Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Anders Johannessen

Affine Alignment of Ultrasound Volumes Using Deep Learning

Master's thesis in Cybernetics and Robotics
Supervisor: Gabriel Kiss, Hans Torp
June 2019

NTNU
Norwegian University of Science and Technology

Anders Johannessen

Affine Alignment of Ultrasound Volumes Using Deep Learning

Master's thesis in Cybernetics and Robotics
Supervisor: Gabriel Kiss, Hans Torp
June 2019

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Summary

This thesis explores the possibilities of deep learning based image registration of 3D ultrasound volumes. Two models for affine transformation of a moving image in reference to a target image is being compared. The first model is inspired by the affine transformation framework proposed by de Voss et al. (2019). This model passes the full ultrasound image directly through a deep learning network to predict the optimal affine transformation matrix that maximizes the Normalized Cross-Correlation (NCC) similarity measure. The second model proposed in this thesis is inspired by the framework for Large Deformation Diffeomorphic Metric Mapping (LDDMM) developed by Yang et al. (2017). This model divides the 3D image into multiple smaller patches of uniform size before passing them through a deep learning network for transformation matrix prediction. The change in NCC between the moving and target image before and after transformation, is being used as a metric for performance of the two models. The results show that the patch based model gives an 0.0044 improved NCC after transformation of the moving image. The full image based model was not able to give any preformed NCC on the test image dataset even though the NCC during training and validation showed an improvement similar to the patch based model.

The image dataset used in the project consists of multiple ultrasound recordings of the left ventricle. Each recording consists of several images take over the course of 3-5 heart cycles. Between recordings, the probe Transesophageal echocardiogram (TEE) transducer probe has been shifted slightly. This gives a misalignment between the different recordings. The purpose of this is to simulate probe movement during a procedure.

Before the ultrasound images are passed through the deep learning network, filtration methods such as Gaussian blur and histogram equalization are applied to the images. The Gaussian blur filtration is done in an effort to reduce the large amount of noise prevalent in ultrasound images. Histogram equalization is applied in order to enhance characteristic features within the image like edges, corners and light patches corresponding to heart wall tissue. The results show that the Histogram equalization is a necessary preprocess in order for the deep learning network to generalize and learn to recognize key features and apply it to affine transformation. The Gaussian blur is not a necessary preprocess for learning, but will improve the difference in NCC after transformation with 66.7%.

From the analysis of the runtime of the two models presented in this thesis, the results show that the full image based model out preforms the patch based model. The average runtime of the full based model per generated image is 891ms, while the average runtime of the patch based model per generated image is 1012ms. The difference in time mainly comes down to the variable and memory handling associated with splitting the full image into multiple patches.

Both models where implemented using CUDA based GPU acceleration.

Sammendrag

Denne masteroppgaven utforsker mulighetene for dyp læringsbasert bilderegistrering av 3D ultralyd volumer. To modeller for affin transformasjon av et bevegelig bilde og i referanse til et fast bilde blir sammenlignet. Den første modellen er inspirert av affin transformasjon rammeverket foreslått av de Voss et al. (2019). Denne modellen sender hele ultralydbildet direkte gjennom dyp læringsnettverket for å predikere den optimale affin transformasjons matrisen som maksimerer den Normaliserte Krysskorrelasjon (NKK). Den andre modelen foreslått i denne masteroppgaven er inspirert av rammeverket for kartlegging av deformasjon av vev utviklet av Yang et al. (2017). Denne modellen deler det tredimensjonale ultralydbildet opp i flere mindre deler med lik størrelse før disse sendes inn i dyp læringsnettverket. Endringen i NKK mellom det bevegelige bildet og det faste bildet før og etter transformasjonen er brukt til å beregne ytelsen til de to systemene. Resultatet viser at den del-baserte modellen har en forbedring på 0.0044 i NKK etter transformasjon. Den fult bilde-baserte modellen viste ingen forbedring i NKK etter tranformasjon på test bildene selv om NKK under trening og validering viste en forbedring i tråd med den del-baserte modellen.

Datasettet med bilder brukt i dette proskjektet inneholder flere ultralyd opptak av venstre ventrikkel. Hver opptak består av flere ultralydbilder tatt over 3-5 hjerte sykluser. Mellom hvert opptak har tranduser proben blitt litt forskjøvet. Dette gjør at bildene mellom hvert opptak blir forskjøvet i forhold til hverandre, noe som simulerer at tranduser proben beveger seg under operasjon.

Før ultralydbildene kan sendes gjennom dyp læringsnettverket blir bildene filtrert med Gaussisk filtrering og histogram utgjevning. Den Gaussiske filteringen er gjort i et forsøk på å redusere den store mengden støy som er til stede i ultralydbilder. Histogram utjevning er anvendt for å forsterke karakteristiske egenskaper i bildet som kanter, hjørner og lyse områder som korresponderer med vev i hjerteveggen. Resultatene viser at histogram utjevnelse er en nødvendig filtrerings metode for at dyp læringsnettverket skal kunne generalisere og lære å gjenskjenne nøkkelegenskaper og bruke dem til affin transformering. Gaussisk filtrering ikke er nødvendig for læring, men fører til en forbedring av systemet på 66.7%.

Fra kjøretidsanalysen til de to modellene presentert i denne masteroppgaven viser resultatene at den fult bilde-baserte modellen har en raskere hastighet enn den del-baserte modellen. Den gjennomsnittlige kjøretiden den fult-bilde baserte modellen bruker på å generere ett transformert bilde er 891ms, mens den gjennomsnittlige kjøretiden til den del-baserte modellen er på 1012ms. Forskjellen i kjøretid kommer hovedsaklig av variabel og minnehåndteringen assosiert med å dele ett bilde opp i flere mindre biter. Begge modelllene er utviklet med CUDA basert GPU akselerasjon.

Preface

This thesis marks the end of my 5 year education at the Norwegian University of Science and Technology (NTNU). I have been so lucky to be a part of the great program: Cybernetics and Robotics from the Department of Engineering Cybernetics. During my studies I have got a unique insight into how the technology around us, that we take for granted, works and its underlying principals. During the last 2 years of my studies I specialized in Artificial Intelligence (AI) and Computer Vision (CV) as well as biomedical cybernetic engineering. This has culminated in this thesis, which is a follow up from my project completed during the Fall of 2018. Therefore some of the theory and method presented in this thesis may be similar to the theory and method presented in the report from last semester.

The data used in this project is 3D B-mode ultrasound volumes acquired from 13 patients that presented to the clinic at St. Olavs hospital for a TEE examination or underwent cardiac surgery that included cardiac bypass. At least 3 complete cardiac cycles were captured for all patients using a GE Vivid E95 or E9 system with a 6VT-D probe (GE Vingmed, Ultrasound, Horten, Norway). No selection of the data was performed, the main clinical contact was Erik Andreas Rye Berg.

I would like to thank Gabriel Kiss at the Department of Circulation and Medical Imaging for being my supervisor and guiding me through this project. He has provided me with ultrasound images and material necessary to complete this project. He has also acted like a bridge between the two topics presented in this thesis: ultrasound imaging and deep learning. Without his enthusiasm for the topic, this thesis would not be possible. I would also like to thank Hans Torp at the Department of Circulation and Medical Imaging for the great course on medical ultrasound imaging. This course was my first introduction to ultrasound imaging and it gave me a new perspective on the underlying principles of ultrasound imaging.

Contents

CHAPTER 1

Introduction

Ultrasound is a medical image modality with many advantages compared to other modalities like X-ray, Computerized Axial Tomography (CAT), Positron-Emission Tomography (PET) and Magnetic Resonance (MR). X-ray, CT and PET expose the patient to harmful ionizing radiation. MR and Ultrasound does not expose the patient, as well as the medical personnel, to harmful radiation because images generated from these modalities comes from exposing the patient to strong magnetic fields and sound waves in the ultrasound spectrum (starting from around 20kHz) respectively. However, ultrasound is in another league compared to MR when it comes to cost of equipment and size of equipment. This makes ultrasound a much more safe and available medical image modality tool at hospitals compared to X-ray, CAT, PET and MR. Artificial Intelligence (AI) and deep learning are at the forefront in the development of new technology for Computer Vision (CV). Equipping the future operation room with the latest and most advanced technology servers the dual purpose of freeing up medical personnel, as well as minimizing the possibility of human errors. This means that procedures can be done in a safer and more efficient manner.

Real time monitoring of the heart functionality is a critical component in the diagnostic phase as well as both during and after procedures such as bypass surgery, vascular surgery and valve related interventions. The introduction of real time 3D Transesophageal Echocardiogram (TEE) has provided medical personnel with a new tool from ultrasound imaging. A study committed by Ning et al. (2008) showed that 12.5% of the patients with mitral valve defects that participated in the study were not diagnosed correctly using traditional 2D TEE, but successfully diagnosed using 3D TEE. This shows that the better overview of 3D images (often referred to as volumes) compared to traditional 2D images may provide the additional information needed to discover possibly life threatening conditions. However, the introduction of an additional dimension creates a new set of degrees of freedom when positioning the transducer probe for

the echocardiographer. Drifting of the transducer probe may therefor be a challenging aspect for the echocardiographer. In this thesis we are going to discuss two models for non-rigid 3D image registration using deep learning techniques. The goal is to propose a model for affine transform of a misaligned ultrasound image due to transducer probe movement, with respect to a reference image with correct frame position using a deep learning approach.

Today, the most image registration methods for 3D images are used in a post-process manner. This is due to the computational requirements of the iterative optimization of 3D image registration algorithms. Deep learning methods such as Convolutional Neural Networks (CNN) has been shown to speed up the the image registration process substantially (Yang et al., 2017). This is because the time consuming iterative optimization is done during the training phase in the development of the deep learning network. For the image registration part, a deep learning based system uses just a single forward pass through the network to predict the transformation parameters. Previous work done on ultrasound are mainly focused on the 3 classical CV tasks: image classification, image segmentation and object detection using labeled data for supervised learning. Ghesu et al. (2016) uses all three task in their model for detection and segmentation of the left ventricle from 3D TEE images. They are using a sparse adaptive deep learning network together with marginal space learning to predict a bounding box around the left ventricle and preform segmentation on the content within the bounding box. The object detection part can be formulated as a patch-wise classification problem. Each patch has a binary label representing whether or not the desired anatomical structure is contain within the patch. By using the pipeline structure: object detection followed by segmentation, they are able to preform fast and accurate segmentation of the left ventricle.

Classical image registration is an extensive research field. Searching for relevant previous work on medical image registration on scientific search engines like Google Scholar[1] or the NTNU university library database Oria[2] results in thousands of hits. Number of citations, publishing year and abstract give good indications on the relevance of an article, and are all used to critically assess the relevance of the study. A model for image registration of ultrasound images has been developed by Danudibroto et al. (2016). They are using a piecewise 1D cubic B-slpine interpolation together with iterative multiscale Farnebäck optic flow to preform temporal, as well as spatial image registration. The temporal image registration detect the cardiac phase of the ultrasound image, while the spatial image registration computes the optimal transformation between two ultrasound images. They show that the field of view increases by 15.6% and 25.4% on 2D and 3D ultrasound images respectively by using the proposed method.

Rajpoot et al. (2011) proposes a model for fusion of multiple ultrasound to create a multiview fused image. This is done by constructing a three-level Gaussian pyramid. Starting from the roughest level, the rigid transformation that results in the highest Normalized Cross-Correlation (NCC) is passed to the next finer level for more refined transformations. This pro-

[1] https://scholar.google.no/
[2] https://www.ntnu.no/ub

cess is iterated though the entire pyramid. The final transformation at the finest level of the Gaussian pyramid results in the best transformation of the two ultrasound images.

A method for compounding ultrasound images acquired from different views has been proposed by Grau and Noble (2005). They are using information about the structural content of the image based on image phase as well as the orientation of the image to weight the contribution of different images. Strong image features are found by using the phase congruency. If the phase congruency returns a large value for a single image, the information from that image is adopted. If multiple images has large phase congruency, a contribution of each image based on their alignment is adopted. For low values of the phase congruency, there are no features within the images and thus considered noise.

This thesis is a follow up to the project completed in the Fall of 2018 (Johannessen, 2018). That project aimed at producing the deformation of two subsequent frames in a ultrasound video stream using the "*Quicksilver*" system implemented by Yang et al. (2017). Quicksilver is a deep learning based system for image registration of MR images of the human brain. A challenging aspect shown in this project how how difficult image registration of ultrasound images can be due to large amount of noise prevalent in ultrasound images. The Quicksilver framework will be discussed further in Section 2.5 Image registration. The aim of this thesis is to develop a framework for affine transformation of ultrasound using deep learning. Two models are being presented inspired by the models for affine transformation using deep learning developed by Yang et al. (2017) and de Voss et al. (2019). Both models uses a deep learning network to predict a transformation matrix and from there uses classical image registration affine transformation to produced an aligned image based on a reference. This is done in an effort to speed up the image registration process for real-time applications.

This thesis is structured as follows: In section Chapter 2 Theory, we are going to take a thorough look the theories behind image registration and deep learning approach to the image registration problem. Chapter 3 Method and Chapter 4 Implementation gives a detailed explanation of the two proposed methods and the tools used in the implementation of the models. The results of the deep learning framework for image registration implemented in this thesis are being presented in Chapter 5 Results, for both a patch based approach and a full image approach. In Chapter 6 Discussion we are going to discuss the results as well as give suggestion on how to extend the system in future work. And finally in Chapter 7 Conclusion, my closing thoughts on the project are being presented.

CHAPTER 2

Theory

2.1 3D transesophageal echocardiogram

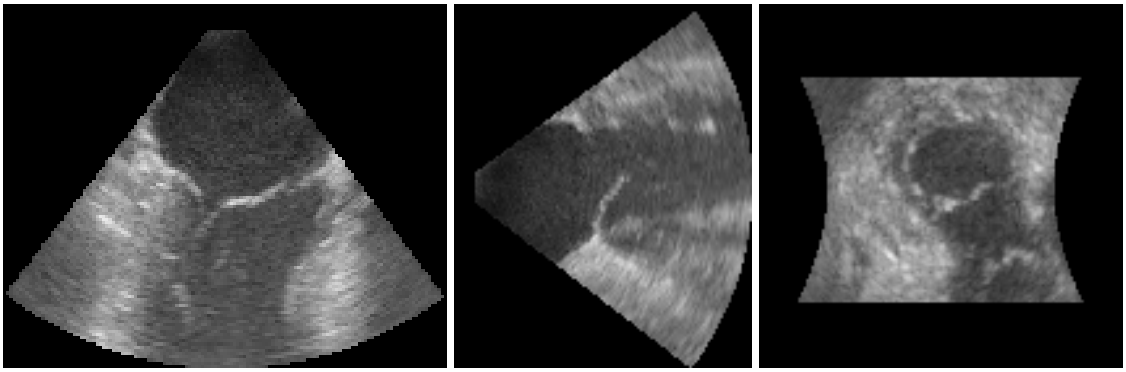
This section is cited from from Johannessen (2018).

Transesophageal echocardiogram (TEE) is a method to generate high quality ultrasound images of the heart (echocadiogam). This is done by lowering a specialized transducer probe through the esophagus of the patient. The clinician positions the probe a few millimeters way from the heart. From this position the probe has easy access to the heart. In contrast to transthoracic echocardiagram (through the chest wall), where the beams have to travel through skin, fat, the rib cage and the lungs before the they reach the heart. The beams from TEE in this position only have to travel a short distance to reach the heart. This shorter distance the beam has to travel increases the received echo signal from the surrounding heart tissue, resulting in a higher quality image.

The probe consists of several piezoelectric crystals that can be activated and oriented individually to generate beams in both the x-y plane and the x-z plane in a fan shaped area (Vegas, 2016). These fan shaped planes generate a pyramid shaped volume. The y-z plane can be extracted from the generated volume. Figure 2.1 shows the generated planes form the center x-y and x-z beams ((**a**) - (**b**)) and an example of the the y-z plane at $x = \frac{l_x}{2}$ where l_x is the length in x-direction. The data is then processed and stored as a 4D grayscale image (3D euclidean space plus one temporal dimension) for future use.

2.2 Image registration

Goshtasby (2005) (p. 1) defines image registration as:



(a) y-z plane **(b)** x-y plane **(c)** x-z plane

Figure 2.1: The probe transmits beams in the x and z directions creating two fan shaped planes, y-z (**a**) and x-y (**b**). This creates a pyramid shaped volume which the x-z (**c**) can be extracted from. The 2D images has been extracted from the center beam.

Definition 2.2.1. Image registration is the process of determining the point-by-point correspondence between two image of a scene.

In other words; image registration uses a *moving image* (I_M) and a *target image* (I_T), with a set of features, and produces a new image where the moving image is aligned to the target image ($I_M \approx I_T$) based on the features. In some literature the moving image and target image may be referred to as *sensed image* and *reference image* respectively (Goshtasby, 2005) or *input image* and reference image respectively (Gonzalez and Woods, 2010). The traditional image registration process can be divided into 5 steps:

1. *Preprocessing*
2. *Feature selection*[1]
3. *Feature correspondence*[1]
4. *Determination of a transformation function*
5. *Resampling*

Preprocessing is process of preparing the images for image registration. This may include include processes such as noise reduction or segmentation. Feature selection finds *keypoint* features within the target image that are of high interest and defining to the image. Examples of such keypoint features may be: lines, corners, or light sources. Feature correspondence is the process of finding the location of keypoint features within the moving image that correspond to the same set of keypoint features within the target image. Determination of a transformation function uses a *similarity measure*, such as Sum of Absolute Differences (SAD) or Sum of

[1] Only for feature-based and area-based methods.

Squared Differences (SSD) to find a *geometric spatial transformation* that maximizes the similarity measure between the corresponding features in the moving and target image. Resampling applies the geometric spatial transformation found in the previous step, as well as an *interpolation method* to reconstruct the moving image. Geometric spatial transformation and intensity interpolation will be discussed further in Section 2.3 Geometric spatial transformations.

Step 2 (feature selection) and 3 (feature correspondence) are only used for *feature-based* and *area-based* image registration methods. An example of this is the model developed by Wu et al. (2016) that uses unsupervised deep learning to predict keypoint features within medical images which are used in image registration. Hu et al. (2017) have developed a model which utilizes labeled segmented areas to train a deep learning network for image registration. Both of these models are examples of image registration methods that uses feature selection and feature correspondence. Models that does not utilize keypoint features are called *intensity-based* image registration methods. Intensity-based uses solely the intensity values of voxels when calculating the transformation between the moving image and the target image (Fitzpatric et al., 2008). Some well established similarity measures for intensity-based methods are: Correlation Coefficient (CC), NCC and Mutual Information (MI). The similarity measure we are going to focus on in this thesis is NCC.

