivo motion data showed to be able to learn to perform registration. However, the relatively small data set size makes it difficult to push the model trained in this thesis into deployment. The data set issues mentioned must be addressed before a feasible DL registration network can replace the current registration method in iMOCO.

Table of Contents

1 Introduction

1.1 Background

Patient motion artifacts in magnetic resonance (MR) examinations are the most common reason for patients to undergo re-examinations resulting in an increased cost for the healthcare (Afacan, et al., 2016). The origin of motion artifacts is primarily due to the relatively long data acquisition in magnetic resonance imaging (MRI). This has prompted the development of shortening acquisition durations such as parallel imaging, compressed sensing, and echo-planar imaging (EPI) (Usman, et al., 2020). Despite the recent advancements, motion still poses a problem since the duration of the acquisition is not short enough to not generate artifacts.

The results are images with worsened diagnostic quality, which requires the healthcare to reschedule patients and prolong the examination times. This is seen in all populations, but it is more prevalent in pediatric patients, patients with anxiety, the elderly, and patients with neurodegenerative diseases (Afacan, et al., 2016; Andersen, et al., 2019; Törnqvist, et al., 2006). Methods of reducing the impact of motion in image quality include the use of sedatives or anesthetic drugs. These methods may be problematic due to the long-term adverse effects of these drugs, and especially a concern in pediatric studies (Afacan, et al., 2016). In a 2015 report, 8.6% of children experienced a long-term effect from a sedation or anesthesia event, where 57.5% of these events were from MR examinations (Havidich, et al., 2016). Alternatively, computed tomography (CT) may be used when the concern for motion is significant. However, this exposes the subject to radiation which is generally considered a larger risk for children than the use of sedatives or anesthetic drugs. In addition, the CT images provide poor contrast in the brain (Afacan, et al., 2016). Furthermore, it is estimated that patient motion can call for healthcare costs of at least $115,000 per scanner per year (US prices). This estimation is based on the prevalence of motion artifacts which lead to an examination being repeated being 16.4% (Andre, et al., 2015). Hence, the investment into motion correction research is beneficial for the healthcare.

Today there exist motion correction techniques for MRI in clinical practice and development phases. These can be divided into two categories, based on when the correction is applied, which are retrospective and prospective motion correction (PMC). Information of motion during a scan can be obtained with different techniques such as with navigators, RF-based tracking systems, and optical tracking systems (Godenschweger, et al., 2016).

Some retrospective motion correction techniques use the principle that a rigid body transformation causes a linear phase shift in k-space. By knowing the translation in each direction from the systems mentioned above, it is possible to determine a phase correction to apply to the acquired k-space data resulting in a motion-corrected image (Bookwalter, et al., 2010). Other retrospective methods uses deep learning (DL) to reduce motion artifacts. This is done by training a network on synthetically corrupted data to learn the mapping from motion-affected images to artifact-free images (Usman, et al., 2020).

One of the proposed methods of PMC in MRI is with optical trackers placed in the bore of an MR camera to acquire translation and rotation information from subject movement to adjust the gradient field direction and radio frequencies (RF) phases and frequencies in real-time to reduce motion artifacts (Qin, et al., 2009). Another method for reducing motion artifacts was found in 1986 when researchers suggested that phase encoding gradients could be adjusted in real-time based on navigation data (Haacke & Patrick, 1986). The use of image-based navigators has been of interest in the recent decade. By exploiting the waiting time for longitudinal relaxation in magnetization preparation imaging sequences, EPI can be used to acquire images of the entire subject volume to determine the subject's position in between the repetitions. The transformation parameters between the two EPI images are determined in real-time, then an image registration is performed on the sequence of interest images (Tisdall, et al., 2012).

This EPI method has further been built on to show its success in other imaging sequences, such as T1-, T2-weighted images, and angiographies. This has been done by adding a motion score threshold for a specific k-space data being reacquired, among other additions (Andersen, et al., 2019). One limitation of this method is that the registration of the reference and moving EPI images is an iterative process, meaning that the solution may not be reached within the desired time frame and that the timing to reach a solution varies. It would be beneficial for sequence design if these limitations were solved.

A DL model used to predict rigid-body transformation parameters from inputting a reference and moving image has shown to be successful with synthetic transformed fat navigators (FatNavs). The advantage of using a DL model is that the predictions of transformation parameters are made in a specific computation time, 40.1 ms with the most successful model. This approach is presented in a bachelor project from the Technical University of Denmark (DTU) (Svane Olsen & Nguyen-Cong, 2021) and is yet to be validated in vivo.

1.2 Aim

The aim of this work is to determine the feasibility of implementing a deep learning model into the PMC pipeline of a Philips 7 T scanner, replacing the current iterative image registration, to generate accurate transformation parameters to reduce motion artifacts, and have a practical computation time for better sequence design.

1.3 Goal and Purpose

The goal and purpose of this work is to implement, validate, and optimize the DL approach proposed by the DTU students (Svane Olsen & Nguyen-Cong, 2021). A method of calling DL models in the host computer of the scanner is needed for the implementation. At the same time, different DL architectures are to be explored if they can produce shorter computational times and more accurate predictions. Finally, the suggested architectures will be tested in vivo on the 7 T scanner.

2 Theory

2.1 Essential Principles of Magnetic Resonance Imaging

2.1.1 General Spin Physics

Atomic nuclei with an uneven number of protons or neutrons exhibit a non-zero spin s. The spin angular moment magnitude is defined as

$$S^2 = s(s+1)\hbar^2 \tag{1}$$

A quantum number associated with the spin is the magnetic spin quantum number m_s. This number ranges from $-s$ to $+s$ in steps of one, which generates $2s+1$ values of m_s as follows

$$m_s = [-s, -(s-1), \dots, s+1, s] \tag{2}$$

A classical analogy to understand the quantum spin is to imagine a charged sphere rotating on its own central axis. Its spinning results in angular momentum and if charged will generate a magnetic dipole moment $\vec{\mu}$. The magnetic dipole moment describes the strength of the interaction the proton will have on an external magnetic field and the strength of the magnetic field it generates itself. It can be described in terms of the $\vec{S}$ and the gyromagnetic ratio γ by

$$\vec{\mu} = \gamma \vec{S} \tag{3}$$

In the presence of an external magnetic field $\boldsymbol{B}$, the potential energy the dipole experiences are described classically as the following

$$U = -\vec{\mu} \cdot \vec{B} \tag{4}$$

This means that the moment will align parallel to the external magnetic field to reach a minimum energy state (Brown, 2014).

There is a large abundance of hydrogen ^{1}H in the human body, primarily due to a large amount of water H_2O. ^{1}H is a nucleus with an uneven number of protons, hence it will exhibit a $s = \frac{1}{2}$ and spin states $m_s = \pm\frac{1}{2}$. These two states are usually defined as spin up ($m_s = \frac{1}{2}$) and spin down ($m_s = -\frac{1}{2}$). In the presence of an external magnetic field $\boldsymbol{B}_z$, these two states will become degenerate. This is an example of the Zeeman effect. The amount of the splitting can be determined using the energy equation 4

$$E\left(m_s = \tfrac{1}{2}\right) = \frac{1}{2}\gamma\hbar B_z$$
$$E\left(m_s = -\tfrac{1}{2}\right) = -\frac{1}{2}\gamma\hbar B_z$$
$$\Delta E = E\left(m_s = \tfrac{1}{2}\right) - E\left(m_s = -\tfrac{1}{2}\right) = \gamma\hbar B_z \tag{5}$$

Due to thermal energy, the spins will populate both the lower and higher energy states (Brown, 2014).

Excitation of the spins from the lower energy state to the higher will require an electromagnetic field with energy equal to the ΔE. For an electromagnetic field, its frequency ν will equal

$$h\nu = \Delta E \Longleftrightarrow \nu = \frac{\gamma}{2\pi} B_z \tag{6}$$

$$For\,^1H\colon \frac{\gamma}{2\pi} = 42.58\ MHz\ T^{-1}$$

This frequency is often denoted as ν_0 and is called the Larmor frequency, for a specific external magnetic field $\boldsymbol{B}_0$ in the z-direction. The Larmor frequency for ^{1}H is in the radio frequency (RF) range. The probability for excitation is the same for deexcitation, however, there is a larger population of spins in the lower energy state. The population distribution can be described as a Boltzmann distribution. This will lead to a collective magnetism $\boldsymbol{M}_0$ parallel to the static magnetic field. The magnitude of this collective magnetism is almost proportional to the field strength of $\boldsymbol{B}_z$ as derived in the following approximation

$$M_0 \approx N \frac{\mu_p^2}{kT} B_z \tag{7}$$

Where N is the proton density, μ_p is the magnetic dipole moment of protons, k is the Boltzmann constant, and T is the temperature (Brown, 2014).

Due to the Heisenberg uncertainty principle, the spins can't be parallel to the magnetic field. This will result in a torque on the spin and it will precess around the axis of the static magnetic field. This precession frequency ω is related to the ν by

$$\omega = 2\pi\nu \tag{8}$$

Sending an RF pulse with a frequency of ω will change the population distribution between the two states over time while it is active. Measuring a signal from the precessing spins can be done via Faraday induction with a coil from the magnetism M_{xy} in the plane orthogonal to the direction of the static magnetic field. The magnitude of M_{xy} can be maximized by manipulating the population distribution so that the net magnetism before the RF signal is completely transferred to the xy-plane, then shutting off the RF (Brown, 2014).

2.1.2 Spatial Encoding

In other imaging modalities, the detection of particles transmitted through or from the patient is used to generate an image. One challenge in MRI is that there are no particles being emitted. Some form or spatial encoding of the spins is needed to be able to distinguish protons at different locations in the body. The main idea of solving this challenge is to use magnetic field gradients to create a varying magnetic field. This will give protons at different locations in the gradient a different ν.

Along the direction orthogonal to the transverse plane of a patient, usually defined as z, a magnetic field gradient is used to set different locations to have different ν. It is commonly desired to only excite a region in the gradient field, such as a head. This then requires a bandwidth of frequencies to be sent in the excitation RF pulse. The name of this process is slice selection, where a slice refers to the region of protons that has been excited (Brown, 2014).

Now the remaining protons in the two directions of the patient need to be encoded. This is done by applying magnetic field gradients in these directions. Consider the x to be encoded first. By activating a gradient in this direction during the signal acquisition, the induced signal frequencies will correspond to the signal produced by protons at different locations in this gradient. The direct Fourier transform of the one signal read out will correspond to the magnitude of signal coming from all protons in the plane orthogonal to the direction of the field. This process is named frequency encoding (Brown, 2014).

Since the frequency of resonance is used to determine the position in the x direction, the same method cannot be used in the y direction, since we will have lost the encoding. Instead, gradients are activated in the y direction before a signal readout. However, the y gradient in the next read-out will have changed in steepness by some increment. This is repeated a number of times with the same increment so that each signal read out has experienced a different steepness in the y direction. The result is that the signal acquired will have phase-dependent on the location of protons in slice orthogonal to the y direction. This process is named phase encoding (Brown, 2014).

2.1.3 Relaxation Theory

After the RF signal, relaxation effects will start to take over and determine the magnitudes of M_{xy} and M_z. The population distribution of spins will return to equilibrium. This is seen in the gradual increase of M_z over time. The reason for this is due to dipole-dipole interactions induced by fluctuations in the local magnetic field. These fluctuations arise from molecular motion including rotations, vibrations, and translations. These fluctuations occur with different frequency components. The frequencies depend on the mobility of the water molecules in a medium. For a simple tissue model, water bounded next to proteins are considered to be relatively less mobile, hence they have an abundance of low-frequency components. While more mobile water molecules have a broader distribution of frequencies. If the water molecules in a specific medium contain an abundance of the ν, the growth of M_z will be more effective as these fluctuations frequencies are enough to stimulate deexcitation. This is referred to as T1 relaxation or longitudinal relaxation (Brown, 2014).

The M_{xy} will also decrease gradually over time, however, this will be due to spatial variations in the magnetic field. If the fluctuations are rapidly changing, the effect is canceled out. However slow fluctuations will cause protons to point in some direction for some time. Which will be enough to exert a bias on the local magnetic field. This will change the ν and thus contribute to the dephasing of the spins. Hence, an abundance of less mobile water molecules means that the decrease in M_{xy} will be more effective. This is named T2 relaxation or transversal relaxation (Brown, 2014).

2.1.4 Ultra-High Field Magnetic Resonance Imaging

It is clear that from equation 1, more signal can be achieved by increasing the static magnetic field strength. This has been one of the reasons for the push for developing stable MR scanners with stronger static magnetic fields. It is expected that signal-to-noise should increase linearly. However, there are many other factors that signal-to-noise such as the imaging sequences, the properties of the object, and magnetic field homogeneity. One notable difference when using ultra-high field (UHF) strengths is that the ν increases linearly, meaning that the T1 relaxation time will increase as well (Olsson, 2021).

2.2 Neural Networks

The field of neural networks (NN) is heavily inspired by the goal to model biological neuron systems. These results of this have proven to be useful in machine learning (ML) tasks. A simple description of a neuron in the brain is that it is one computational unit. The human nervous system contains billions of these neurons, which are each connected to synapses. Signals are received by the neurons through their dendrites. After the neuron processes the signals received, it releases a signal through its single axon. This signal will later branch off to other synapses of other neurons. When this signal enters the dendrites of the other neurons, it is modified by some weight which is specific for signal the link between the two neurons.

A neuron in ML is essentially a number, which is equivalent to the strength of the signal in the biological description. There are four parts of a neuron: (i) a group of inputs (ii) a linear function (iii) a non-linear function (iv) an output. These parts are illustrated in Figure 1. The output of a neuron is determined by

$$y = \sigma\left(\sum_{i=0}^{n} w_i x_i + b_i\right) = \sigma(z) \tag{9}$$

Where σ is the non-linear function which is often defined as the activation function, and the terms inside σ represent the linear function z operating on the input vector x. The linear function has two parameters which are the weights w and biases b. The idea in ML is that the values of these weights and biases of links between neurons are determined through training.

The path from a neuron to a NN is by considering several vector columns of neurons linked in sequence. Each column is defined as a layer. Figure 2 shows the appearance of a multi-layer NN. Equation 1 can be modified to show the values of the neurons in the l'th layer based on the neurons values in the layer before.

$$\begin{bmatrix} x_0^{(1)} \\ x_1^{(1)} \\ \vdots \\ x_k^{(1)} \end{bmatrix} = \sigma\left(\begin{bmatrix} w_{0,0} & w_{0,1} & \cdots & w_{0,n} \\ w_{1,0} & w_{1,1} & \cdots & w_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,0} & w_{k,1} & \cdots & w_{k,n} \end{bmatrix}\begin{bmatrix} x_0^{(0)} \\ x_1^{(0)} \\ \vdots \\ x_n^{(0)} \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_n \end{bmatrix}\right) \tag{10}$$

A more compact formulation is

$$x^{(l)} = \sigma^{(l)}\left(W^{(l)} x^{(l-1)} + b^{(l)}\right) = \sigma^{(l)}\left(z^{(l)}\right) \tag{11}$$

Letting this function iterate over itself the number of layers the network has, L, gives the composite function

$$x^{(L)} = \sigma^{(L-1)}\left(W^{(L-1)}\sigma^{(L-2)}\left(W^{(L-2)} \ldots + b^{(L-2)}\right) + b^{(L-1)}\right) \tag{12}$$

This is essentially the definition of a NN. The L'th layer is named as the output layer, the first layer is named the input layer, and the layer between are named hidden layers. NN

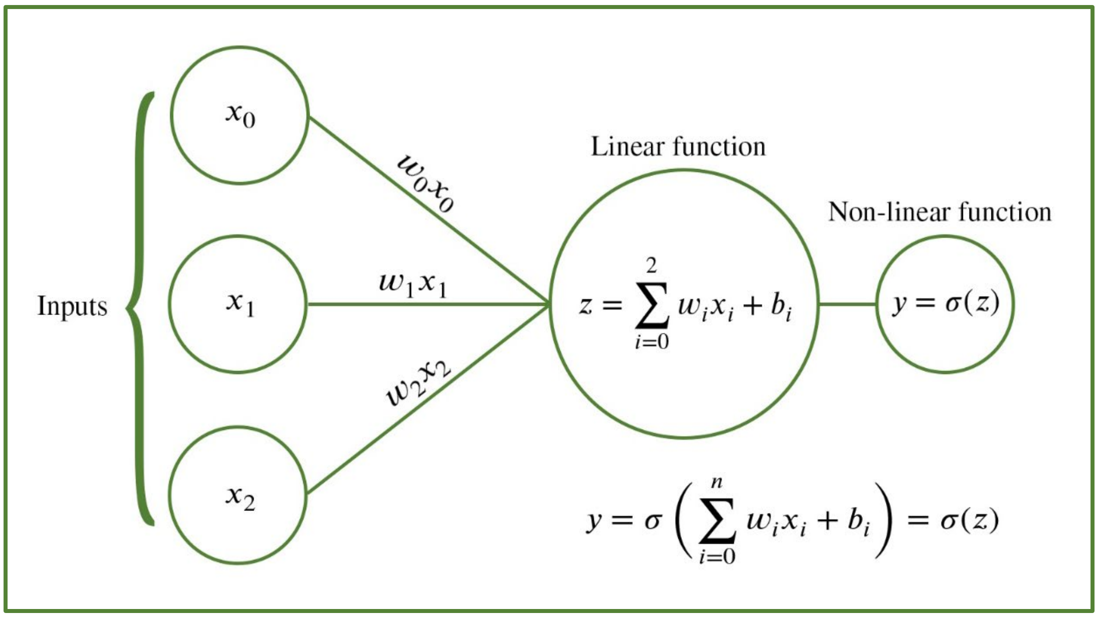

Inputs
x_0
x_1
x_2
w_0x_0
w_1x_1
w_2x_2
Linear function
$z = \sum_{i=0}^{2} w_i x_i + b_i$
Non-linear function
$y = \sigma(z)$
$y = \sigma\left(\sum_{i=0}^{n} w_i x_i + b_i\right) = \sigma(z)$

Figure 1: Temporary figure for single neuron

is generally defined as deep neural networks (DNN) if there are more than two hidden layers in the network ($L > 4$).

In practice, NN is shown to have better performance with more layers than one layer with many neurons. This is likely due to that with each layer, the input is transformed and creates new representations of it. However, an excessive number of layers can impair the performance of the network as well. This is one of the obstacles of DNN, as most methods for pursuing the optimal number of layers rely on trial and error. There are a few methods that will later be mentioned in this section that can improve the performance of DNNs.

