Graduate thesis

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

Sigurd Vangen Wifstad

Reconstruction of Ultrasound Blood Velocity Fields using Deep Learning

Graduate thesis in Electronic Systems Design and Innovation
Supervisor: Lasse Løvstakken
June 2020

NTNU
Norwegian University of Science and Technology

Sigurd Vangen Wifstad

Reconstruction of Ultrasound Blood Velocity Fields using Deep Learning

Graduate thesis in Electronic Systems Design and Innovation
Supervisor: Lasse Løvstakken
June 2020

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Electronic Systems

Abstract

Recent research suggests deviations from normal flow patterns in the left ventricle can be early signs of heart disease. Accurate screenings of left ventricular blood flow can aid doctors provide the correct treatment for patients before the disease develops, potentially reducing prevalence of cardiovascular disease, which we see is increasing in most of the world. Ever since the 1970s ultrasound technology has been a helpful tool for cardiologists to better visualize and measure the blood flow inside the body, a non-invasive technology which is both easy to use and cheap to produce. Vector Flow Imaging (VFI) is one of the recent advancements in cardiac ultrasound, providing a new promising way of visualizing the blood flow as a velocity vector field. However, VFI measurements are prone to noise and struggle with capturing all the important features in the flow, especially the lateral component, as measuring equipment works best for capturing the radial flow. Hence good regularization and reconstruction techniques are essential for uncovering the full potential of VFI. We propose a deep convolutional neural network (CNN) to perform regularization and reconstruction of time series of ultrasound blood velocity field measurements. The CNN is trained on a data set containing slices of a simulated 4D velocity field of a neonate left ventricle. The simulation applies Blood Speckle Tracking to the Fast Ultrasound Simulation in K-Space (FUSK), with a Computational Fluid Dynamics (CFD) phantom as the target labels. Two model types both based on the U-Net architecture are proposed; one trained on static fields using 2D convolutions (2D U-Net) and one trained on a time series of field frames using 3D convolutions (3D U-Net). The models are initially trained and tested on the simulated data set, and then trained and tested on a small data set of real VFI data of the left ventricle from pediatric patients using a transfer learning approach. Both models show great performance on the simulated data, being able to capture a higher order of detail in the flow dynamics which are lost in the standard reconstruction procedure. When evaluated on the real data, the 3D U-Net outperforms the 2D U-Net in reconstructing missing data, showing that temporal flow information is beneficial for learning from sparse data. The 3D U-Net shows to be an improvement from the standard approach, giving a higher field resolution and hence better capturing important features in the flow. However, the model struggles with reconstructing the lateral component, probably due to a combination of insufficient diversity in the training data and limitations in the loss function to capture perceptually important information. Finally, a Generative Adversarial Network (GAN) is created to see if the introduction of an implicit loss metric can improve the performance. When tested on the simulated data it is shown that the GAN can create realistic looking fields. However, they are far from the ground truth. Hence the GAN needs further development to get a proper assessment of the possible gains of using the GAN approach.

Contents

1 Introduction

1.1 Motivation

Cardiovascular disease (CVD) is a term used to describe all diseases which involve the heart or the blood flow circulation system. An extensive study from 2015 [1] showed that CVD is currently the leading cause for death everywhere outside of Africa. They explained that this is likely to arise from the fact that since the overall life-expectancy is increasing, the prevalence of CVD also increases as these diseases mainly occur to middle aged and elderly persons. Additionally, it was shown that death caused by CVD is most prominent in areas where the people are living long enough to the point where CVD is the most prevalent, but where access to medical and surgical treatment is limited. Interestingly, other studies [2][3] suggest that about 90% of all CVD related death could be prevented. This could be done by identifying *risk factors*, which include deviations from normal heart function, high blood pressure and cholesterol levels, and more. Most of these risk factors can be reduced by medical treatment or surgery. Therefore, even though CVD is a major issue in most of the world, its consequences on mortality can be significantly reduced through proper screenings and treatment.

Several types of CVD are believed to be caused by abnormalities in the left ventricular function. The left ventricle (LV) plays a central role in the cardiac cycle. The LV is the biggest chamber in the heart and serves the purpose of pumping oxygenated blood through the aorta and into the body. In the processes of blood filling and ejection, *vortices* are present in the blood flow. A vortex is a well-known phenomena in fluid dynamics which describes circular, inward curling flows. Recent research suggests that LV vortex characteristics might act as risk factors, showing early signs of CVD [4]. Vortices are naturally occurring in the LV. However, vortices are dynamical and instable structures and therefore prone to significant changes in their behavior when exposed to minor changes in the environment. A vortex can easily cause turbulence in the flow and modulate the transfer of energy and momentum, which could alter the balance between the blood flow and the heart chamber, ultimately leading to complications. Therefore, analyzing the LV vortex formation and how it deviates from the normal pattern could be an early indicator of disease. This could allow interventions which prevent disease from occurring at all. Although screening for LV vortex abnormalities can provide life-saving information, the procedure is greatly limited by technology. In order to make certain decisions about whether or not a vortex shows signs of a dysfunctional LV, the cardiologist needs an accurate visualization of the details in the flow field. As vortices are fairly unpredictable, complicated and dynamical phenomena, we require very accurate measurement technology to capture them to the extent we need in order to use them in diagnosis.

The technology which gives the best visualization of intraventricular flow is Phase contrast magnetic resonance imaging (PC-MRI) [5]. PC-MRI is a type of MRI which is used for measuring blood flow velocities. The technique involves inserting the patient into an MRI machine and measuring the phase of the acquired MR signal, which is directly related to the velocity of moving particles in the blood stream. PC-MRI gives astounding 4D (3D plus time) measurements of the blood flow, which both capture the cardiac walls and the blood flow velocity field with high resolution and accuracy. However, MRI machines are expensive and require a lot of time to perform measurements of one patient. This is a challenge when a large demographic might need screening for CVD, especially in areas with limited resources. An alternative to PC-MRI is *cardiac ultrasound*, also known as *echocardiography.* Cardiac ultrasound creates images from acoustic waves transmitted into and received from the cardiovascular system. The received signal can be used to visualize both the anatomy of the heart and the blood flow within. Table 1 shows a comparison between cardiac ultrasound and MRI machines [6][7]. Cardiac ultrasound machines are cheaper, as well as quicker and easier to use. This could make cardiac ultrasound available to a large demographic, as compared to PC-MRI, hence potentially increasing the capability to diagnose many more people with CVD. However, the procedure is limited by its image quality, which can be insufficient for certain applications. In particular, the visualization of vortices is difficult. Capturing the fine details of the flow in both space and time is challenging using only a ultrasound probe, as the signal strength is

Table 1: Comparison of important factors regarding cardiac ultrasound machines and MRI machines.

	Cardiac ultrasound machine	**MRI machine**
Image quality	Limited	Great
Acquisition time	A few seconds / real time	20-60 minutes
Invasive	No	No
Price	$\$20,000 - \$100,000$ [8]	$\$150,000 - \$3,000,000$ [9]
Usability	Easy to use	Requires radiologists
Size	Fits in your pocket [10] – fits on a trolley [11]	Occupies an entire room

highly dependent on the direction of the ultrasound beam.

In recent years there has been extensive research on *Vector Flow Imaging* (VFI) which aims to measure and visualize the blood flow velocity as a vector field overlaid the ultrasound B-Mode image. These methods try to work around the limitations of the ultrasound device, combining different measurements to extract as much information about the velocity field as possible. Some methods worth mentioning are *Vector Doppler*, which combines several doppler velocity measurements from different directions, *Blood Speckle Tracking* (BST) [12], which uses block matching to track speckle patterns in the blood flow, and *Vector Flow Mapping* (VFM) [13], which combines doppler measurements with velocity information from the cardiac walls. All these methods have shown promising results for visualizing vortices, however they are all prone to noise and loss of information. Hence both *regularization* and *reconstruction* is needed in post-processing to get results of sufficient quality. Studies show that imposing domain knowledge about fluid dynamics in combination with the measurements can be beneficial for recreating realistic looking velocity fields. This is model based regularization and reconstruction, where we try to create a model for what we are trying to observe, and then fit the measured data to the model. In this way we combine raw measurements with our physical understanding of the system we are measuring.

Model based data processing is beginning to become very popular in all sorts of scientific research and applications. In particular, we see a big increase in the use of so called *Artificial Intelligence* (AI). This is a broad and vaguely defined term, but it is usually used to describe algorithms that make decisions without the need of explicit programming. AI has been shown to produce results which greatly improve upon what previously was known to be possible in many fields of science. It has even been dubbed *The AI revolution* [14], comparing it to the great impact on society the industrial revolution had in the 19th century. Especially *Machine Learning*, a class of algorithms which learn to do inference from large datasets, has shown themselves to be very useful in many applications where there exists an abundance of data. Machine learning techniques are based on creating models with a large amount of parameters, and then optimizing these parameters to a *training* data set. The algorithms demand large data sets and a lot of computing power to be trained properly, but this is becoming less of an issue as advancements in hardware keeps decreasing computation time and memory limitations. We also see that these techniques are beginning to find their way into medical research and technology. For medical image processing in particular, we see an increasing use of *Convolutional Neural Networks* (CNNs). CNNs are a class of machine learning models which consist of networks of convolutional operations, which essentially can be viewed as image filters when applied to images. CNNs are believed to be similar in function to the visual human cortex [15], breaking down the visual input into individual features. This feature extraction is done automatically through training the model with data, similar to how us humans are trained to recognize the features of what we see through experience. Putting the biological analogy aside, the success of CNNs are explained by their ability to learn the optimal features of the data directly, as opposed to leaving it to the designer to decide which features are important. This removes the constraint of having to carefully choose a model which perfectly suits the problem. Instead, we only need to impose a rough model structure, which is then fine-tuned to fit the data distribution. This is beneficial when working with problems which are very complex and involve large data sets that contain a lot information which is hard to analyse.

CNNs are currently being extensively researched in the context of cardiac ultrasound. Smistad [16] described the potential of using CNN models for segmentation of objects in ultrasound images in an intraoperative setting. Gilbert et. al. [17] showed how CNNs could be used for accurate automatic LV dimension measurements. Senouf et. al. [18] used a CNN to achieve a super-resolution frame rate of cardiac B-Mode images. However, little research has been done on using CNNs for VFI measurement reconstruction, as this research topic is mainly dominated by physically based models. There still might be an advantage to using machine learning and CNNs for VFI measurement reconstruction, as:

- Trained CNNs are significantly faster than physically based models since data fitting is done beforehand.
- There is no need for imposing domain knowledge on the models. Finding good regularization terms grounded in physics is limited to our understanding of the physics.
- Physical model based approaches need boundary conditions. These have to be measured and hence introduce measuring uncertainties into the system.

With this in mind, we might see that CNNs also can be useful for blood flow visualization in cardiac ultrasound machines. If these methods can achieve results which are comparable to, or even better than those of the state of the art techniques, the biggest gain will be the reduced computation time. Pretrained CNN algorithms can be implemented efficiently in hardware, allowing for the production of cheap cardiac ultrasound machines which could be able to quickly and accurately measure and visualize blood flow. In addition, it is of interest to investigate whether or not a CNN can learn the underlying physical aspects of flow out of curiosity. This might help us understand more about blood flow, as well as the capability of CNNs.

1.2 Problem Statement

This thesis aims to apply CNNs to perform regularization and reconstruction of ultrasound blood flow velocity measurements. The algorithms are trained and tested on both simulated flow data, and real flow data from ultrasound blood flow velocity measurements of the left ventricle. The results are compared with those from the state of the art. In particular, the following tasks will be addressed:

- Generate a dataset of realistic simulated ultrasound blood flow measurements using computer simulations and VFI techniques.
- Train CNN models to reconstruct the simulated data and evaluate their performance.
- Further train the models and evaluate them on a dataset of real ultrasound blood flow measurements from the left ventricle of pediatric patients.
- Explore the possibilities and limitations of the proposed models, namely variations of the U-Net and a Generative Adversarial Network (GAN).

2 Background

2.1 A Brief History of Ultrasound Technology in Cardiology

The fundaments of ultrasound technology were laid in the late 19th century, most notably in 1877 when the "the Theory of Sound" was published by Lord Rayleigh, and in 1880 when the piezo-electric effect was discovered by Pierre Curie and his brother Jacques [19]. The first technological applications of these scientific advancements were the development of underwater sound navigation and ranging (SONAR) systems, largely motivated by the Titanic sinking in 1912. The application of high frequency sound waves (ultrasound) to medical imaging first truly saw the light of day in 1966 when there was a push from the industry, as well as research centers around the world, to commercialize experimental ultrasound technology to be sold to hospitals. This quickly made ultrasound imaging equipment available in hospitals and proved itself to be an important tool in several fields of medicine. One of these fields was the field of *cardiology*, concerning diagnosis and treatment of disease in the heart and blood flow circulation system. The first commercial breakthrough in ultrasound for cardiology was the development of the Pulsed Echo Doppler Flowmeter (PEDOF) [20], which was developed by Bjørn Angelsen between 1973 and 1976 in relation to his PhD thesis at NTH. This apparatus measured the velocity of blood flow in the heart and other large cavities by the means of doppler velocity measurements. At the time, there were no similar products available, hence PEDOF was quickly picked up by researchers and hospitals alike to be used for blood flow measurements. In 1977, the Horten based company Vingmed joined the project as an industrial contact, and has ever since been actively focusing on the development of ultrasound technology for cardiology. In 1998 Vingmed was bought by GE Healthcare, and from that point on grew substantially. Today they are the world's biggest company for the development of cardiac ultrasound. With a close contact to the industry, cardiac ultrasound research has been ongoing with researchers at NTNU and St. Olavs hospital in the front line. One of the major advancements was the development of *color flow imaging*, which was able to display blood flow superimposed on B-mode ultrasound images, showing the blood flow magnitude and direction as a color gradient. This technology was largely developed by researchers at St. Olavs hospital and NTNU in the 1980s, and is still being used today [21]. Recent advancements led by the same institution have created *vector flow imaging* (VFI), aiming to further improve visualization of multidirectional blood flow. Technologies worth mentioning are *vector doppler* [22], based on triangulation of several doppler measurements, and *blood speckle tracking* [23], based on tracking patterns of scatterers in the blood stream to estimate the blood flow velocity. These technologies are still being researched today, and combined with the overall evolution of technology in the world, ultrasound based visualization and diagnostics in cardiology keep on improving, and are prospected to be even more accessible and helpful in the near future [24].

2.2 Cardiac Blood Flow

2.2.1 The Cardiac Cycle

The heart is essentially a pump which pumps blood through the body, delivering oxygen and other important chemicals to every cell, thus being essential for our survival. It is an intricate system which consists of several parts and goes through several phases of contraction and relaxation. The heart consists of four chambers called the *left and right atria* and the *left and right ventricles.* The atria are connected on top of the ventricles and act as receiving chambers which push blood into the ventricles. The ventricles perform the primary pumping function. After the ventricles are filled with blood, signals from the nervous system make the heart muscle surrounding the ventricles contract, ejecting the blood out of the heart and into circulation in the body [25]. Figure 1 illustrates the cardiac cycle. It begins with all the chambers being relaxed. Then, during *atrial systole* the atria contract, pushing blood into the ventricles. This is followed by both *atrial diastole*, where the atria relax again, and by *ventricular systole*, where the ventricles begin to contract. During ventricular systole, the ventricles reduce in volume and followingly eject the blood out through the aorta and pulmonary artery. After ejection, *ventricular diastole* begins as the ventricles relax. First, during the *isovolumic ventricular relaxation phase*, the semilunar valves between the arteries and ventricles close to prevent backflow from the arteries due to the pressure drop when the ventricles expand. Then, in the *late ventricular phase*, the blood flows back from the major veins, into the atria, and through the mitral valves connecting the atria to the ventricles as the pressure continues to drop. Blood fills both atria and ventricles as all chambers relax, thus completing the cardiac cycle [26]. The main events of this cycle can be summarized for any of the chambers as consisting of *systole* (contraction) and *diastole* (relaxation). It is during these events the filling and ejection of the chambers occur. In particular, the left ventricular systole and diastole are of interest. During diastole, we expect a vortex known as the *left ventricular diastolic vortex*, which might prove to uncover abnormalities in the cardiac function [27].

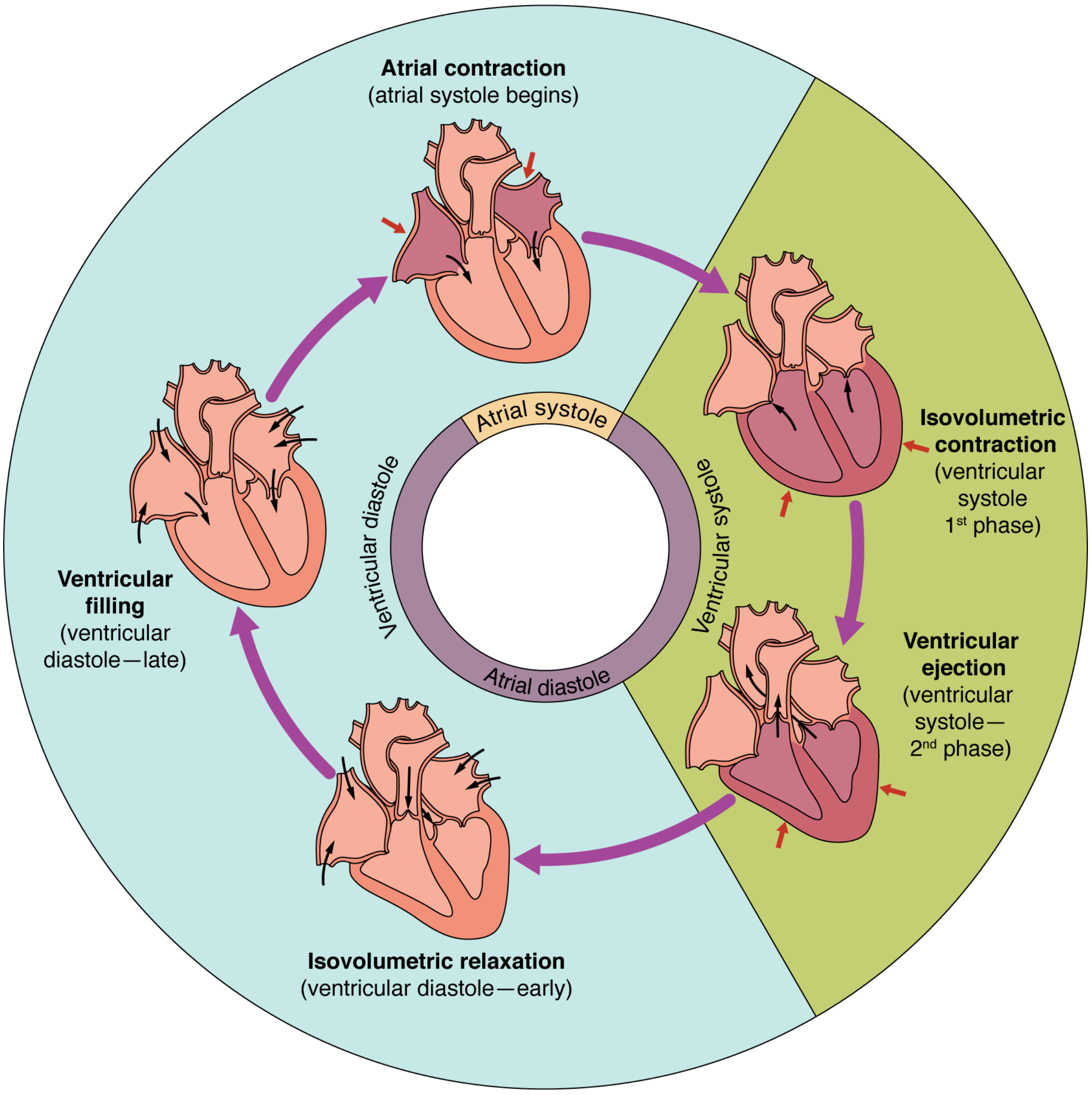


Figure 1: "Overview of the Cardiac Cycle: The cardiac cycle begins with atrial systole and progresses to ventricular systole, atrial diastole, and ventricular diastole, when the cycle begins again. Correlations to the ECG are highlighted." - The illustration and the caption are taken from [26].