2.2.1 NCC

Given a moving image I_M and a target image I_T, and their corresponding mean values $\bar{I_M}$ and $\bar{I_T}$, the NCC is defined as (Schers et al., 2008):

Definition 2.2.2 (NCC)**.**

$$NCC(I_M, I_T) = \frac{\sum_{i \in I_M} \left(I_M(i) - \bar{I_M}\right) \sum_{i \in I_T} \left(I_T(i) - \bar{I_T}\right)}{\sqrt{\sum_{i \in I_M} \left(I_M(i) - \bar{I_M}\right)^2 \sum_{i \in I_T} \left(I_T(i) - \bar{I_T}\right)^2}} \tag{2.1}$$

Equation (2.1) will result in a measurement of similarity in the range $[0, 1]$, where identical images will have NCC value of 1, while completely opposite images will have a NCC value of 0. Note that Equation (2.1) can be turned into a minimization problem by multiplying by -1. This result in the *negative NCC*.

A variation of Equation (2.1) for *template matching* has been developed by Sarvaiya et al. (2009). Given an image I and a template f, the NCC of the image part under the template $g = I \cap f$ is given by:

$$NCC(f, g) = \frac{n \sum fg - \sum f \sum g}{\sqrt{\left(n \sum f^2 - (\sum f)^2\right)\left(n \sum g^2 - (\sum g^2)\right)}}, \tag{2.2}$$

where n is the number of voxels in the template.

Resampling is the part where the moving image is being transformed so it approximate the target image using the transformation found earlier.

Implementation of image registration algorithms are done using an iterative minimization of a *cost function* by using a *registration optimizer* such as Stochastic Gradient Descent (SGD). A similarity measure may not always be sufficient to determine the optimal transformation due to noise (especially for ultrasound images) (Che et al., 2017). The introduction of a penalizing term that reduce the effect of unlikely deformations will improve the performance on noisy images. This term is often called the *smoothing term*. This yields a cost function of a given transformation T that can be expressed mathematically by:

$$C(T) = D(T) + \alpha S(T), \tag{2.3}$$

where C is the cost function, D is the similarity measure, S is the smoothing term and α is the a scalar value that regularizes the smoothing term. The optimal transformation that gives the greatest similarity will be the global minima of Equation (2.3).

Different image registration frameworks has been developed for ultrasound imaging of the heart. Veene et al. (2015) uses non-rigid image registration techniques to track the mitral valve for diagnosis and selection of optimal valve repair strategies. This is done by computing the displacement between the moving image and the target image using a third order B-spline interpolation. The optimal cost is then found by using Limited-Memory Broyden-Fletcher-Goldfarb-Shanno (LMBFGS) optimizer. Zagrodsky et al. (2000) proposes a method for optimal rigid transformation between a moving image and a target image with identical cardiac phase. They are using a mutual information similarity measurement together with a Nelder-Mead direct search algorithm to align two frames from the same ultrasound recording taken 250 ms apart. Woo et al. (2009) are utilizing a combination of intensity values and local phase information as a similarity measure to find the displacement vector between the moving image and the target image for 2D ultrasound images. The displacement of the moving image is done with the displacement vector $\begin{bmatrix} u & v \end{bmatrix}$ such that $I_T(x, y, t) \approx I_M(x + u, y + v, t + 1)$, where I_T is the target image and I_M is the moving image. They are using the subsequent frame in the image sequence as the moving image. The Euler-Langrange method is used as the optimization algorithm.

2.3 Geometric spatial transformations

Geometric spatial transformation is defined as modification of the spatial relationship between voxels in an image (Gonzalez and Woods, 2010). Geometric transformation of a digital image consists of 2 operations: *transformation of coordinates* and intensity interpolation. Transformation of coordinates gives voxels a new position and intensity interpolation assign a new value to the voxels based on certain criteria.

2.3.1 Spatial transformation of coordinates for 3D images

Table 2.1: Comparison of different geometric spatial transformations. The more degrees of freedom the more generalized transformation. The operations in less generalized transformation will be retain in more generalized transformation; however, the more generalized the transformation, the less properties of the image will be preserved after the transformation.

Transformation	Operations	Preserved properties	Degrees of freedom
Translation	+Translation	Distance, Angles, Parallelism	3
Rigid	+Rotation	Distance, Angles, Parallelism	6
Similarity	+Scaling	Angles, Parallelism	7
Affine	+Shearing	Parallelism	12
Perspective		Straight lines	15

Spatial transformation of coordinates consists of mapping voxel with coordinates $\begin{bmatrix} u & v & v \end{bmatrix}$ in the original image to a set of new coordinates $\begin{bmatrix} x & y & z \end{bmatrix}$ in the transformed image. The general expression for voxel-wise spatial coordinate transformation is given by:

$$\begin{bmatrix} x & y & z \end{bmatrix} = T(\begin{bmatrix} u & v & w \end{bmatrix}),$$

where T is the mapping between the original image and the transformed image.

A summary of different transformations is shown in Table 2.1. The different transformations can be sorted into two categories: *rigid* and *non-rigid* transformations. Translation and Rigid from Table 2.1 are rigid transformations and will both retain the distance between object within the image, internal angles, and parallel lines within the original image will remain parallel in the transformed image. Similarity, Affine and Perspective transformation are non-rigid transformations (meaning the spatial relationship between objects within the image are not preserved). Perspective transformation is the most general transformation with the most amount of degrees of freedom, and will only guarantee that straight lines within the original image will remain straight in the transformed image (Che et al., 2017).

Affine transformation is a type of spatial transformation of coordinates often used in image processing. Affine transformation in *homogeneous coordinates* takes the general form:

$$\begin{bmatrix} x & y & z & 1 \end{bmatrix} = \begin{bmatrix} u & v & w & 1 \end{bmatrix} \mathbf{T}, \tag{2.4}$$

where $\mathbf{T}$ is a 4×4 matrix called the *affine matrix*. In some literature Equation (2.4) are presented in terms of its transpose (Solomon and Breckon, 2011):

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \mathbf{T}^\top \begin{bmatrix} u \\ v \\ w \\ 1 \end{bmatrix}.$$

An important property of affine transformation in homogeneous coordinates is that any series of transformations represented by the matrices ($\mathbf{T}_0$, $\mathbf{T}_1$, …, $\mathbf{T}_N$) on an image can be represented by a single matrix through matrix multiplication $\mathbf{T} = \mathbf{T}_0\mathbf{T}_1 \dots \mathbf{T}_N$ for the global transformation of the image (Solomon and Breckon (2011), p. 175).

There are 5 main types of transformation with a corresponding affine matrix: identity, rotation, scaling, translation and shearing.

Identity

Identity transform is the simplest affine transformation there is. It is achieved by using the 4×4 identity matrix as affine matrix. Inserting the identity affine matrix into Equation (2.4) yields:

$$\mathbf{T}_I = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.5a}$$

$$\Rightarrow \begin{bmatrix} x & y & z & 1 \end{bmatrix} = \begin{bmatrix} u & v & w & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.5b}$$

$$\Rightarrow \begin{cases} x = u \\ y = v \\ z = w \end{cases} . \tag{2.5c}$$

From Equation (2.5c) we can see that this will result in a transformed image identical to the original image, as is expected.

Scaling

Scaling is achieved by adding scaling terms along the diagonal of the 4×4 identity matrix. Inserting the scaling affine matrix into Equation (2.4) yields:

$$\mathbf{T}_{Sc} = \begin{bmatrix} c_x & 0 & 0 & 0 \\ 0 & c_y & 0 & 0 \\ 0 & 0 & c_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.6a}$$

$$\Rightarrow \begin{bmatrix} x & y & z & 1 \end{bmatrix} = \begin{bmatrix} u & v & w & 1 \end{bmatrix} \begin{bmatrix} c_x & 0 & 0 & 0 \\ 0 & c_y & 0 & 0 \\ 0 & 0 & c_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.6b}$$

$$\Rightarrow \begin{cases} x = c_x u \\ y = c_y v \\ z = c_z w \end{cases} . \tag{2.6c}$$

This will result in a scaled image that is c_x times larger in x-direction, c_y times larger in y-direction and c_z time larger in y-direction as seen in Equation (2.6c).

Zooming is a special case of scaling where the transformed image is cropped to the same size as the original image.

Rotation

For rotation in 3D image, we first have to introduce rotation in 2D images. Clockwise rotation of a 2D image by θ radians is given by the affine matrix:

$$\mathbf{T}_R = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} . \tag{2.7}$$

For counterclockwise rotation is achieved by the transpose affine rotation matrix in Equation (2.7):

$$\begin{aligned} \mathbf{T}_R^\top &= \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}^\top \\ &= \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

Whereas rotation for 2D images rotates the image around the origin of the image, rotation for 3D images is done around each orthogonal axis separately (Goebel, 2017). In some literature this may be referred to as *simple rotation* (Egeland and Gravdahl, 2002). This results in 3 affine matrices for rotation around each of axis separately:

$$\mathbf{T}_{R_x} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta_x & \sin\theta_x & 0 \\ 0 & -\sin\theta_x & \cos\theta_x & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.9a}$$

$$\mathbf{T}_{R_y} = \begin{bmatrix} \cos\theta_y & 0 & -\sin\theta_y & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta_y & 0 & \cos\theta_y & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.9b}$$

$$\mathbf{T}_{R_z} = \begin{bmatrix} \cos\theta_z & \sin\theta_z & 0 & 0 \\ -\sin\theta_z & \cos\theta_z & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{2.9c}$$

Equation (2.9) shows each of the rotation matrices for rotation around the x-axis by θ_x radians (Equation (2.9a)), y-axis by θ_y radians (Equation (2.9b)) and z-axis by θ_z radians (Equation (2.9c)) respectively. Equation (2.9c) is the 4×4 affine rotation matrix for 3D image equivalence of Equation (2.7) for 2D images. Using the homogeneous coordinates property of affine transformations, a global rotation matrix can be found through matrix multiplication by choosing a sequence of rotations as in Arfken and Weber (2005) (p. 203)

$$\mathbf{T}_R = \mathbf{T}_{R_z}\mathbf{T}_{R_y}\mathbf{T}_{R_x} \tag{2.10a}$$

$$= \begin{bmatrix} \cos\theta_z & \sin\theta_z & 0 & 0 \\ -\sin\theta_z & \cos\theta_z & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta_y & 0 & -\sin\theta_y & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta_y & 0 & \cos\theta_y & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta_x & \sin\theta_x & 0 \\ 0 & -\sin\theta_x & \cos\theta_x & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.10b}$$

$$= \begin{bmatrix} \cos\theta_z\cos\theta_y & \cos\theta_z\sin\theta_y\sin\theta_x + \sin\theta_z\cos\theta_x & \sin\theta_z\sin\theta_x - \cos\theta_z\sin\theta_y\cos\theta_x & 0 \\ -\sin\theta_z\cos\theta_y & \cos\theta_z\cos\theta_x - \sin\theta_z\sin\theta_y\sin\theta_x & \cos\theta_z\sin\theta_x + \sin\theta_z\sin\theta_y\cos\theta_x & 0 \\ \sin\theta_y & -\cos\theta_y\sin\theta_x & \cos\theta_z\cos\theta_y & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{2.10c}$$

The order of multiplication in Equation (2.10a) determines the order of rotation. In this case; the image will rotate around the z-axis first, then around the y-axis, and finally around the x-axis.

Inserting Equation (2.10) into Equation (2.4) yields:

$$\begin{bmatrix} x & y & Z & 1 \end{bmatrix} = \begin{bmatrix} u & v & w & 1 \end{bmatrix} \mathbf{T} \tag{2.11a}$$

$$\Rightarrow \begin{cases} x = v(\cos\theta_z \cos\theta_y) - u(\sin\theta_z \cos\theta_y) + w \sin\theta_y \\ y = v(\cos\theta_z \sin\theta_y \sin\theta_x + \sin\theta_z \cos\theta_x) \\ \quad + u(\cos\theta_z \sin\theta_x + \sin\theta_z \sin\theta_y \cos\theta_x) \\ \quad - w(\cos\theta_y \sin\theta_x) \\ z = u(\sin\theta_z \sin\theta_x - \cos\theta_z \sin\theta_y \cos\theta_x) \\ \quad + v(\cos\theta_z \sin\theta_x + \sin\theta_z \sin\theta_y \cos\theta_x) \\ \quad + w(\cos\theta_z \cos\theta_y) \end{cases} . \tag{2.11b}$$

Translation

The translation affine matrix is achieved by adding translation factors to the bottom row of the identity matrix. Inserting the translation affine matrix into Equation (2.4) yields:

$$\mathbf{T}_T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ t_x & t_y & t_z & 1 \end{bmatrix} \tag{2.12a}$$

$$\Rightarrow \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} u & v & w & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ t_x & t_y & t_z & 1 \end{bmatrix} \tag{2.12b}$$

$$\Rightarrow \begin{cases} x = u + t_x \\ y = v + t_y \\ z = w + t_z \end{cases} . \tag{2.12c}$$

The resulting transformed image will be a copy of the original image where the origin of the image has been translated t_x voxels in the x-direction, t_y voxels in the y-direction and t_z voxels in the z-direction as seen in Equation (2.12c).

Shear

As for rotation, shearing on 3D images is done by fixating each of the orthogonal axis and preforming a 2D shearing on the image for each fixated axis. This gives us 3 affine matrices for shearing for each fixated axis.

$$\mathbf{T}_{Sh_x} = \begin{bmatrix} 1 & s_y & s_z & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.13a}$$

$$\mathbf{T}_{Sh_y} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ s_x & 1 & s_z & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.13b}$$

$$\mathbf{T}_{Sh_z} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ s_x & s_y & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{2.13c}$$

As seen in Equation (2.13), shearing is achieved by adding shearing factos to the off-diagonal elements of the identity matrix. Inserting the global shear affine matrix[2] into Equation (2.4) results in yields:

$$\mathbf{T}_{Sh} = \begin{bmatrix} 1 & s_y^x & s_z^x & 0 \\ s_x^y & 1 & s_z^y & 0 \\ s_x^z & s_y^z & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.14a}$$

$$\Rightarrow \begin{bmatrix} x & y & z & 1 \end{bmatrix} = \begin{bmatrix} u & v & w & 1 \end{bmatrix} \begin{bmatrix} 1 & s_y^x & s_z^x & 0 \\ s_x^y & 1 & s_z^y & 0 \\ s_x^z & s_y^z & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.14b}$$

$$\Rightarrow \begin{cases} x = u + s_x^y v + s_x^z w \\ y = s_y^x u + v + s_y^z w \\ z = s_z^x u + s_z^y v + w \end{cases}. \tag{2.14c}$$

By using the homogeneous coordinate property of affine transformation it is possible to find a global transformation matrix that represent the series of transformations: scaling, rotation around z-axis, rotation around y-axis, rotation around x-axis, translation and shearing, pre-

[2]The upper case notations indicates the fixed axis, and the lower case notation indicates the axis affected by the transformation.

formed on the original image through matrix multiplication:

$$\mathbf{T} = \mathbf{T}_{Sc}\mathbf{T}_{R_z}\mathbf{T}_{R_y}\mathbf{T}_{R_z}\mathbf{T}_T\mathbf{T}_{Sh} \tag{2.15a}$$

$$= \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} & 0 \\ \theta_{21} & \theta_{22} & \theta_{23} & 0 \\ \theta_{31} & \theta_{32} & \theta_{33} & 0 \\ \theta_{41} & \theta_{42} & \theta_{43} & 1 \end{bmatrix} \tag{2.15b}$$

2.3.2 Intensity interpolation

Consider the example where we wish to scale an image by a factor of 2 ($c_x = c_y = c_z = 2$). From Equation (2.6) we can see that voxel $[0, 0, 0]$ will have the coordinates $[0, 0, 0]$ in the transformed image, and voxel $[1, 0, 0]$ will have the coordinates $[2, 0, 0]$. This leaves a gap at $[1, 0, 0]$ with a unknown voxel value. This unknown voxel value is determined through the process called intensity interpolation. Gonzalez and Woods (2010) (p. 87) defines interpolation as:

Definition 2.3.1. Interpolation is the process of using known data to estimate the values at unknown locations.

The simplest solution to this would be to pick the value to be the same as the closest known voxel. This method is known as *nearest neighbor interpolation*. Nearest neighbor interpolation has the advantage of being a simple and fast interpolation method. However, the resulting image will be have a distortion of the transition between colors in the image, creating a more pixelated image.

(a) Nearest neighbor interpolation **(b)** Bilinear interpolation

Figure 2.2: 30 degrees rotated image with different interpolation methods. (**a**) has been rotated using nearest neighbor interpolation, while (**b**) has rotated using bilinear interpolation. The left part of (**a**) and (**b**) shows a enhanced section of the rotated image. From this enhanced section we can clearly see that (**a**) has a greater distortion of the edges between different colors creating a more pixelated image, while (**b**) has a more smooth transition between colors.

Figure 2.2 shows a rotated 2D image which has been interpolated using different interpolation methods. The rotation was done using the affine matrix from Equation (2.7) with $\theta = \pi/6 = 30°$. (**a**) has been interpolated using nearest neighbor interpolation, and (**b**) has been interpolated using bilinear interpolation (2D equivalent of trilinear interpolation). From the figure it is clear that bilinear interpolation creates a smoother transition between colors, while nearest neighbor interpolation creates a more distorted and pixelated image.

Trilinear interpolation

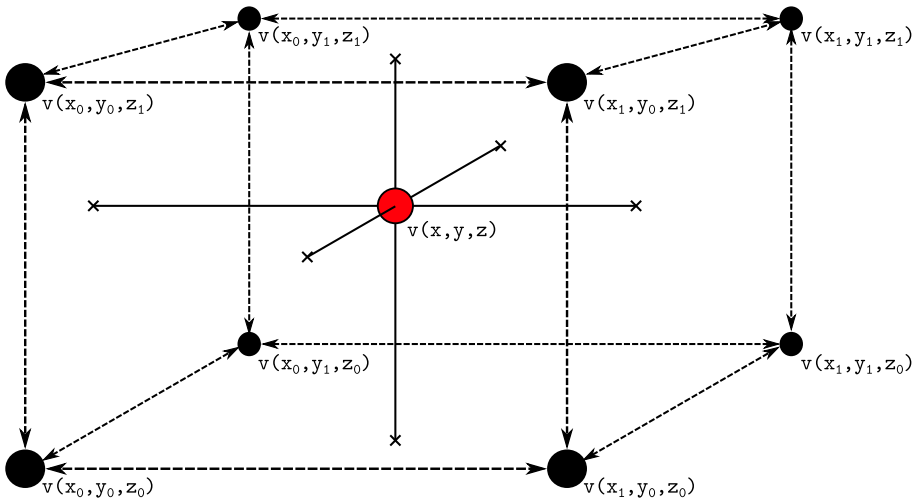


Figure 2.3: Trilinear interpolation uses the 8 closest voxels: $v(x_0, y_0, z_0)$, $v(x_1, y_0, z_0)$, $v(x_0, y_1, z_0)$, $v(x_0, y_0, z_1)$, $v(x_1, y_1, z_0)$, $v(x_1, y_0, z_1)$, $v(x_0, y_1, z_1)$ and $v(x_1, y_1, z_1)$ to calculate the unknown voxel at position $\begin{bmatrix} x & y \end{bmatrix}$. The intensity value $v(x, y, z)$ of the voxel is found by Equation (2.16).