2.2.1 Activation Functions

The activation function is an operation that is performed on the result from the linear function in a neuron. Its result will be the overall output of the function. The purpose of the activation function is to introduce non-linearity into the output of the neuron. If the activation function between two consecutive layers is an identity map, then due to linearity equation 3 becomes

$$\begin{aligned} x^{(l+1)} &= \sigma^{(l)}\left(W^{(l)}\left(W^{(l-1)}x^{(l-1)} + b^{(l-1)}\right) + b^{(l)}\right) \\ &= \sigma^{(l)}\left(\widetilde{W}^{(l)}x^{(l-1)} + \tilde{b}^{(l)}\right) \end{aligned}$$

Where $\widetilde{W}^{(l)} = W^{(l)} \cdot W^{(l-1)}$ and $\tilde{b}^{(l)} = W^{(l)} \cdot b^{(l-1)} + b^{(l)}$. This shows that the neurons in $x^{(l+1)}$ are directly connected to the neurons in $x^{(l-1)}$ without the need for the neurons in between ($x^{(l)}$). If all the activation layers in a NN were identity maps, the whole network would be in principle a linear model. Hence, the activation function is the primary source of non-linearity in NN. It is believed that non-linear activation functions enable better representation of complex relationships by transforming the input data and creating an additional fold.

There are several types of activation functions. One of the first encountered types is sigmoid. A sigmoid function is defined as

$$f(x) = \frac{1}{(1 + e^{-x})} \tag{13}$$

It is a non-linear function whose range covers 0 to 1. For $x = 0$, this function returns $^1\!/_2$. For very positive and negative values, the function returns 1 or 0 respectively. Historically, the sigmoid has been popular for its nice interpretation of the firing of a neuron. 0 for not firing at all and 1 for saturating the firing.

Another type of activation function is the tanh function, which is defined as

$$f(x) = tanh(x) \tag{14}$$

Like the sigmoid function, this function saturates for very positive and negative values. However, the output of the function ranges between 1 and -1, meaning that it is zero-centered. This is one reason why tanh functions would be preferred more than sigmoid functions. It is worth noting that the tanh function is essentially a scaled sigmoid function σ like

$$tanh(x) = 2\sigma(2x) - 1$$

One notable activation function is the rectified linear unit (ReLU) function. It is defined is as

$$f(x) = \begin{cases} x, x > 0 \\ 0, x \leq 0 \end{cases} \tag{15}$$

ReLU use as an activation function has been popular recently for its successes in effective training. The basic idea of the ReLU function is to set the threshold at zero.

The last activation function to mention here is the Leaky ReLU function. It is in principle the attempt in solving the dying gradient problem of using ReLU activation functions. It is defined as

$$f(x) = \begin{cases} x, x > 0 \\ x \cdot a, x \leq 0 \end{cases} \tag{16}$$

2.2.2 Optimization and Training

Solving problems with a DL approach requires a training phase. The purpose of the training phase is to determine the optimal parameters W^* for the network from training data. This is often formulated mathematically as an optimization problem where the goal is to minimize some loss function M. From equation 4, the optimization problem can be expressed as

$$W^* = \min_W M(W) = M\left(\hat{x}, x^{(L)}\right) \tag{17}$$

Where $\hat{x}$ is the desired output of the network. The loss function method is specific to the type of problem the network is attempting to solve. For linear regression problems, the mean square error (MSE) is often used and can be expressed as

$$M\left(\hat{x}, x^{(L)}\right) = \frac{1}{2} \sum_{i=1}^{n_L - 1} \left\| x_i^{(L)} - \hat{x}_i \right\|^2 = E \tag{18}$$

The term E is used to convenience.

The difficulty in finding the optimal parameters for equation 9 is that it is often a non-convex optimization problem, meaning that there are difficulties in finding a global minimum in the function. The most common method used to solve these types of problems is the gradient descent algorithm, in which the gradient of the function is examined to determine which direction decreases the function the quickest (Rojas, 1996). Thus, the gradient of the loss function is needed

$$\begin{aligned} \Delta W &:= -\eta \left(\frac{\partial M}{\partial w_1}, \frac{\partial M}{\partial b_1}, \frac{\partial M}{\partial w_2}, \frac{\partial M}{\partial b_2}, \dots, \frac{\partial M}{\partial w_\kappa}, \frac{\partial M}{\partial b_\kappa} \right) \Big|_{W(old)} \\ &:= -\eta \frac{dM}{dW} \Big|_{W(old)} \end{aligned} \tag{19}$$

Where w_κ are the κ number of weights in the entire network. This is used to determine the increment on the weights and biases to find the minimal of the loss function. In the case of the weights, this is

$$w_k = w_k + \Delta W = w_k - \eta \frac{dM}{dw_\kappa} \tag{20}$$

Here, η is called the learning constant, which is a scalar applied to the increment. Now the optimization problem boils down to determining the partial derivatives of the loss function and iteratively changing the weights until $\Delta M = 0$.

Since the loss function and weights are linked via two different functions and due to the composite function form of the output in the neuron, the chain rule can be applied to determine the gradient in equation 11. This can be seen from equation 10, the partial derivative of the loss function with respect to weight is given by

$$\frac{dM}{dw_{k,n}^{(l)}} = \frac{\partial z_k^{(l)}}{\partial w_{k,n}^{(l)}} \frac{\partial x_k^{(l)}}{\partial z_k^{(l)}} \frac{\partial M}{\partial x_k^{(l)}} \tag{21}$$

The first term describes how the linear function is influenced by some change in the weights in the layer. As a reminder, the linear function z is defined as

$$z^{(l)} = \sum_{i=0}^{n} w_{k,i}^{(l)} x_{k,i}^{(l-1)} + b_{k,i}^{(l)}$$

Since only one term is dependent on the weights, the partial derivative becomes

$$\frac{\partial z_k^{(l)}}{\partial w_{k,n}^{(l)}} = x_{k,i}^{(l-1)}$$

The second term in equation 13 is the values of the neurons which is essentially the output of the neurons. Hence, the partial derivative with respect to linear function is given by

$$\frac{\partial x_k^{(l)}}{\partial z_k^{(l)}} = \frac{\partial \sigma^{(l)}(z_k^{(l)})}{\partial z_k^{(l)}} = \sigma'^{(l)}\left(z_k^{(l)}\right)$$

The final term in equation 13 is the sensitivity of the loss function to a change of the value of a neuron in a layer. If this layer is the output layer of the NN, $l = L$, then the partial derivative becomes

$$\frac{\partial M}{\partial x_k^{(l)}} = \frac{\partial M}{\partial x_k^{(L)}}$$

This means that only the derivatives influenced by the layer before the output layer are needed to determine this. If the loss function is the MSE as in equation 10, the partial derivative of the output layer will be

$$\frac{\partial M}{\partial x_k^{(L)}} = x_k^{(L)} - \hat{x}_k$$

In the case of l being some hidden layer, the derivative is not as straightforward. For comprehensibility, sub-indices are now ignored. Each of the neurons influences the loss function through several different paths. Hence the loss function will be dependent on the linear operations in between the l'th layer and the output layer L. Through the total derivative and the chain rule, the result is

$$\begin{aligned}\frac{dM}{dx^{(l)}} &= \frac{\partial M\left(z^{(l+1)}, z^{(l+2)}, \ldots, z^{(L)}\right)}{\partial x^{(l)}} \\ &= \frac{\partial M}{\partial z^{(l+1)}}\frac{\partial z^{(l+1)}}{\partial x^{(l)}} + \frac{\partial M}{\partial z^{(l+2)}}\frac{\partial z^{(l+2)}}{\partial x^{(l)}} + \ldots + \frac{\partial M}{\partial z^{(L)}}\frac{\partial z^{(L)}}{\partial x^{(l)}} \\ &= \sum_{j\in L}\left(\frac{\partial M}{\partial z^{(j)}}\frac{\partial z^{(j)}}{\partial x^{(l)}}\right) \\ &= \sum_{j\in L}\left(\frac{\partial M}{\partial x^{(j)}}\frac{\partial x^{(j)}}{\partial z^{(j)}}\frac{\partial z^{(j)}}{\partial x^{(l)}}\right) \\ &= \sum_{j\in L}\left(\frac{\partial M}{\partial x^{(j)}}\frac{\partial x^{(j)}}{\partial z^{(j)}} w^{(j)}\right)\end{aligned}$$

Finally, all the terms of equation 13 are defined and the updates to the weights by equation 12 can be determined.

This method of optimizing and training and network can be very costly in memory usage. Finding the global minimum of equation 9 is neither guaranteed. To solve address these issues, researchers have suggested the use of stochastic optimization methods such as stochastic gradient descent (SGD) and adaptive moment estimation (ADAM).

The common idea of stochastic optimization methods is to solve the optimization problem by iterating sets of the full training data set instead of passing the full data set in one iteration. This tackles the memory usage problem. These smaller sets of the training data are called batches and the amount of data in the set is referred to as the batch size. Before data is placed into these batches, the entire training set is shuffled. Updates to the weights are performed after a batch has been passed through the model. Each pass of all batches in the entire dataset is called an epoch. The difference between each epoch is the random shuffling of the data before being assigned to batches. If the number of batches is defined as τ and the size of the batch is χ, then equation 10 will be the following for any loss function

$$E^{(\tau)} = \sum_{i=1}^{\chi} M\left(\hat{x}_{((\tau-1)\chi+i)}, x^{(L)}_{((\tau-1)\chi+i)}\right), m = 1,2,\ldots,\tau \tag{22}$$

The most notable difference to the updates of the weights by using the SGD method is that a momentum term is added to equation 12. The idea is to include the gradients calculated in the previous batch into the new batch calculations. A scalar α is introduced into equation 11 which is multiplied with the previous batch $(\beta - 1)$ gradients.

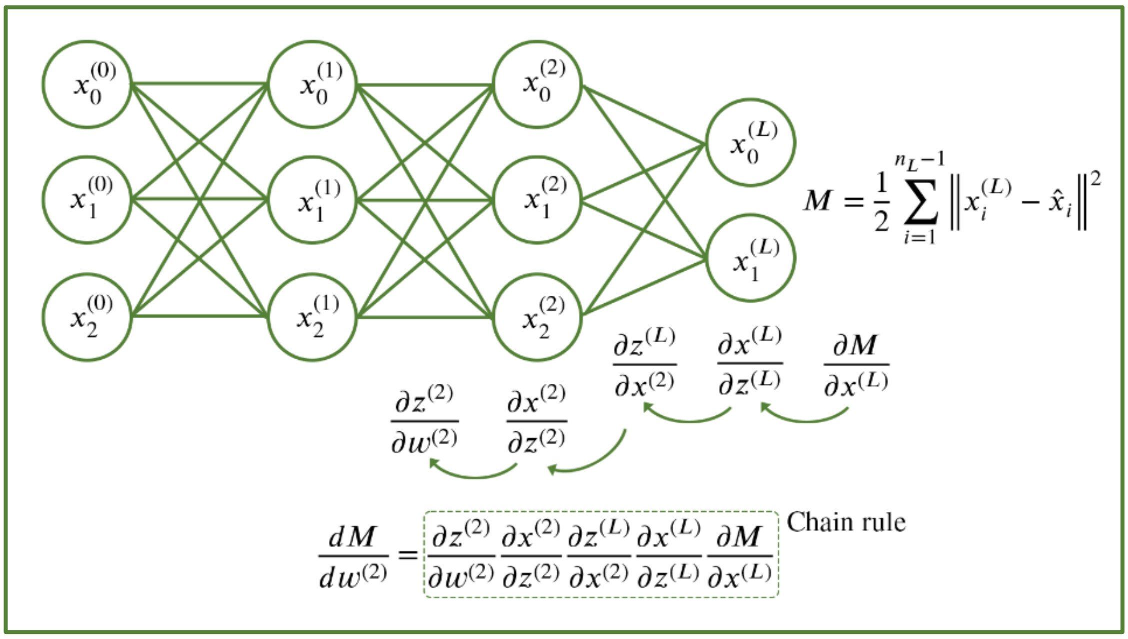

$x_0^{(0)}$
$x_1^{(0)}$
$x_2^{(0)}$
$x_0^{(1)}$
$x_1^{(1)}$
$x_2^{(1)}$
$x_0^{(2)}$
$x_1^{(2)}$
$x_2^{(2)}$
$x_0^{(L)}$
$x_1^{(L)}$
$M = \frac{1}{2}\sum_{i=1}^{n_L - 1}\left\|x_i^{(L)} - \hat{x}_i\right\|^2$
$\frac{\partial z^{(2)}}{\partial w^{(2)}}$ $\frac{\partial x^{(2)}}{\partial z^{(2)}}$ $\frac{\partial z^{(L)}}{\partial x^{(2)}}$ $\frac{\partial x^{(L)}}{\partial z^{(L)}}$ $\frac{\partial M}{\partial x^{(L)}}$
$\frac{dM}{dw^{(2)}} = \frac{\partial z^{(2)}}{\partial w^{(2)}}\frac{\partial x^{(2)}}{\partial z^{(2)}}\frac{\partial z^{(L)}}{\partial x^{(2)}}\frac{\partial x^{(L)}}{\partial z^{(L)}}\frac{\partial M}{\partial x^{(L)}}$
Chain rule

Figure 2: Visualization of the iterative process in determining the change of the loss function (Mean square error in this case) with respect to weights of an inner layer $l = 2$.

$$\Delta W^{(\chi)} := \alpha \Delta W^{(\chi-1)} - \eta \frac{dM}{dW}\bigg|_{W^{(\chi-1)}} \tag{23}$$

This can improve the step size in the gradient descent when the slope is steep, which shortens the time to convergence. Typical values of α are between 0.5 and 0.9 and can result in a factor two faster convergence rate.

The ADAM algorithm was proposed in 2015 for DL and is one of the most frequently used optimization algorithms. It includes the idea of momentum as in SGD, and an adaptive rate scheme meaning that the learning rates will vary per parameter. In mathematical terms, the updates to the weights

$$W^{(\chi)} = W^{(\chi-1)} - \frac{\eta}{\sqrt{\hat{v}^{(\chi)} + \epsilon}} \widehat{m}^{(\chi)} \tag{24}$$

$$\widehat{m}^{(\chi)} = \frac{m^{(\chi)}}{1-\beta_1^{\chi}}, \hat{v}^{(\beta)} = \frac{v^{(\chi)}}{1-\beta_2^{\chi}}$$

$$m^{(\chi)} = \beta_1 m^{(\chi-1)} + (1-\beta_1)\Delta W$$

$$v^{(\chi)} = \beta_2 v^{(\chi-1)} + (1-\beta_2)\Delta W \odot \Delta W$$

$$\Delta W = \frac{dM}{dW}\bigg|_{W^{(\chi-1)}}$$

Where v is the term which modifies the learning rate depending on different parameters, m is the term which includes a moment into the gradient descent, and the respective hat accents $(\hat{v}, \widehat{m})$ on each term represents the normalization of them. The β values are user-defined values and are typically set to $\beta_1 = 0.9$ and $\beta_2 = 0.99$. The ϵ term is usually set to 10^{-18}, its purpose is to avoid a division by zero.

2.2.3 Convolutional Neural Networks

In the analysis of images with NN, convolution operations are frequently used. The major reasons are that there is an explosion of the number of parameters, and the loss of information with respect to how the pixels are interrelated if the usual inner product operation is performed as in a NN. Vectorizing an image with a resolution of 256×256 pixels will result in a column vector containing 65536 elements. Assuming that the number of neurons in the first layer is 1000, then the number of weight parameters connecting the input and first layer will be in the order of $65 \cdot 10^6$. For this number of parameters, the computational cost of the model would be significant, and the generalization performance will take a hit for many parameters. Convolutional neural networks (CNN) are used to solve these issues. The main two operations introduced by CNNs are the convolution step and the pooling step.

As mentioned, a problem with NN in image analysis is that the number of parameters will be very large. A potential solution to this is to share parameters between neurons in the same layer. A convolution operation is mathematically expressed as

$$f(x) \otimes h(x) = \int_{-\infty}^{\infty} f(\tau)h(x-\tau)d\tau$$

$$f(x,y) \otimes h(x,y) = \iint_{-\infty}^{\infty} f(\tau_1,\tau_2)h(x-\tau_1, y-\tau_2)d\tau_1 d\tau_2 \tag{25}$$

for 1D and 2D respectively. The input is defined as f and h is defined as the kernel.

From using convolutions in NNs, weight sharing is possible, meaning that the kernel h is shared with all neurons of the previous layer instead of each neuron having its own weight. Information of the local neighborhood surrounding each neuron is considered with the use of convolutions, which effectively tackles the loss of information of how the pixels are interrelated when using a conventional NN and inner product.

After each convolution, any activation function is applied to all neurons in the output in each channel. The introduction of CNNs also includes another process that can improve the performance networks. Several kernels are often used to extract information from the data, this can lead to memory issues if the resulting number of channels is large. Reducing the dimensions of the data is one method of tackling this. In CNN, this is known as pooling and one of these operations is named max pooling. In max pooling, a convolution process is applied data, where a kernel slides across the data, but instead of the conventional convolution process, a selective kernel is applied which outputs the value of the neuron in the neighborhood with the maximum value. The stride of the convolution is set so that the dimensions of the data are reduced, and the idea is to preserve as much information as possible in the data after the reduction in size. Selecting the max-pooling process assumes that the important information in the data is the largest value. Another pooling process is named average pooling. Here, the output is the average value of the neurons in the neighborhood. This could improve the performance of the network as all information is considered.

2.2.4 Learning Strategies

Learning strategies in DL can primarily be either supervised or unsupervised. Data used in supervised training are labeled as input and expected output. This type of training is straightforward as the optimizing goal is clearly to predict the expected output and penalize the network for incorrect predictions. However, the requirement for supervised training is that such input and output data exist. Knowing the exact ground truth is not always the case. Acquiring such data for supervised training can be difficult, such as segmentation data where segmentations are expected to be drawn from medical annotators, which is time-consuming. Luckily, there exists data that is publicly available that has been gathered that can be used in some DL methods. However, this data might not always be the right to tackle a specific problem. Especially for regression tasks, there have been successes with accurate predictions on synthetic data (Svane Olsen & Nguyen-Cong, 2021). One problem with using synthetic data is that the trained model could be less generalizable to real data.

On the other hand, unsupervised learning aims to find the patterns in the data without feeding the training ground truth or our targeted output.

2.2.5 Deep Learning Frameworks

The most common way to implement DL solution to a problem is by using a programming script. In recent years, several frameworks have been distributed with common DNN and CNN operations already implemented such as activation functions and 3-dimensional convolutions. Most of these can be used with Python and C++. Two of the most popular frameworks in recent years are TensorFlow and PyTorch (Paszke, et al., 2019). The availability of these types of frameworks allows for easy training of NN for researchers without the need of developing all the operations in a programming language.