2.2.2 A Mathematical Model of Blood Flow

This section is taken and adapted from [28] *to be used in the context of this thesis.*

Blood, like any other fluid, has been shown to be quite accurately described by the *Navier-Stokes Equations*, which were developed in parallel by Claude-Louis Navier and George Gabriel Stokes in the mid 19th century [29, p. 6-7]. These equations are given in (1) and (2), respectively known as the *Incompressible Navier-Stokes equation* and the *Incompressible continuity equation* [29, p. 468].

$$\rho\frac{D\boldsymbol{v}}{Dt} = -\nabla P + \rho\boldsymbol{g} + \mu\nabla^2\boldsymbol{v} \tag{1}$$

$$\frac{\partial \boldsymbol{v_x}}{\partial x} + \frac{\partial \boldsymbol{v_y}}{\partial y} + \frac{\partial \boldsymbol{v_z}}{\partial z} = 0 \tag{2}$$

Here, $\boldsymbol{v}$ denotes the flow velocity field, P is the pressure, ρ is the fluid mass density, μ is the fluid viscosity and $\boldsymbol{g}$ is the gravitational acceleration. The Navier-Stokes equations do not have a known analytical solution. For real-world problems, the equations are solved numerically by what is generally called *computational fluid dynamics* (CFD) [29, p. 880]. When solving the equations the *no-slip boundary condition* [29, p. 498] is usually applied, providing a complete solution. The no-slip condition states that a fluid does not slip on the walls of the container containing the fluid, hence the tangential component of the velocity field along the wall must be zero. The no-slip condition is widely accepted as it has been verified extensively through empirical studies. Recent studies suggest it might not hold for flow within some nanometers of the wall, however for larger systems, such as the heart, it is generally accepted.

2.3 Ultrasound Technology for Blood Flow Measurements

2.3.1 Data Acquisition

Ultrasound data acquisition begins with emitting acoustic waves through a medium from a probe and receiving the backscattered signal from the medium. The probe is an array of ultrasonic transducers that generates an acoustic pulse which is transmitted through the body. Because of differences in density of the different materials, different regions of the focused area will reflect the acoustic waves to varying extents in the boundaries between materials. This generates an image where contours are highlighted, what is known as *B-mode ultrasound*. A B-mode image is a cross-sectional image showing blood and tissue of a probed area in a patient.

The transmitted waves are generated by transducers organized in arrays on a probe. There are different types of probes which are used for different medical applications. The most common ones are *linear arrays*, *curved linear arrays* and *phased arrays*. A linear array is made up of transducers in line on a relatively wide flat probe head, creating a linear scan image. A curved linear array is made from transducers placed on a convex probe head, which provides a wider field of view. A phased linear array uses transducers placed on a shorter flat probe head, but the phase of each transducer is controlled to form a wide fan shaped beam. This makes a wide scan available while still keeping a short probe, making certain areas like in between the ribs, easier to image [30, p. 16-17].

Images are generated by emitting pulses of acoustic waves. When using *pulsed wave ultrasound* pulses of ultrasonic waves are emitted in short bursts, often called *shots*, with a high *Pulse Repetition Frequency* (PRF) in the kHz range. The variability within one packet of shots can be used to enhance image quality, or extract information like blood flow velocity. Each packet of pulses is repeated over time with a certain *frame rate*, generating a time series of frames. The received acoustic signal is sampled at sampling rate f_s and stored as *IQ data*, according to (3).

$$
\begin{aligned}
A\cos(2\pi\frac{f_0}{f_s}n+\phi) &= A\cos(\phi)\cos(2\pi\frac{f_0}{f_s}n) - A\sin(\phi)\sin(2\pi\frac{f_0}{f_s}n)\\
&= I\cos(2\pi\frac{f_0}{f_s}n) - Q\sin(2\pi\frac{f_0}{f_s}n)
\end{aligned}
\tag{3}
$$

(3) shows that the amplitude A and phase ϕ of the received signal can be stored as two numbers I and Q, usually represented as the complex number $I + jQ$.

2.3.2 Clutter Filtering

When measuring blood flow velocities using ultrasound we want to avoid measuring the velocities of all the moving tissue which is not blood. All of this tissue is what we call *clutter.* Since blood generally moves at high velocities relative to the surrounding tissue, the interfering signal from the clutter can be removed by high pass filtering the received signal in the time domain [31]. However, this yields a new problem of cutting off the lowest velocities in the blood signal which fall below the cutoff velocity of the filter. Figure 2 shows a simulation where the effects of clutter filtering is illustrated. In the leftmost image we see velocity measurements of some IQ data at a given time frame. The rightmost image shows the velocity measurements after the IQ data has been clutter filtered. Here we can see that the clutter filter has the unfortunate side effect of removing the lowest velocity areas of the signal. This makes the background noise "bleed through" in these regions and information about the blood flow is lost. However, clutter filtering is essential for ultrasound imaging as interfering signals would make the blood signal very difficult to detect without it.

2.3.3 Color Flow Imaging

A common method for VFI is through a technique called *Color Flow Imaging* [32]. The technique is based on the *doppler effect*, which is the phenomena of how waves with center frequency f_0 emitted or reflected from objects at an angle θ experience a frequency shift f_d, proportional to the object's radial velocity v_r. It can be expressed as shown in (4)

$$
v_r = \frac{c}{2f_0\cos\theta} f_d, \tag{4}
$$

where c is the speed of sound in the material. When transmitting ultrasonic waves into the blood stream the waves are reflected by moving particles, hence creating a doppler shift in the received signal. In this way the velocity of the blood flow can be measured. The doppler velocity measurements are visualized through displaying color overlaid the B-mode image, where the intensity increases with the velocity magnitude. The direction of the flow is indicated through different colors, thereby the name color flow imaging. Usually red is used to visualize flow moving towards the probe, while blue is used for flow moving away from it. Figure 3 shows an example of left ventricular flow measured with color flow imaging. The method is simple in concept, and is widely used in cardiac ultrasound technology today. However, it has some limitations. Firstly, it is only possible to measure the velocity along the radial axis of the transmitted beam. This makes reconstruction of the entire velocity field difficult, as the field tends to vary with all three spatial coordinates. Secondly, doppler measurements are prone to *aliasing.* From the *Nyquist sampling theorem* it follows that there exists an upper bound v_{nq} on the measurable velocity, as described in (5).

$$
v_{nq} = \frac{\mathrm{PRF}}{4f_0} c. \tag{5}
$$

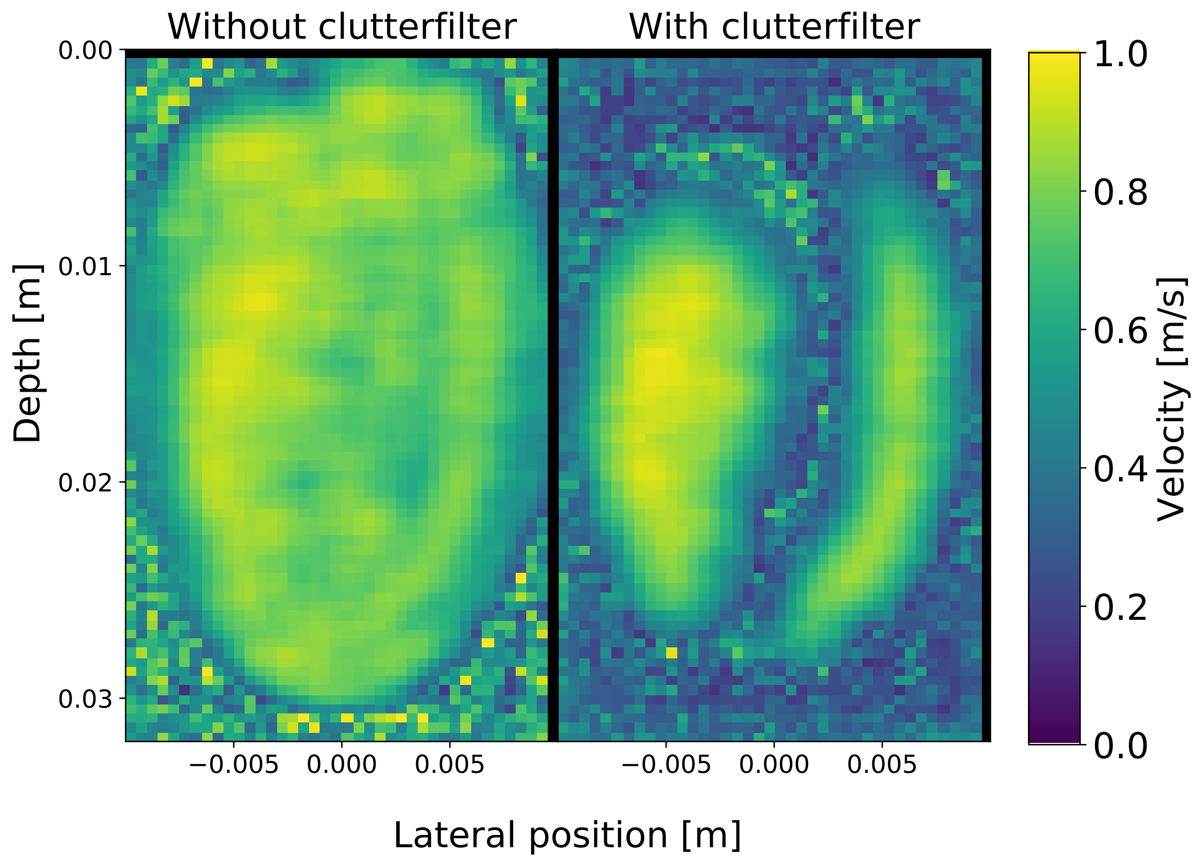


Figure 2: Simulated blood flow measurements (FUSK) with clutter filtering respectively applied and not applied to the IQ data before VFI is performed. The images are showing the velocity magnitude of an apical slice of the left ventricle during diastole.

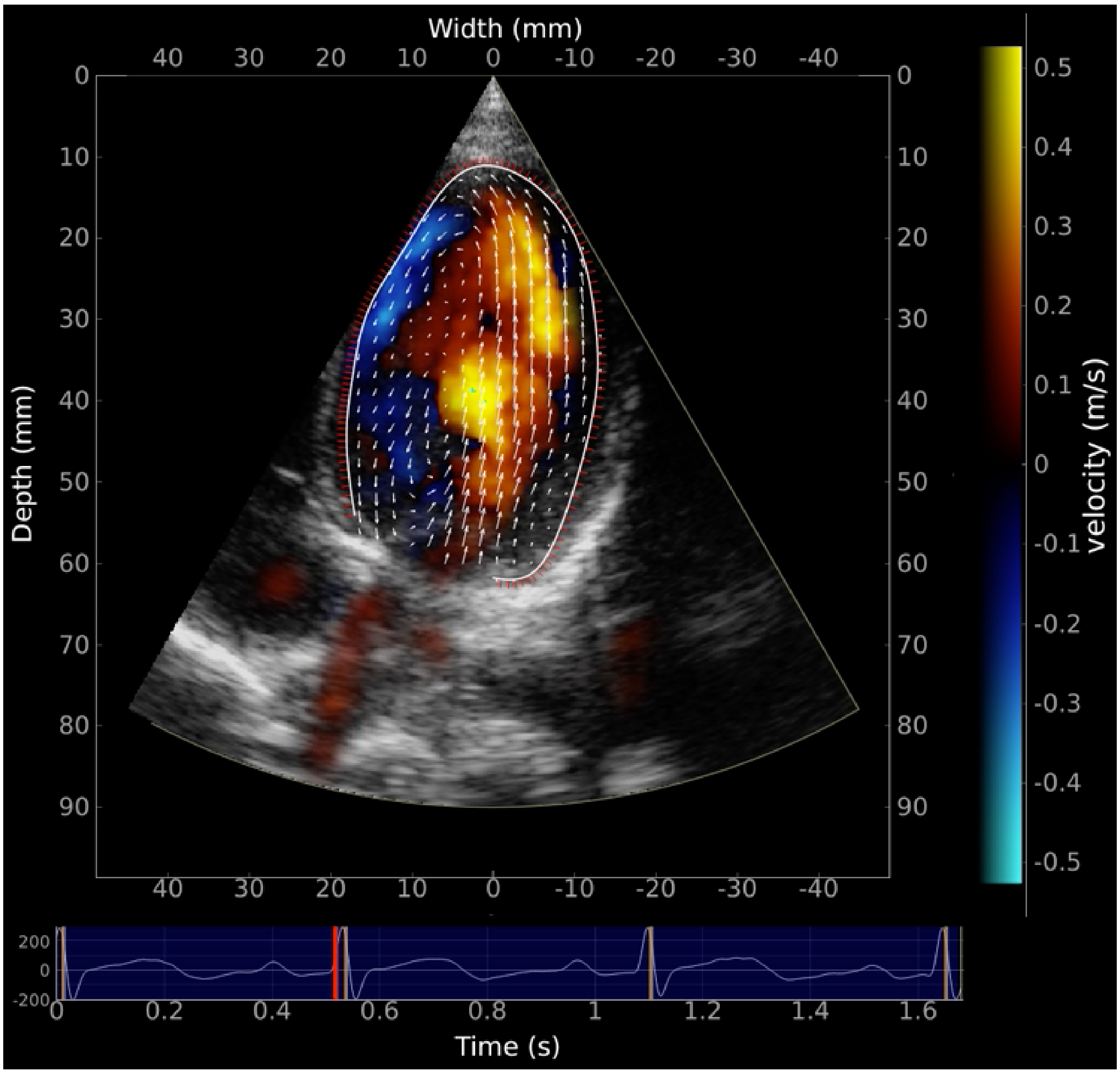


Figure 3: A frame of ultrasound data acquired from measurements of the left ventricle of a pediatric patient. We can observe the B-mode image as the grayscale image showing the contours of the cardiac walls. The color flow is overlaid the B-mode image, showing the radial flow as a red and blue gradient illustrating positive and negative velocities. Additionally, VFI is applied, showing the radial-lateral blood flow velocity vector field inside the heart chamber, which is segmented based on the B-mode information. The graph along the bottom shows the cardiac cycle time series with the frame's position marked with a red vertical line.

Velocities above v_{nq} cannot be reconstructed perfectly. However, they can be reconstructed with some loss of information. This process is known as *antialiasing*. After antialiasing is applied, the radial velocity v_r at each measured point is illustrated as a color superimposed the B-mode image, thereby the name color flow imaging. The color is usually within a range from blue to red or yellow, where the blue colors are mapped to negative velocities, while the red colors are mapped to positive velocities. An in-vivo example showing color flow imaging in action is shown in Figure 3.

2.3.4 Blood Speckle Tracking

Blood Speckle Tracking (BST) is another method for VFI. The method is based on applying block matching to track blood speckle patterns. Blood speckle patterns are patterns which occur in the blood filled areas of ultrasonic B-mode images due to backscattered echoes from random scatterers in the blood. These patterns remain more or less constant over short periods of time as the blood and tissue moves, which makes them useful for tracking [23]. The block matching procedure consists of computing the correlation of a kernel from the image acquired from one shot to a search grid in the preceding image. This would uncover the most likely position $\boldsymbol{r_0} + \boldsymbol{\Delta r}$ to which the kernel has moved to between shots. This can be expressed as in (6)

$$\Delta \boldsymbol{r} = \underset{\Delta \boldsymbol{r}}{\operatorname{argmax}} \sum_{\boldsymbol{r}}^{K_{dims}} \mathrm{R}(\boldsymbol{r_0}, \boldsymbol{r_0} + \Delta \boldsymbol{r}), \tag{6}$$

where R is some correlation metric function and $\boldsymbol{r_0}$ is the center point of the search grid. The velocity $\boldsymbol{v(r_0)}$ is then estimated as in (7)

$$\boldsymbol{v}(\boldsymbol{r_0}) = \frac{\Delta \boldsymbol{r}}{\mathrm{PRF}}, \tag{7}$$

where PRF is the pulse repetition frequency. BST could be performed in two or three dimensions. Often three dimensional measurements are performed using two dimensional BST, combined with doppler measurements in the radial direction, creating a hybrid method which tends to give lower variance in the radial direction [12].

2.3.5 State-of-the-art ultrasound blood flow velocity field reconstruction

As VFI techniques are prone to noise, regularization and reconstruction is needed in post-processing in order to make the measurements interpretable when visualized. The conventional approach is to use a *Gaussian filter* [33, p. 154], which smoothens the flow data along both spatial and temporal axes with a Gaussian function, as described in (8).

$$\begin{aligned} g(\boldsymbol{w}) &= \frac{1}{\sqrt{2\pi^2|\Sigma|}} \exp -\frac{1}{2}\boldsymbol{w}^T \Sigma^{-1} \boldsymbol{w}, \\ \boldsymbol{w} &= \begin{bmatrix} x \\ z \\ t \end{bmatrix}, \\ \Sigma &= \begin{bmatrix} \sigma_x & 0 & 0 \\ 0 & \sigma_z & 0 \\ 0 & 0 & \sigma_t \end{bmatrix}. \end{aligned} \tag{8}$$

$g(\boldsymbol{w})$ denotes the Gaussian smoothing function at point $\boldsymbol{w}$, describing respectively the lateral, radial and temporal positions x, z and t. σ_x, σ_z and σ_t are the corresponding standard deviations for each axes. Increasing the standard deviation will increase the severity of the blur applied along the specific axis. With an increased blur more of the high frequency noise can be reduced. However, the smoothing will also affect the blood flow signal and will erase more details in the signal for a high blur.