Trilinear interpolation uses the 8 nearest corner to compute the value at the desired, unknown voxel. Let $\begin{bmatrix} x & y & z \end{bmatrix}$ be the location of the unknown voxel and $v(x_0, y_0, z_0)$, $v(x_1, y_0, z_0)$, $v(x_0, y_1, z_0)$, $v(x_0, y_0, z_1)$, $v(x_1, y_1, z_0)$, $v(x_1, y_0, z_1)$, $v(x_0, y_1, z_1)$ and $v(x_1, y_1, z_1)$ be the values of the 8 nearby corners with known intensity values as in Figure 2.3. $v(x, y, z)$ is then determined by the equation:

$$v(x, y, z) = ax + by + cz + dxy + exz + fyz + gxyz + h, \tag{2.16}$$

where the coefficients are found by solving:

$$\begin{bmatrix} x_0 & y_0 & z_0 & x_0y_0 & x_0z_0 & y_0z_0 & x_0y_0z_0 & 1 \\ x_1 & y_0 & z_0 & x_1y_0 & x_1z_0 & y_0z_0 & x_1y_0z_0 & 1 \\ x_0 & y_1 & z_0 & x_0y_1 & x_0z_0 & y_1z_0 & x_0y_1z_0 & 1 \\ x_1 & y_1 & z_0 & x_1y_1 & x_1z_0 & y_1z_0 & x_1y_1z_0 & 1 \\ x_0 & y_0 & z_1 & x_0y_0 & x_0z_1 & y_0z_1 & x_0y_0z_1 & 1 \\ x_1 & y_0 & z_1 & x_1y_0 & x_1z_1 & y_0z_1 & x_1y_0z_1 & 1 \\ x_0 & y_1 & z_1 & x_0y_1 & x_0z_1 & y_1z_1 & x_0y_1z_1 & 1 \\ x_1 & y_1 & z_1 & x_1y_1 & x_1z_1 & y_1z_1 & x_1y_1z_1 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{bmatrix} = \begin{bmatrix} v(x_0, y_0, z_0) \\ v(x_1, y_0, z_0) \\ v(x_0, y_1, z_0) \\ v(x_1, y_1, z_0) \\ v(x_0, y_0, z_1) \\ v(x_1, y_0, z_1) \\ v(x_0, y_1, z_1) \\ v(x_1, y_1, z_1) \end{bmatrix}. \tag{2.17}$$

An alternative method to Equation (2.16) can be found by assuming that the cube formed by the 8 surrounding corners (as in Figure 2.3) is a *unit cube*. Using this, the trilinear interpolation method can be simplified to 2 bilinear interpolations and 1 *linear interpolation*. Rajon and Bolch (2003) shows how this is done by first doing a bilinear interpolation on the x-y plane, followed by a new bilinear interpolation on the x-z plane. This results in 2 points in the y-z plane, which can be used to find the intensity value of the desired voxel through *linear interpolation*. These 3 steps: 2 bilinear interpolations and 1 linear interpolation, gives rise to the new equation for trilinear interpolation.

$$\begin{aligned} v(x, y, z) = & v(x_0, y_0, z_0)(1 - y)(1 - y)(1 - z) \\ & + v(x_1, y_0, z_0)x(1 - y)(1 - z) \\ & + v(x_0, y_1, z_0)(1 - x)y(1 - z) \\ & + v(x_0, y_0, z_1)(1 - x)(1 - y)z \\ & + v(x_1, y_0, y_1)x(1 - y)z \\ & + v(x_0, y_1, z_1)(1 - x)yz \\ & + v(x_1, y_1, z_0)xy(1 - z) \\ & + v(x_1, y_1, z_1)xyz \end{aligned}. \tag{2.18}$$

2.4 Computer vision and deep learning

This section is cited and adapted from Johannessen (2018).

Computer vision is task of having a computer retrieve more information from pictures or videos beyond just pixel values. The interesting information we want the computer to know about is often the content of the image such as objects and their position within the image, but also things like illumination, color distributions and key feature points are of interest. Based on the information retrieved, we can do further analysis such as parameter estimation (position within the real world, motion or pose). There are many machine learning methods a computer can use to retrieve the desired information, but here we will discuss the *deep learning* approach

which has gained huge popularity in the later years. Goodfellow et al. (2016) (chapter 1 p. 5) describes deep learning as:

> Deep learning enables the computer to build complex concepts out of simpler concepts. [...] a deep learning system can represent the concept of an image of a person by combining simpler concepts, such as corners and contours which are in turn defined in terms of edges.

This means that in deep learning you break down the image into simpler features such as edges and corners, and teach the computer to recognize these features and what combination of features (edges, corners) that the object consists of. Teaching the computer these features and combination of features can be done through a *Convolutional Neural Network* (CNN). A CNN is a special kind of artificial neural network which works especially well on images (Nielsen, 2015). Instead of using every pixel in the image as input, as you would in a traditional Fully Connected Neural Network (FCNN), you are using a *local receptive field*. This is a small window inside the image that you search for features within. The same feature is searched for in the entire image by sliding the window (sliding window approach) across the image. This process is repeated for each feature. Finding more specific features is done by doing a new pass with a local receptive field, but this time using the generated feature map as input. Adding a FCNN at the end of the CNN will give the network classification capabilities by knowing combinations of features that make up the object in question.

Some important terms used in deep learning and CNN are:

Feature: A feature is pattern of pixels that can be used to defines an object. This could be an edge, a corner, an arch or some other kind of shape. Searching for features in a CNN creates a feature map which is tell the computer whether or not that feature is present in the local receptive field[3].

Weights and bias: Weights, w_i, and bias, b, are numerical values representing the importance of specific inputs, x_i when computing the output, z.

$$z = \sum_i (w_i \cdot x_i) + b \tag{2.19}$$

The weights and biases are the numerical values updated during network training.

Kernel: Kernel (or filter) is a combination of shared weights represented as a matrix. In a CNN the feature map is computed by a convolution[4] between the kernel and the local receptive

[3]This is rarely representing as a binary true or false value, but rather as a probability that the given feature is present within the local receptive field.

[4]Convolution is a specialized kind of linear operator:

$$(K * I)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n)$$

where I is the input image and K is the kernel (Goodfellow et al. (2016) p. 328)

field (hence the name "convolutional neural network").

Stride: Stride is the shift length of the local receptive field between two convolutions. A stride of two means the local receptive field shifts two pixels for each convolution

Padding: Padding creates a border around the image with arbitrary pixel values (often zero, i.e. zero-padding). Consider a starting image with size (m, n), padding of two creates a new image with size $(m + 4, n + 4)$ (2 pixels on each side). Padding and stride length determines the size of the output feature map.

Pooling: Pooling layers reduce the size of the output layer of a convolutional layer. This can be done by for example picking the largest value of a (2×2) window, effectively reducing the size of the layer by half. This is known as max-pooling.

Activation function: The result of Equation (2.19) can be anything in the range $(-\inf, \inf)$. The goal of the activation function is to bring range of possible output down to a more manageable range. Some popular activation functions are:

- Step function:
$$f(x) = \begin{cases} 1, & x \geq \text{threshold} \\ 0, & \text{otherwise} \end{cases} \in [0, 1]$$
- Sigmoid function:
$$f(x) = \frac{1}{1 + e^{-x}} \in [0, 1]$$
- ReLU
$$f(x) = \max(0, x) \in [0, max(x)]$$

,

2.4.1 Backpropagation

Backpropagation is the name of the algorithm used in deep learning to produce the gradients of the cost function C. The cost function is a quantifiable measurement of how well our system is preforming as discussed in Section 2.2 Image registration. The goal of the backpropagation algorithm is to produce the gradients of the cost function with respect to the wights and biases for every layer (L) in the network.

$$\frac{\partial C}{\partial w^{(L)}}, \frac{\partial C}{\partial b^{(L)}}$$

Given a cost function $C(a, y)$ as a function of the output of a neuron, $a = f(z)$, and some desired value, y, the gradient with respect to the weights and bias of the layer L can be determined

from the chain rule:

$$\frac{\partial C}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C}{\partial a^{(L)}} \tag{2.20a}$$

$$\frac{\partial C}{\partial b^{(L)}} = \frac{\partial z^{(L)}}{\partial b^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C}{\partial a^{(L)}} \tag{2.20b}$$

The new partial derivatives introduced in Equation (2.20) can be summarized as follows:

- $\frac{\partial z^{(L)}}{\partial w^{(L)}}$ and $\frac{\partial z^{(L)}}{\partial b^{(L)}}$ are the derivative of output neuron with the respect to the weights and biases respectively. These can be found by taking the partial derivative of Equation (2.19) with input being the output of the neuron from the previous layer, $x = a^{(L-1)}$:

$$\frac{\partial z^{(L)}}{\partial w^{(L)}} = a^{(L-1)} \tag{2.21a}$$

$$\frac{\partial z^{(L)}}{\partial b^{(L)}} = 1 \tag{2.21b}$$

- $\frac{\partial a^{(L)}}{\partial z^{(L)}}$ is identical for both equations in Equation (2.20) and is simply the derivative of the chosen activation function of the neuron output $f(z^{(L)})$.

$$\frac{\partial a^{(L)}}{\partial z^{(L)}} = f'(z^{(L)}) \tag{2.22}$$

- $\frac{\partial C}{\partial a^{(L)}}$ is also identical for both equation in Equation (2.20). Given the cost function as a function of the output of the final neuron in the network and the ground truth $C(a^{(L)}, y)$, the derivative is simply the derivative of the chosen cost function:

$$\frac{\partial C}{\partial a^{(L)}} = C'(a^{(L)}, y) \tag{2.23}$$

Computing the partial derivatives of the cost function with respect to the weights and biases for each layer (L) in a backwards manner (hence the name "backpropagation") results in the gradient of the cost function, ∇C. The gradient are then used to update the weights and biases of the network through a forward pass using an optimization algorithm such as SGD.

2.5 Image registration with deep learning

Jaderberg et al. (2016) proposes a method for geometric spatial transformation based on a deep learning network for the classical example of recognizing handwritten digits. The system has been implemented both for 2D and 3D images. They are using a combination of a *localization network*, a *parameterized sampling grid* and *image sampling* to orient the skewed handwritten digits to an upraised position before passing it though a second deep learning network for

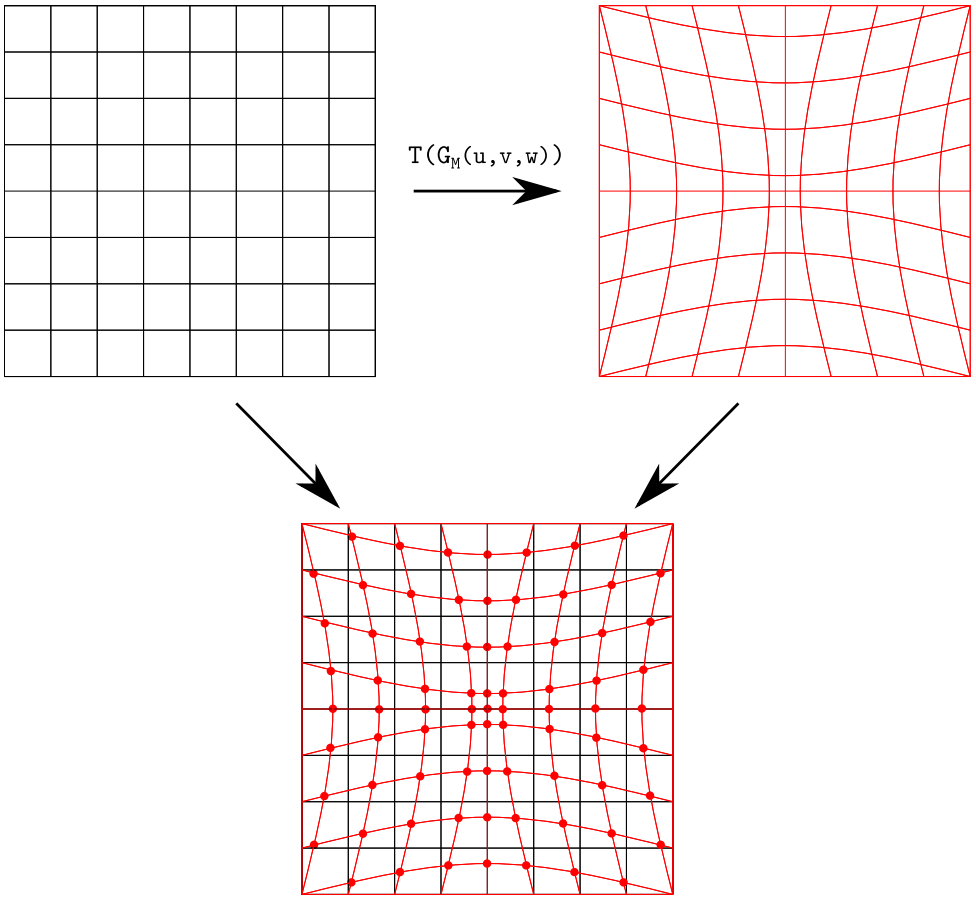


Figure 2.4: A parameterized grid $G_T(x, y, z)$ is generated by applying the affine transformation from Equation (2.4) on a grid corresponding to the original image $G_M(u, v, w)$ with an arbitrary transformation. Aligning the 2 grids upon each other illustrates how the image sampler uses the original grid and the and the intensity values from the original image when determining the intensity values of the transformed image.

classification. The localization network input is an image which has been deformed in various ways (translated, rotated, scaled, sheared and added noise). The output of the localization network is the global affine transformation matrix from Equation (2.15). The parametrized sampling grid uses the affine transformation matrix from the localization network to produce a gird which represent the new spatial transformation of coordinates. Consider a grid $G_M(u, v, w)$ corresponding to the original image with normalized coordinates such that $-1 \leq u, v, w \leq 1$. A parameterized grid $G_T(x, y, z)$ is now generated by applying the affine transformation from Equation (2.4) on $G_M(u, v, w)$:

$$G_T(x, y, z) = T\big(G_M(x, y, z)\big) = \begin{bmatrix} u & v & w & 1 \end{bmatrix} \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} & 0 \\ \theta_{21} & \theta_{22} & \theta_{23} & 0 \\ \theta_{31} & \theta_{32} & \theta_{33} & 0 \\ \theta_{41} & \theta_{42} & \theta_{43} & 1 \end{bmatrix}. \tag{2.24}$$

This process is illustrated in the top part of Figure 2.4 with an arbitrarily transformation. The

image sampling process interpolates each point in $G_T(x, y, z)$ using a interpolation sampling kernel $k(*)$ (i.e. trilinear interpolation), using the position of the point relative to $G_M(u, v, w)$, and the intensity values of the original image. Intuitively, this can be viewed as placing original grid $G_M(u, v, w)$ upon the or parameterized grid $G_T(x, y, z)$ and interpolation each point in the grid using the values in $G_M(u, v, w)$ (bottom part of Figure 2.4). Equation (2.25) shows the mathematical expression for this process:

$$V_i = \sum_n^H \sum_m^W \sum_l^D U_{nml} k(u_i - m) k(v_i - n) k(w_i - l) \; \forall i \in [1, \ldots, HWD], \tag{2.25}$$

where U is the original image, V is the output image and H, W, D is the height, width and depth of the original image respectively. Note that the kernel in Equation (2.25) as been separated into three terms, one for each axis, instead of one interpolation function as in Equation (2.16).

Since the image registration part is placed after predicting the affine transformation matrix and before backpropagation, the gradients of transformed image has to be computed in order to be able to do the backpropagation algorithm during training. Partial derivation of Equation (2.25) with respect to U_{nml}, u_i, v_i and w_i yields:

$$\frac{\partial V_i}{\partial U_{nml}} = \sum_n^H \sum_m^W \sum_l^D k(u_i - m) k(v_i - n) k(w_i - l) \tag{2.26a}$$

$$\frac{\partial V_i}{\partial u_i} = \sum_n^H \sum_m^W \sum_l^D U_{nml} k(v_i - n) k(w_i - l) \tag{2.26b}$$

$$\frac{\partial V_i}{\partial v_i} = \sum_n^H \sum_m^W \sum_l^D U_{nml} k(u_i - m) k(w_i - l) \tag{2.26c}$$

$$\frac{\partial V_i}{\partial w_i} = \sum_n^H \sum_m^W \sum_l^D U_{nml} k(u_i - m) k(v_i - n) \tag{2.26d}$$

for a trilinear kernel as shown by Jaderberg et al. (2016) (p. 5, modified for 3D). The image registration with deep learning training process can be summarized as:

1. Input training example
2. Feedforward through the network.
3. Preform image registration[5].
4. Compute similarity measure.
5. Backpropagate the similarity measure through the network.

[5]New addition to the general training algorithm from Nielsen (2015) (chapter 2, section "*The backpropagation algorithm*") caused by image registration.

6. Update the weights and biases of the network using a optimization algorithm.

There has some previous work done on medical image registration utilizing unsupervised deep learning, but little work has been done on deep learning based registration of ultrasound images. Most of the work has been done on multimodal MR/CT imaging. Yang et al. (2017) have developed a system for image image registration of MR images called "*Quicksilver*"[6]. In their method they are using a deep learning approach to predict the initial momentum m_0 used in the Large Deformation Diffeomorphic Metric Mapping (LDDMM) algorithm for 3D image registration developed by Vialard et al. (2012). LDDMM uses a series of small deformations to produce a larger deformation between the moving image and the target image initialized by the initial momentum parameter. As ground truth the system uses the initial momentum generated by the traditional LDDMM image registration method without deep learning. Another model for image registration proposed by de Voss et al. (2019) uses two deep learning network in series when preforming the image registration. The purpose of the first network is to align the two image using affine transformation. The output of the neural network is the optimal affine matrix as in Equation (2.15). The second part of the system is a neural network for predicting deformations. It uses the aligned images form the previous step to predict a 3D displacement vector, which is used to transform the moving image according such that $I_T \approx I_M(x+u, y+v, z+w)$. As similarity measure for the displacement image registration they are using NCC in combination with a bending energy penalty. Both networks uses the ADAM optimization algorithm.

[6] `https://github.com/rkwitt/quicksilver`

CHAPTER 3

Method

Two models are being proposed in this thesis. The first model is based on the affine transformation framework by de Voss et al. (2019), and is using the entire moving and target image as input to the deep learning network. The second model is based upon the framework proposed by Yang et al. (2017), where the moving and target images are divided into patches of uniform size before being passed through the network.