TensorFlow is Python-based and is the framework with the largest support and usage for its focus on production and scalability in mind, which makes it popular in an industrial setting where quickly pushing prototypes into deployment can be important. It was developed by the Google Brain Team and distributed publicly in 2015, and since then has reached the milestone of having the largest community of all DL frameworks. Since TensorFlow's 2.0 releases, have Keras fully integrated by default. Keras is a high-level wrapper around TensorFlow making for seamless and easy training of networks. Eager Execution is enabled by default in the >2.0 releases allowing for faster development and debugging.

In recent years, PyTorch has received the fastest community growth among all DL frameworks. This is mainly due to its different approach than in TensorFlow and has made it more popular in academia.

There are several more frameworks available for DL, each with its advantages and disadvantages as well as the use of different languages. Both TensorFlow and PyTorch have released C++ API to allow for easy deployment of the models, and due to their popularity, there exists plenty of support for each.

2.2.6 Hyperparameter Tuning

Hyperparameters of a NN refer to the parameters the user must define in the model before training such as the number of hidden layers in a network or the choice of activation functions. There are several hyperparameters in NN, and each has its impact on the training. Hyperparameter optimization refers to the adjustment of these parameters to minimize a defined loss function for a given set of data.

One popular approach to hyperparameter optimization is Bayesian optimization…

2.3 Geometric Transformations

Geometric transformations are essential tools for image analysis.

2.3.1 Rigid-Body Transformation

In Euclidean space, translations, t, in 3D can be represented in a matrix T in the following way

$$T = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (26)$$

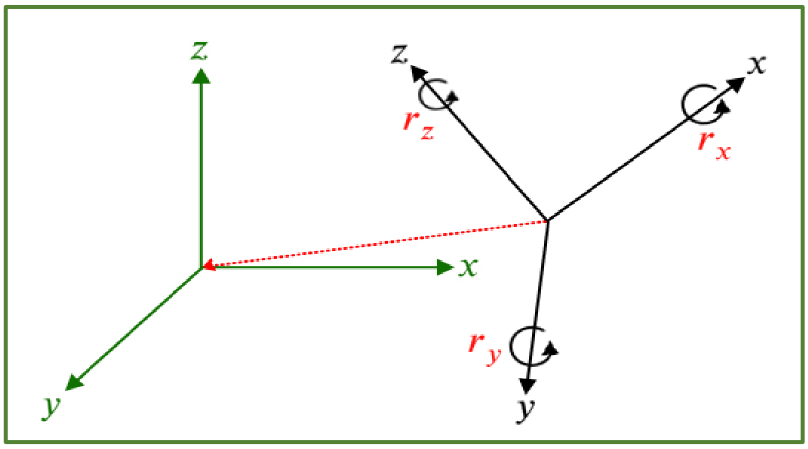


Figure 3: Illustrating the coordinate shift (black to green) from rigid-body transformations (red).

Rotations around each axis can be represented in the matrices R in the following way

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(r_x) & sin(r_x) & 0 \\ 0 & -sin(r_x) & cos(r_x) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{27}$$

$$R_y = \begin{bmatrix} cos(r_y) & 0 & sin(r_y) & 0 \\ 0 & 1 & 0 & 0 \\ -sin(r_y) & 0 & cos(r_y) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{28}$$

$$R_z = \begin{bmatrix} cos(r_z) & sin(r_z) & 0 & 0 \\ -sin(r_z) & cos(r_z) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{29}$$

This gives in total 6 degrees of freedom from the variables in the matrices. Each of these matrices are used in determining the new position of a voxel for a certain set of parameters.

In imaging, rigid-body transformations maintain the original shape of the object and only works as a movement of pixels to a new destination. In a medical imaging setting, this is beneficial for problems where size difference between objects is not of interest. More specifically, rigid-body transformations preserve the distances and angles of an object.

Let Y be the vector describing the coordinates of the voxels after a transformation. For a set of rigid body transformation parameters, the new coordinates Y of the voxels with coordinates X can be found by

$$Y = R_x \cdot R_y \cdot R_z \cdot X + T \tag{30}$$

2.3.2 Affine Transformation

In addition to linear transformations, there is the affine transformation which includes 3 more degrees of freedom, in 3D, than in rigid-body transformations. These parameters introduce scaling to the matrix. In an image transformation sense, this preserves parallelism meaning that lines preserve their ratios of distances along a line. However, absolute distances and angles are not preserved after affine transformations.

A matrix describing shearing in combination with the matrices $18-21$ complete the affine transformation. Scaling in 3D can be represented in the following way

$$S = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{31}$$

2.4 Image Registration

Image registration is a major research field in computer vision that will be difficult to grasp completely in this thesis. It refers to the process of spatial aligning two images so that the coordinates in one image match the coordinates of the other image. It is common to refer to the matching images as a reference and moved images respectively. Then the reference is the image that is kept unchanged, while the moving image is transformed to match the geometry and coordinate system of the reference image. One common use of this is in registering satellite images for mapping, such as when the alignment of two satellite images with different zooms is desired. Image registration has also made its way into medical imaging. There are several problems in medical imaging which can be addressed with image registration, such as when health personnel need spatially aligned images of a patient that were taken at different dates.

2.4.1 Deep Learning Methods

Recently, the state-of-the-art deep learning image registration methods for 3D MRI are based on the Voxelmorph approach. Voxelmorph (Balakrishnan, et al., 2019) is a deep learning-based tool that is publicly available and has been shown to outperform traditional registration methods. The main architecture is based on a U-Net in combination with a spatial transform network (STN). Loss functions are based on traditional similarity metrics such as MSE or cross-correlation. The transformations made in the Voxelmorph are both affine and deformable in this order. This has been shown to improve the accuracy of the final registration.

In comparison to other DL deformable image registration methods, Voxelmorph includes a smoothness regularizer to address 'folds' in the deformation field, which is where the determinant of the deformation field is negative. These can appear in several regions of the deformation field in other methods, making it not physically realistic. Ideally, the deformation field should be smooth and invertible. The number of 'folds' is a criterion that evaluates the smoothness of the deformation field. To tackle this, Voxelmorph introduces the smoothness regularizer to its optimization problem. This uses a diffusion regularizer on the gradients of the displacement field and approximates them using differences between neighboring voxels. This base design has been used to develop other tools such as tools that can generate atlases for 3D brain images and contrast-invariant image registration.

The performance of the Voxelmorph is data-dependent. For example, models trained on T1W image pairs will poorly register T2W image pairs. There is a possibility to consider all combinations of contrasts, however, this would need a large dataset. The contrast invariant method named SynthMorph attempts to train models to register segmentations of images with arbitrary contrasts. In addition to this, these images with arbitrary contrasts are synthetically generated, meaning there is no need to input data. In this way, the researchers hoped to encourage the model to generalize across MRI contrasts.

In SynthMorph, pairs of atlases are generated with arbitrary shapes. Each pair has one randomly generated deformable transformation relationship between them. The moving atlas will be the reference atlas after this deformable transformation. These atlases are pushed into an image sampler which generates a grayscale image using a Gaussian mixture method which will include creating arbitrary contrasts, blurring, and intensity bias. The model is then trained with these two images to optimize a combined loss function of a regularization term and dice term of the warped image. This is a form of unsupervised

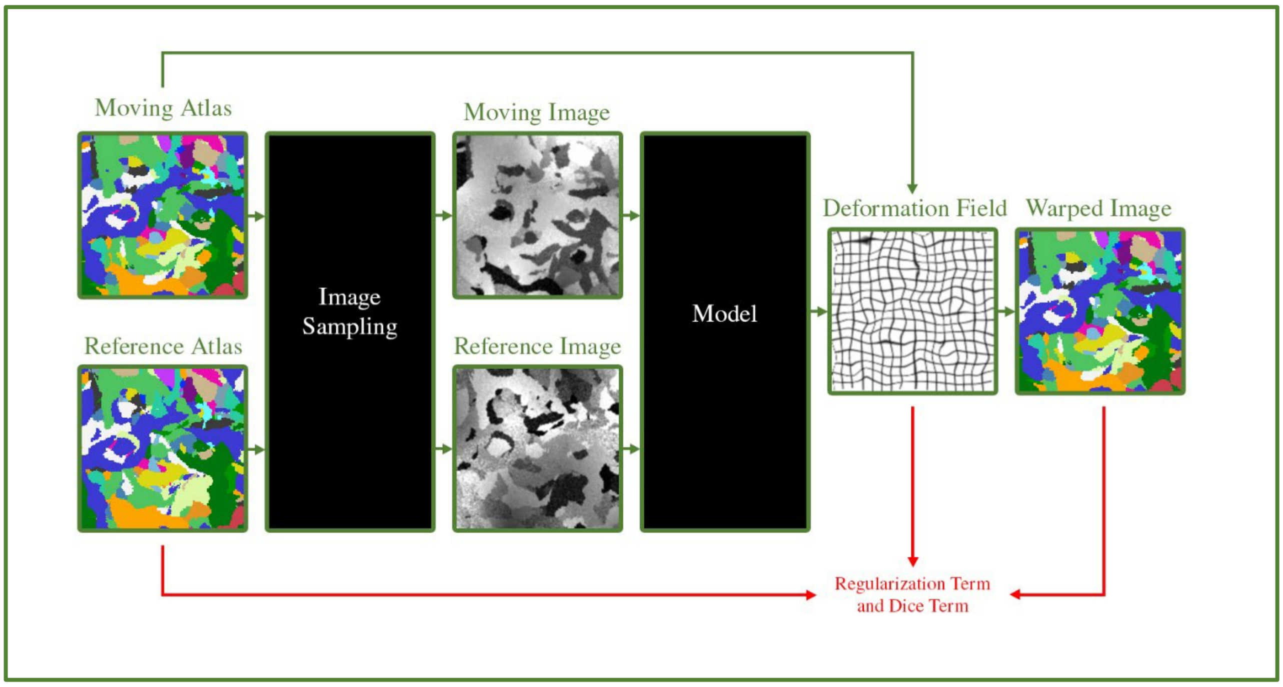


Figure 4: An outline of the SynthMorph pipeline. The initial atlas pair are synthetically generated, then pushed into an image sampling function which generates the corresponding images in grayscale. These are pushed into a NN which generates a deformation field describing how to transform the moving atlas to the reference atlas. Since unsupervised learning is used, red indicates the needed images to determine the loss. Adapted from ??.

learning, and a general overview of the pipeline is outlined in figure 5.

In addition to non-linear registration DL methods, there exist linear registration DL methods. One recent work suggested a network design to predict rigid body transformation parameters that register 3D images of the brain from different modalities. Results from registration of 3D brain computed tomography (CT) and MR images are shown to produce accurate results in comparison to traditional registration methods. This method uses supervised training by augmenting data from CT and MRI with random rotations and translations. The model is then optimized by a loss function describing the difference between the predicted parameters and the augmentation parameters used for the specific image pair. The suggested CNN is relatively large in comparison to VoxelMorph. It can be seen as the encoding part of a U-Net, but larger as shown in figure 6. A complete description of the model can be found in the original paper.

2.5 Motion Correction

Correcting motion artifacts in MRI is another large field of research. As previously mentioned, motion artifacts can lead to several re-examinations which will impact healthcare availability and cost. Focus on reducing motion artifacts is highly desired. The two main approaches to tackling this issue are retrospective and prospective motion correction (PMC). Retrospective means that correction is applied after the data acquisition and prospective meaning that correction is applied during data acquisition. Both approaches have advantages and disadvantages.

2.5.1 Prospective Motion Correction

One type of method for PMC is iMOCO. It is a navigator-based approach where information of the patient's motion is acquired during an examination. Based on this information, updates to the scanning parameters or requiring motion corrupted data can be done which will effectively reduce the risk of motion artifacts. This entire pipeline can be split up into three parts, a navigator, a registration, and a correction.

iMOCO depends on acquiring two 3D images fast during scanning. These images are taken between readout trains in a sequence, thus are desired to be in the magnitude of ms to acquire. Sequences able to produce 3D images in this time are navigators. Navigators are commonly used by radiographers as fast low-resolution images to determine an appropriate FOV for the main imaging sequences. The low resolution is due to the readout technique which is often used for navigators which is EPI. Acquiring 3D navigators can be in the order of 350 ms, and an MRI data collection sense, this is a comparable time. The TI of FLAIR sequences can be a factor 2 larger than this, such as in T1W FLAIR sequences where TI can be around 860 ms. This means that navigators can be acquired in between readout trains of FLAIR sequences and won’t disturb the desired output image data collection. It is also possible to acquire navigators with fat-selective excitations.

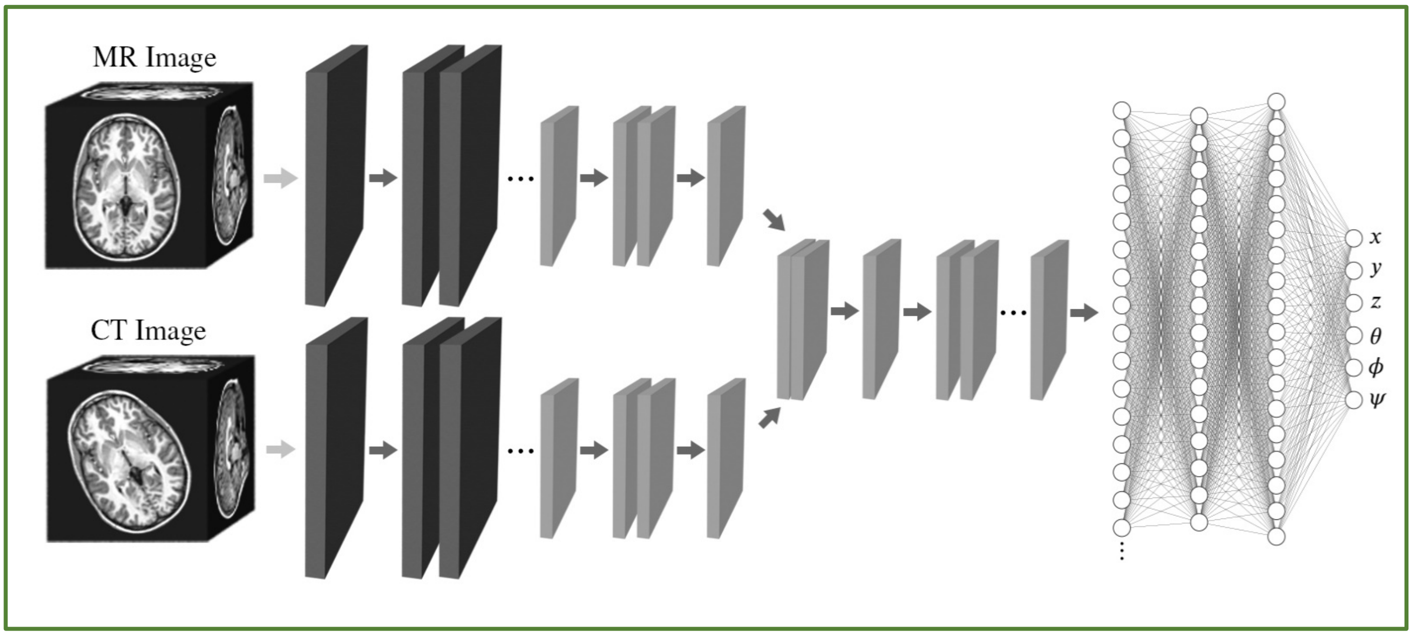


Figure 5: Deep learning network suggested by (Svane Olsen & Nguyen-Cong, 2021). Each image is pushed through a sequence of convolution, batch normalization, Leaky ReLU and max pooling layers until concatenated and pushed into a fully connected dense layers which outputs the predicted rigid-body transformation parameters.

Once two navigators are acquired at different time points, image registration is performed. The earliest navigator is set as the reference image while the other is set as the moving image. Rigid-body transformation parameters are determined with an iterative approach, hence the time it takes to reach a solution is arbitrary. For registering fat-navigators with iMOCO, it is agreed that it takes around 250 – 300 ms. This still is in the order of magnitude for TIs in FLAIR sequences. Together with the acquisition of a new motion navigator, this process can take a total of 650 ms, which is squeezed into the TI.

Once the registration is completed, the parameters are used to calculate a motion score, *score*, using the following equation

$$score = \Delta R + \left(t_x^2 + t_y^2 + t_z^2\right)^{1/2} \qquad (32)$$

Where ΔR is the largest displacement a point experiences on a sphere with a 64 mm radius when rotated by the magnitude of the rotation $|\hat{R}|$. ΔR is given by

$$\Delta R = 64\left(\left(1 - cos\left(|\hat{R}|\right)\right)^2 + \left(sin\left(|\hat{R}|\right)\right)^2\right)^{1/2} \qquad (33)$$

And $\left|\hat{R}\right|$ is given by

$$\left|\hat{R}\right| = \left|cos^{-1}\left(\frac{1}{2}\left(-1 + cos(r_x)\,cos\left(r_y\right) + cos(r_x)\,cos(r_z)\right.\right.\right.$$
$$\left.\left.\left. + cos\left(r_y\right)\,cos(r_z) + sin(r_x)\,sin\left(r_y\right)\,sin(r_z)\right)\right)\right| \tag{34}$$

This score is used to decide whether requisition of motion corrupted data is needed. Today, the integrated iMOCO technique at the Danish Research Center for Magnetic Resonance (DRCMR) uses a threshold of 1 mm for requisition. For scores below this threshold, updates to the FOV are performed so that the next readout train acquires data in the assumed new position of the patient.

This entire correction method has been shown to improve diagnostic quality in T1W-, T2W imaging as well as in time-of-flight angiography. Furthermore, the integration to other sequences than FLAIRs which don't have an obvious long wait time is shown to be possible. Figure 7 illustrates this entire process.