State-of-the-art approaches are model based and try to incorporate physical constraints as well as measured data into the models. One of these are *Intraventricular Vector Flow Mapping* (iVFM), proposed by Assi et. al. [13] which can reconstruct 2D velocity fields in the left ventricle. This model impose a solution to least squares constraint optimization problem in polar coordinates (r, θ) with regularization terms. In particular, they solve the optimization problem shown in (9).

$$\begin{aligned} \boldsymbol{v} &= \underset{\boldsymbol{v}}{\operatorname{argmin}} \boldsymbol{J}(\boldsymbol{v}) \\ \boldsymbol{J}(\boldsymbol{v}) &= \boldsymbol{J_0}(\boldsymbol{v}) + \lambda_1 \boldsymbol{J_1}(\boldsymbol{v}) + \lambda_2 \boldsymbol{J_2}(\boldsymbol{v}) + \lambda_3 \boldsymbol{J_3}(\boldsymbol{v}). \end{aligned} \tag{9}$$

Here, $\boldsymbol{v} = [v_r, v_\theta]$ is the velocity field and $\boldsymbol{J_0}$ is the mean-squared-difference between the doppler estimate and v_r, ensuring a good fit to the data, $\boldsymbol{J_1}$,$\boldsymbol{J_2}$ and $\boldsymbol{J_3}$ are the regularization terms, which provide a more physically realistic solution. $\boldsymbol{J_1}$ describes the null-divergence constraint, which comes from the Incompressible continuity equation, as described in 2.2.2. $\boldsymbol{J_2}$ are the no-slip boundary conditions along the cardiac walls, and $\boldsymbol{J_3}$ is a smoothness constraint which ensures a smooth velocity field. The regularization weights $\lambda_1, \lambda_2, \lambda_3 \geq 0$ are chosen automatically through hyperparameter optimization. iVFM proved itself to give quite accurate results in simulations, as well realistic looking fields from in-vivo tests. However, they also state that the current method cannot be expanded to 3D/4D velocity measurements, hence limiting its use.

Another advancement in model based regularization and reconstruction of VFI data is by B-spline interpolation [34, p. 60]. Splines are functions which fit curves to a set of discrete query points $\boldsymbol{w}$. Any spline function $s(w)$ of degree n can be represented as an affine combination of the B-splines (basis splines) of the same degree, as described in (10).

$$s(\boldsymbol{w}) = \sum_i \boldsymbol{c_i} N_i^n(\boldsymbol{w}). \tag{10}$$

Here, $N_i^n(\boldsymbol{w})$ are the B-spline functions of degree n with minimal support and $\boldsymbol{c_i}$ are the control points for the curve. Gomez et. al. [35] expanded this theory to a scalar spline field of arbitrary finite dimension and used it to do reconstruction of 4D blood flow measurements from the left ventricle in pediatric patients. The proposed method was to combine estimated wall motion from B-mode image sequences with 1D doppler measurements from different beam directions using a least squares solution of a B-spline based linear system to find the optimal control points. Using this framework they were able to compute control points with a regularization term dependent on the velocity field divergence, as shown in (11).

$$\boldsymbol{C} = \left[A_{proj} + \frac{\lambda}{1-\lambda} A_{div} \right]^{-1} \boldsymbol{b}, \tag{11}$$

where $\boldsymbol{C}$ are the control points, $\boldsymbol{b}$ is the data vector, A_{proj} is the least squares projection matrix, A_{div} is the divergence regularization matrix and $\lambda \in [0, 1)$. $\boldsymbol{b}$ is dependent on the doppler and wall motion measurements, while A_{proj} is dependent on the vector doppler beam directions. A_{div} is proportional to the velocity divergence $\boldsymbol{\nabla} \cdot \boldsymbol{v}$, hence enforcing the incompressibility of blood flow. This incorporates the incompressible continuity equation into the solution. This aims to give a more physically realistic description of the flow. However, the regularization term is not able to incorporate the incompressible Navier-Stokes equation, hence not capturing all the known flow properties. This is due to the constraints of the linear system which demand linear functions. Because of the non-linear terms in the incompressible Navier-Stokes equation, it cannot be described in the linear system.

Grønli et. al. [36] expanded on the method by implementing it in the Tensorflow [37] framework for fast GPU accelerated tensor differentiation and spline calculation. Tensorflow is usually used for machine learning, and is specifically designed for fast constraint optimization of non-linear functions using gradient descent. Using this framework, Grønli et. al. were able to implement the method imposed by Gomez et. al. without the constraints of a linear system, hence allowing for arbitrary functions to be used to compute the control points. This way, any set of physical constraints can be imposed on the system. Followingly, they were able to impose both a weak compressibility condition and the incompressible Navier-Stokes equation, as well as no-slip boundary conditions and the condition that the radial component of the solution coincides with the doppler measurements. Ultimately, this gives a solution which agrees with our understanding of blood flow to a large extent, while still being efficiently computable. The framework was shown to give sufficient performance for bedside applicability requirements in an in-silico analysis and realistic results in an in-vivo demonstration.

2.4 Machine learning

Machine learning describes a wide range of algorithms which learn and make decisions from empirical data. In recent years these kinds of algorithms have shown themselves useful for many different applications, often outperforming the classical methods. The algorithms are all based on learning from a training dataset, in a process which is often dubbed "training" or "model fitting". Machine learning is often divided into *supervised* and *unsupervised learning.* In supervised learning, the training data is labeled. The model then tries to learn the correlation between the labels and the data such that it can predict the labels of new, unseen data. In unsupervised learning there are no such labels. The model then tries to learn the features of the data and make decisions for new unseen data based on the learnt features. Additionally, the models can be either *discriminative* or *generative.* Discriminative models use the training data to create a mapping from the input data to the output labels, hence discriminating the input data into different types of output data. These models are entirely deterministic. Generative models, on the other hand, are probabilistic models which learn the underlying distribution of the input data features. They can then generate new, unseen data by sampling from this distribution.

Deep learning describes a specific type of machine learning algorithms which are built up from many different nonlinear mapping nodes, connected to each other in a network. These nodes are often called "neurons", because of the apparent similarities to the structure of neural networks in the human brain. The terms "Deep Learning", "neural networks", and "deep neural networks" are often used interchangeably. "Deep" usually refers to neural networks which have several layers of neurons in cascade. The more layers, the deeper the network. For deeper networks the amount of learnable model parameters increases, allowing for more complex models which can perform more complex tasks. In fact, according to the Universal Approximation Theorem [38], if one allows for a neural network with infinitely many learnable parameters, then it can approximate *any* function. This implies that a neural network could, in theory, perform any task given enough data and modelling complexity. With all the different applications of Deep learning we see in research and in the industry today it might seem like this might in fact be valid in practice as well, as Deep learning is beginning to replace all kinds of other problem solving mechanisms.

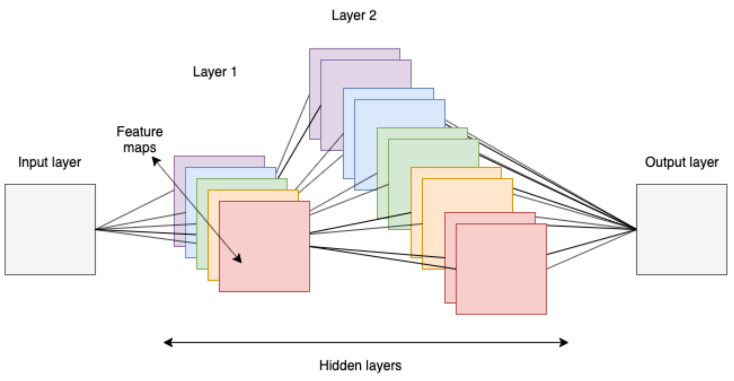


Figure 4: An example CNN structure with two hidden layers. Layer 1 has five feature maps, while layer 2 has two feature maps.

2.4.1 Convolutional Deep Neural Networks

The following section is taken from [28] *and adapted to the context of this report.*

Convolutional neural networks (CNNs) are Deep Learning models which apply convolutional operations at each neuron. Conceptually CNNs work by breaking down the input images into a set of *feature maps* which highlight different features of the image. For a deep CNN, the deeper layers extract sub-features of the feature maps, making deeper networks able to encode more information about the input data. A CNN is essentially a cascade of linear convolutions, but with an added bias term and a nonlinear activation function at each layer of the cascade. The behavior of this *convolutional layer* is defined mathematically in the *forward pass* equation (12).

$$\begin{aligned} \boldsymbol{z}^l &= \boldsymbol{b}^l + \boldsymbol{w}^l * \boldsymbol{a}^l \\ \boldsymbol{a}^{l+1} &= h(\boldsymbol{z}^l) \end{aligned} \tag{12}$$

Here $\boldsymbol{a}^l$ is the *activation* at *layer* l. $\boldsymbol{b}^l$ and $\boldsymbol{w}^l$ are respectively the *bias* and *shared weights* at layer l. h denotes the *activation function*. Each layer can have a number of parallel activations with different parameters. The activations are then often called feature maps, as each activation correspond to different features of the input data. $\boldsymbol{b}^l$ and $\boldsymbol{w}^l$ are learnt parameters which are optimized based on the *loss function* L [39, chap. 6]. For regression problems, a common loss function is the *mean-squared-error* (MSE). This is computed on the predicted output tensor $\hat{\boldsymbol{y}}$ and the target tensor $\boldsymbol{y}$ as in (13).

$$L = \frac{1}{N} \sum_{x_0, x_1, \dots, x_{n-1}} (\hat{\boldsymbol{y}} - \boldsymbol{y})^2, \tag{13}$$

where the summations is across all n variables $(x_0, x_1, ..., x_{n-1})$ of the tensor with a total of N points across the n dimensions, producing a scalar value for L. L is minimized through gradient descent, by what is known as the *backpropagation equations* [40, p. 976], expressed for 2D CNNs in (14). The equations for 3D CNNs are similar.

$$\begin{aligned} \delta^l_{x,y} &= \frac{\partial L}{\partial z^l_{x,y}} \\ \frac{\partial L}{\partial w^l_k} &= \boldsymbol{\delta}^l_k * \operatorname{rot180}\left(\boldsymbol{a}^{l-1}\right) \\ \frac{\partial L}{\partial b^l} &= \sum_{x,y} \delta^l_{x,y}, \end{aligned} \tag{14}$$

where $\delta^l_{x,y}$ is the backward propagated loss at layer l for position (x, y) in the image. The index k denotes the feature map index at layer l. rot180 denotes a rotation by 180 degrees. All shared weights and biases are then updated through gradient descent. The activation function h can be any nonlinear function. Deciding which one to use is ultimately a design choice which alters the performance of the model in certain ways. A handful of commonly used activation functions [41] are:

- **Rectified Linear Unit (ReLU)**, shown in (15). The ReLU is a popular choice which speeds up learning and is computed quickly, but is prone to causing the problem of certain neurons never activating, known as the "dying ReLU problem".
- **Leaky Rectified Linear Unit (LReLU)**, shown in (17). The LReLU aims to solve the "dying ReLU problem" by allowing weak gradients for inactivated neurons. Research shows varying results when applying these as compared to ReLUs.
- **Exponential Linear Unit (ELU)**, shown in (16). The ELU aims to solve the "dying ReLU problem" similarly to the LReLU, with some reports of significant performance enhancement [42].
- **Sigmoid**, shown in (18). The sigmoid transforms the input to an output in the range $[0, 1]$ and is historically been a popular choice because of its behavior being analogous to the behavior of biological neurons. However, it is has a major issue with saturating and causing vanishing gradients in deep networks.
- **Hyperbolic tangent (tanh)**, shown in (19). The tanh is similar to the sigmoid, but outputs values in the range $[-1, 1]$. The outputs being zero-centered dampens oscillations during gradient update, hence stabilizing the learning phase.

$$\mathrm{ReLU}(x) = \begin{cases} \mathrm{x} & \text{if } x \geq 0 \\ 0 & \text{if } x < 0, \end{cases} \tag{15}$$

$$\mathrm{ELU}(x) = \begin{cases} \mathrm{x} & \text{if } x \geq 0 \\ \alpha(\exp(x) - 1) & \text{if } x < 0, \end{cases} \tag{16}$$

$$\mathrm{LReLU}(x) = \begin{cases} \mathrm{x} & \text{if } x \geq 0 \\ -\alpha x & \text{if } x < 0, \end{cases} \tag{17}$$

$$\mathrm{Sigmoid}(x) = \frac{1}{1 + \exp(-x)} \tag{18}$$

$$\tanh(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}, \tag{19}$$

$$\tag{20}$$

where α is some constant. In recent years CNN architectures have evolved from sequential structures to *residual* structures, introducing *skip-connections* between layers. Residual networks have several paths from input to output, with varying lengths. This combines the benefit of deep connections being able to achieve high accuracy with the benefit of shallow connections being easier to train [43]. The term *hidden layer* is often used to describe any layer in the network which is not the input or the output layer. In addition to convolutional layers, the hidden layers can consist of a range of other layer types which serve different purposes. Some which are worth mentioning are:

- **Pooling layers** [39, chap. 6] reduce the dimensionality of the image by applying a pooling function with stridden steps across the image. Variations include *Max pooling*, where the function output is the maximum value of the input pixels at each stride, and *average pooling*, where the average of the pixels is computed.

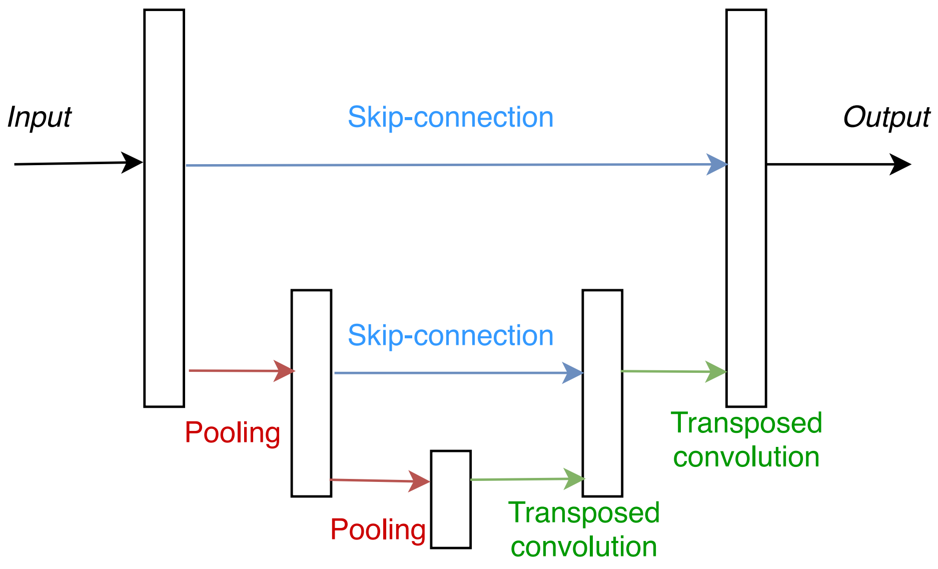


Figure 5: Illustration of a basic U-Net structure. Convolutional layers are connected by max-pooling and transposed convolutional layers, creating a compression/decompression type structure. Skip-connections connect the final layers at each compression level.

- **Transposed convolutional layers** [39, chap. 6] increase the dimensionality of the image by stridden convolution with the output having an increased number of pixels compared to the input.
- **Dropout layers** [39, chap. 3] are a form of *regularization* of the parameters, which aim to reduce *overfitting* on the training data. These work by randomly setting some activations to zero in a given layer, with a given probability in the training phase.
- **Concatenation layers** [43] are used in Residual Networks to merge different layers together by concatenating the outputs from two hidden layers together.
- **Batch Normalization layers** [44] normalize the output from the preceding hidden layer, aiding to stabilize the network by countering vanishing and exploding gradients.

2.4.2 U-Net

Among the many different network structures the *U-Net* has shown itself to be quite useful for several applications within image processing. This architecture, originally proposed by Ronneberger et. al. [45] in 2015, was used for cell segmentation in microscopy images. Following the publication of this article the U-Net has been a popular choice for segmentation problems, but also for other tasks, such as regression [46] [47]. The basic U-Net structure is shown in Figure 5. The intuition behind the U-Net is to use pooling and transposed convolutional layers to create a compression/decompression pipeline. The pooling layers gradually reduce the dimensionality of the input images, creating compressed versions which capture the more general, low level features of the inputs. Then the compressed images are decompressed by the up-convolutional layers, aiming to reconstruct the target images from the low-level features. In addition, the U-Net is a residual network and introduces skip-connections between every compression level. This makes sure that the information at every compression level contributes to the final reconstruction of the output image. In this way, the U-Net can encode data with complex feature spaces and reconstruct them without too much loss of information.

2.4.3 Optimizers

The training of neural networks is done through the backpropagation equations, as described in 2.4.1. These involve calculating the loss function and updating the model weights through *gradient descent*. The most straight-forward way of doing gradient descent is known as batch gradient descent, as described in (21).

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta \nabla_{\boldsymbol{w}_t} L(\boldsymbol{w}_t) \tag{21}$$

In (21) the entire gradient $\nabla_{\boldsymbol{w}} L(\boldsymbol{w})$ is computed at once and the weights $\boldsymbol{w}$ are updated accordingly with learning rate η. For very large datasets, the computation of the gradient can be intractable. A variation which often is used is the Stochastic Gradient Descent (SGD) optimizer, a described by (22).