Both models can be divided into 4 steps: Pick a target recording from a group of moving recordings, load the images and preform preprocessing, pass the moving and target images through the deep learning network and preform the affine transformation, and store the transformed moving images in a separate file. A set of recordings is here referred to as a set of files containing multiple ultrasound images. A set of images is referred to as the ultrasound images within one file. A single selected target recording is used as reference target images. This is done so that the system will align the moving recordings to the target recordings, and use it as a ground truth. For each recording in the set of moving recordings, the moving and target images are loaded into memory and the preprocessing is executed. Now the system iterates through every ultrasound image in the loaded set of moving images, and passes one moving image and one target image through the CNN to produce a transformation matrix, which in turn is used to align the moving and target image. The sequence of transformed moving image is then stored using the same file format as the original recordings together with the total execution time and the loss of the transformation for each image. The flow of information through the framework is illustrated in Figure 4.1.

3.1 Data

The data used in this project contains 123 3D TEE, anonymized 3D ultrasound images of the left ventricle. The images are are a part of a larger set of recordings collected from 13 patients which all underwent either coronary artery bypass surgery or a routine checkup at the echocardiography lab at St. Olavs hospital. There are 3D recordings of the left ventricle from different views as the 3D probe was sightly rotated or translated between each recording. This was done in order to simulate probe movement. At least 4 3D recordings were acquired for each patient. The ultrasound scanners used were Vivid S70 and Vivid E9 from GE Vingmed in Horten. A 3D TEE probe 6VT-D was used to acquire the images. These images are recorded over 3 to 5 heart cycles. The recordings were not subjected to any form of selection process. The ground truth for the clinical datasets was landmark based manual registration from a physician.

Table 3.1: Information about recordings done for 1 patients. Each recording has its own file containing multiple 3D ultrasound image. The resolution, voxel size and origin are all information about the images stored in the file together with in images themselves. The resolution is given in pixels while voxel size and origin is given in mm.

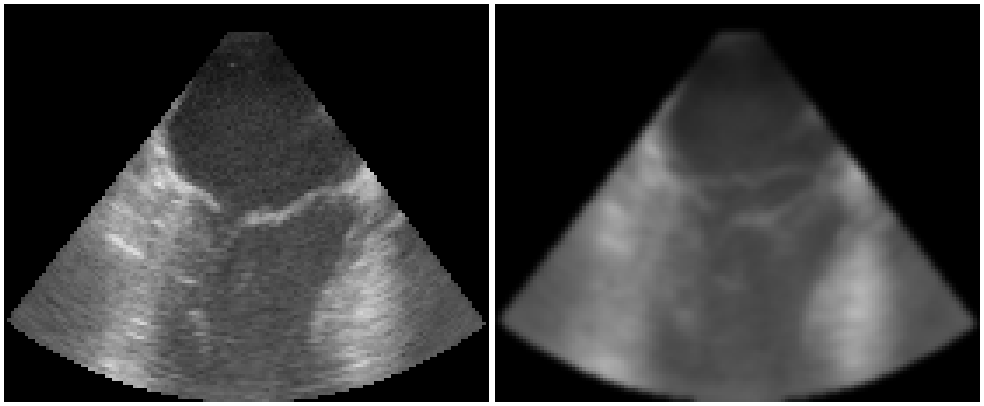
Recording	Frames	Resolution [px]	Voxel size [mm]	Origin [mm]
1	5	$222 \times 157 \times 222$	$0.7 \times 0.7 \times 0.7$	$-77.8 \times 0 \times 77.8$
2	5	$222 \times 157 \times 222$	$0.7 \times 0.7 \times 0.7$	$-77.8 \times 0 \times 77.8$
3	5	$222 \times 157 \times 222$	$0.7 \times 0.7 \times 0.7$	$-77.8 \times 0 \times 77.8$
4	46	$97 \times 114 \times 97$	$0.7 \times 0.7 \times 0.7$	$-33.8 \times 0 \times 33.8$
5	31	$69 \times 114 \times 69$	$0.7 \times 0.7 \times 0.7$	$-24.1 \times 0 \times 24.1$
6	29	$69 \times 114 \times 69$	$0.7 \times 0.7 \times 0.7$	$-24.1 \times 0 \times 24.1$

The recordings are stored as *HDF5*[1] files. Each file contains multiple 3D ultrasound images from different points during the heart cycle. In addition, the files also contain information about the geometry of the images such as number of frames, image resolution, as well as voxel size and origin converted to real world coordinates. A summary of information for one patient can be seen in Table 3.1. The implementation of image loading is presented in Section 4.1 Loading data and preprocess.

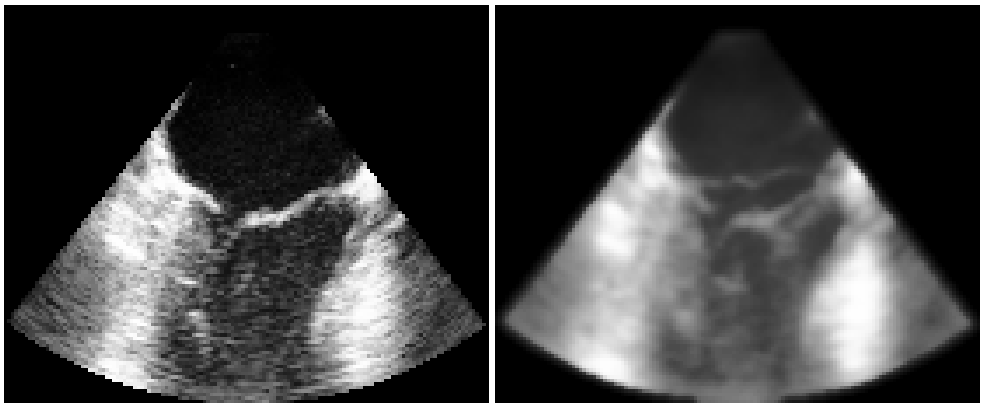
3.1.1 Preprocess

Preprocessing of the ultrasound images in this framework involves *Gaussian blur* and *histogram equalization*. How the Gaussian blur and histogram equalization are implemented can be seen in Section 4.1 Loading data and preprocess.

[1] `https://portal.hdfgroup.org/display/HDF5/HDF5`



(a) Original image **(b)** Gaussian blur



(c) Histogram equalization **(d)** Gaussian blur and histogram equalization

Figure 3.1: Comparison of images before and after preprocessing. (**a**) is the original, unfiltered image. In (**b**) a Gaussian blur filter has been applied to the original image with $\sigma = 1$. The goal of the Gaussian blur is to reduce noise within the image. In (**c**) a histogram equalization has been preformed on the image. By doing a histogram equalization you get higher difference between high and low values within the image. (**b**) show the final image after both Gaussian blur and histogram equalization has been applied. By doing both these preprocessing steps, the aim is to reduce noise and at the same time preserve edges within the image.

Gaussian blur

Gaussian blur is done using kernel convolution. A kernel of size $N \times N \times N$ is determined from the Gaussian function:

$$f(x, y, z) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2+z^2}{2\sigma^2}},$$

where σ is the standard deviation of the Gaussian function (Solomon and Breckon (2011) p. 95). The kernel convolution is then done by sliding a window of size $N \times N \times N$ over the image and computing the convolution between the window and the kernel.

Images generated from ultrasound recording are susceptible to a large amount of noise such as speckle noise. The following about speckle noise is cited from Johannessen (2018). Speckle noise phenomenon which appears when a signal hits a rough surface (rough relative to the

wavelength) (Boncelet, 2005). The roughness gives the returning signal a random variation in phase and amplitude, and in some voxels this will result in a heavily amplified or weakened signal. Speckle noise appears as bright or dark voxels within the ultrasound image. This is especially true around the edges (of the actual ultrasound image and not the black boarder around the image), which are, for the most part, uninteresting areas. Applying a Gaussian blur filter is done in an effort to reduce the effect noise will have on the later in the system. However, by applying a Gaussian blur to the image, will also result in less defined edges. Figure 3.1 (**b**) shows image (**a**) where a Gaussian blur filter has been applied.

Histogram equalization

Histogram equalization aims at enhancing contrasts within the image by adjusting the intensity levels. This can be seen in Figure 3.1 (**c**) where histogram equalization has been applied to the original image (**a**). This shows that after the histogram equalization the dark areas of the image has had their intensity values pushed further down, while the light areas has had their intensity values enhanced. Histogram equalization is done by treating the histogram of the image as the Probability Density Function (PDF) $p(X_k)$, where X_k is a specific intensity level of the image ($k = [0, \ldots, 255]$ for grayscale images) (Kim, 1997). The Cumulative Density Function (CDF) can then be computed from the PDF:

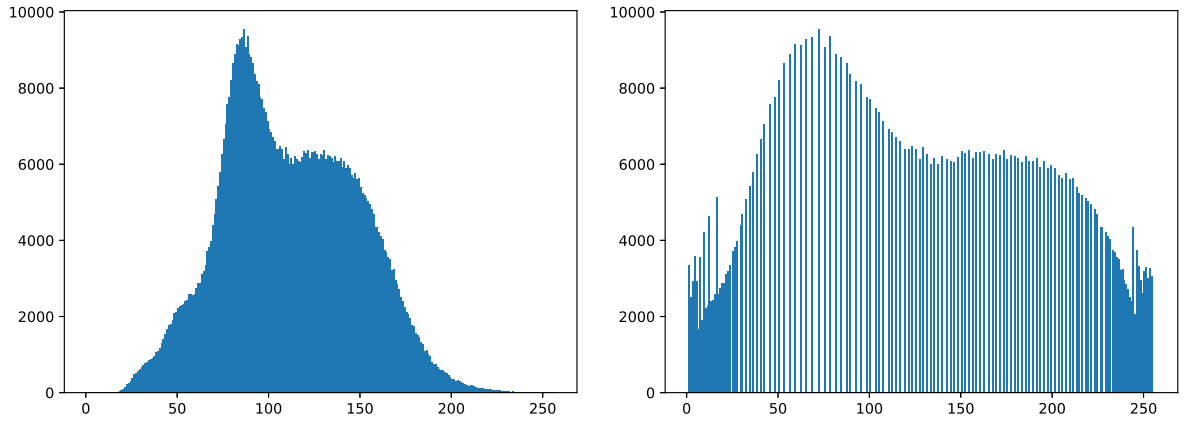
$$CDF(x) = \sum_{j=0}^{k} p(X_j).$$

By using the CDF as a transformation function $T(x_k) : I_M(x_k) \rightarrow I_T(x_k)$ for $k = [0, \ldots, 255]$, the moving image I_M will be transformed into a target image I_T which has a histogram using the entire dynamic range $[X_0, \ldots, X_{255}]$ as can be seen in Figure 3.2. The transformation which achieves this map is given by:

$$T(x) = X_0 + (X_{255} - X_0)CDF(x).$$

Figure 3.2 shows the histogram of the images given in Figure 3.1 (**a**) and (**c**). (**a**) in Figure 3.2 correspond to the histogram of the original image ((**a**) in Figure 3.1), and (**b**) in Figure 3.2 correspond to the histogram of the image after histogram equalization has been applied ((**c**) in Figure 3.1).

For the patch based model presented in this thesis, a third preprocessing technique is used. This preprocess involves dividing the moving and target images into patches and remove entire black patches. In ultrasound, the content of the image is being presented as a cone. A black boarder is then placed around the cone to make it into a square image (cube volume). For a patch based model these black patches can be removed from the set of patches by looking at the



(a) Original image histogram

(b) Histogram after histogram equalization

Figure 3.2: Histograms of the image in Figure 3.1 before and after histogram equalization has been applied. (**a**) correspond to the histogram of the original image (**a**) in Figure 3.1, and (**b**) correspond to the histogram of the image after histogram equalization has been applied (**c**) in Figure 3.1.

sum of intensity values within the patch. If the sum of intensity values within the patch are 0, then the entire patch consists just of the black boarder and back be removed from consideration. This is summarized in Algorithm 1. The first step is to divide the image, I, into patches which is stored in a flattened array. Then we iterate through every patch and calculate the intensity sum. If the sum is not 0 then we know that the patch has a part of the cone inside it and we give that patch the value 1 in the a masking array. Finally a masked selection of the patches are preformed. This result in only the patches containing the part of the ultrasound cone is left.

3.2 Deep learning network

Both models presented in this thesis uses a deep learning network to predict the 12 parameters of the transformation matrix $\mathbf{T}$ from Equation (2.15). The networks consist of a CNN connected to a FCNN. The CNN has a parallel structure as in de Voss et al. (2019) and Yang et al. (2017), one pipeline for the moving image and one for the target image. Both of the CNN for the moving image and the target images pipelines are identical. The output from the last convolutional layer of both the moving image pipeline and the target image pipeline concentrated into one tensor which is used an input to the FCNN. Both the full image based model and the patch based model uses this network structure, but the layer details are different. The weights and bias of the final layer was initialized so that the output of the network at first training iteration is the identity transformation as in Equation (2.5a). As a similarity measure, both networks are using the negative NCC.

Algorithm 1 Remove black patches from the image I

```
I: Input image
p: Patch size
s: Step size
procedure REMOVEBLACKPATCHES(I, p, s)
    I ← Divide I into patches based on p and s
    n_patches ← Number of patches in I
    flatIdx ← Initialize array of size (n_patches, p, p, p)
    i ← 0
    for all patch ∈ I do
        flateIdx(i) ← patch
        i ← i + 1
    end for
    idxSelect ← Initialize zero array of size (n_patches)
    for all i ∈ idxSelect do
        if ∑ flatIdx(i) is not 0 then
            idxSelect(i) ← 1
        end if
    end for
    I ← Preform masked selection of flatIdx based on idxSelect
end procedure
```

Table 3.2: CNN structure.

Layer	Input	Output	Kernel	Stride	Pooling	Activation
1	1	32	$5 \times 5 \times 5$	1	Avg.	ReLU
2	32	32	$5 \times 5 \times 5$	1	Avg.	ReLU
3	32	64	$5 \times 5 \times 5$	1	Avg.	ReLU
4	64	64	$3 \times 3 \times 3$	1	Avg.	ReLU
5	64	64	$3 \times 3 \times 3$	1	Global Avg.	-

3.2.1 Full image network

The full image based network model consists of 5 convolutional layers with a 4 average pooling layers in between each convolutional layer. For the final convolutional layer, a global average pooling layer is being used. The reasoning behind this is to make the output of the final CNN a tensor of shape $features \times 1 \times 1 \times 1$. This is desired because in medical imaging a common practise is to take multiple recordings with different beam angles and depths to make get a good overview. This result in a large variety of image sizes, and the global average pooling layer gives a uniform output regardless of the input without resulting to padding. A Rectifier Linear Unit (ReLU)[2] activation function is used as activation function for the first 4 convolutional layers. The specifications for each layer in the CNN network structure is given in Table 3.2. Using the information from Table 3.2 the input of the first fully connected layer becomes 256 (2 times 128

[2] $ReLU(x) = \max(0, x)$

Table 3.3: FCNN structure.

Layer	Input	Output	Activation
6	128	64	ReLU
7	64	32	ReLU
8	32	12	-

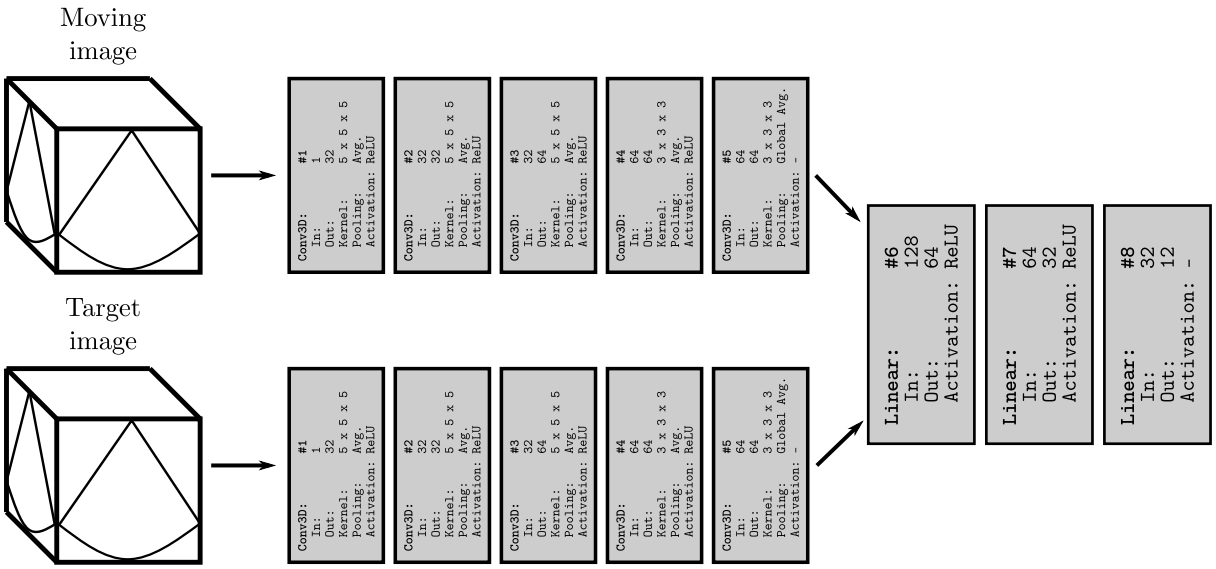


Figure 3.3: Full image based deep learning network. Two pipelines, one for each of the moving image and the target image, consisting of 5 CNN. The output from the final convolutional layer is used as the input to the FCNN. The FCNN consists of 3 linear layers where the output of the final layer is the 12 parameters of the affine transformation matrix Equation (2.15). The details from each layer can be seen in Table 3.2 and Table 3.3.

because of the concentration of the moving image and target image pipeline from the CNN). ReLU is used as the activation function here as well. Table 3.3 shows the layer details for the FCNN. The full network structure for the full image based network can be seen in Figure 3.3.

3.2.2 Patch based network

Table 3.4: network structure

Layer	Input	Output	Kernel	Stride	Pooling	Activation
1	1	32	$5 \times 5 \times 5$	1	Avg.	ReLU
2	32	64	$3 \times 3 \times 3$	1	Avg.	ReLU
3	64	64	$3 \times 3 \times 3$	1	-	-

Since the input of the patch based network model is patches of uniform size, this model does not have the same problem with changing image sizes that the full image based network has. The patch based network model consists of 3 convolutional layers with 2 average pooling

Table 3.5: FCNN structure.