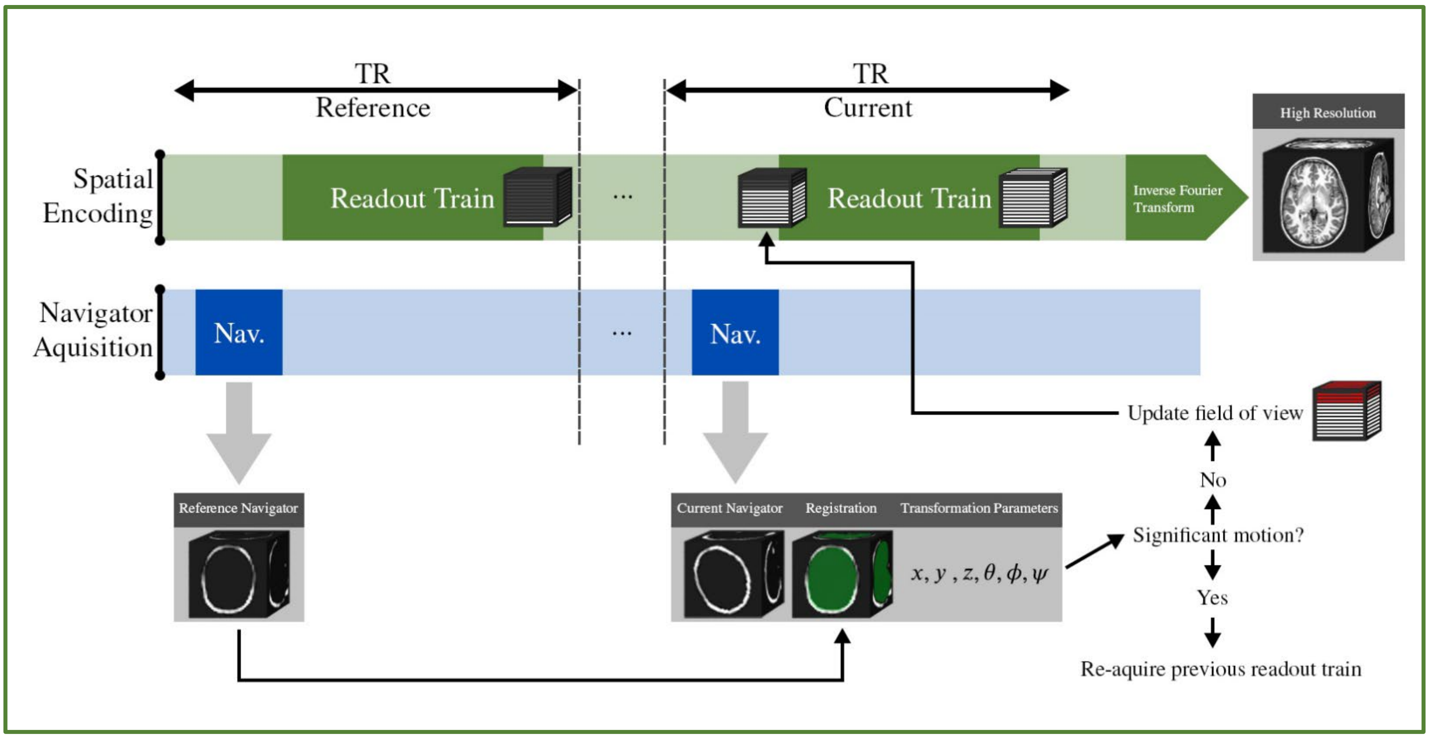


Figure 6: iMOCO workflow. Green sequence indicates the main desired sequence, with 3D cubes representing the amount of k-space data that has been acquired. In-between read-out-trains navigators are acquired to determine the subjects position at these dead-times. A reference navigator is used to register the current navigator with a rigid body transformation. These parameters are used to calculate a score which determines whether data should be required or if scanning parameters should be updated.

3 Method

3.1 Datasets and Environment

The same dataset as in (Svane Olsen & Nguyen-Cong, 2021) was used as the main dataset for model training. This consisted of 18000 3D fat navigators acquired from 110 subjects, both male and female, with the Philips 7 T MRI scanner at DRCMR in Hvidovre, Copenhagen. Each navigator had 23 slices, each with dimensions 48×48 pixels. The resolution of each voxel is $7.00\,\text{mm} \times 5.33\,\text{mm} \times 5.33\,\text{mm}$.

iMOCO from (Andersen, et al., 2019) is already implemented on the same scanner at DRCMR as a C++ program built into the host computer. Users can select whether this type of PMC will be used during a scanning procedure simply by toggling the motion correction parameters in the main UI of the scanner. Then two arrays of single-precision floats containing a reference and current fat navigator are sent to the iMOCO program. These are quickly sent to an iterative solver which finds a solution to rigid body transformation parameters which minimize the difference between the two images. By default, the time the scanner allows the iMOCO program to take before continuing to the next read-out train is 300 ms. This restriction can be varied in the scanner UI from the Edit Scan Parameters category with the parameter PMC Lead Time. The rigid body transformation parameters are sent back to the PDF for assessment of whether updates to the scan FOV or reacquisition should be made.

The majority of the programs that run on the host computer of the scanner are written in C++, which include thousands of lines of code. A simple way to swap the iterative solver in the iMOCO program won't be as straightforward. An optimal method of replacing the iterative solver would be to avoid the need to rebuild a larger portion of the scanner's code, and instead just replace the shared library file of the PMC. This way, deployment to other scanners will be easier. For this project, this will be the goal to achieve in implementation.

Another challenge of this host computer is that it does not have a GPU. Convolutions are known to be a computational heavy operation. However, GPUs are designed to be able to perform these operations at quicker speeds than GPUs. Without a GPU, all predictions will have to be made on the host computer's CPU if the method is to be fully implemented on the scanner.

All training was performed on a system with $2 \times$ GeForce RTX 3090s with 24 GB each.

3.2 Initial Data Augmentations

From (Svane Olsen & Nguyen-Cong, 2021; Islam, et al., 2021), it is clear supervised training can generate models to predict rigid body transformation parameters. In both cases, randomly generated transformations were applied to the data to have an image pair of a reference and
a transformed image that were related via these transformation parameters. In (Islam, et al., 2021), angles for rotations were generated randomly between the interval $[-15°, 15°]$. A random axis was chosen to perform the rotation. Then a random distance between the interval $[-5, 5]$ was chosen to
perform a translation across the coordinate axis. It unclear whether the researcher performed only one translation on a single axis or one on each axis. In any case, the

researchers stated that this was not the focus of their study. In (Svane Olsen & Nguyen-Cong, 2021), angles for rotations were generated randomly between $[-15°, 15°]$ and transformations were randomly generated between $[-15\text{ voxels}, 15\text{ voxels}]$ for each axis. These were applied to the datasets using the SimpleITK toolkit (Yaniv, et al., 2018) in Python.

One concern with rigid body transformations on images is the method of handling edges in the images after a transformation. Since the coordinates of an image are shifted in the same coordinate space, there will appear undefined pixels. The most common way of handling these undefined pixels is to set them to zero. For 2D axial MRI images, this would not cause too much of an issue unless there is heavy noise, as the skull is shown in the center of the image. In (Islam, et al., 2021), Otsu thresholding was applied to all images. This can effectively reduce the noise of anatomical structure images since the distribution of voxel intensities is crowded at different intervals, as seen in Figure 7. However, for 2D sagittal and coronal MRI images, setting the undefined pixels to zero can introduce unrealistic edges in the transformed image. It is unclear how this is addressed in (Islam, et al., 2021). These issues appeared in (Svane Olsen & Nguyen-Cong, 2021) as well. The distribution of voxel intensities is not similar to the distribution present in structural images, as seen in Figure 7. Hence, simple thresholding of the voxel intensities was made to address noise at the edges.

Since the goal of this project is to register fat navigators, the initial approach to data pre-processing will be the same as in (Svane Olsen & Nguyen-Cong, 2021), where a simple threshold is applied first for reducing noise. The transformation was then applied using the SimpleITK toolkit in Python. For each of the 18000 3D fat navigators, a random rigid body transformation was applied, using the same random intervals as in (Svane Olsen & Nguyen-Cong, 2021). Transformations were applied to the center of the 3D image. Resampling was performed using a B-spline interpolator. To ensure that the same pre-processing steps were applied to the reference image, the same functions were applied but with transformation parameters equal to zero. In total, 18000 image pairs of a reference and transformed images were generated with six single-precision floats of the rotations in degrees and translation in voxels applied on each axis.

3.3 Initial Neural Network Design

Earlier studies have shown that CNNs can perform registration tasks (Svane Olsen & Nguyen-Cong, 2021; Balakrishnan, et al., 2019; Islam, et al., 2021) on medical images. Both approaches in (Svane Olsen & Nguyen-Cong, 2021; Islam, et al., 2021) use supervised training, and present CNN designs where an image pair of a reference and transformed image were fed into their own sequence of layers. Figure 5 shows this separate sequence of layers. Due to the appearance of the network, it will be named as a Y-branch network. The idea is that for each layer in each branch of the network, additional folds are created from the data. The results at the end of the branches are concatenated. (Islam, et al., 2021) proceeds with flattening and then with a FCNN which outputs six numbers. In (Svane Olsen & Nguyen-Cong, 2021), the concatenated result is pushed through another sequence of layers before flattened, and then pushed through a FCNN.

The sequence of layers mentioned is a 3D convolution layer, followed by a batch normalization layer and a ReLU layer. In the case where the data size exceeds hardware limits such as in (Islam, et al., 2021), 3D max pooling layers were added after some ReLU layers to reduce the dimensions.

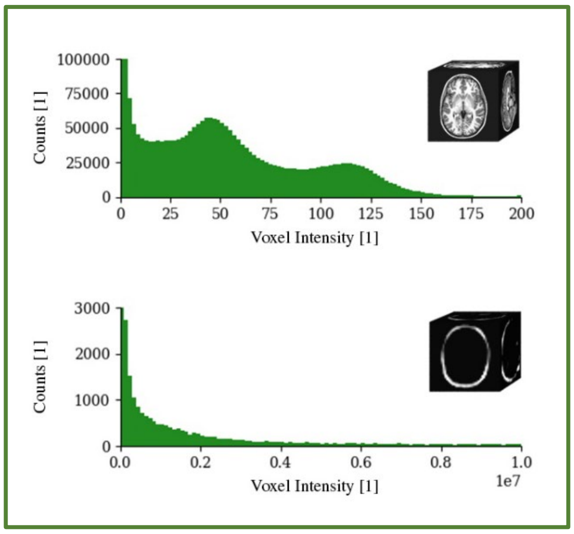


Figure 7: Histograms from two 3D images. Top is a histogram from a publicly available anatomical T1W MRI image and bottom is a histogram from a fat navigator used in (Svane Olsen & Nguyen-Cong, 2021).

Hyperparameter	Search Parameter
Number of convolution layers in a branch	Minimum value: 4 Maximum value: 20 Steps: 2
Symmetrical kernel size for a convolution layer in a branch	Minimum value: 1 Maximum value: 3 Steps: 1
Number of filters for a convolution layer in a branch	Minimum value: 2 Maximum value: 1024 Steps: Power of two
Symmetrical stride for a convolution layer in a branch	Minimum value: 1 Maximum value: 3 Steps: 1
Dropout rate for dropout layers in a branch	Minimum value: 0.0 Maximum value: 0.5 Steps: 0.1
Activation function for every convolution layer in the network	Sigmoid, Tanh, ReLU, or Leaky ReLU
Leaky ReLU scalar parameter	0.5, $1 \cdot 10^{-1}$, $1 \cdot 10^{-2}$, or $1 \cdot 10^{-3}$
Number of convolution layers in concatenated sequence	Minimum value: 1 Maximum value: 6 Steps: 1
Symmetrical kernel size for a convolution layer in concatenated sequence	Minimum value: 1 Maximum value: 3 Steps: 1
Number of filters for a convolution layer in concatenated sequence	Minimum value: 2 Maximum value: 512 Steps: Power of two
Activation function for every dense layer in the network	Sigmoid, Tanh, ReLU, or Leaky ReLU
Number of dense layers in the FCNN	Minimum value: 1 Maximum value: 5 Steps: 1
Number of neurons for a dense layer	Minimum value: 16 Maximum value: 512 Steps: Power of two
Learning rate of optimizer	$1 \cdot 10^{-1}$, $1 \cdot 10^{-2}$, $1 \cdot 10^{-3}$, $1 \cdot 10^{-4}$, or $1 \cdot 10^{-5}$
Momentum of optimizer	0, $1 \cdot 10^{-1}$, or $1 \cdot 10^{-2}$

Table 1: Search parameters for the initial Bayesian hyperparameter optimization of a supervised training network design based on (Svane Olsen & Nguyen-Cong, 2021; Islam, et al., 2021) using the same dataset as in (Svane Olsen & Nguyen-Cong, 2021) with similar augmentations, as described in section 3.2.

A Bayesian approach to hyperparameter optimization was initiated, using TensorFlow 2.7, with search parameters based on the network designs in the mentioned studies. The goal of the hyperparameter search was to minimize a loss function on the validation dataset. The loss function was defined to be the predefined MSE function in the Keras library. A summary of these search parameters is found in Table 1. A Y-branch network was searched for.

The number of convolution layers for a branch was varied in steps of two between the interval $[4, 20]$. For each convolution layer that will be generated in a branch, a number for kernel size, stride, and filter parameter will be varied. This selection will be the same in both branches. Kernel size and stride will be varied between $[1, 3]$ in steps of one. This number will be applied to each direction of the kernel size and stride, making it symmetrical. The number of filters for a convolution layer in the branches varied between the intervals $[2, 1024]$ in steps of power of twos. The selection of activation layers used after every convolution layer in the network was varied between using a sigmoid, tanh, ReLU, and Leaky ReLU function. Then the data is pushed through a batch normalization layer. At every fourth convolution layer, a dropout layer was added. Max pooling layers were not added due to the already relatively low dimensions of the input data. Reduction of the dimensions was thought to

Once each input has been pushed through each layer of their respective branches, the outputs are concatenated. Another sequence of convolutions and activation layers are varied in a similar way as in the branches. However, batch normalizations and dropout layers are not added. The output of the final convolution layer is flattened before being sent into a FCNN. The number of dense layers is varied between the interval $[1, 5]$ in steps of 1, and the number of neurons each contains varied between the interval $[2, 512]$ in steps of the power of two. The last layer of the entire network will be a dense layer with six neurons

The network will be trained using the SGD optimization method, which is pre-defined in Keras. Its learning rate is varied between the interval $[1 \cdot 10^{-1}, 1 \cdot 10^{-5}]$ and the momentum is varied between the interval $[0, 1 \cdot 10^{-1}, 1 \cdot 10^{-2}]$ in steps of the power of ten. The Bayesian hyperparameter optimizer used was the predefined optimizer in the Keras library, with the alpha and beta parameter set to $1 \cdot 10^{-4}$ and 2.6 respectively. 20% of the training dataset was used as a validation set, while the remaining 80% was used to train networks. Networks with no change greater than 1 in validation loss

Hyperparameter	Importance	Correlation
Learning Rate of optimizer	0.217	0.246
Number of convolution layers in a branch	0.146	-0.189
Number of filters for second convolution layer in a branch	0.084	0.111
Momentum of optimizer	0.083	0.271
Symmtrical kernel size for first convolution layer in concatenated sequence	0.052	0.106
Symmtrical kernel size for second convolution layer in a branch	0.038	-0.089

Table 2: Hyperparameter importance and correlation with respect to the MSE of the validation set from the initial Bayesian hyperparameter optimization. This was

after five epochs were aborted in training, and its current weights are saved on the online toolkit for visualizations of machine learning experiments Weights & Biases (Biewald, 2020).

Weights & Biases offers an analysis of parameter importance and correlation with respect to a metric. This method is inspired by Jeremy Howard, founder of the machine learning education platform Fast.ai. A more detailed motivation for this type of analysis is available in the lecture notes (Howard, et al., 2018). From training around 200 models based on the Bayesian hyperparameter optimization initiated, the learning rate was the most important parameter with respect to the MSE of the validation set. A summary of other notable hyperparameter importance and correlation is shown in Table 2. This gives a more general idea of what type of network design would yield the lowest MSE. An increase in the learning rate would increase the loss function. This is most likely due to Missing the global minimum during the gradient descent. The number of convolution layers in each branch was positively correlated to the loss function, meaning that networks with more convolutions in the branches will perform better in predictions than networks with fewer. This is likely due to the that important features, or 'folds', of the initial input data, are extracted the deeper the network the data is in. Both filter number parameters presented in the table show a positive correlation to the loss function. This indicates that it is not the number of feature extractions that are important for this task. Rather, the correlation leans towards that well-fined filters to create relevant 'folds' in the data is more important.

From these 200 models, the model with the lowest validation loss was selected for further training and use. A summary of the exact network parameters is shown in Figure 8. This model was trained for 30 epochs with the training dataset. 20% of this dataset was used as a validation set. This reached an MSE on the validation set of 1.3253772 and the interference time was 40 ms on a GeForce RTX 3090 with TensorFlow 2.7, Cuda 11.2, and Cudnn 8.1.1.33. The corresponding loss and interference time is comparable to the results obtained by (Svane Olsen & Nguyen-Cong, 2021), this shifted the focus of the project towards deployment of the model onto the 7 T scanner.

3.4 Deployment of Deep Learning Model

Interference time is the main factor in this deployment challenge. Since the consensus is that Python performs slower and occupies more memory in several tasks than C++ (Zehra, et al., 2020), it was desired to avoid using a wrapper to Python to call the DL libraries and generate a prediction. Instead, it was desired to do this entirely inside the environment of the host computer and the iMOCO program, which was in C++. Due to the recent popularity of DL, deployment is a frequently discussed topic, and there exist several documentation and libraries that tackle this challenge. Due to time constraints and personal experience, it was not attempted to use the C++ API directly.

3.4.1 Cppflow

Cppflow is a C++ library that is publicly available on GitHub (Izquierdo, 2019), which allows for easy calling of TensorFlow's C++ API without the need to build TensorFlow C. This will automatically enable eager execution mode for fast training and debugging of networks.

For easy testing of this approach for implementation, a Python script will simulate calling the Cppflow program for loading a pre-trained model. Then the Python side will load an image pair from the test dataset and send it to the Cppflow program as an array of single-precision floats in a similar way to how the iMOCO program receives its input arrays. The Cppflow program points to this array as inputs for making a prediction with the pre-trained model. The predictions are sent back to the Python script so that a simple timer can be wrapped around this process. Predictions of the same input images in a C++ and Python environment using the same model were confirmed to be the same.

Building the Cppflow library on the host computer would not be as straightforward. The program written needed to work in the same C++ environment as in the host computer. Up to here, it had been built with C++17 since the standard release of Cppflow was written for it. The majority of the host computer programs had been built on C++98 and Visual Studio 2008, meaning that several syntax differences needed to be addressed. Due to time constraints and experience, syntax changes were made that made Cppflow compatible with C++14. Further syntax changes would involve figuring out what type of variable each auto is creating. This is a time-consuming process. The Cppflow changes for C++14 are available on GitHub. Focus shifted then to another approach for deployment.

3.4.2 LibTorch

LibTorch is the official PyTorch C++ API which is available to the public on the PyTorch website. It is not clear at first whether this library would be easier to make compatible with the host computer C++ environment. Nevertheless, the hyperparameters which gave the lowest MSE from the Bayesian hyperparameter optimization were written entirely in PyTorch to generate a LibTorch compatible model file. This model was trained with the same training dataset for 30 epochs. The necessary PyTorch scripts are available on GitHub.