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta \nabla_{\boldsymbol{w}_t} L(\boldsymbol{w}_t; x^{(i)}, y^{(i)}), \tag{22}$$

where $x^{(i)}$ and $y^{(i)}$ are respectively sample i of the training input data and target data. SGD updates the weights for each data sample pair $(x^{(i)}, y^{(i)})$, greatly simplifying the computation of the gradient for large datasets. One can also divide the dataset into *mini-batches* and compute the gradient of each mini-batch. This is known as *Mini-batch gradient descent*. In addition to the gradient, one can add a *moment* to the weight update, as described in (23).

$$\begin{aligned} \boldsymbol{v}_t &= \gamma \boldsymbol{v}_{t-1} + \eta \nabla_{\boldsymbol{w}} L(\boldsymbol{w}) \\ \boldsymbol{w}_{t+1} &= \boldsymbol{w}_t - \boldsymbol{v}_t, \end{aligned} \tag{23}$$

where $\boldsymbol{v}_t$ is the moment at time t and $\gamma \in [0, 1)$. The purpose of this is to give the optimizer a momentum which dampens oscillations around local minima, thus improving convergence. Several popular optimizers use adaptive moments. These work by adapting the hyperparameters to the data such that sparse data samples or mini-batches have higher impact on the update. This means that the optimizer will favor rare data over common data, emphasizing the spread in the data. There are many variations that try to achieve this. The by far most used one is the *Adaptive Moment Estimation* (ADAM), described by (24)

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \frac{\eta}{\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon} \hat{\boldsymbol{m}}_t, \tag{24}$$

where $\hat{\boldsymbol{m}}_t$ and $\hat{\boldsymbol{v}}_t$ are respectively unbiased estimates of the first and second order moments of the gradient. ϵ is some very small number to prevent division by zero. ADAM has been shown empirically to outperform most other optimizers and is hence a popular choice [48].

2.4.4 Transfer Learning

Transfer learning is an area of research within machine learning. It covers a range of methods for inference from datasets where the available annotations of the target data is sparse, or even unavailable. This is common for medical imaging, where it is hard to have real ground truth data for supervised learning problems since this would require extremely accurate imaging technology. For some applications it would simply be too costly or impractical to create the "true" data labels. Another problem is the amount of unlabeled data which often is not sufficiently large enough as compared to the complexity of the models, again because of practical constraints in data acquisition. Transfer learning seeks to "fill in the gap" of the missing data. This is done by improving the learning process through the learning of another similar

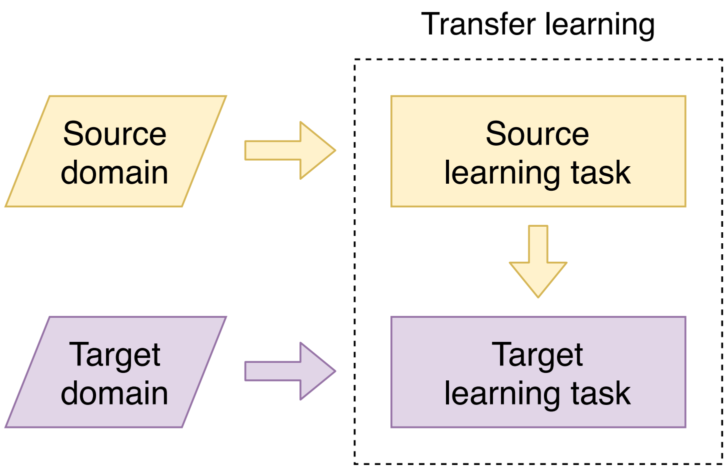


Figure 6: Illustration of the transfer learning problem. A source domain learning task is used to improve the learning task of the target domain.

domain, as described in Figure 6. Here the problem at hand is to estimate the learning task from the *target domain*. This learning task is based both on the target data, and the learning task estimated from the *source domain*. The source data, unlike the target data, is plentiful and properly labeled. The two domains are not the same, but similar enough such that the learning task from the source domain can give information about the target domain to the target domain learning task.

When transfer learning is applied to Deep Learning it is often addressed as *Deep Transfer Learning* (DTL). There are several different ways of approaching DTL. The most straight-forward one is *Network-based DTL*. In this model, a network is trained on the source data. Its parameters are then fixed and used as a feature extractor at the beginning of a larger model with some additional hidden layers at the end. This top level model is trained on the target data. There are two ways of implementing network-based DTL. One way is to pre-train a model on the source domain and connect a part of it, or all of it, to a new model, as illustrated in Figure 7. The additional layers are trained on the target domain data through backpropagation, while the layers of the pre-trained model are kept static. In this way, the layers of the new model are fine-tuned to transfer the source domain prediction to a target domain prediction. Another approach is to train the entire pre-trained model on the target domain data. This way the learning from the target domain data speeds up as the weights are already primed on the source domain data, as initial predictions are closer to the target [49].

2.4.5 Generative Adversarial Networks

A *Generative Adversarial Network* (GAN) is a framework for training neural networks via an adversarial process. The framework is illustrated in Figure 8. It consists of simultaneously training two models: the *generator* and the *discriminator*. These models compete against each other in a two player game. The generator's task is to learn the distribution of the training data, and to be able to generate samples which closely resembles the training data. The discriminator's task is to distinguish the generated data from the real data. During training, the generator is trained to maximize the probability that the discriminator will evaluate the generated data as coming from the real training distribution, while the discriminator is trained to maximize the probability that it is able to correctly distinguish the real data from the fake data. The GAN was originally proposed by Goodfellow et. al. in 2014 [50]. Here they defined the GAN training process as the two-player minmax game in (25).

$$\min_G \max_D V(D, G) = \mathbb{E}\left[\log D(x)\right] + \mathbb{E}\left[\log 1 - D(G(z))\right], \tag{25}$$

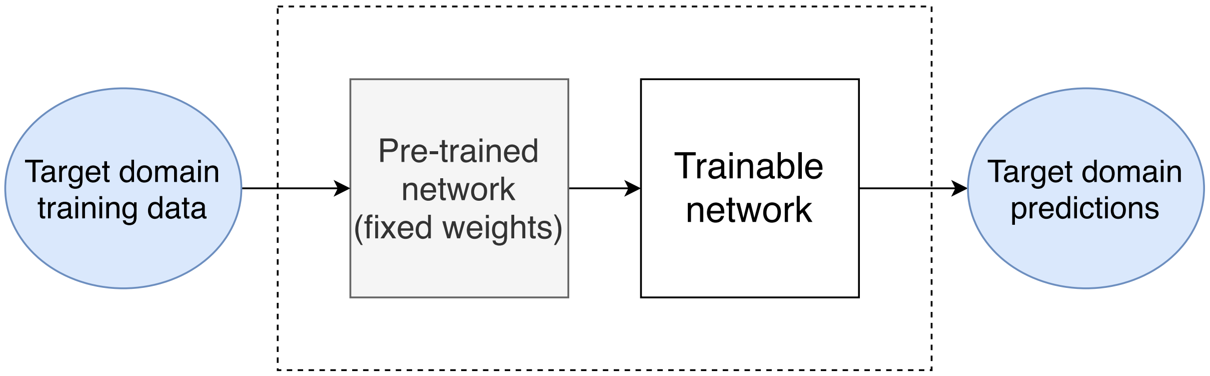

Target domain training data
Pre-trained network (fixed weights)
Trainable network
Target domain predictions

Figure 7: Block diagram of network-based DTL during the transfer learning phase. A network pre-trained on the source and data is used as a fixed feature extractor for a trainable neural network. The top level model is trained on the target data, while the pre-trained network's weights are not updated during the target domain training.

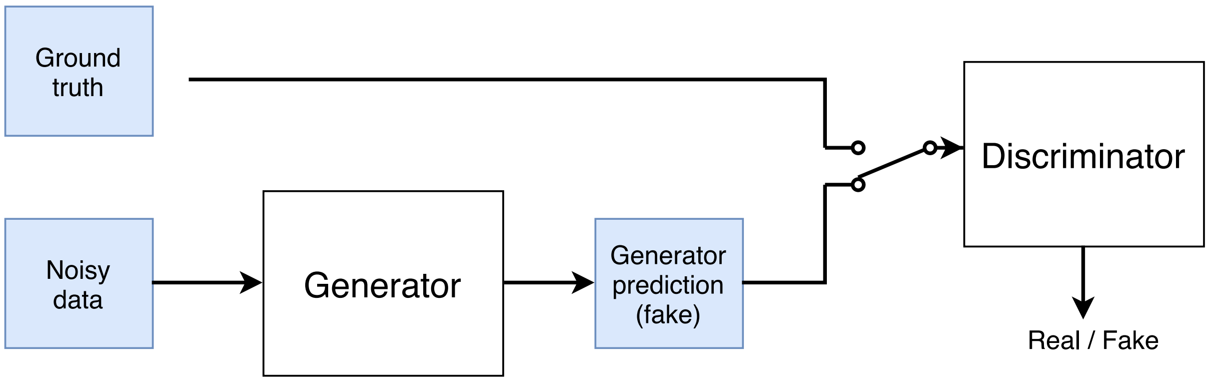


Figure 8: Illustration of a GAN for data reconstruction. The generator network is given a noisy data sample and tries to reconstruct the ground truth in order to fool the discriminator network. The discriminator is trained to distinguish the real ground truth data from the generator's prediction.

where V is the dual objective function, x is the training data, z is the generator input and G and D are the functions corresponding to respectively the generator and the discriminator. They also provided a theoretical analysis where they showed that for arbitrary functions G and D there exists a unique solution to the two-player minmax game where G perfectly approximates the training data distribution and D is $\frac{1}{2}$ everywhere. This is the global minimum for the game. For real world data it is only possible to find this global minimum with infinite model capacity and training data, however it is more viable to find local minima which are known as *Nash equilibriums*. A Nash equilibrium is a point where both G and D have minimal cost with respect to each other's parameters [51]. In other words, both models are happy with their own state given they know the state of the adversary. Finding a Nash equilibrium through gradient descent is in general a very difficult problem, which in turn makes GANs very difficult to train. When training GANs we often encounter one or several of the following issues:

- **Failure to converge**, where the parameters of both models oscillate indefinitely.
- **Mode collapse**, where the generator's learnt distribution collapses and only produces a limited variety of samples.
- **Vanishing gradients**, where the discriminator gets too good at its task, making gradients vanish and thus preventing the generator from learning.
- **High hyperparameter sensitivity**, where altering some hyperparameters slightly may cause the training process to suddenly fail completely.

There has been done a lot of research of how to improve upon these issues, and it is still a hot topic within Machine Learning research. However the model will ultimately depend on the training data, hence making it up to the designer to fine-tune the models to make them capable of solving the problem at hand.

Despite the difficulties that come with GANs, they have shown remarkable performance on several different tasks, but in particular on image generation tasks. Most notably is StyleGAN [52] by Karras et. al. from 2018, a GAN trained to generate human portrait images with such high quality that it can even fool humans to believe that the images are real. The advancements in GAN related image processing originate from a paper published in 2016 by Radford et. al. [53], where they introduced the *Deep Convolutional* GAN (DCGAN). The DCGAN uses an encoder-like structure for the discriminator and a decoder-like structure for the generator, similar to the encoder-decoder structure of the U-Net. They showed that their proposed model was able to do representation learning for several different image data sets, being able to generate images which captured many of the underlying features of the training data. Although a lot of the research regarding GANs is based around generating data from randomly sampling from the generator's distribution, they have also been proposed for regression tasks. For regression, the generator's input is not just random noise, but noisy data from which we want to reconstruct clean data. Placing a neural network tasked with regression in the role of the generator creates a regression model which has no explicit way of learning the optimal regression task. Instead, its performance is evaluated implicitly by the discriminator on how well the generated data resembles the real data. This gives an added benefit of not having to hand craft a loss function for the regression model which determines how to evaluate its performance. Instead, this is learned directly from the data.

2.4.6 Deep Learning for Flow Reconstruction

Although little research is known to have been done concerning reconstruction of ultrasound blood flow velocity fields using Deep Learning, recent research suggests that Deep Learning can be used for reconstruction of flow fields in other applications. In 2020 Bo et. al. [54] applied a Deep Learning based model for super-resolution reconstruction of turbulent flows. In this paper they present residual CNN structures trained to reconstruct downsampled versions of 2D slices from simulated 4D turbulent flows. They present two different models, one which is trained on static flow images, and another which is trained on a time series of consecutive flow images. They showed that both models were capable of substantially improving the reconstruction of the fields as compared to standard methods for super-resolution, potentially capturing all the features of the turbulent flows. When comparing the two models, they discovered that the one trained on time series data was better capable of reconstructing finer details in the flow, hence concluding that the additional temporal information introduced into the network was beneficial for reconstructing physically realistic looking fields.

Another 2020 study by Tianyuan et. al. [55] used a GAN to reconstruct simulated nanofluid pressure, temperature and velocity fields from limited measurements of the fluid. The approach was to train a deconvolutional generator network to predict the fields from sparse measurements, and a convolutional discriminator network to distinguish the predicted fields from the true fields. They reported that the proposed method was able to successfully reconstruct all fields with high accuracy for even a small number of training samples. The performance was however limited to the measuring uncertainty of the input data, giving a lower prediction accuracy for uncertain inputs. The approach is similar to that of a 2017 study by Lee and You [56]. In this study they used a GAN to predict the next time frame of a time series of images showing unsteady laminar vortex shedding over a circular cylinder. They emphasized the network's capability to learn the Navier-Stokes equations, and concluded that the GAN was successful of predicting both the spatial and temporal characteristics of the flow.

As a preliminary study to this thesis [28] the potential use of Deep Learning to reconstruct ultrasound blood flow velocity fields was explored. Here two CNNs, a Convolutional Autoencoder (CAE) and a U-Net, were trained to reconstruct the velocity field from 2D slices of a CFD phantom velocity field of the left ventricle with added white Gaussian noise. It was shown that while the CAE did not have sufficient complexity to reconstruct the spatial dynamics of the flow, the U-Net was able to reconstruct the fields with high accuracy. However, the trained U-Net struggled with generalizing to test data with more realistic noise characteristics. The conclusion was that the U-Net structure has potential to reconstruct blood flow velocity fields in a realistic setting, but would need to be trained on a more realistic data set in order to achieve satisfactory results.

3 Methodology

3.1 Overview

Figure 9 shows the data generation and training pipeline for the model creation. The overall idea is to fit a model to do reconstruction of simulated (in-silico) ultrasonic cardiac blood flow measurements and use this model as a baseline for transfer learning applied to real (in-vivo) ultrasound blood flow measurements of the left ventricle. The in-silico data is generated by taking a computational fluid dynamics (CFD) phantom of a neonate left ventricle and generating realistic 3D ultrasound images through the FUSK simulation. Blood speckle tracking (BST) is performed on these images, creating 3D blood flow velocity estimates. A dataset of 2D blood flow data is generated by rotating the 3D blood flow data in random ways and slicing it. The same rotation and slicing is performed on the CFD phantom, creating a data set of clean-noisy field pairs. The base model is trained through supervised learning to reconstruct the CFD data from the tracking data. Finally, in-vivo tracking data is used to fine-tune the DTL model to improve the performance of reconstructing in-vivo blood flow velocity fields. This is also done through supervised learning. However, since no ground truth velocity fields are available for in-vivo data, results from B-spline interpolation are used as targets for the reconstruction. The B-spline results reconstruction results are computed using the framework proposed by Grønli et. al., as described in 2.3.5. The CNNs are implemented in Python using the Keras framework with a Tensorflow [37] backend. They are trained and tested on a NVIDIA Quadro P5000 GPU.

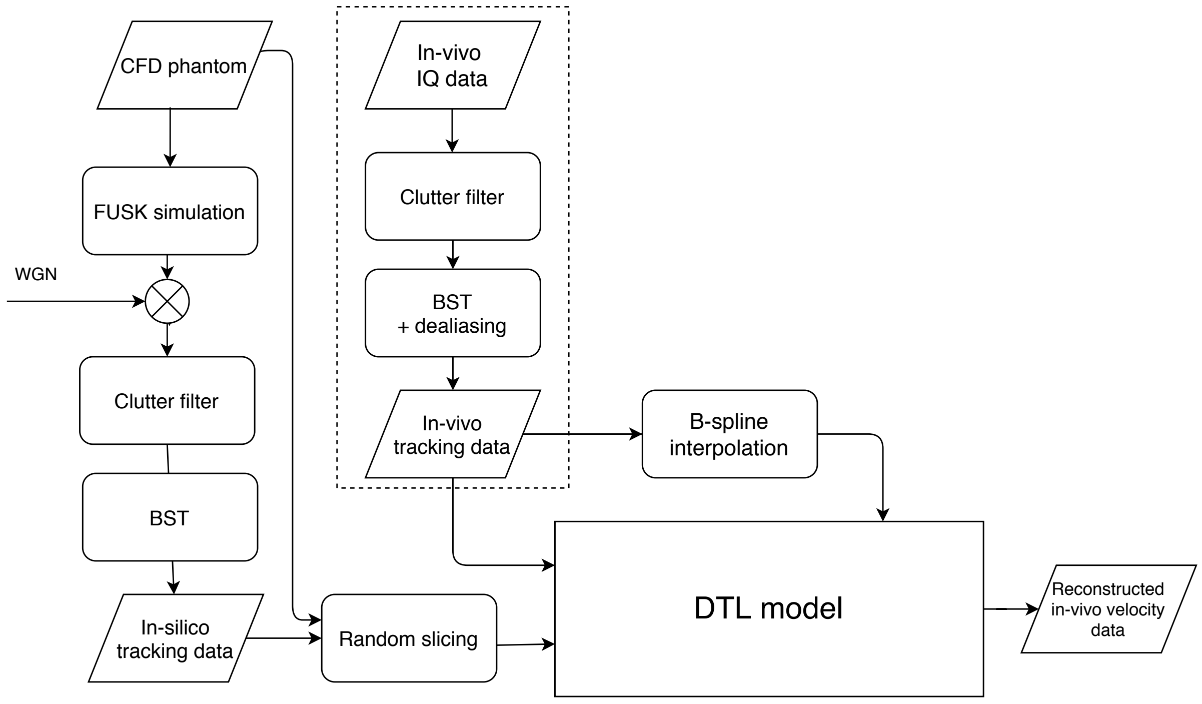


Figure 9: Overview of the model training. VFI measurements are simulated by using FUSK data, adding white Gaussian noise (WGN), clutter filtering and applying full SSD BST. The tracking data is sliced in random orientations to obtain 2D time series from the 3D time series object. The in-silico data is used to pre-train a DTL model. The in-vivo tracking data is obtained in a similar way from raw IQ data, however performed beforehand, thus not described explicitly in the methodology. The in-vivo data is used to fine-tune the DTL network to further improve its capability of reconstructing the velocity data.