Layer	Input	Output	Activation
4	3456	64	ReLU
5	64	32	ReLU
6	32	12	-

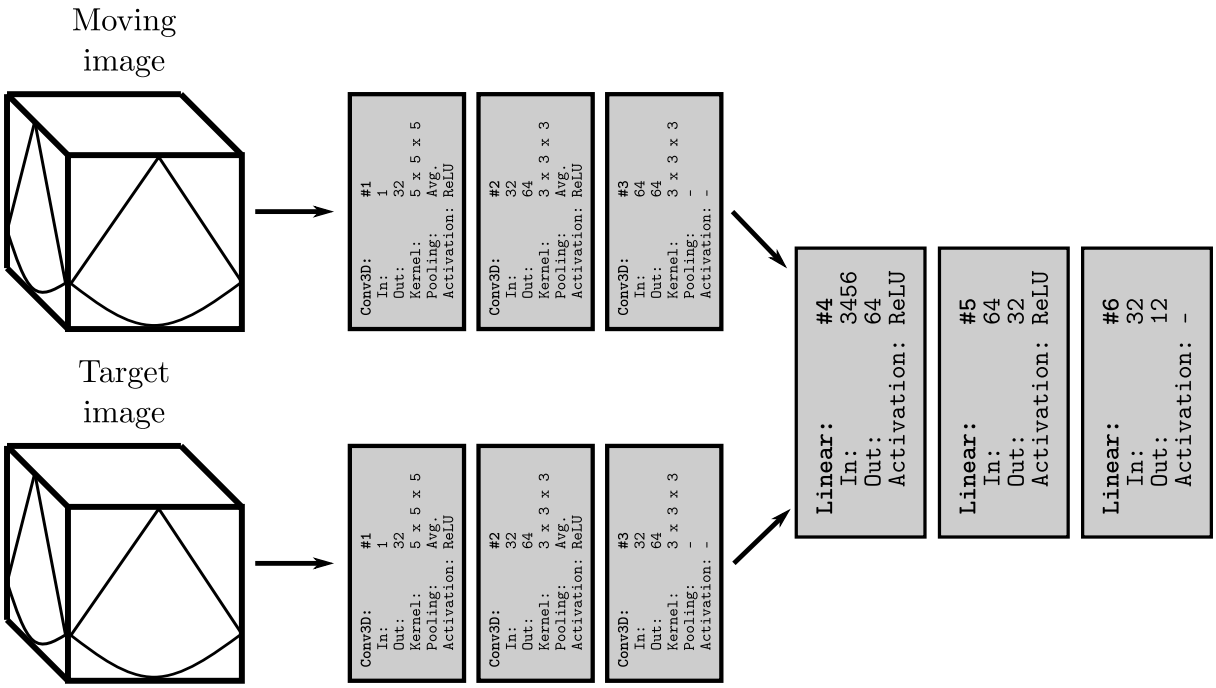


Figure 3.4: Patch based deep learning network. Two pipelines, one for each of the moving image and the target image, consisting of 3 CNN. The output from the final convolutional layer is used as the input to the FCNN. The FCNN consists of 3 linear layers where the output of the final layer is the 12 parameters of the affine transformation matrix Equation (2.15). The details from each layer can be seen in Table 3.4 and Table 3.5.

layers in between the convolutional layers. Table 3.4 shows the CNN structure for the patch based model. As for the full image based model, the output of the two CNN pipelines are used as input to the FCNN, but here without a global pooling layer. Choosing patch size $p = 30$ and step size $s = 29$ gives us an output tensor from the CNN of size $128 \times 3 \times 3 \times 3$. By reshaping this tensor to a 1-dimensional tensor gives a input tensor of size 3456 to the FCNN. Table 3.5 shows the specifications of the FCNN part of the patch based network. The patch based network is illustrated in Figure 3.4.

3.3 Affine transformation

The affine transformation process is done in 2 steps, as discussed in Section 2.5 Image registration with deep learning. The fist step is to produce the 3D affine grid for each batch N. This is

Algorithm 2 Affine grid generator for 3D images

T: Transformation matrix
N: Batch size
H: Image height
W: Image width
D: Image depth
procedure AFFINEGRIDGENERATOR3D(**T**, N, H, W, D)
    $h \leftarrow$ Generate linspace($\min = -1, \max = 1,$ length = H)
    $w \leftarrow$ Generate linspace($\min = -1, \max = 1,$ length = W)
    $d \leftarrow$ Generate linspace($\min = -1, \max = 1,$ length = D)
    for all $n \in N$ **do**
        $G_M(n) \leftarrow$ Generate new grid(h, w, d)
        $G_T(n) \leftarrow G_M(n) \cdot \mathbf{T}^\top$
    end for
end procedure

done to by first by creating 3 linearly spaced vectors in the range $[-1, 1]$. Each vector represent one of the dimensions of the images: height, H; width, W; and depth, D. This result in 3 vectors, **x**, **y** and **z** with length H, W and D respectively. These 3 linearly spaced vectors can then be combined into a 4D base grid:

$$G_M = \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \\ \mathbf{z} \\ \mathbf{1} \end{bmatrix},$$

G_M is then used to create the parameterized sampling grid, G_T, as in Equation (2.24) using the predicted transformation matrix **T**. An algorithm for how the parameterized sample grid is generated can be seen in Algorithm 2.

Algorithm 3 Affine transformation for 3D images

I_M: Moving image
T: Transformation matrix
procedure AFFINETRANSFORM(I_M, **T**)
    $N, H, W, D \leftarrow I.shape$
    $G_T \leftarrow$ AFFINEGRIDGENERATOR3D(**T**, N, H, W, D)
    for all $n \in N$ **do**
        $I_T(n) \leftarrow$ SampleGrid$(I_M(n), G_T(n))$
    end for
end procedure

The next step is to use intensity interpolation on the parameterized sampling grid G_T to compute the transformed moving image $T(I_M)$. How this is implemented will be discussed further in Section 4.3 Affine transformation.

The affine transformation method is summarized in Algorithm 3.

CHAPTER 4

Implementation

Both the full image based model and the patch based model have been developed in *Python* using the *PyTorch*[1] library for deep learning implementation of image registration. Other Python libraries used in this framework are:

- PyTorch
- NumPy[2]
- SciPy[3]
- Matplotlib[4]
- H5PY[5]

The implementation of the framework is publicly available on GitHub[6].

4.1 Loading data and preprocess

Reading HDF5 files was done using the *H5PY* Python library. The content of the `.h5` files are imported using the `library.hdf5_image.HDF5Image` class. This is a custom built Python class which stores the 3D ultrasound images as a 4D PyTorch tensor. The dimension

[1] `https://pytorch.org/`
[2] `https://www.numpy.org/`
[3] `https://www.scipy.org/`
[4] `https://matplotlib.org/`
[5] `https://www.h5py.org/`
[6] `https://github.com/andejoha/Ultrasound-Affine-Transformation`

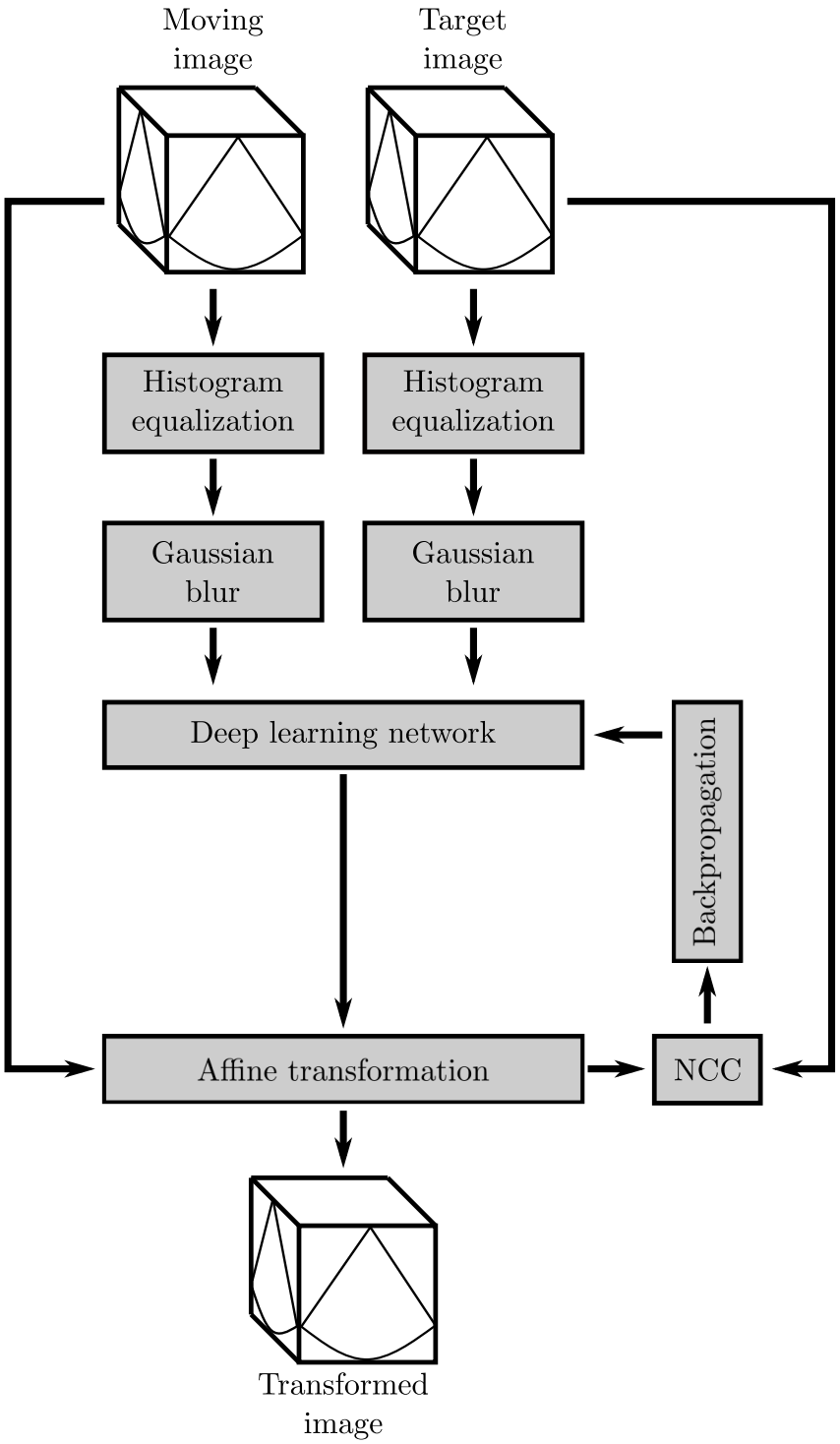
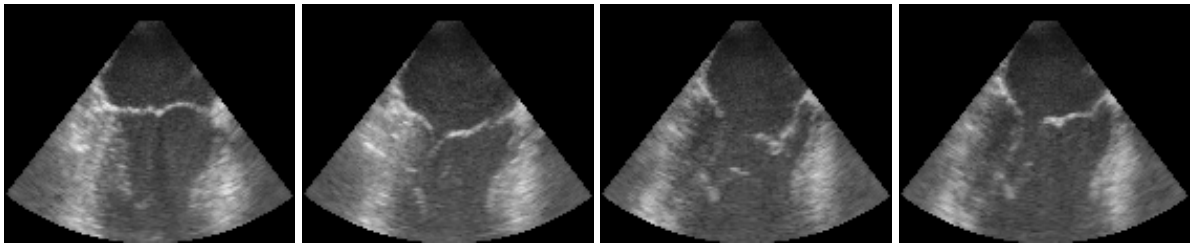


Figure 4.1: Flow of information illustration. The moving and target images are first put through histogram equalization and Gaussian blur preprocessing. The resulting images are then passed as input into a deep learning network. The predicted transformation from the deep learning network is then used to transform the moving image. During training the transformed moving image is also compared to the target image using NCC similarity measure, which is then used during the backpropagation algorithm to update the weights and biases of the deep learning network.



(a) **(b)** **(c)** **(d)**

Figure 4.2: Opening of the mitral valve. In image (**a**) the valve is fully closed. In (**b**) the opening of the valve has started, and in (**c**) the valve is at it fully open position. Finally, image (**d**) shows the valve beginning to close.

sizes of the tensor represent the 3D Euclidean space of the image as well as the temporal domain. This representation can be viewed as multiple 3D images in a time series creating a live video feed. An example of this can be seen in Figure 4.2, where a 4 image series of the opening of the mitral valve.

4.1.1 Gaussian blur

The implementation of Gaussian blur utilizes the `gaussian_filter` function from the SciPy Python library. This function uses a 1 dimensional Gaussian filter for every axis of the input array. The kernel size is determined by the chosen standard deviation, σ, according to the equation:

$$w = 2\lfloor 4\sigma + 0.5 \rfloor + 1,$$

where w is the width of the kernel and $\lfloor * \rfloor$ indicates the floor function. The Gaussian blur is implemented as a method in the `library.hdf5_image.HDF5Image` class.

4.1.2 Histogram equalization

Histogram equalization is implemented as described in OpenCVTeam (2015). For every image in the image tensor, the histogram is calculated using the NumPy `numpy.histogram` function. The cumulative sum of the histogram is then used in combination with the `numpy.ma.maske_equal` and `numpy.ma.filled` to compute the equalized histogram as in OpenCVTeam (2015). The histogram equalization is also implemented as a method in the `library.hdf5_image.HDF5Image` class.

4.2 Deep learning

The two deep learning networks has been implemented as two separate Python classes. The class implementation can be found in `library.network`. Both networks are implemented

with the PyTorch class `torch.nn.Module` as their parent class. This is critical due to the inheritance of the `.backward` method from the parent class. The purpose of this method is to automatically compute the gradients of the entire system based on the similarity measure through backpropagation. An explanation on how backpropagation in PyTorch works can be found in Appendix A.1 Introduction to backpropagation in PyTorch.

The patch based network can be accessed by calling the `library.network.PatchNet` class and the full image based network can be accessed by calling the `library.network.PatchNet` class. Upon initialization of any of these classes, both the CNN and the FCNN will be set up using the `torch.nn.Sequential` functionality. Inside this PyTorch function the full image based network and the patch based network are implemented in a sequential order according to Table 3.2, Table 3.3 and Table 3.4, Table 3.5. PyTorch has a large library of available network layers, pooling layers and activation functions. The layers used in the implementation of the two networks are:

- `torch.nn.Conv3d(in_channels, out_channels, kernel_size)`
- `torch.nn.Linear(in_features, out_features)`
- `torch.nn.AvgPool3d(kernel_size, stride)`
- `torch.nn.ReLU()`

The forward pass through the network is done by calling the `.forward` method with the an input tensor. The input tensor has a shape of $N \times 2 \times H \times W \times D$, where N is the batch size, and H, W and D are the dimension sizes of the images (the moving and target image must have identical dimensions for the full image based network), and the number 2 in the second position of the tensor comes from the combining the moving and target input into one tensor. This is done to combine the two inputs into a single tensor for backpropagation.

4.2.1 Negative NCC loss function

A custom implemented similarity measure for negative NCC is used in this system as a loss function. The negative of the NCC is used to turn the NCC into a minimization problem instead of a maximization problem as given in Equation (2.1). The similarity measure is implemented as a Python class with the PyTorch class `torch.nn.Module` as its parent class.

The negative NCC loss class implementation can be found in `library.ncc_loss.NCC`. An object of the `library.ncc_loss.NCC` class can be initialized at any time, but the computation of the negative NCC value is only done when the `.forward` method is called. This function takes two images as input: the transformed moving image, $I_M(N, H, W, D)$; and the target image, $I_T(N, H, W, D)$. The output is the negative NCC for each batch, $NCC(N)$. The mathematical representation presented in Section 2.2.1 NCC has been optimized for tensors

with batches. The introduction of batches means that the computations of the mean and summation values of the images has to take batches into account. This is done by utilizing the `dim` and `keepdim` arguments in the `.mean` and `.sum` PyTorch tensor functions. By choosing `dim=(1,2,3)` and `keepdim=True` the system will only consider the mean and sum of the images themselves and avoid cross batch influence. All other operations used to compute the negative NCC are elementwise tensor operations (`.sub`, `.sqrt`, `.pow`, `.mul` and `.div`), and are not affected by the batch dimension. To avoid division by 0, a small $\epsilon = 1 \cdot 10^{-7}$ will be added to the denominator of Equation (2.1).

4.3 Affine transformation

The affine transformation implementation is divided into 2 section: grid generation and grid sampling as discussed in Section 3.3 Affine transformation. To generate the three linearly spaces vectors, the PyTorch `torch.linspace` function is utilized with -1, 1 and W, H or D as arguments. A new `torch.tensor` is then initialized with the shape $(N, H, W, D, 4)$. The linearly spaced vectors are then placed into the the last dimension of the tensor with the last element being a `torch.ones` tensor. Batchwise matrix multiplication between the resulting base grid and the θ-tensor from the prediction network is done using the PyTorch `troch.bmm` function. This gives us the desired parameterized sampling gird represented as a $(N, D, H, W, 4)$ tensor.

By having the parameterized sampling gird normalized (on the form $-1 \leq g \in G \leq 1$) we can use the built-in function `torch.grid_sample` from the PyTorch library with the moving image tensor and parameterized sampling grid tensor, as arguments. This computes the transformed moving image using the chosen interpolation method (default: trilinear interpolation). Note that the PyTorch implementation of image sampling also require a channel dimension C. For grayscale images we have $C = 1$, and can be added by using `torch.Tensor.unsqueeze` at the 2nd position of the images. This will give the input images the shape $I(N, C, H, W, D)$. implementation of the framework for affine transformation is available in
`library.affine_transformation`.

CHAPTER 5

Results

5.1 Noise reduction

The Gaussian blur preprocessing is done in an effort to reduce noise. As a metric for noise reduction we are going to look at the negative NCC between a set of arbitrary moving and target images and compare results. To see the effect of Gaussian blur we are looking at the min, max, mean and variance of the negative NCC between the moving and target image. Table 5.1 shows the results of the Gaussian blur without histogram equalization. A standard deviation of the kernel ranging between $\sigma \in [0, 2]$ with 0.1 increment has been used to generate this table. In Table 5.2 the same set of images are used, but histogram equalization has also been applied here.

It is more difficult to give a quantification for the effect of histogram equalization. This is because the purpose of histogram equalization is more directly integrated into the deep learning process. Histogram equalization is done to give the images a more clear patches within the image corresponding to features which (in theory) can be recognize by the deep learning network easily. From (**c**) and (**d**) in Figure 3.1 we can see that the tissue corresponding to the heart wall has a higher contrast compared to the surroundings when histogram equalization has been applied. These are patches can be recognized by the deep learning system and used to predict the transformation that align the moving image to the target image.

5.2 Training results

During training one ultrasound recording from the set of moving ultrasound recordings is removed from the set and used for validation. For every epoch during training every image in

Table 5.1: The negative NCC between a set of arbitrary moving and target images are being used to see the effect of Gaussian blur noise reduction without histogram equalization applied. A standard deviation of the kernel ranging between $\sigma \in [0, 2]$ with 0.1 increment has been used to blur the images and reduce noise.