During the development of a working LibTorch script, it was noted that PyTorch is not compatible with 32-bit systems, which the host computer runs on. Hence, another shift of focus was made in the approach for deployment.

3.4.3 keras2c

The final C deployment library attempted to use for implementation was keras2c (Conlin, 2021). In contrast to the previous two methods, keras2c avoids the use of the

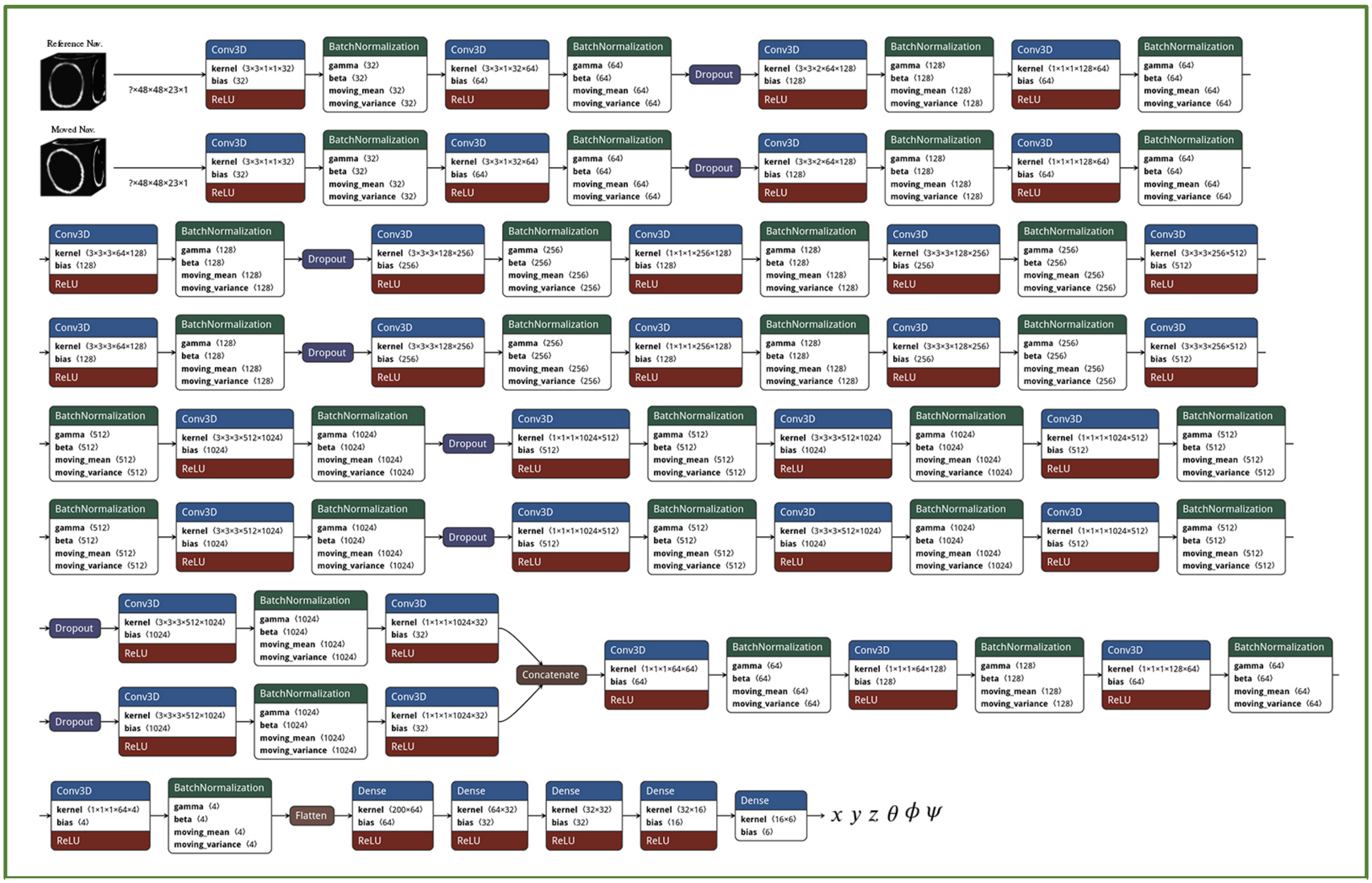
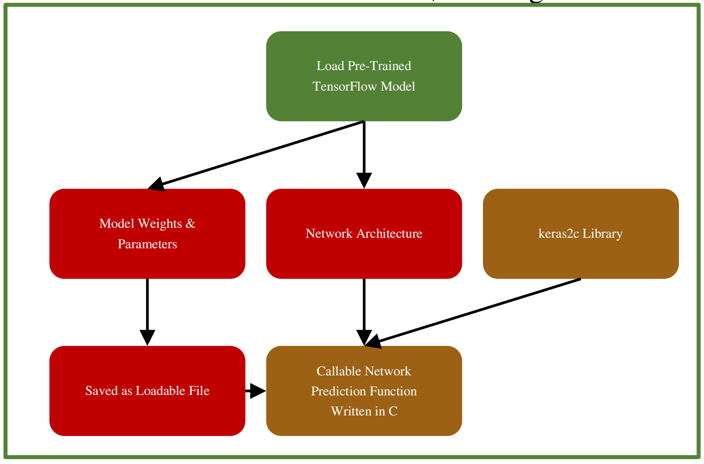


Figure 8: Model selected for deployment testing from the initial Bayesian hyperparameter optimization described in section 3.3.

C++ libraries distributed by TensorFlow. Instead, core functionalities that are required to perform an operation with each layer in a network are written entirely in C99, using only standard library functions. This would also be functional on 32-bit systems but would not implement GPU support out-of-the-box. All layer types from Figure 8 are supported with this library. The workflow of this deployment method is illustrated in Figure 9.

A Python script is used to load a pre-trained model with TensorFlow. Then it extracts all the necessary weights, parameters, and architecture design to make a prediction. These weights and parameters are saved into a separate CSV file and the architecture is automatically translated to the syntax of keras2c. This translation is a C file that describes the functions to call to make a prediction in C. A simple test script can call this prediction function and measure the interference time. In the keras2c paper (Conlin, et al., 2021), differences in interference times with respect to model size between TensorFlow and keras2c were mentioned, as well as differences in prediction results. Hence, different models from the hyperparameter optimization in section 3.3 with sizes with comparable MSE to the model in Figure 8 had their predictions between TensorFlow and keras2c compared, and their interference times in keras2c measured.

The paper (Conlin, et al., 2021) presented results of interference times with TensorFlow, TensorFlow Lite (a lightweight deployment framework for TensorFlow), and keras2c with different types of common layer operations. The measured interference times in our study matched these trends. The timings increased with increasing model size, which can be explained by the larger number of operations. For the model in Figure 8, the interference time on keras2c was 206.25 s. However, a strange result was that the difference in predictions between TensorFlow and keras2c on the same input data was dependent on model size. For smaller model sizes, this difference increased. It is unclear what could be the reason for this result, and this would be the ultimate reason to rethink the approach to deployment.


Load Pre-Trained TensorFlow Model
Model Weights & Parameters
Network Architecture
keras2c Library
Saved as Loadable File
Callable Network Prediction Function Written in C
Figure 9: Diagram showing the workflow of keras2c. (green) indicates Python environment, (red) indicates datafiles, and (brown) indicates C environment.

3.4.4 Network Interface

One Python library which is included in the default distribution of Python is socket. Socket enables client and server to communicate with transmission control protocol (TCP) and user datagram protocol (UDP). These types of communications can enable information to be sent between computers. Client-Server architecture is a standard type of workflow for these communications in a network. In a nutshell, the server refers to the computer that offers an application service to a client. This server creates a socket, that enables a connection and communication through a port in the server computer, which goes into a waiting state which listens for incoming connection requests from other computers, named clients. The main difference between TCP and UDP protocols is speed. The reason for this is that TCP handles multiple clients in a different way than UDP does. UDP handles all incoming data from multiple clients through the same socket, whilst TCP assigns a unique dedicated socket for each incoming connection (Jones & Ohlund, 2002).

This way that UDP handles multiple connections can introduce issues with the ordering of the client data. Incoming data from multiple clients can overlap, which leads to this issue. Hence, a TCP network type will be used.

At the 7 T scanner in DRCMR, there is exists a computer on the same network as the host computer. This computer has a GPU on it and runs Windows 10, and it will be referred to as the GPU-computer. A TCP connection between the GPU computer and the host computer can be established, which can then send the data being sent to the iterative algorithm to the GPU computer. This eliminated the compatibility issues of DL frameworks that were caused by the host computer's environment. Now the data could be processed in a 64-bit Windows 10 environment. The only requirement for the server-side was that it sent an array of six single-precision floats which describe the rigid-body registration in the same way as the iterative algorithm would have outputted it. This was done to reduce the number of needed changes to the actual PMC program and to increase the robustness and deployment to other scanners of this DL approach.

Due to time constraints, a complete replacement of the registration method on the host computer was not made. Instead, the PMC program sends the reference and current fat-navigator through the TCP connection to the GPU computer, then it waits for the six single-precision floats before continuing in the program and generating a registration with the iterative algorithm. The entire server side of the TCP connection on the GPU computer was coded in Python for flexibility, and speed was not prioritized. In the end, two arrays will exist in the host computer's memory, one being the six floats from the TCP connection and the other being the six floats from the iterative method. However, the PMC will only continue the program with the six floats received from the DL approach. The general workflow of the scripts used is illustrated in Figure 10. Timings can be measured of the DL approach, the DL approach, and the iterative approach combined.

One clear obstacle now was the matching of the coordinate systems. To make sure that the coordinate system of the scanner matches the coordinate system of the coordinate system used in the data augmentations, as described in section 3.2, measurements with a phantom and of a subject were performed.

For the measurements with a phantom, the standard fat navigator sequence for the iMOCO program was used. The server side made no predictions. Instead, after the images have been received, the server sent to the client six single-precision floats containing a shift, with a magnitude of 2, in one of the degrees of freedom for ten fat navigators. This shift is the same for each of the ten fat navigators. Another set of ten navigators received another shift in another degree of freedom. In this way, the PMC will apply these transformations to the FOV, since it will be below the threshold of 1 mm in the motion score from equation 32. Then the way the PMC applies the transformations can be assessed based on which direction the phantom is transformed across the ten fat navigators. Since no predictions are made, the timing for the connection workflow can be measured. This was measured to be roughly 15 ms on average across all fat navigator acquisitions.

For the measurements with a subject, the same fat navigator sequence was used. The server side made no predictions, but it sends an array containing six zeros to the client, meaning no transformations should be made to the current navigator. The images received from the client side also saved immediately as binary files on the hard drive of the GPU computer, and the corresponding registration computed by the iterative method was saved together with the binary files. The subject was asked to move their head corresponding to one of the degrees of freedom in rigid body transformations at the fifth fat navigator acquisition. Then the first five navigators will in the reference image position, and the last five navigators will be located at another position. Six sets of ten navigators were acquired, where at each set, the last five navigators had moved in one of the degrees of freedom. The fourth and ninth navigator of these sets were inputted into the pre-trained model from section 3.3. The output predictions are compared to the corresponding iterative computations. It the dimensions of the model predictions were shifted and inverted to identify which order the PMC expected the dimensions to be in the six floats array.

Based on these phantom and subject measurements, no major changes to the dimensions needed to be made, other than that the PMC expects the slice direction dimension to be first, i.e., 7 mm × 5.33 mm × 5.33 mm. Now knowing the difference between the coordinate systems, predictions

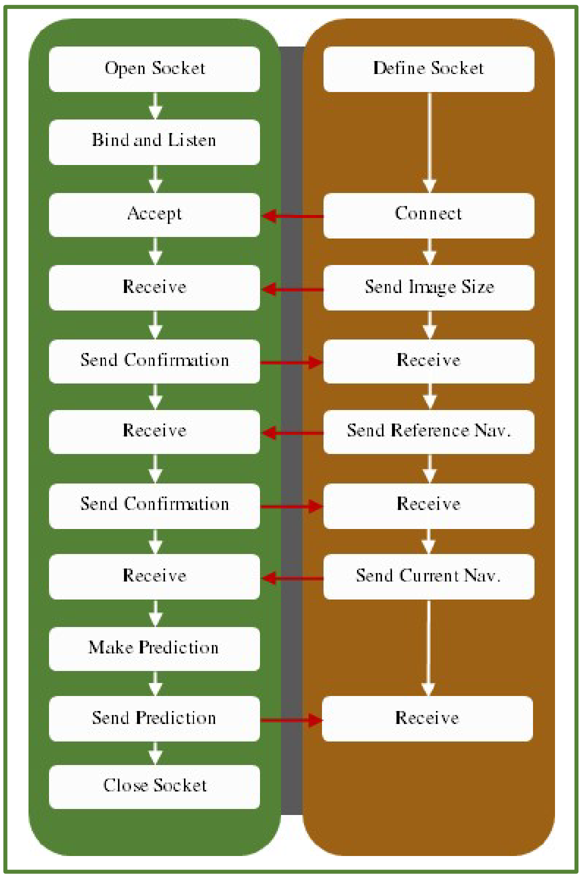


Figure 10: Workflow of the TCP communication scripts between the host computer (brown) and the GPU computer (green).

on the server-side with the initial model were compatible with the PMC program by some shift in dimensions.

The 40 ms interference time measured for the initial model used different hardware than what is available on the GPU computer. Interference times on the GPU computer reached roughly 100 ms on average. This realization prompts the interest in designing smaller models to reduce computation times. However, even with the 100 ms time, this network workflow is comparable to the iMOCO method.

3.5 In-Vivo Testing

One of the most interesting questions to answer in this project is to determine whether models trained in a supervised fashion with synthetic transformations can perform accurate registration with real data. This was tested on the data gathered for the coordinate system verification in section 3.4.4. The six parameters from the iterative method on the scanner and the initial pre-trained model were used to apply transformations on the ninth navigator of each set. These were visually inspected. It was clear that the iterative method produced correct transformations, i.e., that the transformed and reference navigators were accurately registered. Registration with the DL parameters did not lead to any acceptable results, both visually and numerically by assuming the iterative method is a ground truth.

Hyperparameter	Search Parameter
Number of convolution layers	Minimum value: 4 Maximum value: 10 Steps: 2
Number of filters for a convolution layer	Minimum value: 2 Maximum value: 512 Steps: Power of two
Activation function for every convolution layer	ReLU, or Leaky ReLU
Number of dense layers in the FCNN	Minimum value: 2 Maximum value: 6 Steps: 1
Number of neurons for a dense layer	Minimum value: 16 Maximum value: 512 Steps: Power of two
Optimizer choice	SGD, or ADAM
Learning rate of optimizer	$1 \cdot 10^{-1}$, $1 \cdot 10^{-2}$, $1 \cdot 10^{-3}$, $1 \cdot 10^{-4}$, or $1 \cdot 10^{-5}$

Table 3: Search parameters for the new Bayesian hyperparameter optimization of a supervised training network design using a dataset with similar augmentations, as described in section 3.5.

It was first assumed that this deviation was due to the issues mentioned in section 3.2 of the handling noise and clipping at edges due to transformations. Hence, a new approach to data augmentations was taken. Rotations were limited to the interval $[-5°, -5°]$, and translations were limited to the interval $[-2, 2]$. This was considered to be realistic magnitudes of motion within the head coil of the 7 T. Note that the translation interval is denoted in voxels, thus the maximum amount of motion this would include is 14 mm in the slice direction. Thresholding of setting all pixels below 10% of the maximum intensity value in the navigator was set to zero.

One realization was that the network architecture could be re-designed. Instead of having two branches of convolution layers, the subtracted image of the reference and current navigator could be inputted into a single branch network. This would significantly reduce the number of operations in the network but keep information from both images in it.

Based on these new approaches, a new Bayesian hyperparameter optimization was initiated which searched hyperparameters with this new dataset and architecture that optimized an MSE loss function on a validation set. This was done in the same way as in section 3.3, but with the following differences based on the results from Table 2 and intuition. A summary of the hyperparameter search is available in Table 3.

The total number of convolution layers varied between the intervals $[4, 10]$ in steps of one. Each convolution layer had a kernel size of $(4, 4, 2)$ and a stride of $(1, 1, 1)$. The number of filters varied for a convolution layer between the intervals $[4, 9]$ in steps of the power of two. The final parameter that varied for the convolution layers was the activation function. This was varied between using ReLU and Leaky ReLU activation function, where the Leaky ReLU scalar was set to 0.3.

After each convolution layer, the output passes a batch normalization layer, then a dropout layer. This dropout layer dropped 10% of the data. At the third and second last convolution layers, the data passed through a 3D Max Pooling layer with a pool size of $(2, 2, 1)$.

After the final dropout layer, the data is flattened before being sent into an FCNN. The number of dense layers varied between the intervals $[2, 6]$ in steps of one. The number of units in a dense layer varied between the intervals $[4, 9]$ in steps of the power of two. The selection of activation function for all dense layers, besides the output layer, was the tanh function.

The optimizer choice varied between using the SGD or the ADAM algorithms with varying learning rates between the interval $[1 \cdot 10^{-1}, 1 \cdot 10^{-5}]$ in steps of the power of ten.

The model with the lowest MSE was selected for further training with the training dataset and 30 epochs. This reached a validation loss of 0.2034, which is at least a factor 6 less than the loss of the initial Y-branch model. Predictions were made on the coordinate system verification data. Visually, these predicted transformations were more accurate than the Y-branch model. The numerical differences in the six transformation parameters from the iterative and the new DL model were less than with the Y-branch model. However, the magnitude of the difference was still too large to consider the DL model as an accurate predictor of registration parameters. This still leaned towards the conclusion that a more accurate model could be achieved from the improvement of the architecture and data pre-processing.

3.6 Unsupervised Training

An unsupervised training approach was attempted with the same network-style as in Figure 11. However, the outputs of the network had a spatial transformer layer (STL) and the six transformation parameters. The idea of using an STL comes from previous DL registration studies (Jaderberg, et al., 2016). The STL will apply the predicted transformation parameters to the input moving image. Then the loss function can be adapted to minimize the difference between the STL output and the reference image. This difference is evaluated by using a normalized cross-correlation (NCC).

Since the use of synthetic motion data can be avoided, a new data set was collected from two subjects. Each subject was instructed to move their head during an examination. Hence, around 2000 image pairs were collected. Each pair consisted of a moving image and a reference image.

In a similar way as in the supervised training approaches, the network was trained on these 2000 image pairs with the goal of minimizing the NCC in the data set. The trained network was used to pass the entire data set again through the network and evaluate the outputs NCC. Additionally, the data set used to determine the scanner coordinate system was also passed through the network and the outputs NCC was evaluated.