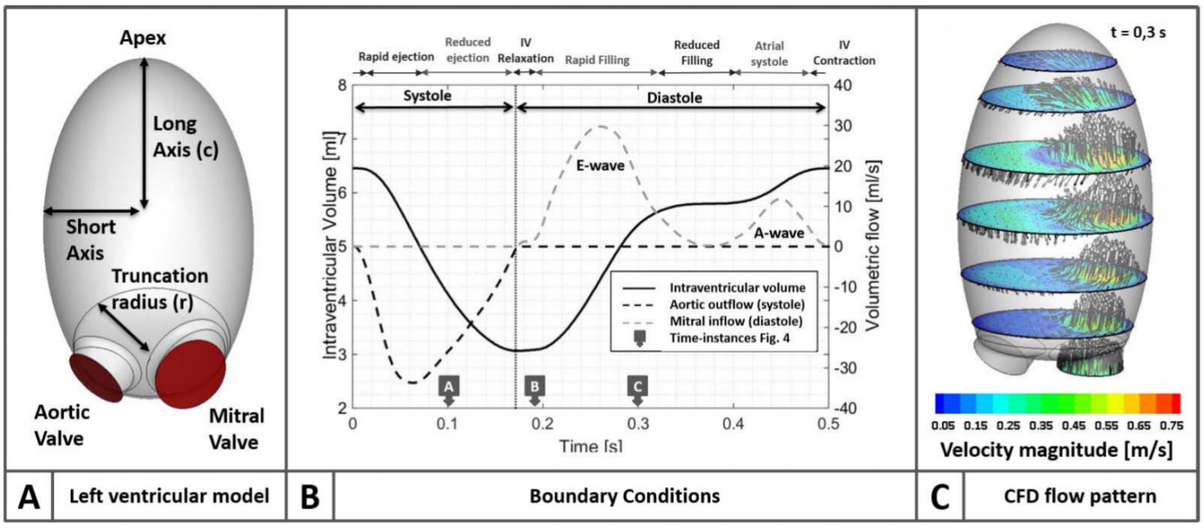


Figure 10: "(A) Simplified computational flow phantom representing the neonatal left ventricular cavity was created with the CAD program ANSYS DesignModeler. (B) Appropriate boundary conditions needed to be applied during both the systolic and diastolic phases indicated by, respectively, the black and gray dashed lines. (C) Time-varying 3-D flow field was obtained with a finite volume method in the commercial CFD solver ANSYS Fluent. The time instance shown in (C) represents the end of the rapid filling phase." - The illustration and the caption are taken from [27].

3.2 CFD Phantom

The 4D CFD phantom is used as an input to the FUSK simulation and acts as a ground truth for the simulated velocity measurements. The phantom is generated by using the commercial CFD solver ANSYS Fluent. The geometrical boundary conditions of the phantom are created with ANSYS DesignModeler. The geometry is created to accurately represent the function of a neonate left ventricle. Figure 10 describes the geometry of the model. The geometry is based on a truncated prolate spheroid with an ellipsoidal inlet representing the mitral valve, and a circular outlet representing the aortic valve. The intraventricular volume, the mitral inflow and the aortic outflow are dynamically modelled using a known mathematical model. The end-systolic volume is set to 3.2 mL, and the heart rate is set to 120 beats/min. Then, a mesh is generated for the volume and the velocity field is computed, solving the Navier-Stokes equations with a finite volume method and modelling the blood as a Newtonian fluid with a dynamic viscosity of 5.5 mPa·s and a density of 1060 kg/m^3. See [27] for the complete description of the methodology.

3.3 FUSK Simulation

Ultrasound data was simulated using the Fast Ultrasound imaging Simulation in K-space (FUSK). The FUSK simulation generates realistic ultrasound images of the CFD phantom as if it were embedded in a human body and scanned with a transducer probe. Firstly, a transmit waveform is generated based on geometrical properties, like focal point depth and aperture dimensions, as well as center frequency, bandwidth and apodization filtering. It then inserts random scatterers as antialiasing filters into the object and calculates their K-space frequency response. The received waveform can then be computed. Please see [57] for a complete description of the methodology. An example result from the simulation is shown in Figure 11. This figure shows the speckle pattern appearing when plotting the squared magnitude of the IQ data on a logarithmic scale. The main parameters used in the simulation are given in Appendix A. The IQ data generated from FUSK is clutter filtered with a FIR filter. The filter

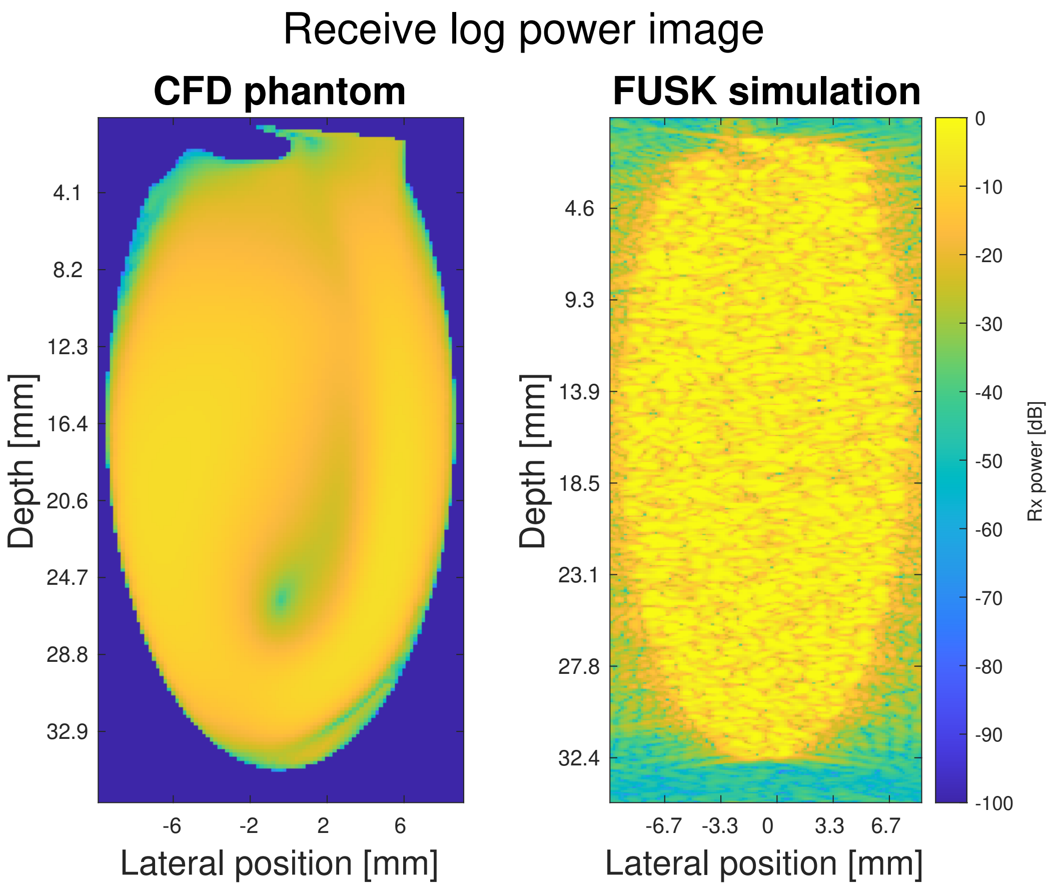


Figure 11: Log receive power image $|\mathrm{IQ}|^2$ of an apical slice of the left ventricle during diastole simulated with FUSK, in comparison to the corresponding CFD phantom slice. The blood speckle pattern can clearly be seen in the FUSK data.

coefficients are provided by GE Vingmed.

3.4 Blood Speckle Tracking

Blood Speckle Tracking (BST) was performed on the FUSK data using an implementation of the method described in [12]. Block matching was performed in all three directions. The radial doppler velocity is not used. This makes sure all slices of all rotational configurations are equally realistic, as we avoid having a stronger signal along a transformed axis somewhere in the slice. The applied correlation metric function R, as mentioned in (6), is the sum-of-squared-differences (SSD). This is computed as described in (26).

$$\mathrm{R}(\boldsymbol{r_0}, \boldsymbol{r_0} + \Delta\boldsymbol{r}) = \left(|\mathrm{IQ}|(\boldsymbol{r_0} + \Delta\boldsymbol{r}) - |\mathrm{IQ}|(\boldsymbol{r_0})\right)^2 . \tag{26}$$

Here $\boldsymbol{r}$ is the positional vector describing a point in the IQ data volume. The search space is limited by bounds on maximum and minimum velocity, following the relation in (7). For the in-silico data the tracking is performed on a 3D-grid with full SSD block matching. The search is limited by the search

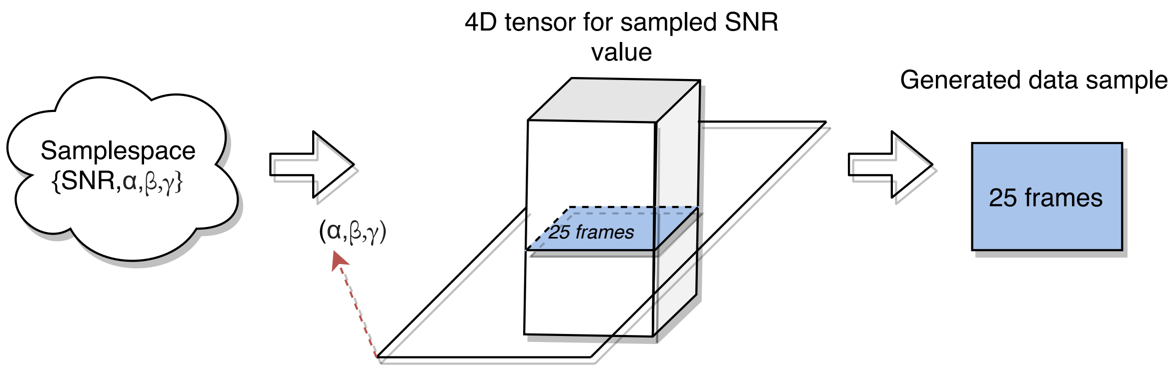


Figure 12: Illustration of sampling and slicing of the noisy FUSK data object to generate training and test data for the CNNs. A configuration of SNR value and slicing rotation (α, β, γ) is sampled without replacement. The corresponding 4D tensor is sliced by a plane with orientation (α, β, γ) along all 25 time frames, producing the generated data sample.

extent, block matching kernel dimensions and velocity limits. The in-vivo data is only two dimensional, meaning tracking is only performed in the lateral and radial directions. Here, doppler measurements are combined with the block matching to enhance the estimation quality. Dealiasing is performed on the doppler measurements according to a dealiasing limit. When the doppler estimate quality measure is below this limit, the dealiasing is aborted and the estimate is set to zero. See Appendix B for the full tracking configuration. The end result of the tracking is a 4D tensor containing the velocity field estimate at each point in space and time. Additionally, the BST algorithm returns R for each point, giving a correlation map which can be interpreted as the quality measure of the tracking at each point for each field component.

3.5 In-Silico Data Set Generation

A dataset of simulated time series of 2D ultrasound blood flow velocity fields is generated to be used as training and test data for the CNNs. This is done by rotating and slicing the CFD phantom and FUSK data object in random orientations and slicing them. Before the FUSK data is generated from the CFD phantom, white Gaussian noise is added to the CFD phantom at one out of four different signal-to-noise ratio (SNR) values (5 dB, 10 dB, 15 dB, 100 dB). The SNR values are chosen to represent a gradient of data quality, ranging between ”poor”, ”moderate”, ”good” and ”perfect”, as illustrated in Figure 13. The noise is added to each data point of the IQ data as described by (27).

$$
\begin{aligned}
\tilde{\mathrm{IQ}} &= \mathrm{IQ} + \frac{1}{\sqrt{2}}(n_r + jn_i) \\
n_r, n_i &\sim \mathcal{N}(0, \sigma_n) \\
\sigma_n^2 &= \frac{\sigma_s^2}{10^{\frac{\mathrm{SNR[dB]}}{10}}} \\
\sigma_s^2 &= \frac{1}{K_x K_y K_z} \sum_{i,j,k} |\mathrm{IQ}|^2(t_0, x_i, y_j, z_k).
\end{aligned} \tag{27}
$$

The signal power σ_s^2 is estimated from computing the mean power over some $K_x \times K_y \times K_z$ thick slice of the data object at some time frame t_0. Followingly, the noisy IQ data is filtered with a clutter filter. BST

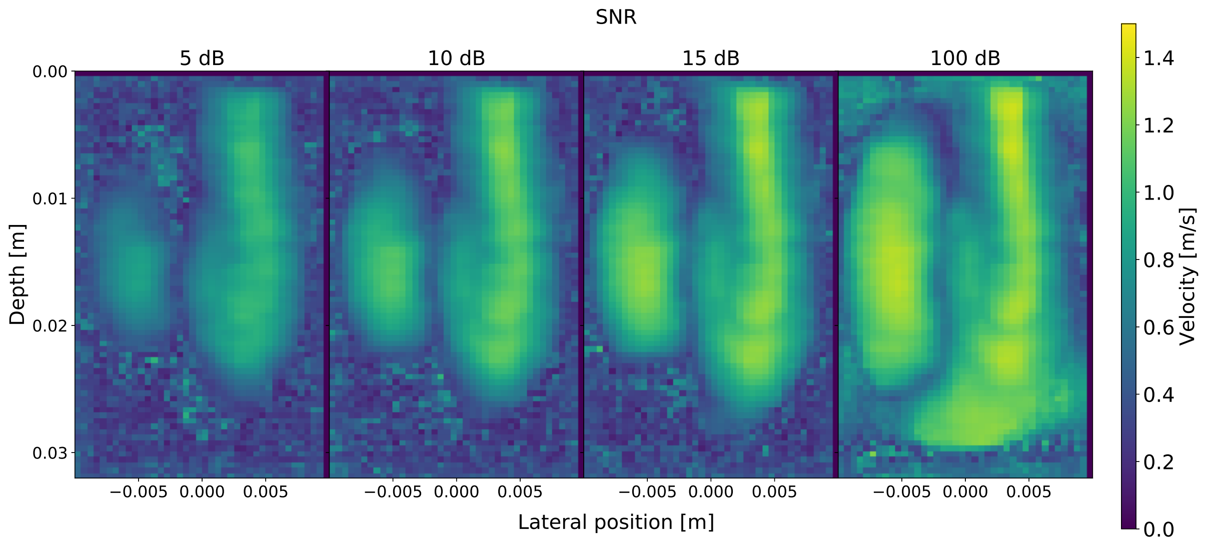


Figure 13: Velocity magnitude of an apical slice of the in-silico tracking data during diastole with added noise for different SNR values. The SNR values are chosen to cover the range of qualities one can expect for ultrasound acquisition (5 dB: poor, 10 dB: moderate, 15 dB: good, 100 dB: perfect). We can observe that lower SNR values give larger cut-offs due to clutter filtering.

is applied to the filtered data as described in 3.4, generating tracking estimate tensors. The resulting data object is a 4D tensor of shape $T \times N \times M \times C$, where $T = 25$ is the number of time frames, $N = 80$ is the image height, $M = 80$ is the image width and $C = 3$ is the number of velocity field components.

The different SNR values in combination with the different rotational angles α, β, γ create a basis for a sample space of possible slice configurations. One data sample is generated from one configuration sample as illustrated in Figure 12. With 4 SNR values, 25 time frames and using a step size of 10° for each angle, this gives a sample space of $4 \cdot 36^3 = 186624$ possible configurations. Each data sample consists of 25 consecutive time frames containing the blood flow velocity field tracking estimate, the correlation map for each field component and a mask containing the object segmentation information. The rotation and slicing is also performed on the CFD phantom, creating a ground truth velocity field as compared to the estimate. Figure 14 shows an example of data generated from a random slicing configuration.

3.6 U-Net

3.6.1 Model Architecture

The structure of the CNN is based on the U-Net [45] architecture. The model trained on static velocity field slices using 2D convolutional layers (*2D U-Net*) is shown in Figure 15. This model is built up of 30 hidden layers, including four 2×2 max-pooling layers and four 2×2 transposed convolutional layers, summing up to a total of 1,941,122 model parameters. Skip-connections connect the final layer of each compression level on the shallow side to the first layer of each decompression level on the deep side. The data which passes through the skip connections are concatenated to the data on the receiving end. All activation functions of the hidden layers are ELUs, as described in 2.4.1. The input layer takes in a 4D tensor of size "?" $\times N \times M \times C_{in}$, where "?" is the batch size, N is the image height, M is the image width and C_{in} is the number of channels at the input. Similarly, the output layer has dimensions "?" $\times N \times M \times C_{out}$, where C_{out} is the number of channels at the output. Because of the four 2×2 max-pooling layers, the image dimensions at the highest compression level will be $\frac{N}{16} \times \frac{M}{16} = \frac{NM}{256}$. Because of this, it is necessary to choose both M and N to be divisible by 16. C_{out} corresponds to the

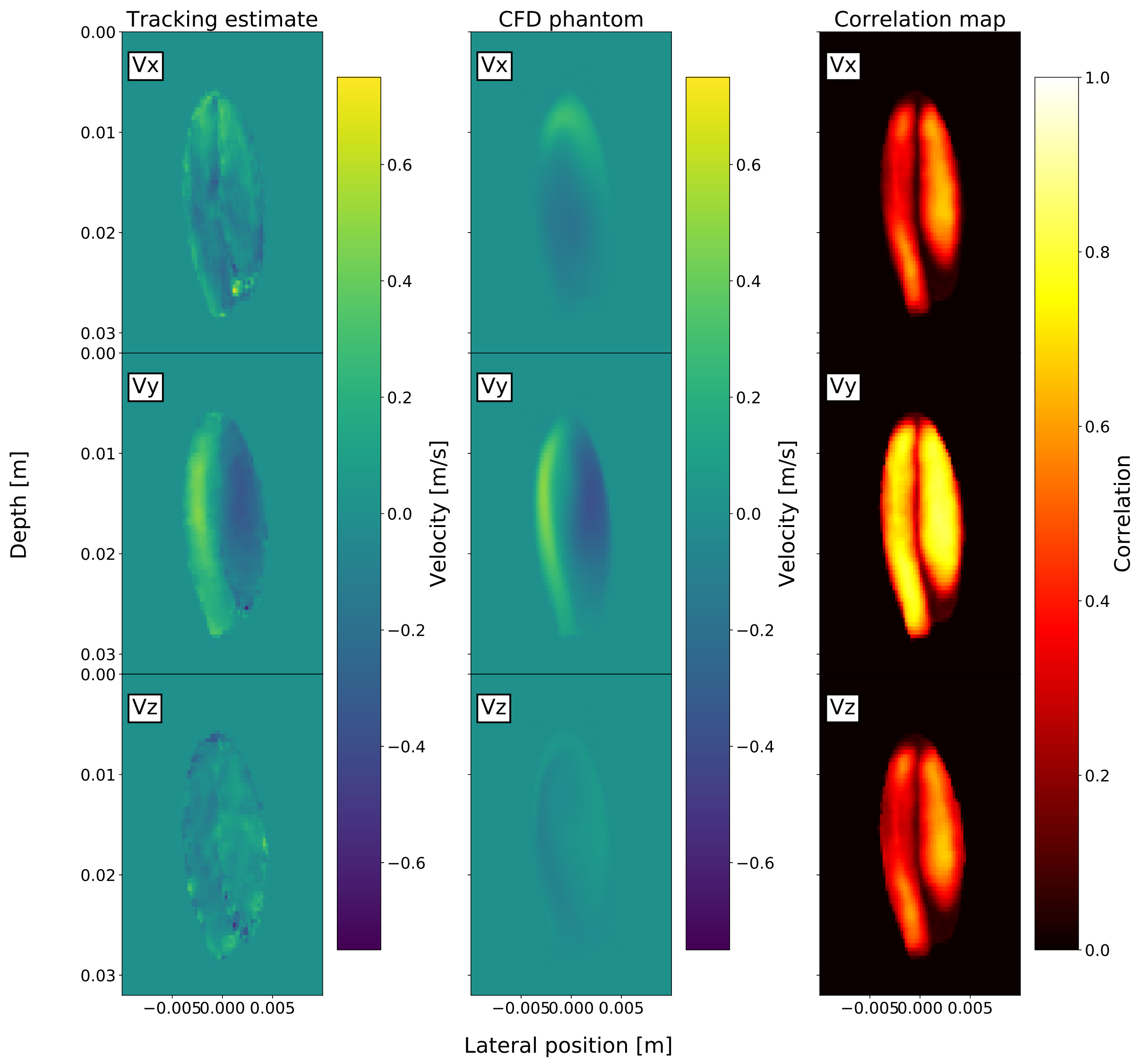


Figure 14: Example of a time frame of a data sample generated from a random slicing configuration (the segmentation mask is not shown). Each image represents a frame along the lateral (x) and radial (z) axes of an ultrasonic recording for all components (V_x, V_y, V_z). The *Tracking estimate* is the result of BST applied to the FUSK data with added noise. The *CFD phantom* is the ground truth velocity field for the Tracking estimate. The *Correlation map* is an additional output from BST which gives the correlation metric for each velocity measurement, giving information about the estimation strength at each point.

number of field velocity components of the output images. C_{in} corresponds to the number of velocity field components of the input images, with the addition of correlation maps and mask. For example, if the network is estimating a velocity field with three components from a noisy velocity field with three components, the corresponding correlation maps and the mask, then $C_{out} = 3$ and $C_{in} = 3 + 3 + 1 = 7$.