σ	Min	Max	Mean	Variance
0	-0.8808	-0.8570	-0.8651	$4.927 \cdot 10^{-5}$
0.1	-0.8808	-0.8570	-0.8651	$4.927 \cdot 10^{-5}$
0.2	-0.8808	-0.8570	-0.8651	$4.927 \cdot 10^{-5}$
0.3	-0.8814	-0.8578	-0.8656	$4.944 \cdot 10^{-5}$
0.4	-0.8866	-0.8631	-0.8706	$5.124 \cdot 10^{-5}$
0.5	-0.8938	-0.8693	-0.8775	$5.463 \cdot 10^{-5}$
0.6	-0.8987	-0.8738	-0.8825	$5.718 \cdot 10^{-5}$
0.7	-0.9020	-0.8772	-0.8860	$5.863 \cdot 10^{-5}$
0.8	-0.9047	-0.8800	-0.8889	$5.945 \cdot 10^{-5}$
0.9	-0.9071	-0.8827	-0.8915	$5.989 \cdot 10^{-5}$
1.0	-0.9092	-0.8851	-0.8940	$6.006 \cdot 10^{-5}$
1.1	-0.9112	-0.8874	-0.8963	$5.999 \cdot 10^{-5}$
1.2	-0.9131	-0.8895	-0.8986	$5.970 \cdot 10^{-5}$
1.3	-0.9148	-0.8915	-0.9007	$5.923 \cdot 10^{-5}$
1.4	-0.9166	-0.8935	-0.9027	$5.859 \cdot 10^{-5}$
1.5	-0.9183	-0.8954	-0.9047	$5.781 \cdot 10^{-5}$
1.6	-0.9199	-0.8972	-0.9065	$5.690 \cdot 10^{-5}$
1.7	-0.9214	-0.8989	-0.9083	$5.588 \cdot 10^{-5}$
1.8	-0.9229	-0.9006	-0.9101	$5.478 \cdot 10^{-5}$
1.9	-0.9243	-0.9022	-0.9118	$5.360 \cdot 10^{-5}$
2.0	-0.9256	-0.9038	-0.9134	$5.237 \cdot 10^{-5}$

the set of moving recordings are passed through the deep learning network together with an images from the target recording. At the end of the epoch the validation set is passed through the network in evaluation mode to act as a objective metric for performance as the images was not a part of training. The full image based network and the patch based network were trained separately and on the same data set.

5.2.1 Full image based network

The full image network follows the traditional training process. A set of training images are passed through the network, the backpropagation is computed on based on the NCC loss function and the weights are updated using an optimizing algorithm. The network was trained for 30 epochs using a set of 71 images. All images used during training have the resolution: $214 \times 214 \times 214$. The Adam optimization algorithm was used with a learning rate of $1 \cdot 10^{-5}$. The details of the training are listed in Table 5.3. Note that the patch size parameter is only used for in the patch based model, and the batch size has been limited to 1 due to limited GPU

Table 5.2: The negative NCC between the same set of arbitrary moving and target images as in Table 5.1 are being used to see the effect of Gaussian blur noise reduction with histogram equalization applied. A standard deviation of the kernel ranging between $\sigma \in [0, 2]$ with 0.1 increment has been used to blur the images and reduce noise.

σ	Min	Max	Mean	Variance
0	-0.8608	-0.8391	-0.8475	$4.521 \cdot 10^{-5}$
0.1	-0.8608	-0.8391	-0.8475	$4.521 \cdot 10^{-5}$
0.2	-0.8608	-0.8391	-0.8475	$4.521 \cdot 10^{-5}$
0.3	-0.8615	-0.8396	-0.8481	$4.547 \cdot 10^{-5}$
0.4	-0.8673	-0.8440	-0.8534	$4.801 \cdot 10^{-5}$
0.5	-0.8751	-0.8504	-0.8606	$5.211 \cdot 10^{-5}$
0.6	-0.8802	-0.8550	-0.8656	$5.501 \cdot 10^{-5}$
0.7	-0.8836	-0.8583	-0.8690	$5.678 \cdot 10^{-5}$
0.8	-0.8863	-0.8611	-0.8718	$5.798 \cdot 10^{-5}$
0.9	-0.8886	-0.8636	-0.8743	$5.887 \cdot 10^{-5}$
1.0	-0.8907	-0.8659	-0.8765	$5.956 \cdot 10^{-5}$
1.1	-0.8926	-0.8681	-0.8787	$6.007 \cdot 10^{-5}$
1.2	-0.8944	-0.8701	-0.8807	$6.041 \cdot 10^{-5}$
1.3	-0.8960	-0.8720	-0.8826	$6.059 \cdot 10^{-5}$
1.4	-0.8977	-0.8738	-0.8845	$6.062 \cdot 10^{-5}$
1.5	-0.8993	-0.8756	-0.8863	$6.049 \cdot 10^{-5}$
1.6	-0.9009	-0.8773	-0.8880	$6.024 \cdot 10^{-5}$
1.7	-0.9023	-0.8789	-0.8896	$5.985 \cdot 10^{-5}$
1.8	-0.9037	-0.8805	-0.8913	$5.935 \cdot 10^{-5}$
1.9	-0.9050	-0.8820	-0.8928	$5.875 \cdot 10^{-5}$
2.0	-0.9063	-0.8835	-0.8944	$5.805 \cdot 10^{-5}$

memory.

Table 5.3: Full image based network training parameters.

Epochs	Learning rate	Patch resolution	Batch	Optimizer	Training images
30	$1 \cdot 10^{-5}$	-	1	Adam	71

The resulting negative NCC training and validation loss graphs can be seen in Figure 5.1.

5.2.2 Patch based network

The patch based network model has a different training process then the full image based network. In stead of using the loss calculated from the patches directly; instead, it takes the maximum negative NCC loss from the patch calculation (the patch with the largest difference, i.e the worst case scenario) and use the corresponding transformation matrix to compute the transformation on a random selected full image. This is done because the large amount of pure noise

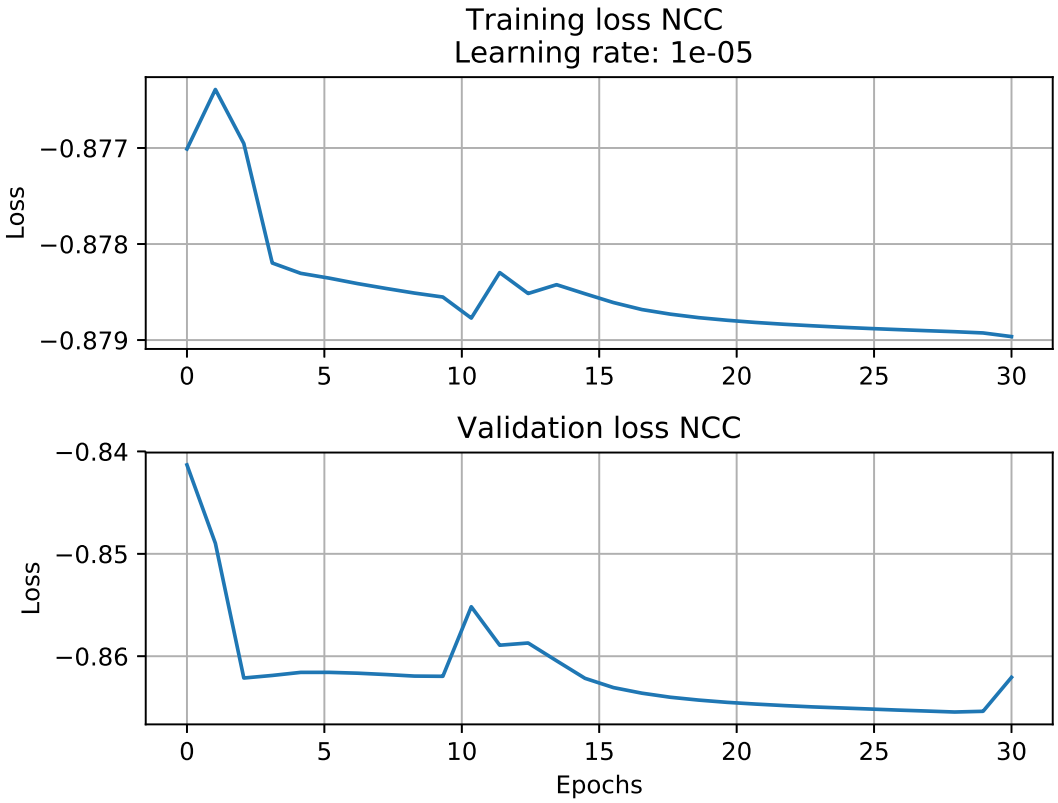


Figure 5.1: Training (top) and validation (bottom) loss graphs for training of the full image based network. The training parameters used to train this network can be seen in Table 5.3.

patches will give a incorrect NCC if the mean of batch mean of the transformation matrix is used.

The training of the network was done using a patches with resolution $30 \times 30 \times 30$ voxels and a batch size of 64. Training lasted for a total of 150 epochs using the Adam optimizer with a $1 \cdot 10^{-5}$ learning rate. A summary of the training details can be seen in Table 5.4. The same set of 71 images has been used for the patch based network as for the full image based network.

Table 5.4: Patch based network training parameters.

Epochs	Learning rate	Patch resolution	Batch	Optimizer	Training images
150	$1 \cdot 10^{-5}$	$30 \times 30 \times 30$	64	Adam	71

The resulting negative NCC training and validation loss graphs can be seen in Figure 5.2.

5.3 Test results

Testing is done on a separate set of ultrasound images from the same patient. The test set contains 1 ultrasound recording containing 25 images each with resolution $214 \times 214 \times 214$ voxels.

Figure 5.2: Training (top) and validation (bottom) loss graphs for training of the patch based network. The training parameters used to train this network can be seen in Table 5.4.

As a metric for the system performance, the difference in negative NCC between the moving and target image before and after transformation has been used. This gives an indication of the the effect the transformation has on the negative NCC. Given the moving image I_M, the target image I_T and the transformation T, the negative NCC difference, δ, is given by:

$$\delta = NCC\big(I_M, I_T\big) - NCC\big(T(I_M), I_T\big) \tag{5.1}$$

Table 5.5 and Table 5.6 shows the calculations of $NCC\big(I_M, I_T\big)$, $NCC\big(T(I_M), I_T\big)$ and δ from Equation (5.1) for every image in the training set. The average across the whole training set is given given in the final row.

An example of a transformed image from the test set can be seen in Figure 5.3 and Figure 5.4. The images in both figures are taken from the same frame (#1) in the test set, but using a different transformation prediction method. In Figure 5.3 the transformation has been predicted using the full image based network, while in Figure 5.4 the patch based network has been used.

Table 5.5: Change in negative NCC before and after transformation as well as the difference of the two for the patch based model.

Frame	$NCC(I_M, I_T)$	$NCC(T(I_M), I_T)$	δ
1	-0.8780	-0.8620	-0.0160
2	-0.8819	-0.8658	-0.0161
3	-0.8858	-0.8701	-0.0157
4	-0.8869	-0.8725	-0.0144
5	-0.8850	-0.8726	-0.0125
6	-0.8817	-0.8712	-0.0105
7	-0.8787	-0.8693	-0.0093
8	-0.8768	-0.8678	-0.0090
9	-0.8760	-0.8668	-0.0092
10	-0.8757	-0.8661	-0.0096
11	-0.8761	-0.8663	-0.0098
12	-0.8774	-0.8675	-0.0100
13	-0.8790	-0.8689	-0.0101
14	-0.8791	-0.8687	-0.0104
15	-0.8771	-0.8661	-0.0110
16	-0.8740	-0.8623	-0.0117
17	-0.8709	-0.8586	-0.0123
18	-0.8681	-0.8557	-0.0124
19	-0.8655	-0.8534	-0.0121
20	-0.8632	-0.8516	-0.0116
21	-0.8616	-0.8504	-0.0112
22	-0.8607	-0.8498	-0.0108
23	-0.8604	-0.8500	-0.0104
24	-0.8605	-0.8506	-0.0100
25	-0.8571	-0.8476	-0.0095
Avg.	-0.8735	-0.8621	-0.0114

5.4 Runtime

The project was conducted by GPU acceleration with *CUDA*[1]. PyTorch is closely integrated with CUDA and switching between running on GPU and CPU is done with the `.cuda()` and `.cpu()` tensor commands respectively. The GPU used for this project was the NVIDIA GeForce GTX 1070 Ti.

The computation time of system is divided into 4 parts: loading images, Gaussian blur, histogram equalization, and image registration. Loading images and preprocessing are shared for both the full image based model and the patch based model. However, the prediction and affine transformation varies depending on which model you use. The image registration consist of both transformation matrix prediction and affine transformation.

[1] `https://developer.nvidia.com/cuda-toolkit`

Table 5.6: Change in negative NCC before and after transformation as well as the difference of the two for the patch based model.

Frame	$NCC(I_M, I_T)$	$NCC(T(I_M), I_T)$	δ
1	-0.8780	-0.8778	-0.0002
2	-0.8819	-0.8814	-0.0005
3	-0.8858	-0.8853	-0.0005
4	-0.8869	-0.8870	0.0001
5	-0.8850	-0.8864	0.0014
6	-0.8817	-0.8844	0.0027
7	-0.8787	-0.8820	0.0033
8	-0.8768	-0.8801	0.0032
9	-0.8760	-0.8788	0.0028
10	-0.8757	-0.8779	0.0023
11	-0.8761	-0.8779	0.0019
12	-0.8774	-0.8792	0.0017
13	-0.8790	-0.8809	0.0019
14	-0.8791	-0.8813	0.0022
15	-0.8771	-0.8796	0.0025
16	-0.8740	-0.8765	0.0025
17	-0.8709	-0.8733	0.0024
18	-0.8681	-0.8705	0.0025
19	-0.8655	-0.8684	0.0029
20	-0.8632	-0.8667	0.0034
21	-0.8616	-0.8656	0.0040
22	-0.8607	-0.8651	0.0044
23	-0.8604	-0.8652	0.0048
24	-0.8605	-0.8656	0.0050
25	-0.8571	-0.8615	0.0044
Avg.	-0.8735	-0.8759	0.0024

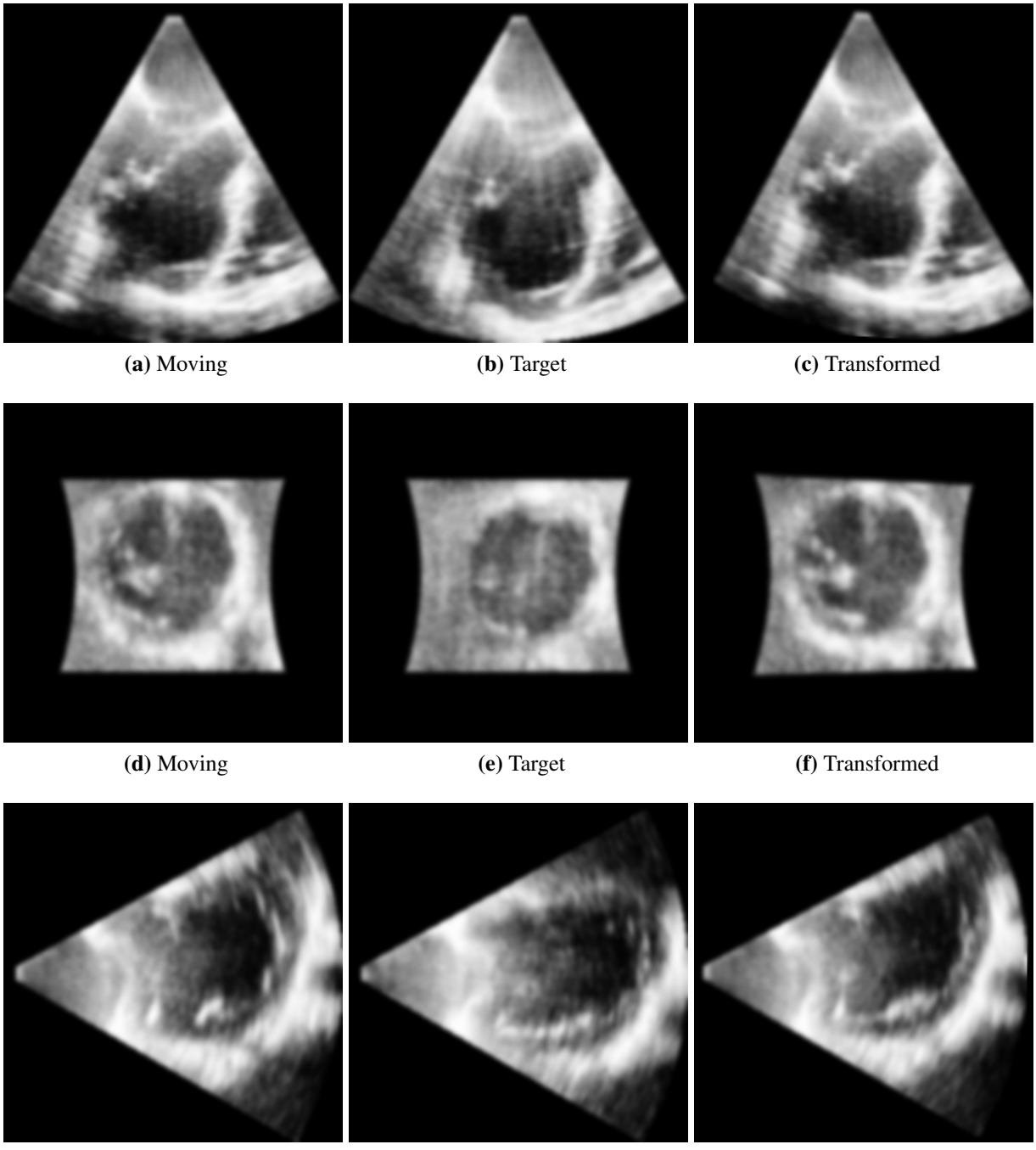
Table 5.7 shows the computation time per image for the full image based model. This computation time per image is calculated from the same test set as before. The total time is calculated from entire registration process of the 25 images in the test set and divided by the number of images. Table 5.8 shows the same test set, but now registered using the patch based model. Note that the percentages does not add up to 100%. This is because of computation time spent on other processes such as variable initialization and memory handling. These processes can not be placed inside a specific category, but is still a part of the program, and thus included in the total time and not for any specific sub-processes. This adds up to 20.8% and 22.2% of the total computation time for full image based model and patch based model respectively.

Table 5.7: Computational time per image for full image based model.

Process	Time [ms]	% of total
Load image	139 ± 9	15.6%
Gaussian blur	156 ± 8	17.5%
Histogram equalization	87 ± 1	9.8%
Image registration	323.5 ± 3.5	36.3%
Total	891	100%

Table 5.8: Computational time per image for patch based model.