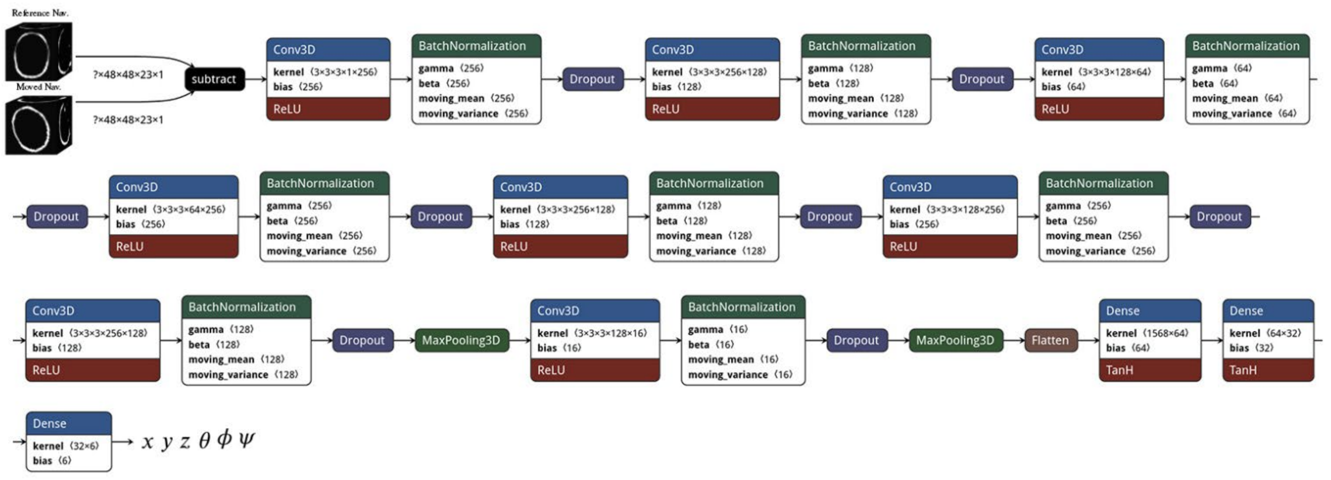


Figure 11: Model selected from the second Bayesian hyperparameter optimization described in 3.5.

4 Results

As mentioned in the methods section, a DL pipeline for image registration was embedded on the Philips 7 T scanner at DRCMR. Different approaches to this were investigated and the findings relevant to the conclusions made are presented here. Development of a DL network that accurately registers fat navigators was investigated. The results from the development stages are presented here as well.

4.1 Initial Supervised Training Network Design

The initial supervised network architecture based on previous studies designs was hyperparameter optimized using a Bayesian approach to the initial data augmentations dataset with the goal of minimizing the MSE of the validation data set. Around 200 models were trained with this search, as summarized in Table 1. Figure 12 shows the validation loss calculated at each epoch. Only models reaching validation losses below 5 are displayed.

The model design that reached the lowest loss was used and trained further with 30 epochs on the same training data set. The validation loss calculated at each epoch on this training is shown in Figure 13. This model will be named as the supervised trained Y-branch model (SYM). The model predicted transformation parameters for the image pairs in the testing data set. The difference between these predictions and the actual transformations of the testing data set is plotted as a histogram in Figure 14. A random image pair in the testing dataset is used to visualize the DL registration. These results are shown in Figure 15.

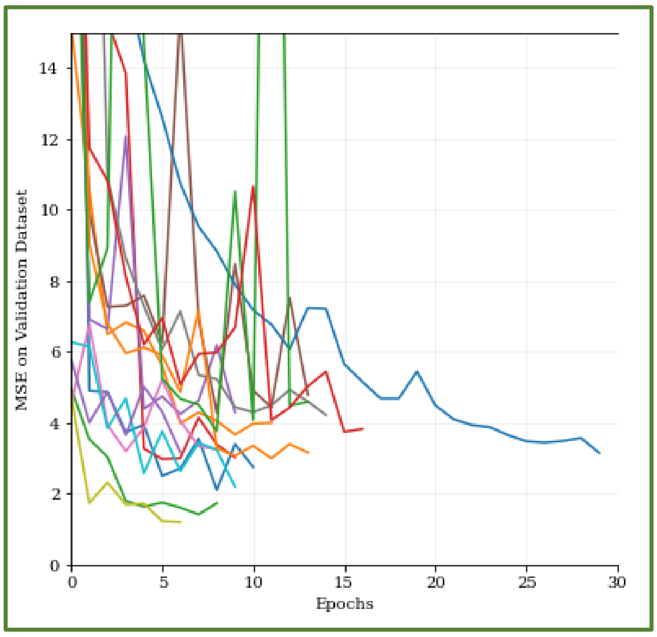


Figure 12: MSE on the validation dataset versus epochs of models trained in the first Bayesian hyperparameter optimization.

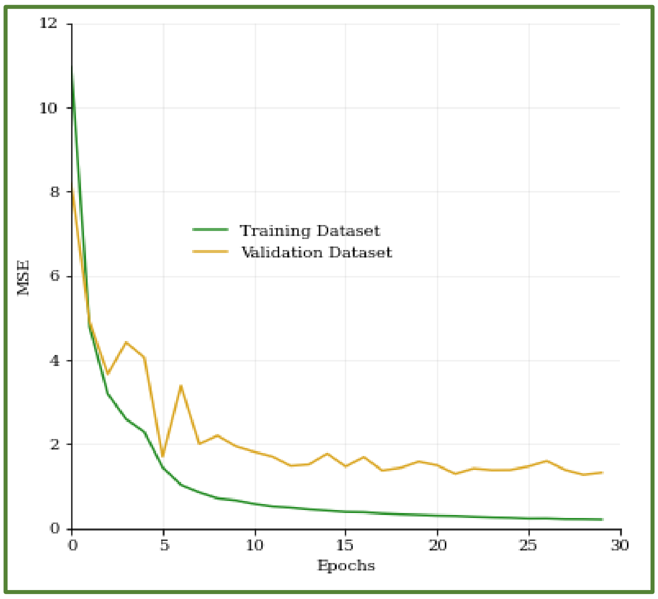


Figure 13: The model which reached the lowest MSE in Figure 12 was used for training with 30 epochs. The results from the training for each epoch is presented in the plot.

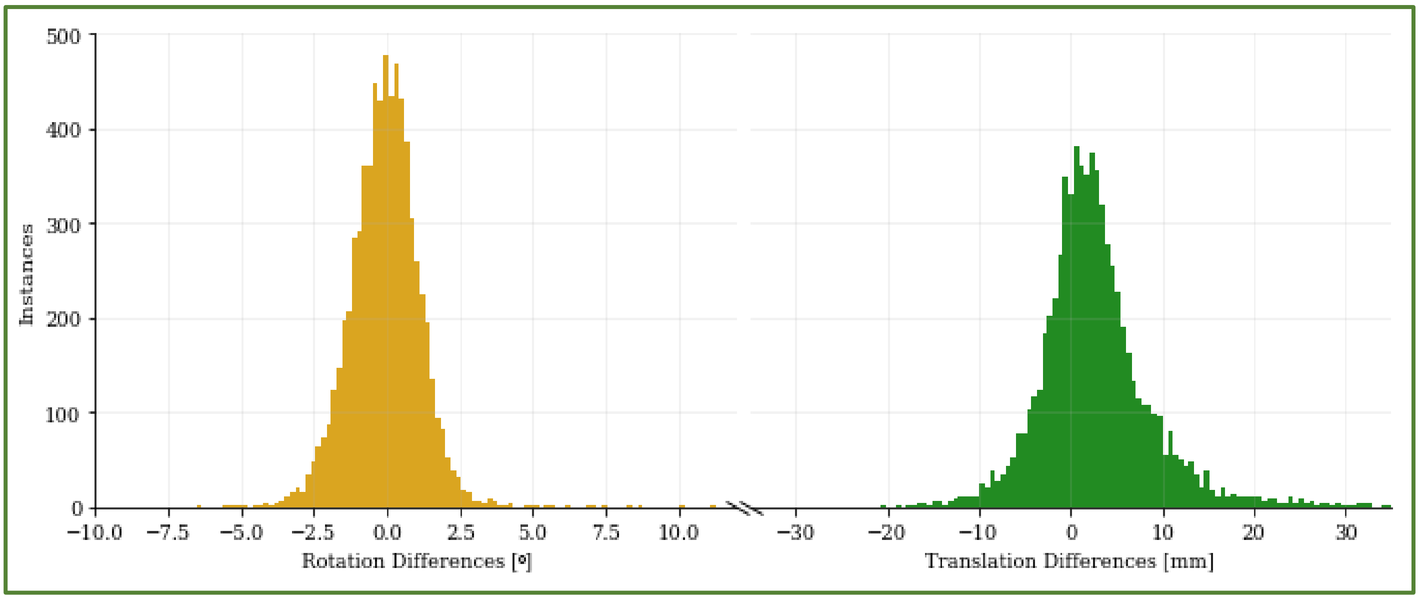


Figure 14: Histogram of differences in predictions made on the testing dataset with the model trained in Figure 12 and the actual transformation parameters.

4.2 Deployment of Deep Learning Model

The interference time of the SYM with Cppflow and keras2c was tested. Table 4 shows these results. The dependence of interference time in keras2c with the number of parameters in a model was investigated. The SYM and two other models which reach similar MSE with the validation data set were used to make a prediction on the same input data using keras2c. This result is presented in Figure 15.

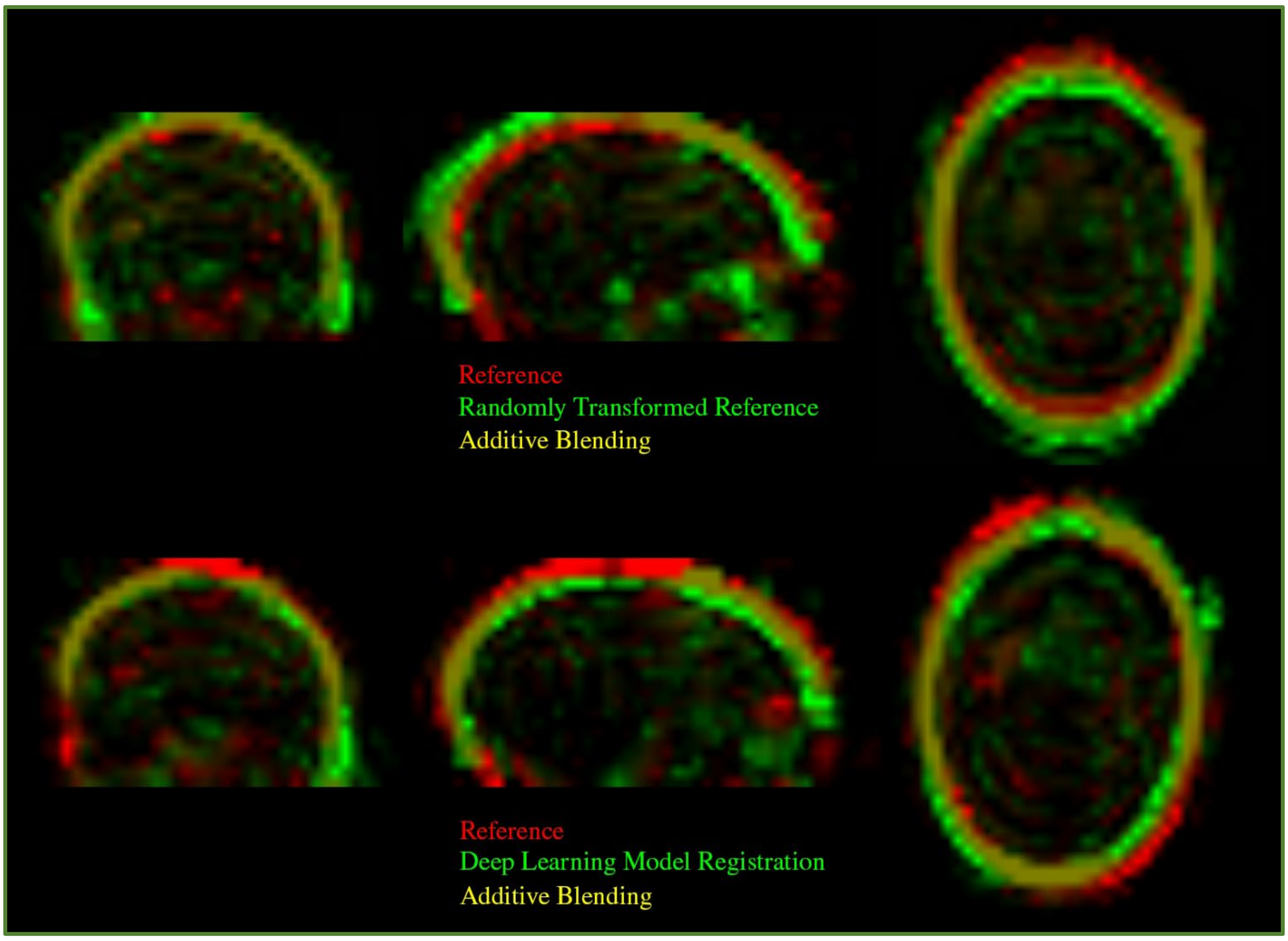

Reference
Randomly Transformed Reference
Additive Blending
Reference
Deep Learning Model Registration
Additive Blending

Figure 15: Registration example of one of the image pairs in the testing data set of the first data augmentations. The first row of slices in the coronal, sagittal and transversal planes show the raw input into the SYM, meaning no registration performed. Green shows the randomly generated motion image and red shows the image without any motion (i.e., reference). The yellow colors indicate where the two images overlap, due to additive blending. The second row of slices show the same planes, but the green image now shows the suggested transformation by the SYM of the motion image for registration.

Predictions made with SYM on Cppflow were the same as in TensorFlow using the same input data. However, this was not the case with keras2c. The same models used to determine the interference time dependence were used to calculate the absolute difference in predictions. The maximum difference is shown in Figure 16 where it is plotted against of the number of parameters in the model.

Framework	Interference Time [s]
Python TensorFlow CPU	~10
Python TensorFlow GPU	~0.040
C++ Cppflow	~8
C keras2c	~200

Table 4: Interference time of SYM using different DL deployment frameworks with the same input data.

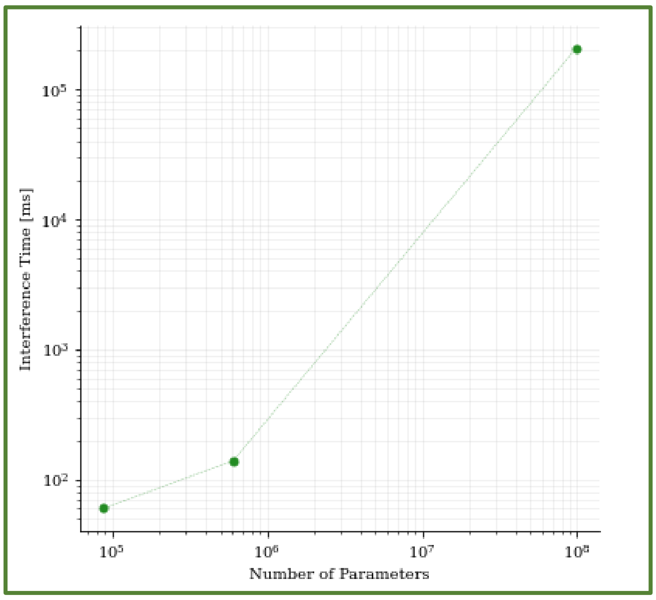


Figure 16: Three trained DL TensorFlow models with different number of parameters used to make a prediction on keras2c with the same input data. The interference time is plotted.

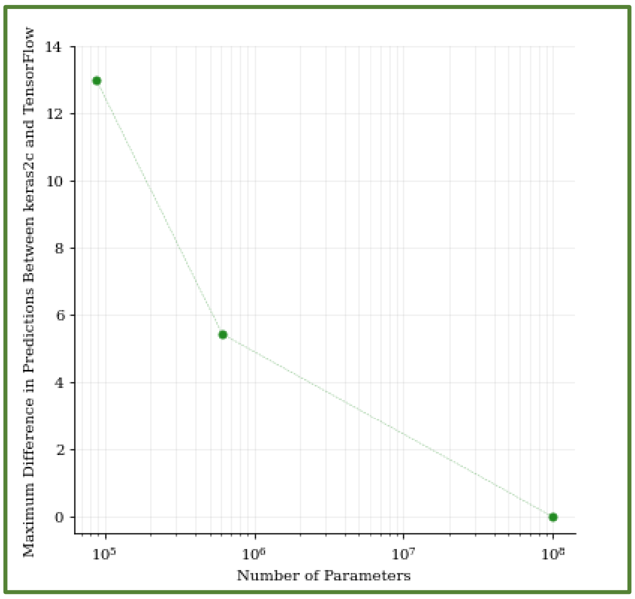


Figure 17: The absolute maximum difference between TensorFlow models predictions made on TensorFlow(Python) and keras2c(C) using the same input data.

The network interface approach for deployment was benchmarked. The timer on the host computer side measured the TCP communication speed which includes sending roughly 4.85 MB. This was found to be around 20 ms. Due to the hardware on the GPU computer, the interference time was twice as long as measured on the offline testing environment.

In combination of determining the coordinate system of the data the PMC program uses, DL registration applied offline was compared to the scanners registration of known motion directions. Meaning, data of actual motion. The difference between the transformation parameters for registration of an image determined by the scanner and the SYM is shown in Table 5.

4.3 Improvements to Supervised Training Network Design

As it was roughly twice as long to make a prediction with the SYM model on the GPU computer, it was convenient to discover that the computational cost of the initial network design could almost be halved by inputting the difference between the volume pairs into the model instead of a Y-branch approach. This was confirmed from the models trained in the second Bayesian hyperparameter search. This search trained around 200 models on the second data augmentations (i.e., the data set with less extreme transformations). Figure 18 shows the validation loss calculated at each epoch for 20 models which reached the lowest loss.

As with the SYM, the model reaching the lowest validation loss was trained further for 100 epochs on the training data set, and the validation loss calculated at each epoch is shown in Figure 19. This trained model was used to predict transformation parameters for registration of an image pair for an entire test data set, as with the SYM. A histogram of the differences in transformation parameters is shown in Figure 20.

Transformation parameters for registering the data set used for determining the coordinate system in the PMC program were calculated using this new subtraction style model. Then, in the same way as with the SYM, the difference is presented in Table 6. An example of the extreme motion registration is visualized in Figure 21.