Two other models are designed for time series of velocity field slices. The models process the input data using 3D convolutional layers. The input layer takes a 5D tensor of shape "?" $\times T \times N \times M \times C_{in}$, and the output layer outputs a 5D tensor of shape "?" $\times T \times N \times M \times C_{out}$, where T is the number of time frames. The *3D U-Net*, shown in Figure 16, applies max pooling and transposed convolutions only along the image height and width dimensions. Because of limitations in the used framework, the 3D U-Net cannot process tensors with arbitrary shapes, like the 2D U-Net can. Hence, the shapes are fixed to $N = 80$, $M = 80$ and $T = 25$, in order to perfectly fit the shape of the simulated data. The total number of parameters for the 3D U-Net sums up to 5,471,714. The *3D U-Net with temporal compression*, shown in Figure 17, applies max pooling and transposed convolutions along the image height, image width and time frame dimensions. Since the four pooling layers require that the shape of each dimension is divisible by 16, we choose $T = 32$, which implies the need of further processing of the simulated ultrasound data in order for it to fit the model. The total number of parameters in the 3D U-Net with temporal compression is 5,645,794.

3.6.2 Training and Testing with In-silico Data

The 2D U-Net is trained, validated and tested on respectively 6400, 1600 and 2000 data samples, which are generated as described in 3.5. All 25 time frames of each data sample is processed as independent data points. As the 3D U-Net architectures have approximately 2.82 times more parameters than the 2D U-Net, they are trained with 2.82 times more data. Additionally, one sample from the time series data set contains 25 times more data than a sample from the static data set. Hence they are trained, validated and tested on respectively 786, 196 and 246 data samples. Here, the data is processed including the dependency between time frames. The loss function used is a variation of MSE, which is expressed in (28) and is dubbed the *masked mean-squared-error* (MMSE).

$$L_{\text{MMSE}}(x) = \frac{(\boldsymbol{\hat{y}}(x) - \boldsymbol{y}(x))^2}{C_{out} \sum\limits_{z} \boldsymbol{m}(z)}. \tag{28}$$

$\hat{y}$ is the model output tensor and $\boldsymbol{y}$ is the target tensor. $\boldsymbol{m}$ is the corresponding 4D mask. The MMSE is equivalent to computing the MSE for exclusively the data within the segmented area given by the mask. The MMSE loss will be invariant to the amount of area outside the mask. On the other hand, the MSE would be lower for images with smaller masks. It would also decrease with increased padding. The MMSE is thus used to avoid dependency on such features. The function is scaled by C_{out} to make sure the metric is of the same order of magnitude when models using different number of output channels are compared, as the numerator will be proportionate to C_{out}. The gradient of can be computed according to (29), which shows the partial derivative of the loss function with respect to a variable x.

$$\frac{\partial}{\partial x} L_{\text{MMSE}}(x) = \frac{2(\boldsymbol{\hat{y}}(x) - \boldsymbol{y}(x))}{C_{out} \sum\limits_{z} \boldsymbol{m}(z)} \frac{\partial \boldsymbol{\hat{y}}(x)}{\partial x}. \tag{29}$$

We can see from (29) that the gradient is proportional to that of a regular MSE loss function, but will be larger for samples with smaller masks, adjusting for the smaller squared-difference in the numerator. The chosen hyperparameters are based on the ones recommended by Ronneberger et. al [45]. They propose using a SGD optimizer with a learning rate of 0.01 and a momentum of 0.99. The hyperparameters

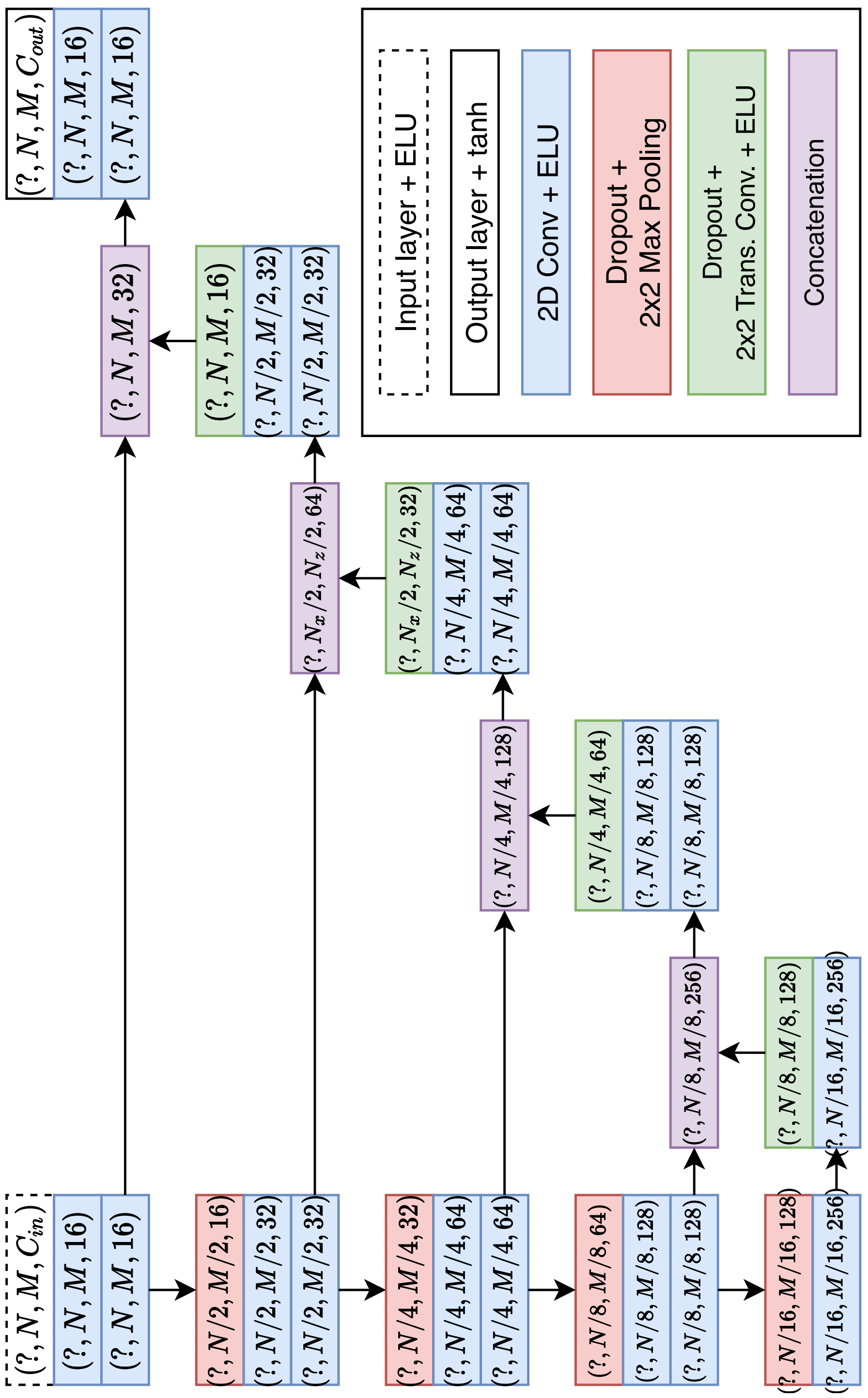


Figure 15: 2D U-Net architecture. The tuple in each layer represents the output shape of the tensor output by the layer. The tensors are four dimensional. The first dimension is the batch size, an arbitrary integer represented by "?". The second, third and fourth dimensions are respectively the image height, image width and channels.

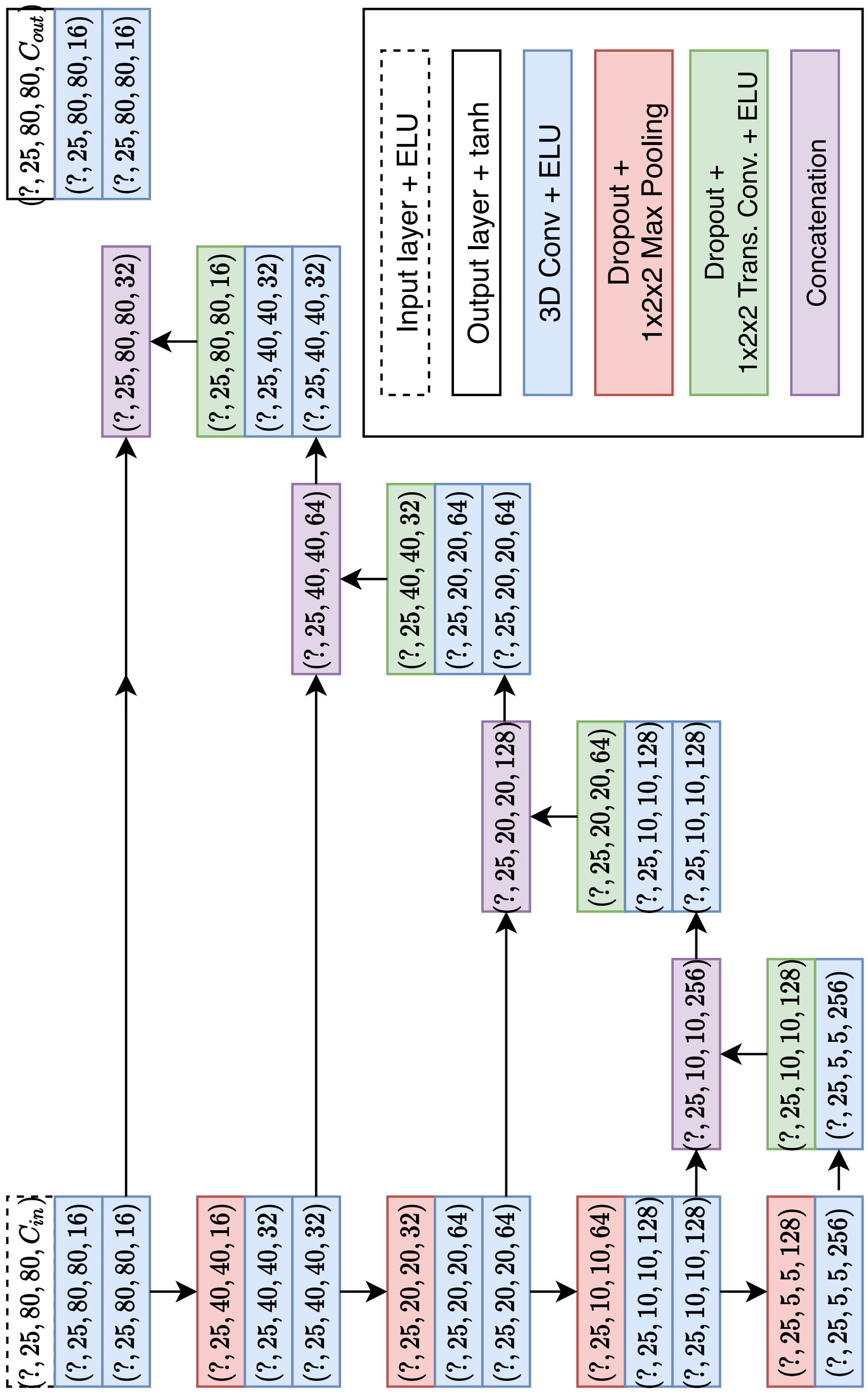


Figure 16: 3D U-Net architecture. The tuple in each layer represents the output shape of the tensor output by the layer. The tensors are five dimensional. The first dimension is the batch size, an arbitrary integer represented by ”?”. The second dimension is the number of time frames. The third, fourth and fifth dimensions are respectively the image height, image width and channels.

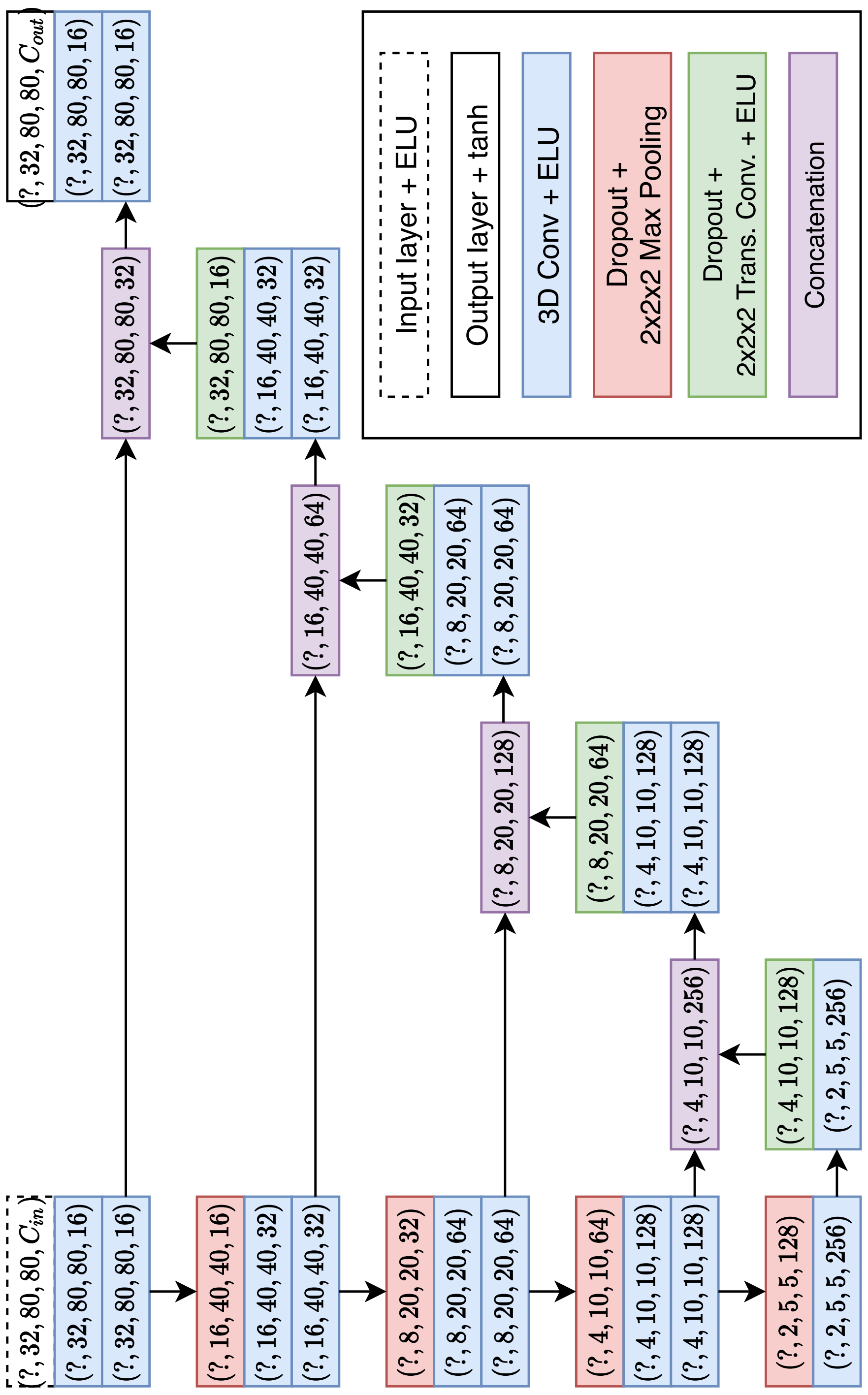


Figure 17: 3D U-Net with temporal compression architecture. The tuple in each layer represents the output shape of the tensor output by the layer. The tensors are five dimensional. The first dimension is the batch size, an arbitrary integer represented by "?". The second dimension is the number of time frames. The third, fourth and fifth dimensions are respectively the image height, image width and channels.

Table 2: Variations in model training.

Name	Input field	Output field	Correlation maps	(C_{in}, C_{out})
U-Net 3-3	(x, y, z)	(x, y, z)	None	$(4, 3)$
U-Net 3-3 w/ correlation maps	(x, y, z)	(x, y, z)	(x, y, z)	$(7, 3)$
U-Net 2-3	(x, z)	(x, y, z)	None	$(3, 3)$
U-Net 2-3 w/ correlation maps	(x, z)	(x, y, z)	(x, z)	$(5, 3)$
U-Net 2-2	(x, z)	(x, z)	None	$(3, 2)$
U-Net 2-2 w/ correlation maps	(x, z)	(x, z)	(x, z)	$(5, 2)$

are changed to using the ADAM optimizer with the Keras default parameters: a learning rate of 0.001, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 1e - 07$ and a batch size of 32. This was verified to give similar performance to using high momentum SGD, with the added benefit of decreased training time, as shown in Figure 18. Figure 18 shows the U-Net model trained and validated on respectively 6400 and 1600 images, with no correlation maps. Figure 18(a) shows the model loss for the SGD optimizer which is identical to the one described in [45]. Figure 18(b) shows the same network training for the ADAM optimizer, with the Keras default parameters and a batch size of 32. The results show that both training procedures achieve similar convergence at the same rate, the difference being that ADAM tends to cause more oscillation than SGD. Because of the increased batch size, the training with ADAM took only a fraction of the time as compared to using SGD. Hence it is more viable to use the ADAM training procedure as it achieves the same results as the proposed training procedure described in the original article.