Process	Time [ms]	% of total
Load image	139 ± 9	13.7%
Gaussian blur	156 ± 8	15.4%
Histogram equalization	87 ± 1	8.5%
Image registration	407 ± 5	40.2%
Total	1012	100%

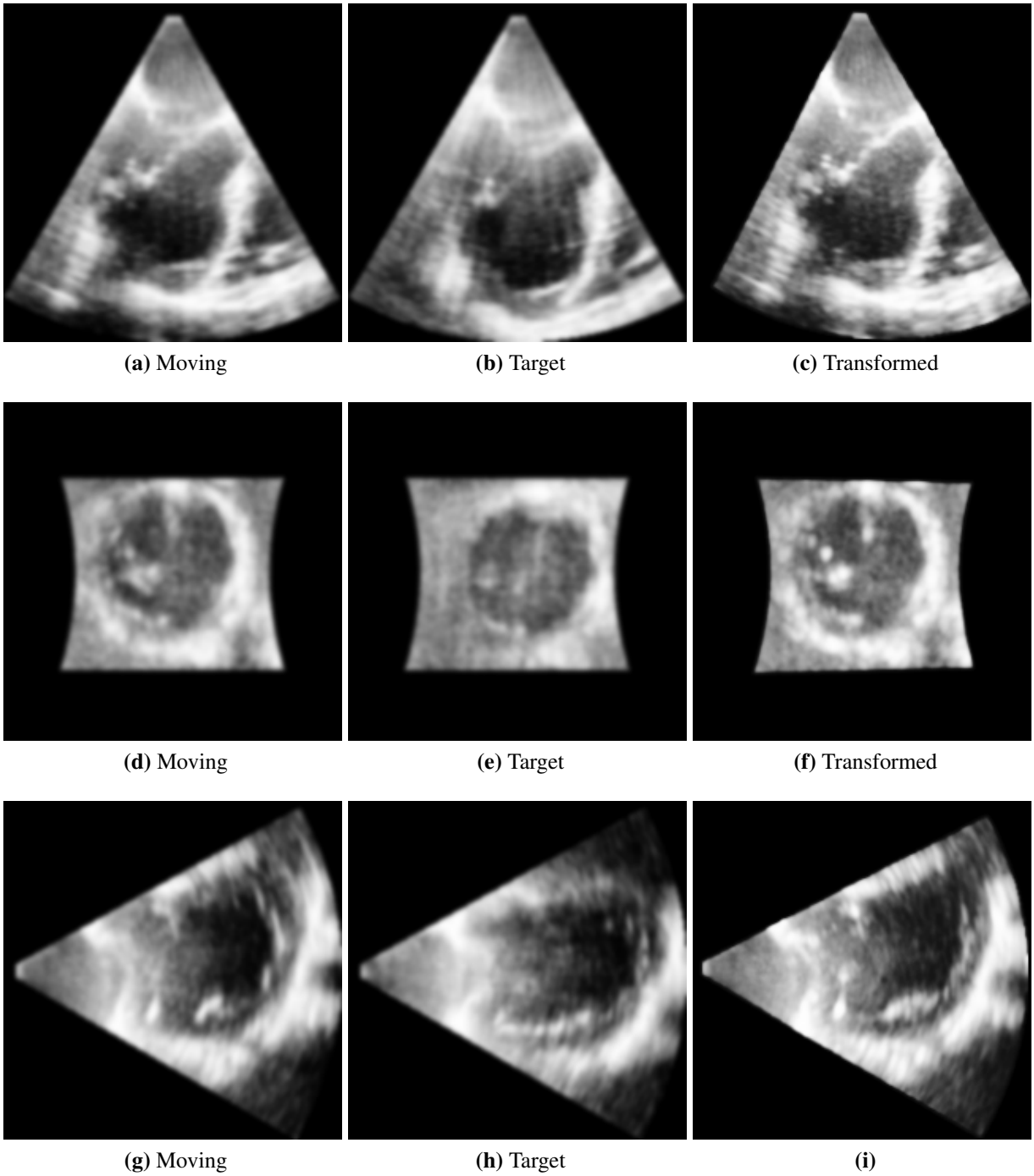


(a) Moving **(b)** Target **(c)** Transformed

(d) Moving **(e)** Target **(f)** Transformed

(g) Moving **(h)** Target **(i)**

Figure 5.3: Example of a transformation of a moving image (first column) with respect to target image (second column) using the full image based network. The resulting transformation can be seen in the third column. The images are taken from the center slice of each dimension of the 3D ultrasound image: first row, y-z plane; second row, x-z plane; third row, x-y plane.



(a) Moving **(b)** Target **(c)** Transformed

(d) Moving **(e)** Target **(f)** Transformed

(g) Moving **(h)** Target **(i)**

Figure 5.4: Example of a transformation of a moving image (first column) with respect to target image (second column) using the patch based network. The resulting transformation can be seen in the third column. The images are taken from the center slice of each dimension of the 3D ultrasound image: first row, y-z plane; second row, x-z plane; third row, x-y plane.

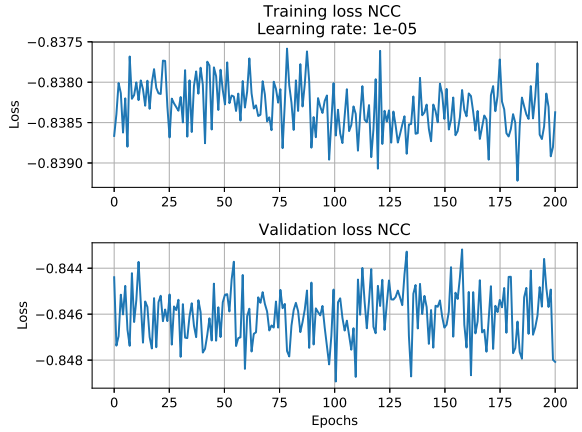
CHAPTER 6

Discussion

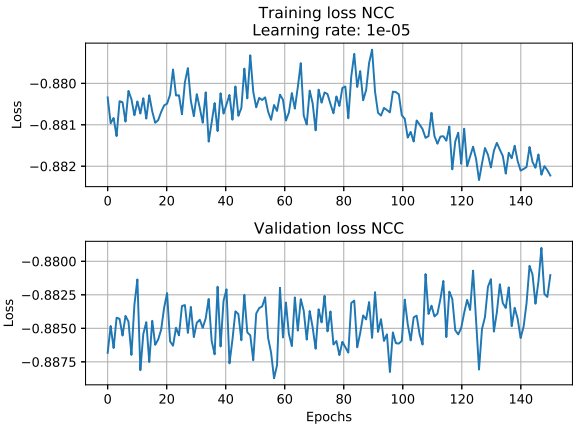
6.1 Noise reduction effect on training

From Table 5.1 and Table 5.2 we can get a understanding on how the Gaussian blur and histogram equalization effects the negative NCC. The min, max and mean are all decreasing as the value of σ increases. This is the expected behaviour as the the more blurred out the images become, the more similar they appear. The interesting part; however, is the variance. From Table 5.1 we can see that the variance of the negative NCC increases as σ increases until it hit a maximum at $\sigma = 1.0$ where Var$(NCC) = 6.006 \cdot 10^{-5}$. After this point the variance decreases once more. The same is true for Table 5.2 where the maximum variance is when $\sigma = 1.4$, at which Var$(NCC) = 6.062 \cdot 10^{-5}$. The variance give an indication on the spread of negative NCC values from their mean value. In the case, the same dataset is being used with different degrees of filtering before transformation. The interesting part is not whether or not the images are similar, but rather the ability of the loss function to detect differences between the moving and target image. A larger variance in negative NCC between different values of σ gives an indication for which filtration combination the loss function preforms optimally.

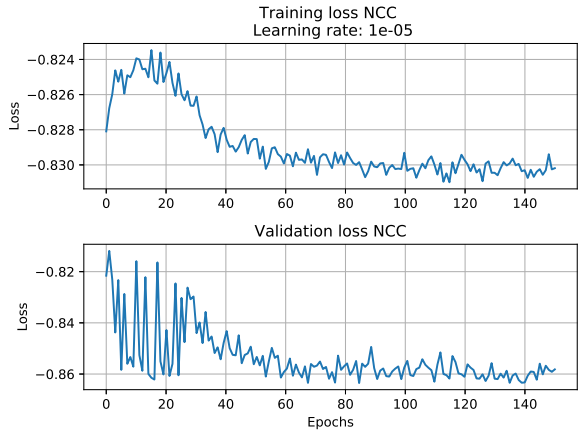
Figure 6.1 shows the training loss and validation loss for the patch based network with different combination of preprocess filters. (**a**) shows the training loss with neither Gaussian blur nor histogram equalization applied. From the graph we can see that the system is not able to train the patch based network when no filtration is being used. (**b**) shows the training with only Gaussian blur filtration. This graph shows a small decline in training loss at around epoch number 90, but at the same time the validation loss graph is increasing. This is an example of *overfitting* and will be discussed further in Section 6.2.1 Overtraining. In (**c**) no Gaussian blur is being applied, but histogram equalization is being preformed. From the graph we can see



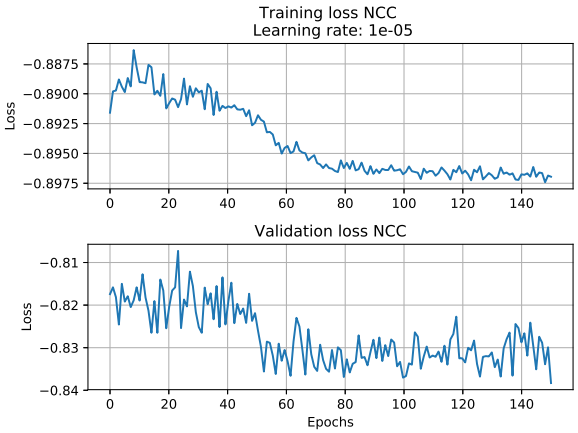
(a) No blur and no histogram equalization



(b) Blur and no histogram equalization



(c) No blur and histogram equalization



(d) Blur and histogram equalization

Figure 6.1: Training loss with different combinations of preprocess filters. In (**a**) no filter has been used, in (**b**) only Gaussian blur has been used, in (**c**) only histogram equalization has been used, and in (**d**) both Gaussian blur and histogram equalization has been used. All of these loss graphs are generated from training the patch based network.

that the histogram equalization made the system able to recognize features of the ultrasound ultrasound images and use them to find affine transformations, and thus being able to train the network. (**d**) shows training loss with both Gaussian blur and histogram equalization applied. For (**b**) and (**d**), the optimal σ from Table 5.1 and Table 5.2 has been used for the Gaussian blur.

Figure 6.1 tells us that histogram equalization is the significant preprocessing filtration method for deep learning network training for affine transformation of ultrasound images. Comparing the starting negative NCC values and convergence negative NCC values of (**c**) and (**d**), we can see that starting point when only histogram equalization has been used is -0.828, while -0.892 when both Gaussian blur and histogram equalization are being used. The difference in starting point comes from the effect Gaussian blur has on the negative NCC as shown in Table 5.1 and Table 5.2. At the end of the training process, the training loss graph from (**c**) converges towards -0.831 and graph from (**d**) converges towards -0.897. This gives a differ-

ence between initial the negative NCC and the negative NCC at the end of training of 0.003 and 0.005 respectively. The introduction of Gaussian blur in addition to histogram equalization gives a training loss improvement of 66.7%. Whether or not this is a significant improvement to justify the increase in runtime that Gaussian blur introduces will be discussed in Section 6.3 Runtime analysis.

6.2 Performance

As stated in Section 5.3 Test results, the difference in negative NCC between the moving and target image before and after the transformation is applied has been used as a metric for analyzing the performance on the test images. Based on the sign of δ in Table 5.5 and Table 5.6 we can make the following conclusions about the transformation:

$\delta < 0$ The transformed image has a larger negative NCC (less similar) then the original moving image in reference to the target image. From this we can conclude that the transformation does not give an improved alignment.

$\delta = 0$ The transformed image and the original moving image has an identical negative NCC. We can conclude from this that the transformation is the identity transformation.

$\delta > 0$ The transformed image has a smaller negative NCC (more similar) then the original moving image in reference to the target image. This tells us that the transformation gives an improved alignment of the moving image.

Using this as a guidelines, we can from Table 5.3 see that the difference in NCC before and after transformation is negative for all images in the training set ($\delta < 0$). This tells us that the transformation does not improve the alignment of the images. Table 5.4, on the other hand; has predominantly positive values of δ. From this we can conclude that the images transformed using the patch based network transformation have an improved alignment. An average improvement of 0.0044 correspond nicely to the training improvement shown in Figure 5.2.

Because of the "black box" nature of a deep learning with CNN it is difficult to give a definitive answer to why the patch based prediction model outperforms the full network based prediction model. A key difference between the full image based prediction model and the patch based prediction model is the amount of training data. Even though the same dataset is used when training the two networks, the process of dividing the image into several smaller patches creates a larger amount of data as well as more localized images used for training. The effect of this can be seen in the difference in convergence time for the two systems. Where the patch based network was trained for 150 epochs before convergence, the full image based network converged after only 30 epochs. This gives an indication that the patch based prediction model is able to generalize and recognize more advanced features of the transformation better then the full image based prediction model is able.

Table 5.5 and Table 5.6 shows the results from running the full image based system and patch based system respectively on a set of test images from the same patient. Each recording in the test set has also been generated by slightly moving the probe between recordings to illustrate probe movement. This can be seen in Figure 5.3 and Figure 5.4 column 1 and 2. From these images we can see that the left ventricle chamber in the moving image (**a**) is slightly misaligned with respect to the target image (**b**). The transformation is easily visible in column 3 (**a**) and (**b**). In (**a**) there is a clear translation of the image to the right as well as rotation/shear of the image to make it fit the frame. (**c**) does not show any clear translation, but the left side has been stretched out and the right side has been squeezed. Note that this images show the images with histogram equalization and Gaussian blur, but this is not necessary. The histogram equalization and Gaussian blur preprocessing techniques are only used during the prediction of the transformation matrix. Once the transformation matrix is found, the transformation can be applied to the original images before the preprocessing.

6.2.1 Overtraining

Two forms of overtraining where encountered during training these models. The first one is the traditional overfitting example, where the training loss continue to decrease while the validation loss increases. An overfitted model tends to remember the structures of the training images instead of generalizing the features that represent training image. This is illustrated in Figure 6.2. From the top graph in this figure we can see that the training loss decreases throughout the entire training process, and converges towards -0.879 around epoch 120. From the bottom graph we can see that the training loss also decreases together with the training loss, but around the epoch number 80, the loss is starting to increase again. This is due to the network is starting to assume that the input will always be the training data, and thus finds a solution that works well for all training images, but does not work in the general case.

The second type of overtraining was encountered during experimentation with different loss functions for the full image based network. Figure 6.3 shows training using the *Binary Cross Entropy with Logits* loss function. Around epoch number 180 the loss decreases substantially on a short amount of time. Figure 6.4 shows an example of a transformed image using the weights corresponding to the training shown in Figure 6.3. The transformed image is a enhancement of a light patch within the image so that it covers the entire image. This is most likely due to the network finding a local minima with a lower loss value representing enhancement of light patches then the local minima representing the desired optimal affine transformation. Initializing the network to the identity transformation will place the starting point closer to the local minima representing the optimal affine transformation rather then extreme transformations such as the one presented in this example.

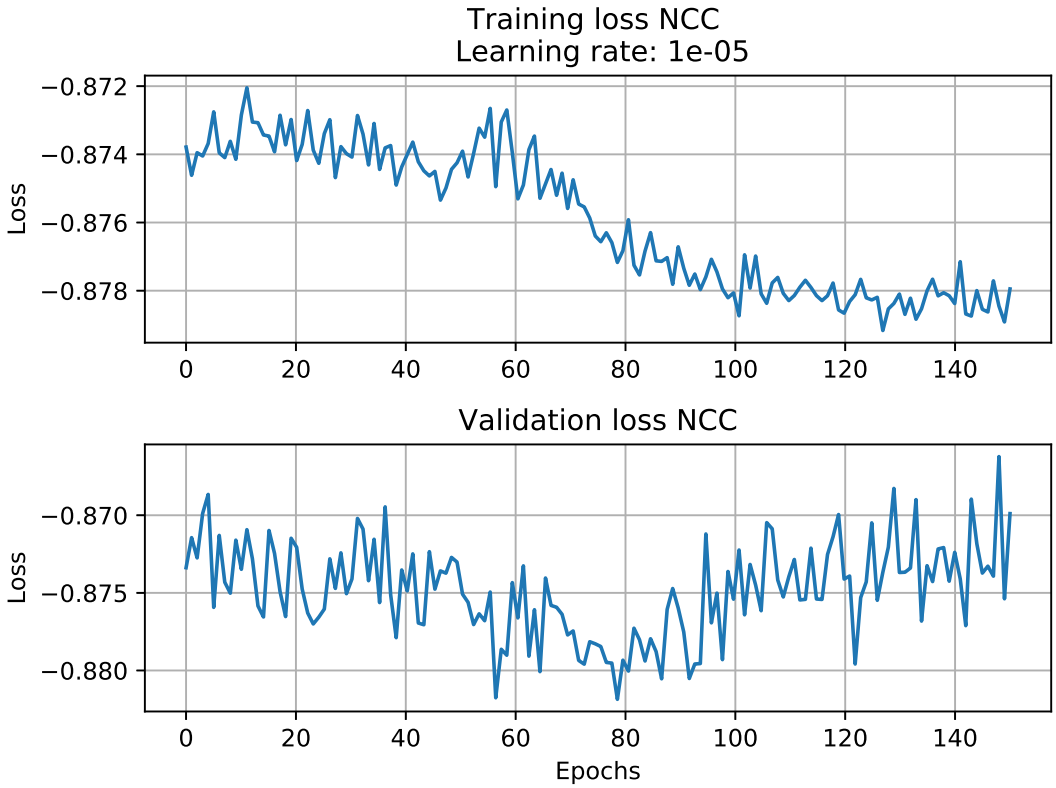


Figure 6.2: Example of an overfitted model. The training loss (top part) is decreasing through the entire training process, but the validation loss (bottom part) reach a point where it start to increase increases again. This is due to the overfitting, where the network remembers the structures of the training images instead of generalizing.

6.3 Runtime analysis

Runtime analysis is an important aspect to consider for any system that may be used in a real time setting. Computational time of different sub-processes can give an idea of potential bottlenecks in your system and processes that can be speed up to reduce overall computation time.

From Table 5.7 and Table 5.8 we can see that prediction of 1 image takes on average 891ms for the full image based prediction model and 1012ms for the patch based prediction model. In other words, the patch based model has an average 13.6% lower computational time. This increase computation time comes primarily from the patch calculations as discusses in Section 3.1.1 Preprocess.