Motion	Registration	x rotation [°]	y rotation [°]	z rotation [°]	x translation [mm]	y translation [mm]	z translation [mm]
Forward Motion	Deep Learning	-0.3058	0.6716	0.6166	0.8891	-4.7260	9.6780
	Scanner	-0.9611	-0.0620	-0.4982	0.9349	-0.2397	4.6859
	Difference	0.655	0.734	1.115	0.046	4.486	4.992
Right Motion	Deep Learning	-2.4686	1.4410	3.9160	9.5846	10.0013	1.3631
	Scanner	0.1701	-0.4008	-0.2243	1.2409	5.2418	1.8220
	Difference	2.639	1.842	4.140	8.344	4.760	0.459
Left to Right (extreme motion)	Deep Learning	-3.2371	0.4992	3.7185	11.5507	11.6363	5.4866
	Scanner	2.1561	-0.5233	-1.6450	1.6396	7.5922	2.9342
	Difference	5.393	1.022	5.363	9.911	4.044	2.552
Head Towards Feet	Deep Learning	1.4212	0.7230	0.3191	7.0169	-2.6217	-2.2353
	Scanner	0.0592	-1.3242	-0.4443	2.3815	-0.6369	0.4915
	Difference	1.362	2.047	0.763	4.635	1.985	2.727
Looking Right	Deep Learning	0.1554	-2.0589	3.0236	5.1428	3.4053	8.9064
	Scanner	3.4745	-0.7812	0.9748	0.4802	2.6974	1.1133
	Difference	3.319	1.278	2.049	4.663	0.708	7.793
Looking up	Deep Learning	1.7253	13.2859	2.6475	12.0396	-1.4133	-17.4395
	Scanner	-0.0296	4.9990	-0.2260	7.2105	-0.7089	-3.2666
	Difference	1.755	8.287	2.874	4.829	0.704	14.173
Anti-Clockwise rotation	Deep Learning	-0.0683	2.3281	1.2897	3.5457	-5.1324	-0.9067
	Scanner	-1.0445	1.1430	-3.6544	0.5592	-3.2164	-0.8386
	Difference	0.976	1.185	4.944	2.987	1.916	0.068

Table 5: Difference in transformation parameters for registration of image pairs with known motion directions from the scanner and the SYM. Green values indicate positive values and red values indicate negative values. Golden bars indicate the relative magnitude of the difference between the transformation parameters in a single degree of freedom.

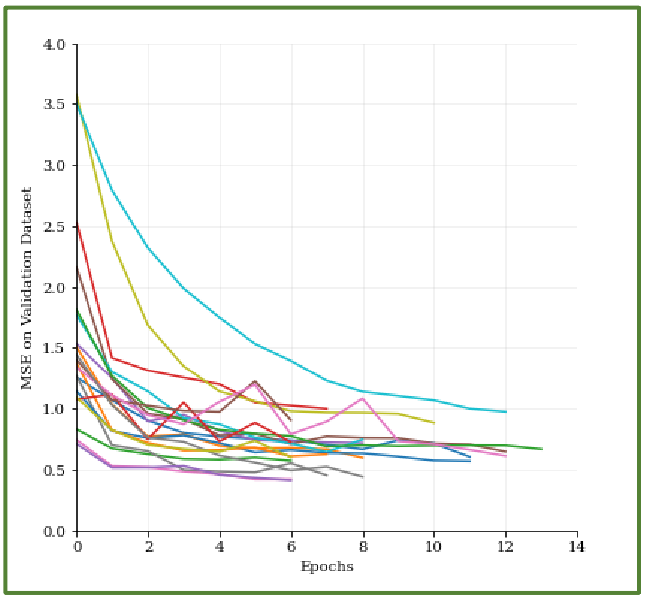


Figure 18: MSE on the validation dataset versus epochs of models trained in the second Bayesian hyperparameter optimization of the subtraction model.

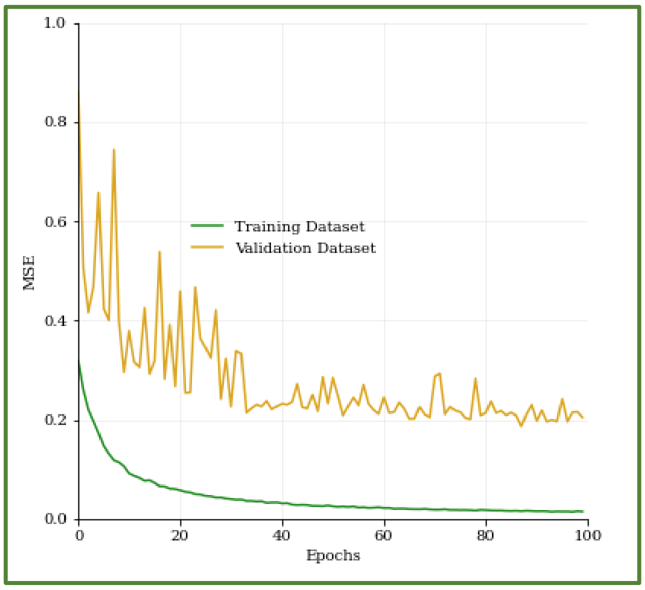


Figure 19: The model which reached the lowest MSE in Figure 18Figure 12 was used for training with 100 epochs. The results from the training for each epoch is presented in the plot.

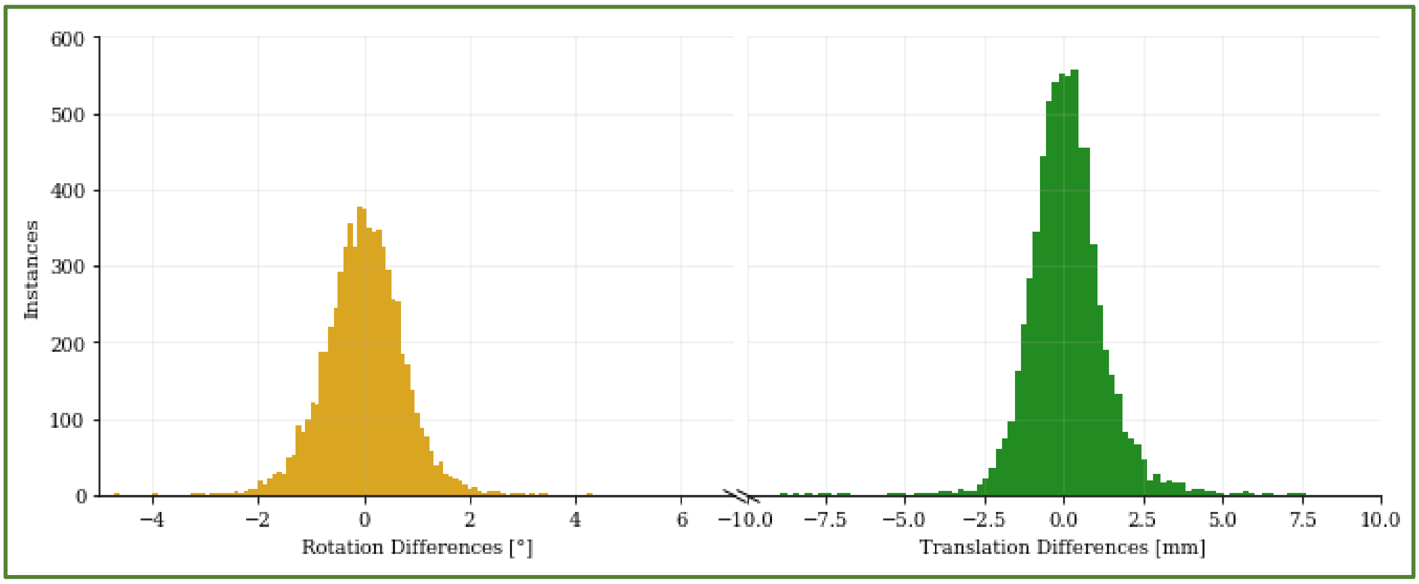


Figure 20: Histogram of differences in predictions made on the testing dataset with the model trained in Figure 20 and the actual transformation parameters.

Motion	Registration	x rotation [°]	y rotation [°]	z rotation [°]	x translation [mm]	y translation [mm]	z translation [mm]
Forward Motion	Deep Learning	0.3084	-0.4953	-0.1798	0.6272	0.0288	3.5858
	Scanner	-0.9611	-0.0620	-0.4982	0.9349	-0.2397	4.6859
	Difference	1.269	0.433	0.318	0.308	0.269	1.100
Right Motion	Deep Learning	-0.7413	0.0780	2.2470	2.4410	5.6787	1.2223
	Scanner	0.1701	-0.4008	-0.2243	1.2409	5.2418	1.8220
	Difference	0.911	0.479	2.471	1.200	0.437	0.600
Left to Right (extreme motion)	Deep Learning	-0.8891	0.0762	1.7563	4.7033	7.2104	2.5639
	Scanner	2.1561	-0.5233	-1.6450	1.6396	7.5922	2.9342
	Difference	3.045	0.600	3.401	3.064	0.382	0.370
Head Towards Feet	Deep Learning	-0.3892	-0.3071	0.3082	2.9211	1.2215	-0.1875
	Scanner	0.0592	-1.3242	-0.4443	2.3815	-0.6369	0.4915
	Difference	0.448	1.017	0.752	0.540	1.858	0.679
Looking Right	Deep Learning	0.5610	0.1060	1.9804	3.7438	3.8187	2.0148
	Scanner	3.4745	-0.7812	0.9748	0.4802	2.6974	1.1133
	Difference	2.913	0.887	1.006	3.264	1.121	0.902
Looking up	Deep Learning	0.0412	1.6048	-1.3230	1.0610	-1.1035	0.7835
	Scanner	-0.0296	4.9990	-0.2260	7.2105	-0.7089	-3.2666
	Difference	0.071	3.394	1.097	6.149	0.395	4.050
Anti-Clockwise rotation	Deep Learning	-1.5180	0.6999	-1.1210	0.7943	-0.3918	0.6283
	Scanner	-1.0445	1.1430	-3.6544	0.5592	-3.2164	-0.8386
	Difference	0.474	0.443	2.533	0.235	2.825	1.467

Table 6: Difference in transformation parameters for registration of image pairs with known motion directions from the scanner and the subtraction style model. Green values indicate positive values and red values indicate negative values. Golden bars indicate the relative magnitude of the difference between the transformation parameters in a single degree of freedom.

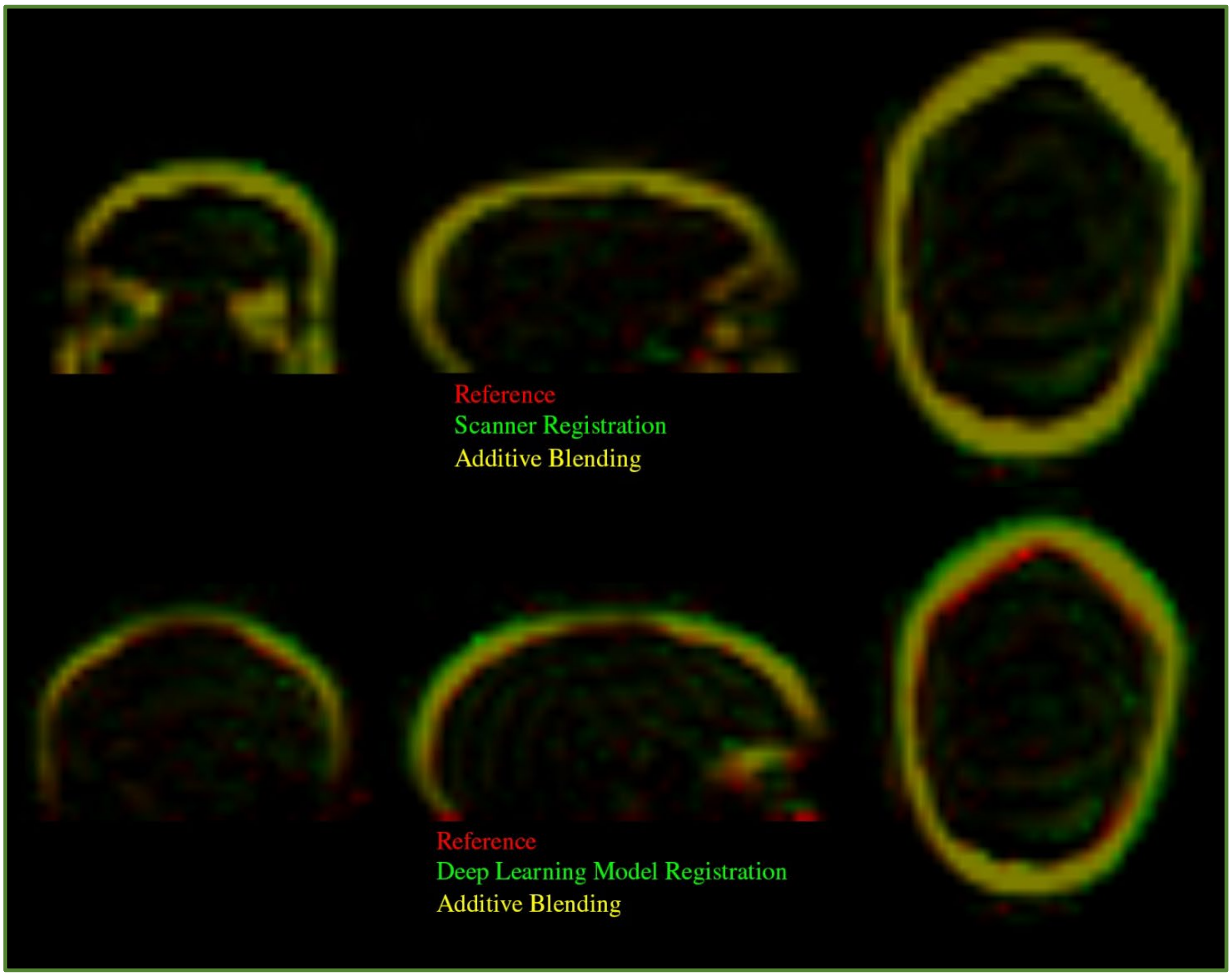


Figure 21: Registration example of the “Left to Right” image pair of the coordinate system determination data set. The first row of slices in the coronal, sagittal and transversal planes show the registration made by the scanner. The second row shows the suggested registration by the subtraction style model. Green shows the transformed image by the respective registration method. The yellow colors indicate where the two images overlap, due to additive blending.

4.4 Unsupervised Learning Network Design

With the goal of minimizing a NCC between a registration of image pairs, a subtraction model was searched for using a random search hyperparameter optimization approach. Around 50 models were trained in this search. Figure 22 shows the NCC at each epoch of 20 of these models which reached the lowest NCC.

The model reaching the lowest NCC was used for further benchmarking. First, the entire data set used to train the model was passed through the trained model to determine the distribution of NCCs. The same input data was passed through the FreeSurfer rigid body registration program. The results of this are presented in Figure 23. To avoid bias, the small data set used to determine the coordinate system of the scanner was also passed through the same registration methods. These results are presented in Figure 24.

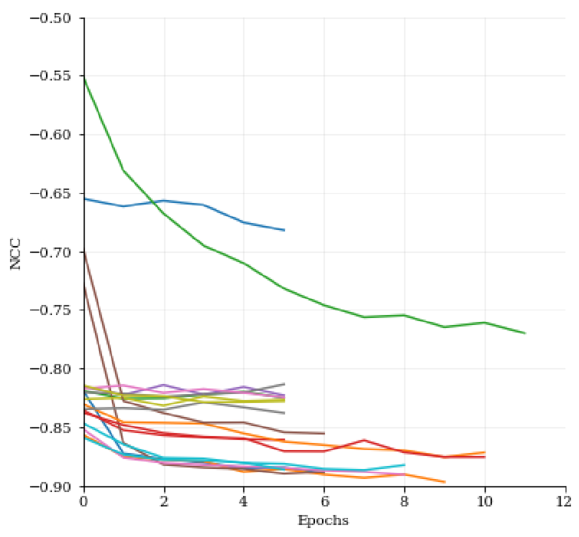


Figure 22: NCC on 200 random images versus epochs of models trained in the second random search hyperparameter optimization of the subtraction model.

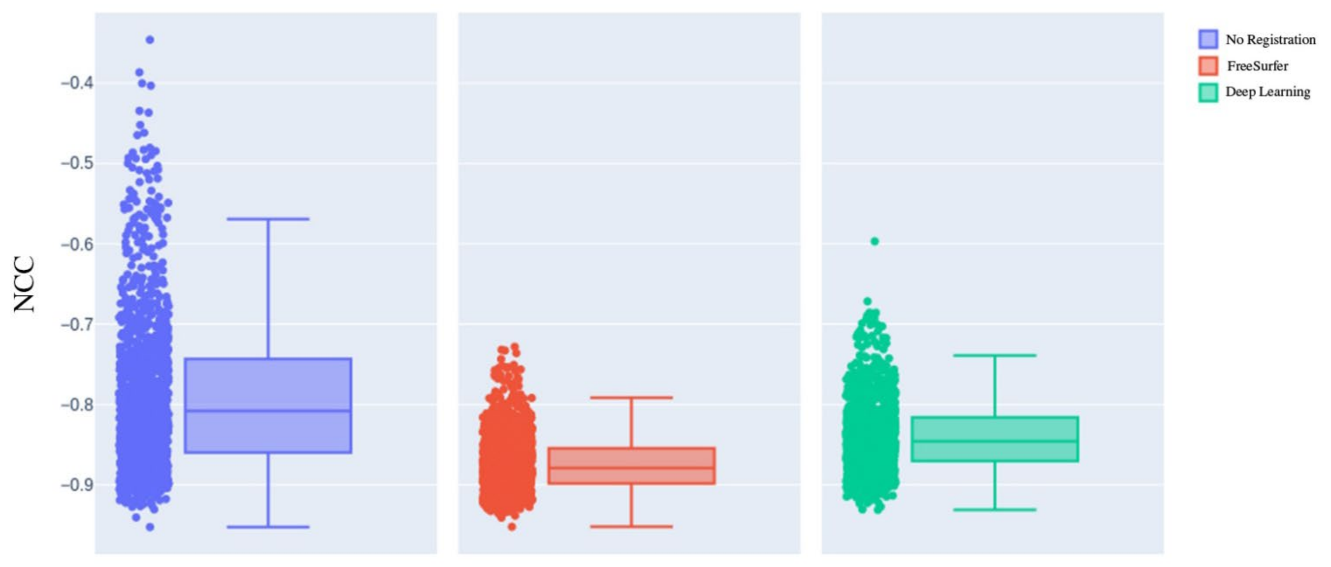


Figure 23: Box plot of NCC distribution of (left) input data, (center) output registration from FreeSurfer, and (right) output registration from DL. Data is the same training data used for the training the model.