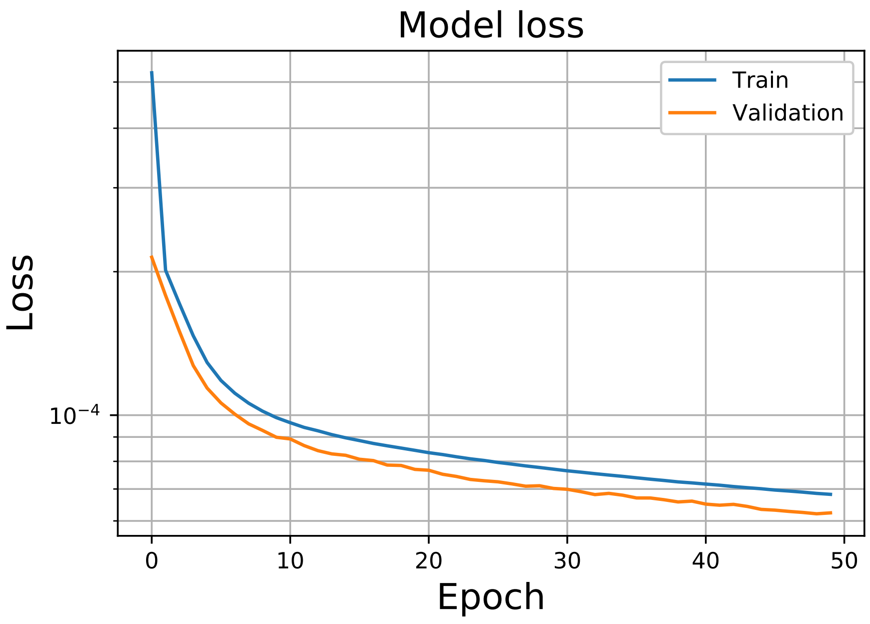
The models are trained using different amounts of velocity field components at the input and output layers. We vary C_{in} and C_{out} to analyze how performance varies with the amount of input information. There are a total of six different training variations for each of the 2D U-Net and 3D U-Net models. The model variations are described comprehensively in Table 2. Here, x, y and z respectively denote the lateral, out-of-plane and radial components of the velocity field.

3.6.3 Transfer Learning with In-vivo Data

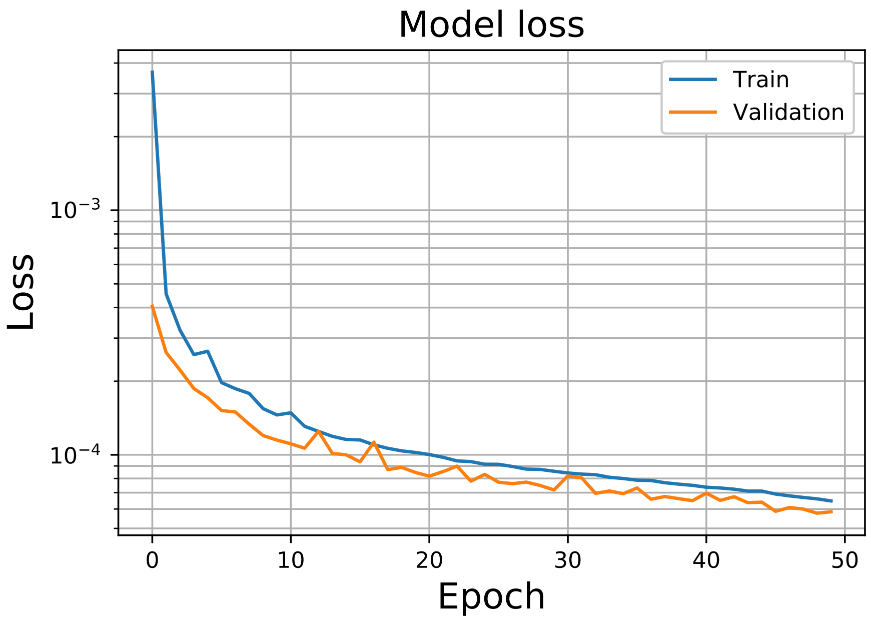
The in-vivo data set consists of 58 different velocity measurements of the left ventricle from pediatric patients. Each measurement lasts for the duration of one cardiac cycle. 12 of the measurements are used for testing and 46 are used for training and validation. The velocity field in each measurement contains the lateral and radial component, hence the U-Net 2-2 model types are chosen. No ground truth data is available for the in-vivo measurements. Instead, we use the results from B-spline interpolation using the framework proposed by Grønli et. al., as described in 2.3.5. The motivation behind this is that this method is heavily grounded in the Navier-Stokes equations, hence giving reconstructed flow fields which coincide with realistic flow behavior to a large extent. This way, even though these results cannot be said to be entirely "true", they accurately describe flow in the left ventricle. We hope that the machine learning models can learn these features, and thus also being able to output equally realistic flow fields.

To evaluate performance of the transfer learning approach, five different training procedures are done for both the 2D U-Net and 3D U-Net, creating five different model variations for each architecture:

- **In-silico base model**, only trained on the in-silico data.
- **Deep layer fine-tune model**, with all the layers firstly trained on the in-silico data, and secondly the deep half of the model layers fine-tuned on the in-vivo data.
- **Shallow layer fine-tune model**, with all the layers firstly trained on the in-silico data, and secondly the shallow half of the model layers fine-tuned on the in-vivo data.
- **Full in-vivo model with pre-trained weights**, with all the layers firstly trained on the in-silico data, and secondly all the layers trained on the in-vivo data.



(a) SGD (U-Net standard)



(b) ADAM

Figure 18: Model loss for the 2D U-Net compiled with SGD optimizer (learning rate: 0.01, momentum: 0.99, batch size: 1) and ADAM (learning rate: 0.001, β_1: 0.9, β_2: 0.999, ϵ: 1e-07, batch size: 32). Both choices of hyperparameters give the same convergence rates. Although the ADAM optimizer causes more oscillation it is beneficial since the training time using mini-batch descent with ADAM is only a fraction of the training time using SGD.

- **Full in-vivo model with randomly initialized weights**, only trained on the in-vivo data.

Because of the variations in the model architectures between the 2D and 3D U-Net models, the training procedures and data pre-processing for the different architecture types are slightly different. The training procedures are illustrated in Figure 19. Since the measurements are of different shapes in all dimensions they cannot be stacked directly into one data set. The 2D U-Net is capable of handling data with arbitrary shapes in the spatial dimensions, and is therefore trained by using a single measurement from a patient as one batch of the training set, where the shape of the time dimension then corresponds to the batch size. The training is iterated over all training batches and repeated for 100 epochs. The test data is padded in the same way. The image heights N and widths M are zero padded to the closest numbers $N + N_{pad}$ and $M + M_{pad}$ which are divisible by 16. The padding numbers N_{pad} and M_{pad} are computed for each measurement batch as described in (30)

$$\begin{aligned} N_{pad} &= 16 - N\%16 \\ M_{pad} &= 16 - M\%16, \end{aligned} \tag{30}$$

where % denotes the modulo operator. The 3D U-Net requires that the data has fixed shape. Hence the data is interpolated to a regular grid of shape $(46, 25, 80, 80, 2)$ for the 3D U-Net and $(46, 32, 80, 80, 2)$ for the 3D U-Net with temporal compression. The data set is then split into a training set of size 42 and a validation set of size 4. The test data is interpolated in the same way. The model is trained with full batch gradient descent for 100 epochs. Since the real velocity flow field measurements are unlabeled it is not possible to to coherent supervised learning. Instead, the B-spline interpolation algorithm is applied to the measurements, giving realistic looking reconstructed flow fields which are validated by experts to be valid targets for the noisy data. In addition, a data set of smoothened data is generated by applying Gaussian smoothing with a kernel size of $(5 \times 5 \times 50\text{ms})$ to the noisy measurements. The smoothened data is used as a benchmark when comparing the model prediction results.

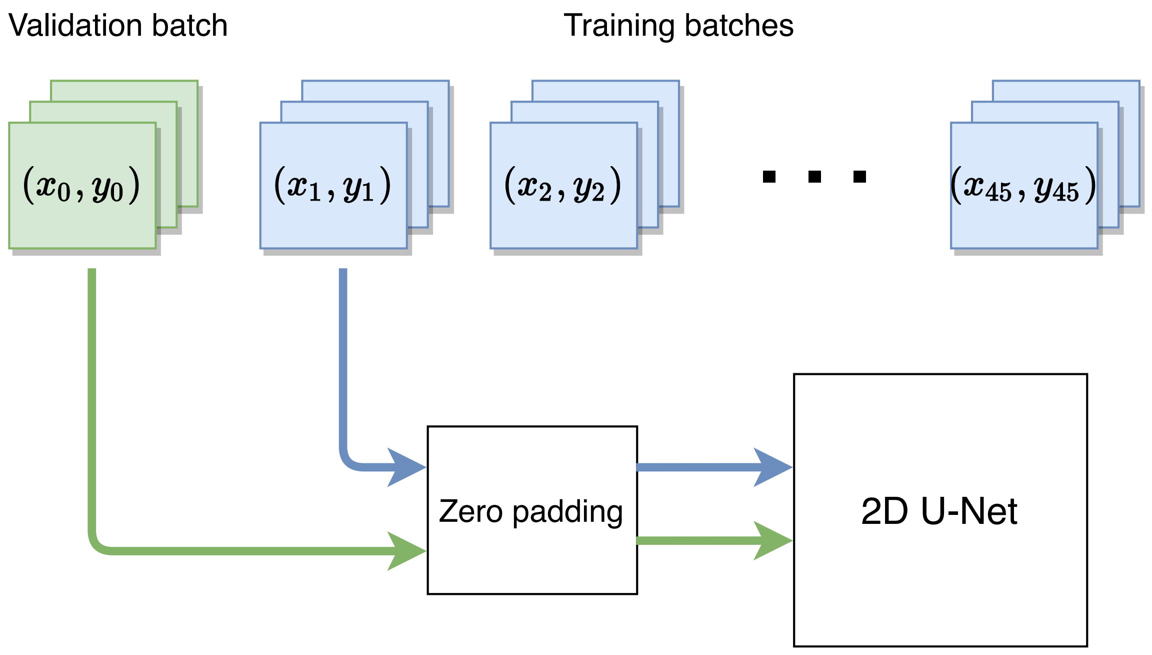
3.7 Generative Adversarial Network

3.7.1 Model Architecture

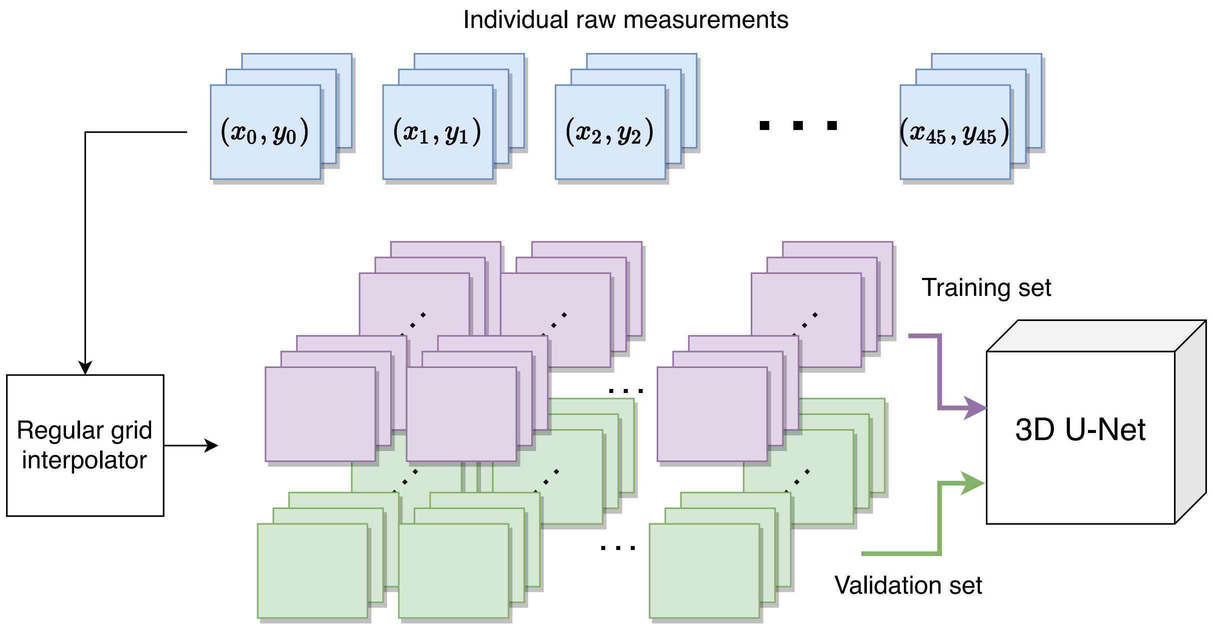
The GAN consist of two neural networks, the generator and the discriminator. The generator has the general structure of the 2D U-Net, but with some modifications. This is done as the original 2D U-Net structure is unable of predicting meaningful results when used as the generator in the GAN. The original structure fails to converge during training, hence not being able to produce any meaningful results. In accordance with the suggestions proposed by Radford et. al. [53], the following modifications are mode:

- **Stridden convolutions in place of pooling operations**. This allows the network to learn its own downsampling, putting fewer restrictions on the model.
- **Introduction of batch normalization layers**. This stabilizes learning and prevents vanishing and exploding gradients in the deeper layers.
- **Use of the ReLU activation function for all hidden layers**. The use of a bounded activation function allows the model to more quickly learn the distribution of the training data.

The full generator architecture is shown in Figure 20. It has a total of 5,525,010 parameters, which is approximately the same as for the U-Net models previously described in 3.6.1. The discriminator architecture, shown in Figure 21 is similar to the first half of the generator architecture. It consists of stridden convolutions which gradually compress the input data. At the deepest compression layer, the layer output is flattened and fully connected to the output layer, that produces a binary output which represents whether the discriminator evaluates the input as real or fake. In contrast to the generator, the discriminator uses LReLU activations for all the hidden layers, which is recommended by Radford et. al. as they find this improves higher resolution modelling of the input space. The discriminator has a total of 101,393 parameters. This is substantially less than the generator has. This is necessary



(a) 2D U-Net training procedure



(b) 3D U-Net training procedure

Figure 19: The training procedures for the different architectures. The 2D U-Net is trained on each measurement batch wise, while the first measurement is kept as a validation batch. The 3D U-Net is trained by first interpolating all the measurements to a regular grid, and then stacking them together into training and validation sets.

as image discrimination is a much simpler task than image reconstruction, hence not requiring as much complexity in the model. When making the architectures for the GAN it is indeed important to balance the complexities of the generator and the discriminator, as an imbalance would make an ”unfair game”. This would make it too easy for one of the models to win over the other, thus making it difficult for any of the models to learn.

3.7.2 Adversarial Training

The GAN is trained and evaluated on the in-silico ultrasound blood flow measurement data set, which is generated as described in 3.5. We use all three velocity components. The GAN adapts a structure similar to the 2D U-Net 3-3, and is hence trained on static velocity field frames. To evaluate the performance of the GAN approach the generator predictions are compared to the predictions of the same generator model which is trained directly using a MSE loss function, i.e. with an explicit loss and with no adversarial training. In addition to the ultrasound dataset, the GAN is also trained and evaluated on the MNIST handwritten digits [58] data set in the same way. The MNIST data is added white Gaussian noise and the generator is tasked with denoising these images. This is done to see how the GAN performs on a simpler task than the reconstruction of blood flow measurements.

The training procedure consists of cascading the generator and discriminator, and compiling the discriminator with a binary cross-entropy loss function and the ADAM optimizer. The optimizer uses a low learning rate of 0.0002 and a low momentum term β_1 of 0.5. This was again done in accordance with recommendation by Radford et. al. The entire training procedure is illustrated in Figure 22 and consists of the following 10 steps, which are repeated for a large number of iterations (epochs):

1. Sample 1/2 batch of ground truth data.
2. Train the discriminator on the ground truth samples with true labels.
3. Sample 1/2 batch of noisy data.
4. The generator predicts fake (reconstructed) data from the noisy data.
5. Train the discriminator on the fake samples with true labels.
6. Sample a full batch of noisy data.
7. Fix the discriminator weights.
8. The generator predicts fake data from the noisy samples.
9. The fake samples are given to the discriminator with false labels. The generator updates its weights by learning from the discriminator’s evaluation.
10. Unfix the discriminator weights.