When it comes to the sub-processes, we can see that the image registration is the most time consuming sub-process with 36.3% and 40.2% of the computation time for the full image based model and the patch based model respectively. This is the main part of the process and minimal improvements such as general optimization of memory processes can be done. The second highest computational demanding process is the Gaussian blur with 17.5% and 15.4% of the total computation time. From the results discussed in Section 6.1 Noise reduction effect on training, one can argue that the Gaussian blur is not significant enough to include in a real

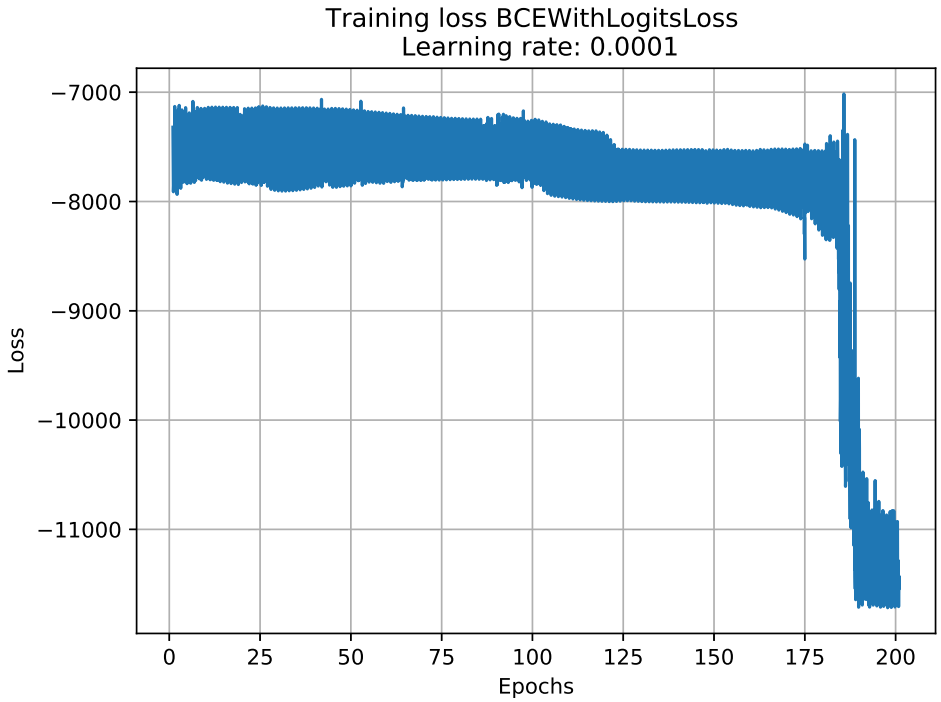


Figure 6.3: Overtraining

time setting because of the computation time. By leaving out the Gaussian blur preprocess the computation speed of the system will increase by 17.5% and 15.4% respectively. Loading of data into memory is in this case mostly used for testing and training purposes. In a real time setting, the images would be fed into the system from a live stream and thus loading and storing images would not be necessary outside for preprocessing purposes. Optimization when it comes to how the images are loaded into memory can be done here as well to increase computation speed. The fastest sub-process is the histogram equalization. As histogram equalization is a necessary preprocessing filtration to ensure training convergence, it would be more difficult to argue leaving out histogram equalization. Both Gaussian blur and histogram equalization are implemented using the Numpy and SciPy libraries, which does not have backend GPU acceleration built in. That means that the all preprocessing is done entirely on the CPU. GPU implementation of these processes may reduce computation time substantially.

6.4 Future work

The results from training of the patch image based model shows promising results. A limitation of the trained network as it is presented in this thesis, is when it comes to multiple patients. The system has only been trained on a limited amount of ultrasound recordings taken from the same patient. More work needs to be done to ensure satisfying results on multiple patients and take with different image sizes. The full image based model did not preform as intended. It showed improvement on the training and validation data, but did not produce satisfying results on the

(a) y-z plane **(b)** x-y plane **(c)** x-z plane

Figure 6.4: The resulting transformation using a overtrained network using binary cross entropy with logits loss. The loss function consider an enhanced white patch a better transformation then a small affine transformation we get by using NCC.

test data. Further training and refinement of the full image based model is needed to see the potential of this method. Areas with improvement potential for the full image based model are, but not limited to: the network structure, more training data, and improving the NCC similarity measure to ignore unlikely deformations.

The framework presented shows potential for real-time applications. Incorporating the framework into a real-time system is possible with little work necessary. Optimization of the Gaussian blur and histogram equalization to run using GPU acceleration could reduce the computation requirements substantially. We have also seen that approximately 20% of the computation time is spent on other processes that can not be directly associated with any of the main parts of the method. Improvements on memory handling may increase computational speed as well. The GPU do also play an important role when it comes to computation time. This project was done using a NVIDIA GeForce GTX 1070 Ti, which is substantially slower then the industry standard GPU NVIDIA Titan X. The additial memory a more powerful GPU includes will also reduce the amount of memory handling required.

CHAPTER 7

Conclusion

The goal of this project was to develop a framework for alignment of 3D ultrasound images based on deep learning. This was done in an effort to improve the computational time of 3D image registration processes, which due to their high computational requirements does not see any use today outside being a post-processing tool. To solve this we have look at two different methods using a full image based model and a patch based model. All ultrasound image used during training and testing was taken from the same patient. The images was misaligned purposely by shifting the TEE transducer probe slightly to simulate probe movement during procedures. Before the images were fed forward through the deep learning networks they were filtrated using Gaussian blur and histogram equalization. The training showed that both models where able to learn features in the images and use them to align the images accordingly when histogram equalization was applied. The Gaussian blur gave an improvement to the accuracy but was shown not to be necessary for training convergence. During testing; however, only the patch based model showed an improved alignment between the moving image and the target image after transformation. The results from the full image based model was inconclusive and require further work.

From the runtime analysis we can see that the models preforms well in compared to traditional image registration methods like LDDMM. As always when it comes to real-time implementations there is a trade of between accuracy and computation speed. Whether or not certain preprocessing operations, such as Gaussian blur gives a high enough accuracy improvement to justify the computational requirement have to be discussed before any implementation of the system in a real-time environment.

Bibliography

Arfken, G., Weber, H., 2005. Mathematical Methods for Physicists, 6th Edition. Elsevier.

Boncelet, C., 2005. Image noise models. In: Bovik, A. (Ed.), Handbook of Image and Video Processing (Second Edition), 2nd Edition. Academic Press, pp. 397–409.
URL http://www.sciencedirect.com/science/article/pii/B9780121197926500875

Che, C., Mathai, T., Galeotti, J., 2017. Methods 115, 128–143.
URL http://www.sciencedirect.com/science/article/pii/S1046202316304789

Danudibroto, A., Bersvendsen, J., Gérard, O., Mirea, O., D'hooge, J., Samset, E., 2016. Spatiotemporal registration of multiple three-dimensional echocardiographic recordings for enhanced field of view imaging. Journal of Medical Imaging 3 (3).

de Voss, B., Berendsen, F., Viergever, M., Sokooti, H., Staring, M., Išgum, I., 2019. A deep learning framework for unsupervised affine and deformable image registration. Medical Image Analysis 52, 128 – 143.
URL http://www.sciencedirect.com/science/article/pii/S1361841518300495

Egeland, O., Gravdahl, T., 2002. Modeling and Simulation for Automatic Control. Marine Cybernetics.

Fitzpatric, J., Hill, D., C.R. Maurer, J., 2008. Image registration. In: Sonka, M., Fitzpatric, J. (Eds.), Handbook of Medical Imaging, Volume 2. Medical Image Processing and Analysis. SPIE, Ch. 8, pp. 447–514.

Ghesu, F., Krubasik, E., Georgescu, B., Singh, V., Zheng, Y., Hornegger, J., Comaniciu, D., 2016. Marginal space deep learning: Efficient architecture for volumetric image parsing. IEEE Transactions on Medical Imaging 35 (5), 1217–1228.

Goebel, R., 2017. Spatial Transformation Matrices. BrainVoyager.
URL http://brainvoyager.com/bv/doc/UsersGuide/CoordsAndTransforms/SpatialTransformationMatrices.html

Gonzalez, R., Woods, R., 2010. Digital Image Processing (3rd Edition). Pearon.

Goodfellow, I., Bengio, Y., Courville, A., 2016. Deep Learning. MIT Press, http://www.deeplearningbook.org.

Goshtasby, A., 2005. 2-D and 3-D Image Registration for Medical, Remote Sensin, and Industrial Applications. Wiley-Interscience.

Grau, V., Noble, J., 2005. Adaptive multiscale ultrasound compounding using phase information. Medical Image Computing and Computer-Assisted Interventions 8 (1), 589–596.

Hu, Y., Modat, M., Gibson, E., Ghavami, N., Bonmati, E., Moore, C., Emberton, M., Noble, J., Barratt, D., Vercauteren, T., 2017. Label-driven weakly-supervised learning for multimodal deformable image registration. CoRR abs/1711.01666.
URL http://arxiv.org/abs/1711.01666

Jaderberg, M., Simonyan, K., Zisserman, A., Kavukcuoglu, K., 2016. Spatial transformer networks. CoRR abs/1506.02025.
URL http://arxiv.org/abs/1506.02025

Johannessen, A., 2018. Deep learning based registration of ultrasound volumes. Master's thesis, Norwegian University of Science and Technology.

Ketkar, N., 2017. Introduction to PyTorch. Apress, pp. 195–208.
URL https://doi.org/10.1007/978-1-4842-2766-4_12

Kim, Y.-T., 1997. Contrast enhancement using brightness preserving bi-histogram equalization. IEEE Transactions on Consumer Electronics 43 (1), 1–8.

Nielsen, M., 2015. Neural Networks and Deep Learning. Determination Press, http://neuralnetworksanddeeplearning.com/index.html.

Ning, M., Zhi-an, L., Xu, M., Ya, Y., 2008. Live three-dimensional transesophageal echocardiography in mitral valve surgery. Chinese Medical Journal 121 (20), 2037–2041.

OpenCVTeam, 2015. Histograms - 2: Histogram Equalization.
URL https://docs.opencv.org/3.1.0/d5/daf/tutorial_py_histogram_equalization.html

Rajon, D., Bolch, W., 2003. Marching cube algorithm: review and trilinear interpolation adaptation for image-based dosimetric models. Computerized Medical Imaging and Graphics 27 (5), 411–435.
URL http://www.sciencedirect.com/science/article/pii/S0895611103000326

Rajpoot, K., Grau, V., Szmigielski, J. N. C., Becher, H., 2011. Multiview fusion 3-d echocardiography: Improving the information and quality of real-time 3-d echocardiography. Ultrasound in Medicine & Biology 37 (7), 1056–1072.
URL http://www.sciencedirect.com/science/article/pii/S0301562911002158

Sarvaiya, J., Patnaik, S., Bombaywala, S., 2009. Image registration by template matching using normalized cross-correlation. 2009 International Conference on Advances in Computing, Control, and Telecommunication Technologies, 819–822.

Schers, J., Troccaz, J., Daanen, V., Fouard, C., Plaskos, C., Kilan, P., 2008. 3d/4d ultrasound registration of bone. CoRR abs/0801.2823.
URL http://arxiv.org/abs/0801.2823

Solomon, C., Breckon, T., 2011. Fundamentals of Digital Image Processing: A Practical Approach with Examples in Matlab. Wiley-Blackwell.

Veene, H. D., Bertrand, P., Popovic, N., Vandervoort, P., Claus, P., Beule, M. D., Heyde, B., 2015. Automatic mitral annulus tracking in volumetric ultrasound using non-rigid image registration. In: 2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC). pp. 1985–1988.

Vegas, A., 2016. Three-dimensional transesophageal echocardiography: Principles and clinical applications. Ann Card Anaesth, 35–43.

Vialard, F.-X., Risser, L., Rueckert, D., Cotter, C. J., 2012. Diffeomorphic 3d image registration via geodesic shooting using an efficient adjoint calculation. IJCV (97), 229–241.

Woo, J., Hong, B.-W., Hu, C.-H., Shung, K., Kuo, C.-C., Slomka, P., 2009. Non-rigid ultrasound image registration based on intensity and local phase information. Journal of Signal Processing Systems 54 (1), 33–34.
URL https://doi.org/10.1007/s11265-008-0218-2

Wu, G., Kim, M., Wang, Q., Munsell, B., Shen, D., 2016. Scalable high-performance image registration framework by unsupervised deep feature representations learning. IEEE Transactions on Biomedical Engineering 63 (7), 1505–1516.

Yang, X., Kwitt, R., Styner, M., Niethammer, M., 2017. Quicksilver: Fast predictive image registration - a deep learning approach. arXiv:1703.10908.

Zagrodsky, V., Shekhar, R., Cornhill, J., 2000. Mutual information-based registration of cardiac ultrasound volumes. Proceedings of SPIE - The International Society for Optical Engineering.

Appendix

A.1 Introduction to backpropagation in PyTorch

PyTorch is a open-source machine learning library based on the Lua library *Torch* (Ketkar, 2017). Compared to other machine learning frameworks like *Tensorflow*, the main differences to keep in mind is that PyTorch uses a imperative programming philosophy while Tensorflow uses a symbolic programming philosophy. Imperative programming executes programs line-by-line, meaning both defining and computation is done at the same line in the program. In symbolic programming; on the other hand, definition and computation is done separately. This means that a symbolic program starts by defining a graph, and computation will only be done when the graph is called. Imperative programming makes debugging easier as an error will be raised at the line where the error is encountered, and not at the line where the graph is called. Also the dynamic nature of imperative programs gives the model greater adaptability and thus well suited for research purposes. However, symbolic programming gives increased speed because defining the graph is only done once, and computation can be done multiple times using the same graph. To summarize: imperative programming executes a mathematical function as it stands, while symbolic programming creates a graph that represent the mathematical which can be executed multiple times at high speed.

An important package in PyTorch is the `torch.autograd` package. This package handles the computation of the gradient of the transformed moving image with respect to the target image. All all tensors in PyTorch (`torch.Tensor`) has a `.requires_grad` option (default: false). Enabling this options tells PyTorch to compute the gradient of the output with respect to these tensors. The `torch.autograd` package does this by storing all operation preformed on the tensor from the moment the `.requires_grad` option is enabled until the backpropagation algorithm is call. Every operation in PyTorch has a its own gradient function which is stored in a list and called for in reverse order during the backpropagation. The resulting gradients are stored in `.grad` for all tensors with the `.requires_grad` option enabled. Consider the simple example in where we wish to compute the gradient of C with respect to A:

$$C = AB + A \tag{A.1a}$$

$$\frac{\partial C}{\partial A} = B + 1 \tag{A.1b}$$

For $A = 5$ and $B = 3$ this gives us $C = 20$ and $\frac{\partial C}{\partial A} = 4$. The same example can be implemented in PyTorch:

```
1     >>> A = torch.tensor(5., requires_grad=True)
2     >>> B = torch.tensor(3.)
3     >>> C = A*B + A
4     >>> print(C)
5     tensor(20., grad_fn=<AddBackward0>)
6     >>> C.backward()
7     >>> print(A.grad)
8     tensor(4.)
```

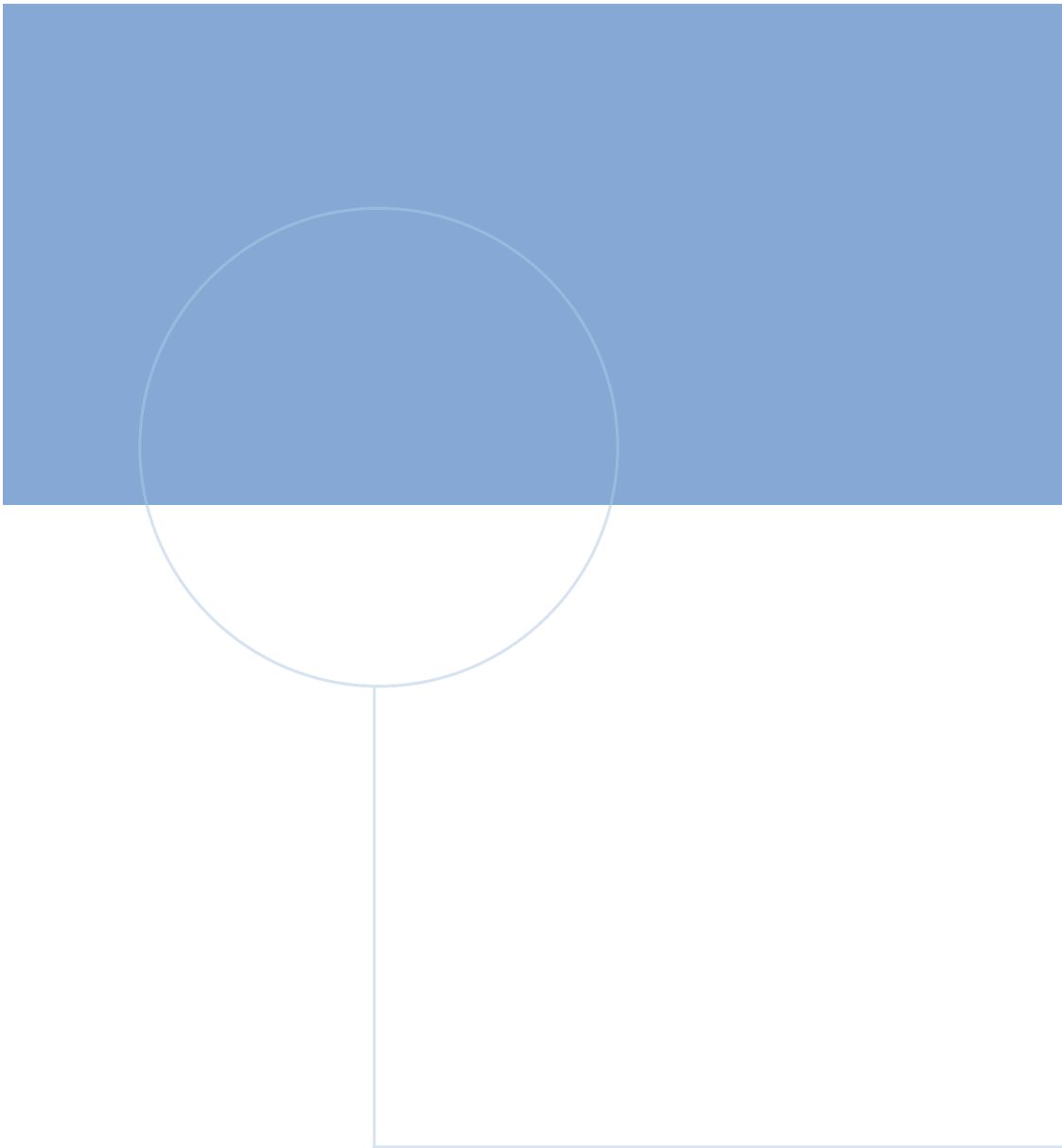
Here we can see from line 5 that

```
grad_fn=<AddBackward0>
```

which indicates that the last operation done on C is an addition. By calling

```
C.backward()
```

the gradient is computed in a backwards manner, starting with the addition followed by the multiplication. The result is identical to what we would expect from Equation (A.1). This functionally allows computation of the gradient of complicated mathematical functions to be done automatically.



NTNU
Norwegian University of
Science and Technology