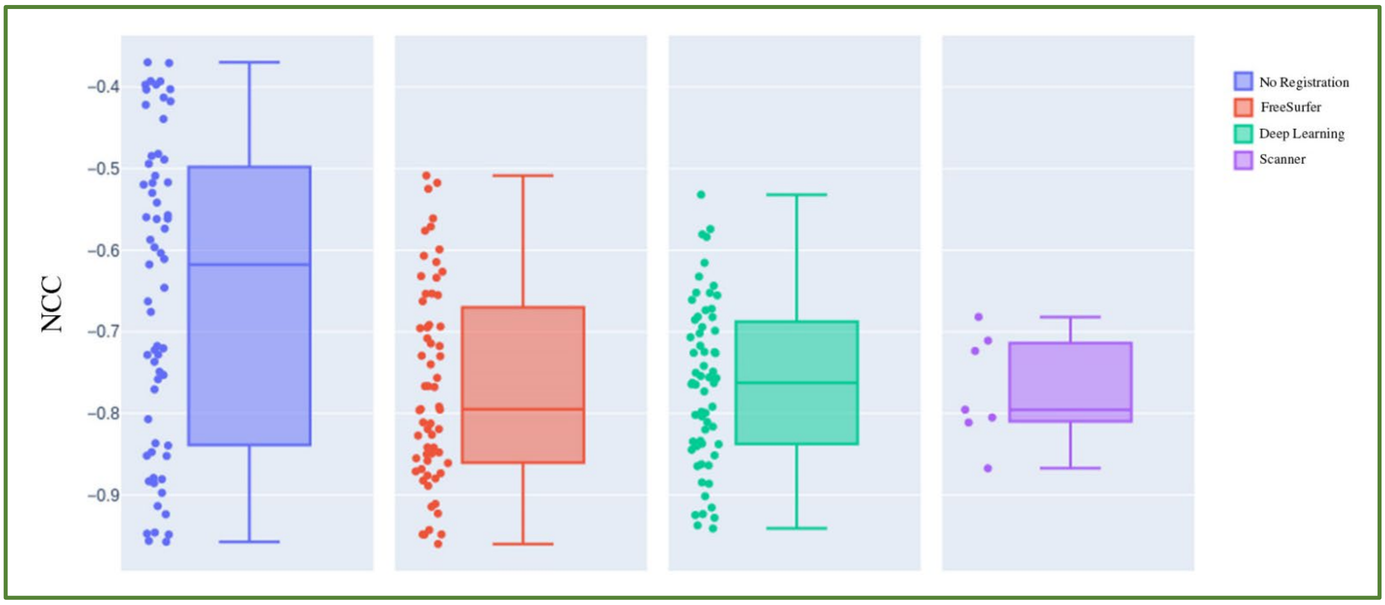


Figure 24: Box plot of NCC distribution of (left) input data, (center-left) output registration from FreeSurfer, (center-right) output registration from DL, and (right) output registration from the scanner. Data is the coordinate system determination data. There are less data points in the scanner distribution due to missing information of the scanners transformation parameters for registration for 80% of the data set.

5 Discussion

The results from this thesis show that DL can accurately perform rigid-body registration of fat navigators obtained at 7 T to some extent. The handling of the data and the type of motion data used for training are shown to be important in generating a feasible network for registration. This is a consensus in DL, that data plays a major role in training a feasible network. Different approaches to manipulation of the input data in supervised training were investigated, as well as training on actual motion data in an unsupervised fashion. Furthermore, the interference times of performing DL registration of fat navigators outrun the current iterative method used in iMOCO. Therefore, it is possible to replace the iterative method in iMOCO with a DL approach with respect to speed, however several general DL issues need to be addressed first to be able to count on the registrations made from the network.

From following approaches that recent DL registration in MRI researchers have used (Svane Olsen & Nguyen-Cong, 2021; Islam, et al., 2021), a supervised trained network was obtained. From the initial hyperparameter search and data manipulations, Figure 12 shows that out of around 200 models, the network reaching the lowest validation loss was used for further training. It would be interesting to investigate further hyperparameter optimizations. However, the validation loss reached by the selected model was comparable to the losses obtained in the previous study which had the same initial dataset (Svane Olsen & Nguyen-Cong, 2021), and due to time constraints, focus on deployment of this network onto the scanner became a priority. Figure 13 shows this network trained and that it reaches convergence within in the number of epochs used. The distribution of errors in Figure 14 agreed with the results obtained in the previous study (Svane Olsen & Nguyen-Cong, 2021). However, visually observing the registrations as in Figure 15 shows unacceptable results of registration to be expected in the iMOCO workflow.

The conclusion from this poor result was that the data manipulations to generate fat navigators with motion were too extreme. Additionally, speed of this DL registration method was an important parameter to investigate. Reconsiderations of the suggested network design were made, and the SYM model reached similar validation losses with the same input data as the Y-branch model did. This design choice almost halved the interference time without the cost of accuracy.

After reducing the distribution of translations and rotations, the error distribution was reduced by almost a factor four in translations as shown in Figure 20. This showed that the generation of motion data played a role in the accuracy of the DL registration predictions. An interesting result is that the distribution of errors in rotations did not decrease at the same grade as in translations. This shows that the network had a hard time learning rotation and that only changes to the degree of transformations in the training data was not enough to improve this ability. The results from both Table 5 and Table 6 support this conclusion. When isolating one rotation degree of freedom, differences between the iMOCO program and DL network rigid body registration parameters were larger than when isolating one translation degree of freedom. To confirm the accuracy of the registration from iMOCO, registration pairs were visually inspected as well as the NCC for seven fat navigator pairs is presented in Figure 24. Both comparisons support the use of the parameters of the scanner as a comparable registration.

Still, registration differences after the improvements to the supervised training data generation were not acceptable for replacing the iterative method in iMOCO. Reductions in differences were significant after the change in data generation, as seen in Table 6. Yet, the differences could reach 4 mm and 3.3°. These motions presented in the table were simulated to be as realistic as possible. Therefore, when the iMOCO program detects motions of around 7.5 mm or 4°, this means that the registration by the DL method can be expected to be at least 50% incorrect, which wouldn't be feasible to use in iMOCO.

One limitation not completely addressed in the supervised training approach was the clipping that arises from the transformations applied to the fat navigators. This was partially addressed by the windowing applied to the images before inputted into the network. This limited the effect that noise had on the clipping. Undefined voxels that appeared due to clipping in the volume were assigned to be zero. Hence, the noise present outside the skull did exist in the zero-filled areas. Removal of the noise was successful to some extent by visually testing different window settings on a few selected fat navigators. Simulating this noise was not attempted in this thesis. It would be interesting to attempt this to improve the accuracy of the registration network. Since by eliminating noise in the volumes with windowing, important information for NN can be lost.

Other DL registration methods for MRI involve some preprocessing steps for data inputted into the network and have shown to produce accurate registrations of more complex geometries compared to fat navigators. One recent study used these types preprocessing steps as well as simulating MRI artifacts on images taken with a MR scanner. Registration networks trained on this data could match the accuracy of state-of-the-art registration methods in MRI (Hoffmann, et al., 2021).

Setting a larger FOV for when acquiring fat navigators was also considered in this thesis. However, the first dataset was assumed to be enough at first to train an accurate registration network. The idea of an increase in FOV was to make it easier to reduce the effects of the clipping when applying transformations to the volumes. The issue will still exist in the edges of the volume towards the body of the subject. Another potential solution would be to replace the undefined voxels with the actual volume flipped along that edge. This would hopefully lead to more accurate registrations on real motion data, since the DL network would be trained on data set which lacked unrealistic motion volumes. Other solutions could involve reducing the FOV of the training data set after transformations has been applied. Then use the DL network to acquire transformation parameters which register a smaller FOV of actual acquired motion data. This will lower the chance that the network will hunt for clipping artifacts in actual motion data. Thus, improve the accuracy of the DL registration. However, this will also reduce the amount of information fed into the network which could play a role in accurate DL registration. The significance of the using the entire FOV in comparison to using a smaller FOV was not investigated in this thesis.

Additionally, the supervised training data set consisted of each individual fat navigator volume being assigned a random transformation once to create a pair. Other studies (Islam, et al., 2021) have shown that a larger number of randomly assigned transformations to a single image can have a positive impact on the accuracy of DL rigid body registration. By which, the data set increases in size, and potentially a sufficient number of synthetic motion volumes are generated for the network to learn accurate registration.

Up to this point, only the supervised training approach has been discussed. In principle, common data set issues in DL have emerged in this project with this approach. The reasoning for this approach was to acquire some sort of ground truth in the predictions that we wanted the DL network to make based on some image pair with motion and without. Defining a patient's motion and linking it to some acquired fat navigator is a difficult task to assure that it is accurate. One method could be to acquire data on patient's head position with lasers or pressure plates. However, the motion determined here might not accurately represent the motion of the brain. The accuracy of these types of methods can also limit the validity of these motion measurements. In cases where ground truth data is unobtainable for supervised training, it is common to consider an unsupervised approach.

Based on previous studies (Balakrishnan, et al., 2019; Hoffmann, et al., 2021) approach to MRI registration using DL, an unsupervised training approach was attempted to solve the lack of ground truth data issue. A network similar to the SYM was used, with the addition of a spatial transformer layer (Jaderberg, et al., 2016) at the end. While being trained on several image pairs only acquired from two subjects, the network was still able to improve the correlation of a data set of image pairs from a completely new and unseen subject, as seen in Figure 24. However, FreeSurfer's rigid body registration still outperformed the DL registration when comparing the change in correlation.

One assumption that needs to be taken into consideration is whether NCC is a good indicator of registration efficiency. The previous studies (Balakrishnan, et al., 2019; Hoffmann, et al., 2021) used NCC as their loss function when training their networks. However, these studies performed deformable registration to MR images with different contrasts. Since the aim of this thesis was to register exclusively fat navigators to each other, a simpler loss function could potentially have been used, such as an MSE. This was rapidly tested when designing the unsupervised network. It was quickly concluded that the NCC would result in better performance. This conclusion could be flawed due to other limitations of this unsupervised training approach. Furthermore, observing a random registration visually from using the DL method or an iterative method as in Figure 21, it can be assumed that the magnitude of NCC achieved by the iterative methods corresponds to acceptable registration since the amount of additive blending in the iterative method registration dominates and is greater than in the DL registration. This trend was observed in all slices of the volumes.

One limitation of the unsupervised training approach is that fat navigators in the data set were acquired from only two subjects. The low number of subjects was due to the time constraints in this project. Typical data set sizes used for training DL registration networks are around 400 subjects (LaMontagne, et al., 2019). With data sets, this large, DL registration networks have been trained to match state-of-the-art registration programs in MRI (Hoopes, et al., 2021). The limited number of subjects in this thesis does impair the generalizability of the network to new subjects. This can be seen in the differences of NCC between the training data set and a new subject data set, as seen in Figure 23 and Figure 24 respectively. For the unseen subject, the network performed worse than on the training data set. This is generally an expected result in machine learning, and it could be seen in Figure 13 and Figure 19 for the supervised training approach as well. One notable result is that even the iterative methods performed worse on this unseen subject data set, as seen in Figure 24. This could indicate that this specific data set was difficult to perform accurate registration for. Furthermore, the performance difference between the iterative and DL registration methods were similar between two data sets. The unseen data set was initially used to determine how the scanner defined its coordinate system. Hence, it contained extreme subject motion in each of the six degrees of freedom. If this data set contained difficult registration problems, then a part of the decreased performance of the DL registration can be associated to this. Assuming this, then the DL registration performed adequately well for only being trained two subjects. The need for a relatively large data set might not be needed to tackle this machine learning problem. This could be due to the somewhat simpler geometry of fat navigators that desired to be registered in comparison to other DL registration studies where T1W volumes are registered to DWI volumes.

As previously mentioned, a smaller FOV might have been a solution to the clipping artifacts arising from the synthetic motion volumes generation in the supervised training approach. In an attempt to improve the accuracy of the unsupervised training approach, the calculation of the NCC was restricted to an inner cube of the original FOV. Specifically, three voxels inwards from each edge of the volume became the new volume to be considered in the NCC. A quick comparison of accuracy by doing this showed positive results. This could indicate that the relevant information for DL registration of fat navigators is contained in the smaller volume.

The spatial temporal profile of MRI images has shown to be compressible. Thus, hinting that the dimensionality of this registration problem could be reduced. One possible solution could be to describe the image at a certain time point with a low rank approximation or another compact representation. By predetermining a reasonable dimensionality reduction of this problem in before-hand would decrease the number of parameters. This would potentially speed up the iterative method, but also the generalizability of trained DL models.

6 Conclusion

Overall, the feasibility of replacing the iterative registration method in the iMOCO framework with a DL registration method is poor. A method for testing DL registration network's performance in conjunction with iMOCO was successfully implemented onto the Philips 7 T scanner at DRCMR. Creating an accurate DL registration network is heavily dependent on the data set used. For supervised trained networks, the use of less extreme random transformations in the generation of training data improved registration accuracy significantly. For unsupervised trained networks, it is possible to generate accurate DL registration networks with a limited number of subjects. This also showed that the ground truth of transformations for registration is not essential.

In conclusion, more data acquisition and research are needed to reach a feasible network. The performance of DL registration networks generated in this thesis are inferior to the current iterative method in iMOCO.

7 References

Afacan, O. et al., 2016. Evaluation of motion and its effect on brain magnetic resonance image quality in children. *Pediatric Radiology,* November, 46(12), pp. 1728-1735.

Andersen, M. et al., 2019. Improvement in diagnostic quality of structural and angiographic MRI of the brain using motion correction with interleaved, volumetric navigators. *PLOS ONE,* May, 14(5), p. e0217145.

Andre, J. B. et al., 2015. Toward Quantifying the Prevalence, Severity, and Cost Associated With Patient Motion During Clinical MR Examinations. *Journal of the American College of Radiology,* July, 12(7), pp. 689-695. 10.1016/j.jacr.2015.03.007.

Balakrishnan, G. et al., 2019. VoxelMorph: A Learning Framework for Deformable Medical Image Registration. *IEEE Transactions on Medical Imaging,* 38(8), pp. 1788-1800.

Biewald, L., 2020. *Experiment Tracking with Weights and Biases.* [Online] Available at: https://www.wandb.com/ [Accessed 24 12 2021].

Bookwalter, C., Griswold, M. & Duerk, J., 2010. Multiple Overlapping k-Space Junctions for Investigating Translating Objects (MOJITO). *IEEE Transactions on Medical Imaging,* February, 29(2), pp. 339-349.

Brown, R., 2014. *Magnetic resonance imaging: physical principles and sequence design.* 2. ed ed. Hoboken, NJ: Wiley Blackwell.

Conlin, R., 2021. *f0uriest/keras2c: v1.0.2.* [Online] Available at: https://zenodo.org/record/5708865 [Accessed 26 12 2021].

Conlin, R., Erickson, K., Abbate, J. & Kolemen, E., 2021. Keras2c: A library for converting Keras neural networks to real-time compatible C. *Engineering Applications of Artificial Intelligence,* Volume 100, p. 104182.

Godenschweger, F. et al., 2016. Motion correction in MRI of the brain. *Physics in Medicine and Biology,* March, 61(5), pp. R32-R56.

Haacke, E. & Patrick, J. L., 1986. Reducing motion artifacts in two-dimensional Fourier transform imaging. *Magnetic Resonance Imaging,* January, 4(4), pp. 359-376.

Havidich, J. E. et al., 2016. Preterm Versus Term Children: Analysis of Sedation/Anesthesia Adverse Events and Longitudinal Risk. *Pediatrics,* March, 137(3), p. e20150463.

Hoffmann, M. et al., 2021. SynthMorph: learning contrast-invariant registration without acquired images. *IEEE Transactions on Medical Imaging,* pp. 1-1.

Hoopes, A. et al., 2021. HyperMorph: Amortized Hyperparameter Learning for Image Registration. *arXiv:2101.01035 [cs, eess].*

Howard, J., Thomas, R. & Gugger, S., 2018. *4— Feature Importance, Tree Interpreter.* [Online] Available at: https://course18.fast.ai/lessonsml1/lesson4.html [Accessed 25 12 2021].

Islam, K. T., Wijewickrema, S. & O'Leary, S., 2021. A deep learning based framework for the registration of three dimensional multi-modal medical images of the head. *Scientific Reports,* 11(1), p. 1860.

Izquierdo, S., 2019. *Cppflow.* [Online] Available at: https://github.com/serizba/cppflow/ [Accessed 25 12 2021].

Jaderberg, M., Simonyan, K., Zisserman, A. & Kavukcuoglu, K., 2016. Spatial Transformer Networks. *Advances in neural information processing systems 28.*

Jones, A. & Ohlund, J., 2002. *Network programming for Microsoft Windows.* 2nd ed ed. Redmond, Wash: Microsoft Press.

LaMontagne, P. J. et al., 2019. OASIS-3: Longitudinal Neuroimaging, Clinical, and Cognitive Dataset for Normal Aging and Alzheimer Disease. *Radiology and Imaging.*

Olsson, H., 2021. Gradient echo-based qantitative MRI of human brain at 7T: mapping of T1, MT saturation and local flip angle. *Thesis.*

Paszke, A. et al., 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. *Curran Associates, Inc.,* pp. 8024-8035.

Qin, L. et al., 2009. Prospective head-movement correction for high-resolution MRI using an in-bore optical tracking system. *Magnetic Resonance in Medicine,* October, 62(4), pp. 924-934.

Rojas, R., 1996. *Neural Networks.* Berlin, Heidelberg: Springer Berlin Heidelberg.

Svane Olsen, O. & Nguyen-Cong, C., 2021. *Deep Learning-based Motion Correction in Magnetic Resonance Imaging.* Copenhagen: Technical University of Denmark.

Tisdall, M. D. et al., 2012. Volumetric navigators for prospective motion correction and selective reacquisition in neuroanatomical MRI: Volumetric Navigators in Neuroanatomical MRI. *Magnetic Resonance in Medicine,* August, 68(2), pp. 389-399.

Törnqvist, E., Månsson, Å., Larsson, E.-M. & Hallström, I., 2006. Impact of extended written information on patient anxiety and image motion artifacts during magnetic resonance imaging. *Acta Radiologica,* June, 47(5), pp. 474-480.

Usman, M. et al., 2020. Retrospective Motion Correction in Multishot MRI using Generative Adversarial Network. *Scientific Reports,* December, 10(1), p. 4786.

Yaniv, Z., Lowekamp, B. C., Johnson, H. J. & Beare, R., 2018. SimpleITK Image-Analysis Notebooks: a Collaborative Environment for Education and Reproducible Research. *Journal of Digital Imaging,* 31(3), pp. 290-303.

Zehra, F., Javed, M., Khan, D. & Pasha, M., 2020. Comparative Analysis of C++ and Python in Terms of Memory and Time. *preprint.*