The least intuitive step is step number 9. When the generator gives its fake data to the discriminator, the discriminator will predict whether the given data is real ground truth data or fake data. The generator benefits from the discriminator being fooled by the generated data. Hence it is beneficial for the generator when the discriminator makes the wrong decision, and this is why the generator is updated by training with false labels. As the discriminator’s weights are fixed at this point, the false labels do not cause the discriminator’s learning process to deteriorate. To evaluate the performance of the GAN its results are compared with the results of a model with the generator’s architecture, but trained directly with a MSE loss function. The GAN and the directly trained model are trained with approximately the same amount of data. Let N_{train} be the number of training data samples. B_{GAN} and E_{GAN} respectively denote the batch size and the number of epochs used when training the GAN. Similarly, B_{direct} and E_{direct} denote the batch size and number of epochs used to train the direct model. Since the direct model is trained with all the training samples each epoch, while the GAN’s generator is only trained with the number of

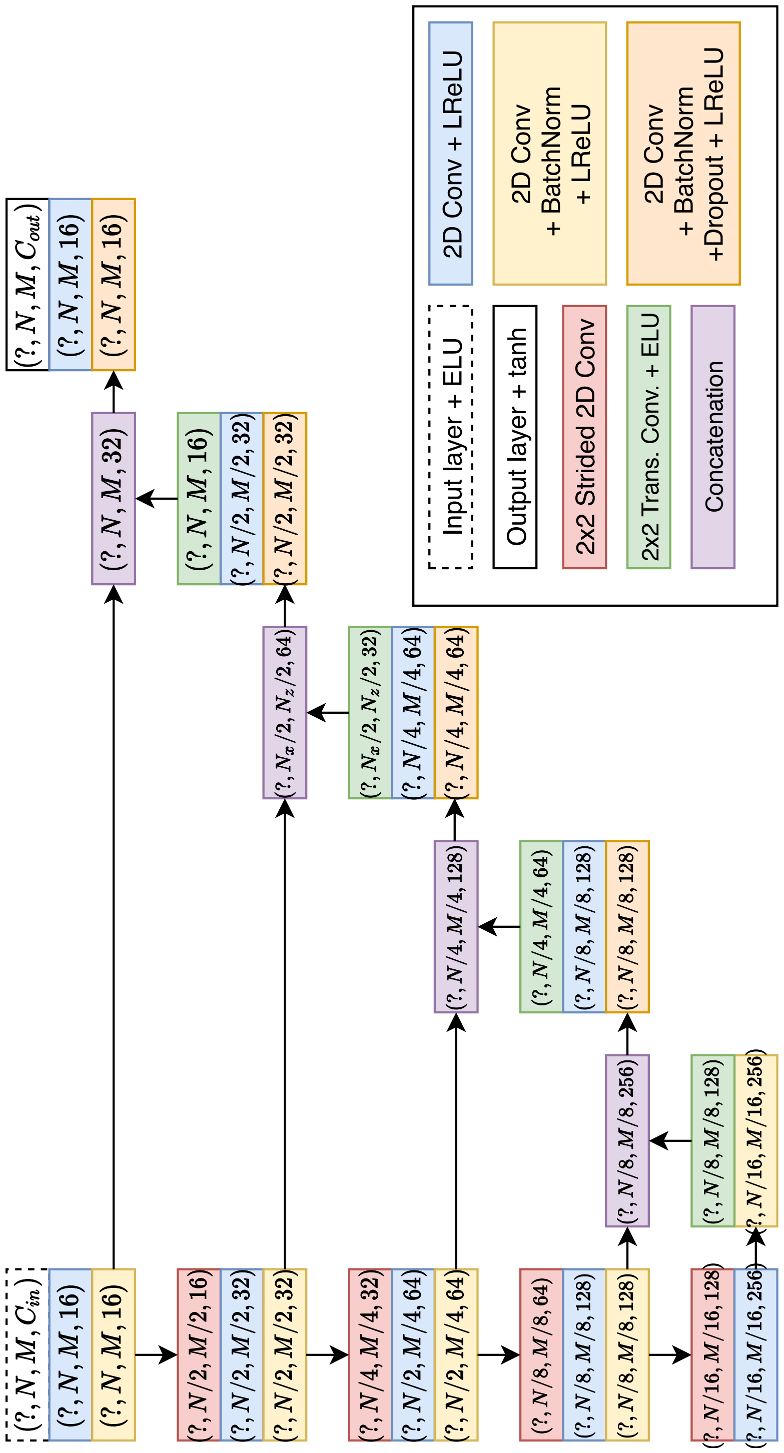


Figure 20: Generator architecture. The tuple in each layer represents the output shape of the tensor output by the layer. The tensors are four dimensional. The first dimension is the batch size, an arbitrary integer represented by "?". The second, third and fourth dimensions are respectively the image height, image width and channels.

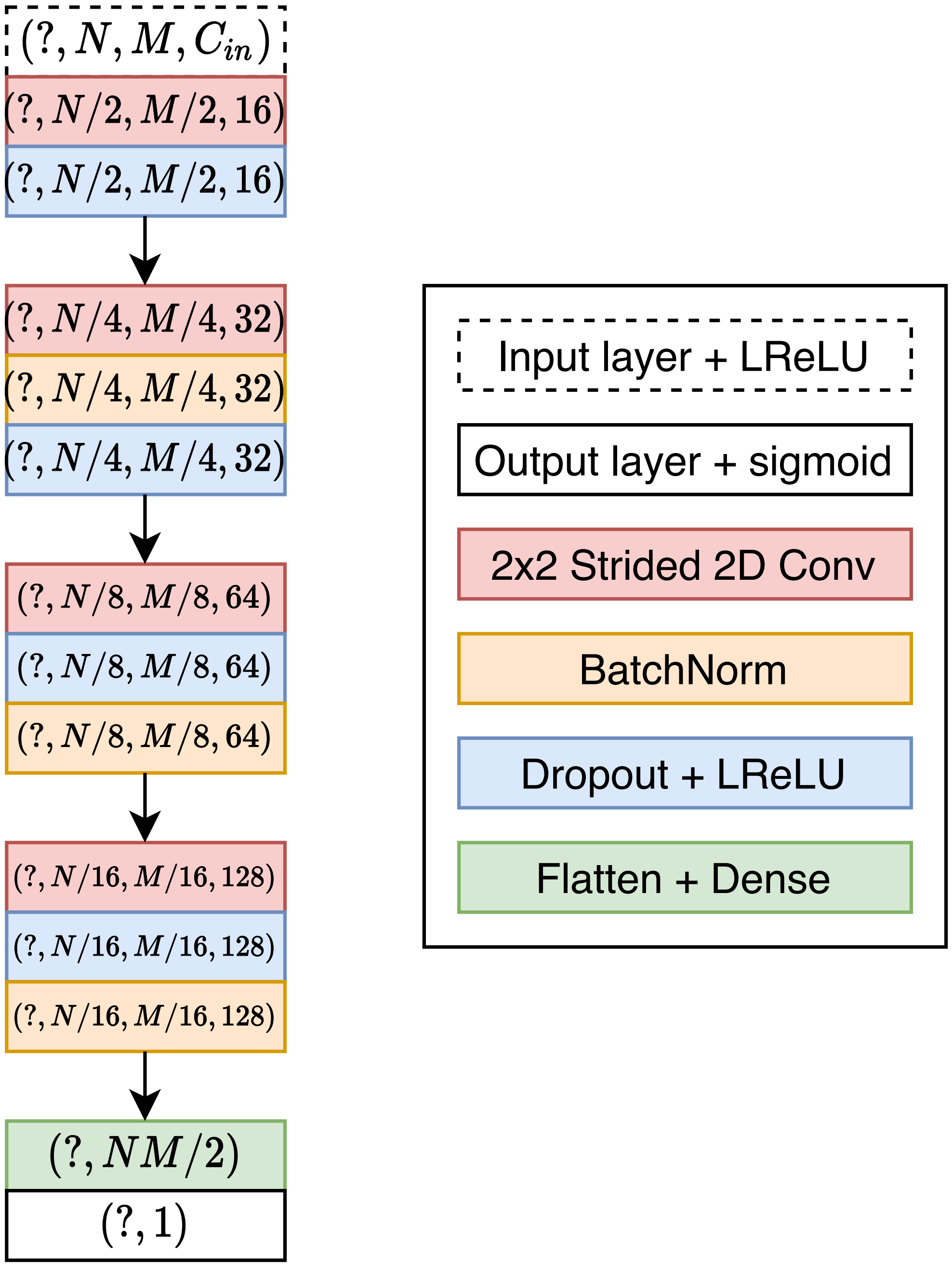


Figure 21: Discriminator architecture. The tuple in each layer represents the output shape of the tensor output by the layer. The tensors are four dimensional. The first dimension is the batch size, an arbitrary integer represented by "?". The second, third and fourth dimensions are respectively the image height, image width and channels.

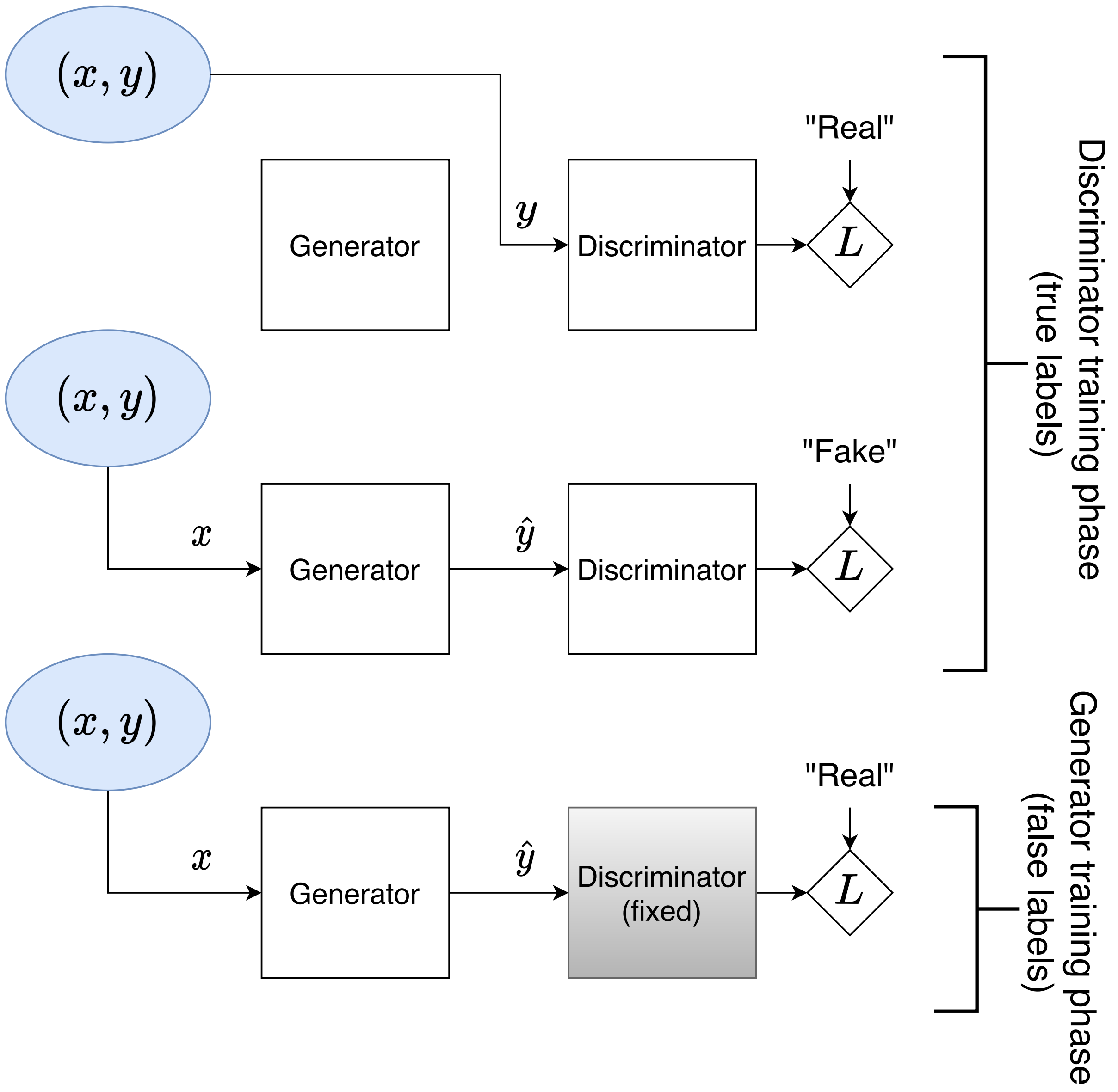


Figure 22: GAN training procedure. Firstly the discriminator is updated in two turns, once with the ground truth y, and once with the fake data $\hat{y}$ generated from the noisy data x, learning through the evaluation of the loss function L. Secondly the generator is updated by attempting to fool the discriminator with its generated fake data, labeling it as real.

Table 3: Training variable sizes for the adversarially and directly trained generator models.

	In-silico ultrasound velocity data	**MNIST handwritten digits**
N_{train}	256	60,000
B_{GAN}	64	64
E_{GAN}	10,000	2000
B_{direct}	64	64
E_{direct}	40	1

samples corresponding to the batch size each epoch, the relation between the parameters are as shown in (31). The parameter values are displayed in Table 3

$$E_{direct} \approx \frac{B_{GAN}E_{GAN}}{N_{train}}. \tag{31}$$

4 Results

4.1 In-Silico Results

This section contains the results related to the U-Net trained and tested purely on the simulated ultrasound blood velocity data. The data set used for the 2D U-Net contains 10 000 data samples, where 6400 are used for training, 1600 for validation and 2000 for testing. Each data sample contains a 2D image with three channels, each channel corresponding to a vector field component. The data set contains noisy BST data slices of the FUSK data, and slices of the CFD phantom as target labels. There are in addition correlation maps for each component of the BST data. The data set used for the 3D U-Net contains 1128 data samples, where 786 are used for training, 196 for validation and 246 for testing. Each data sample consists of 25 time frames of a noisy BST slice changing dynamically in time, and labels in the form of the corresponding CFD phantom time frames. There are in addition correlation maps for each component of the BST data.

4.1.1 Exploring the Model Capabilities

Figures 23-25 show model predictions from the 2D U-Net architecture using different training procedures. They vary in which field components are inserted to and output from the model, as well as whether or not correlation maps are concatenated to the input layer as an additional piece of information fed to the network. The figures each show a different data sample from the test set and the corresponding model predictions. The data is visualized as the output field (V_x, V_z) overlaid the output velocity magnitude. Figures 26-28 show the corresponding predictions from the 3D U-Net. Figure 29 shows the MMSE of the model predictions averaged over all the data points in the entire test set. Figures 30 and 31 show the MSE of the 2D and 3D U-Net model predictions averaged over all the data points in the entire test set, for each time frame.

4.1.2 Reconstruction of the Out-of-Plane Component

Figures 32 and 33 show the noisy input data, ground truth and U-Net predictions of the out-of-plane component (V_y) for two different samples. The *U-Net 2-3* models do not have access to V_y at the input and are therefore predicting V_y only from V_x and V_z. Figure 34 shows the training history for the 3D U-Net 2-3 models with and without correlation maps.

4.1.3 Compression in Three Dimensions

Figures 35 and 36 show a comparison between the model predictions of the 3D U-Net 2-2 architecture which only performs pooling and transposed convolutions in the spatial domain, and the 3D U-Net 2-2 architecture which additionally performs pooling and transposed convolutions in the time domain. Figure 38 shows the same comparison, but visualized as the radial component velocity heat map with no segmentation mask. Figure 37 shows the mean-squared-difference between the predictions of model variations and the result from B-spline interpolation, averaged over all measurements.

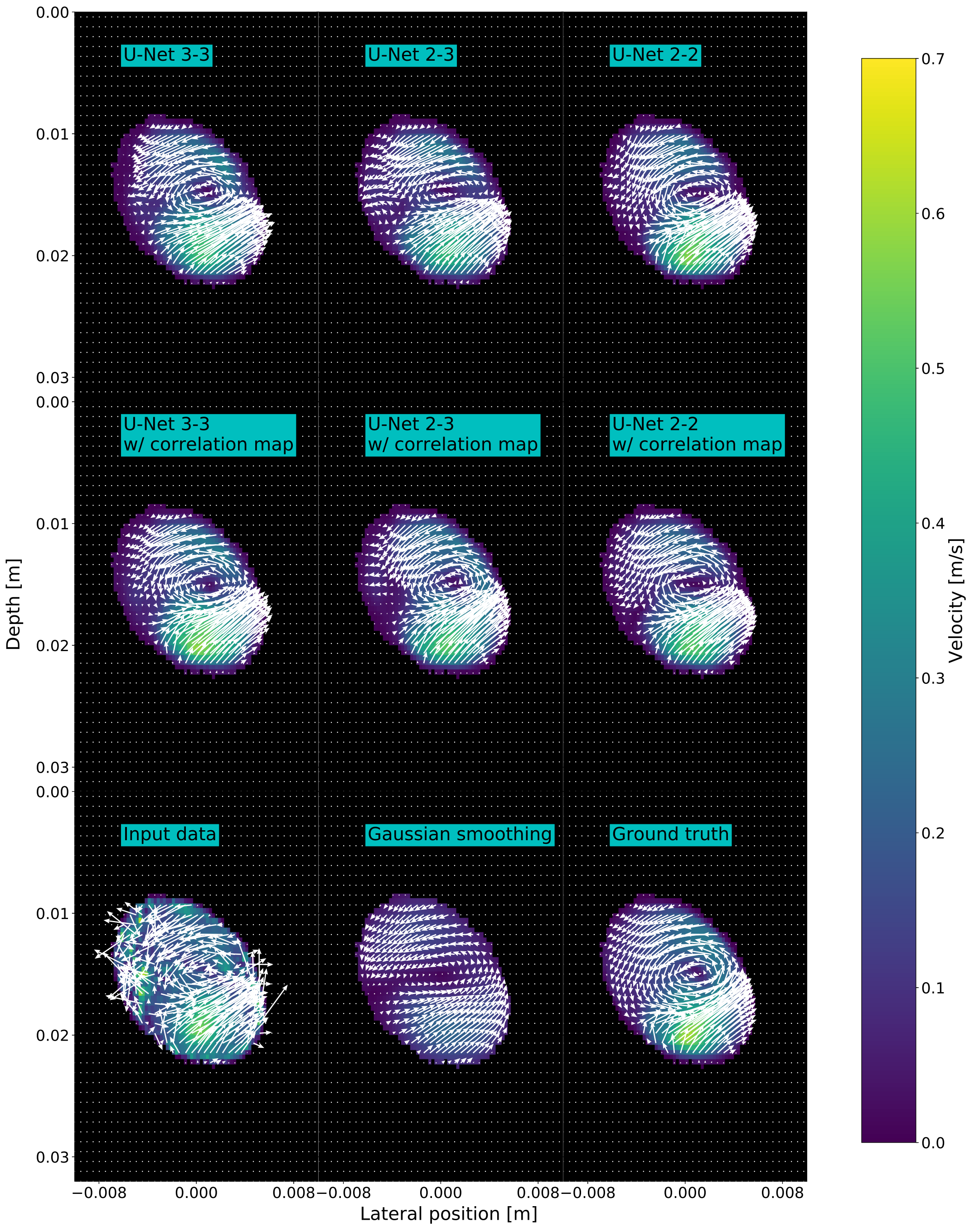


Figure 23: 2D U-Net predictions on simulated ultrasound blood velocity field data, sample 1, frame 15. The data is visualized as the (V_x, V_z) field overlaid the velocity magnitude. The results are showing one frame predicted from the same *input data* from the different models. *U-Net 3-3* denotes a 2D U-Net trained to output velocity field components (x, y, z) from input components (x, y, z). *U-Net 2-3* denotes a 2D U-Net trained to output velocity field components (x, y, z) from input components (x, z). *U-Net 2-2* denotes a 2D U-Net trained to output velocity field components (x, z) from input components (x, z). The models trained */w correlation maps* have the correlation maps corresponding to the input components concatenated at the input layer.

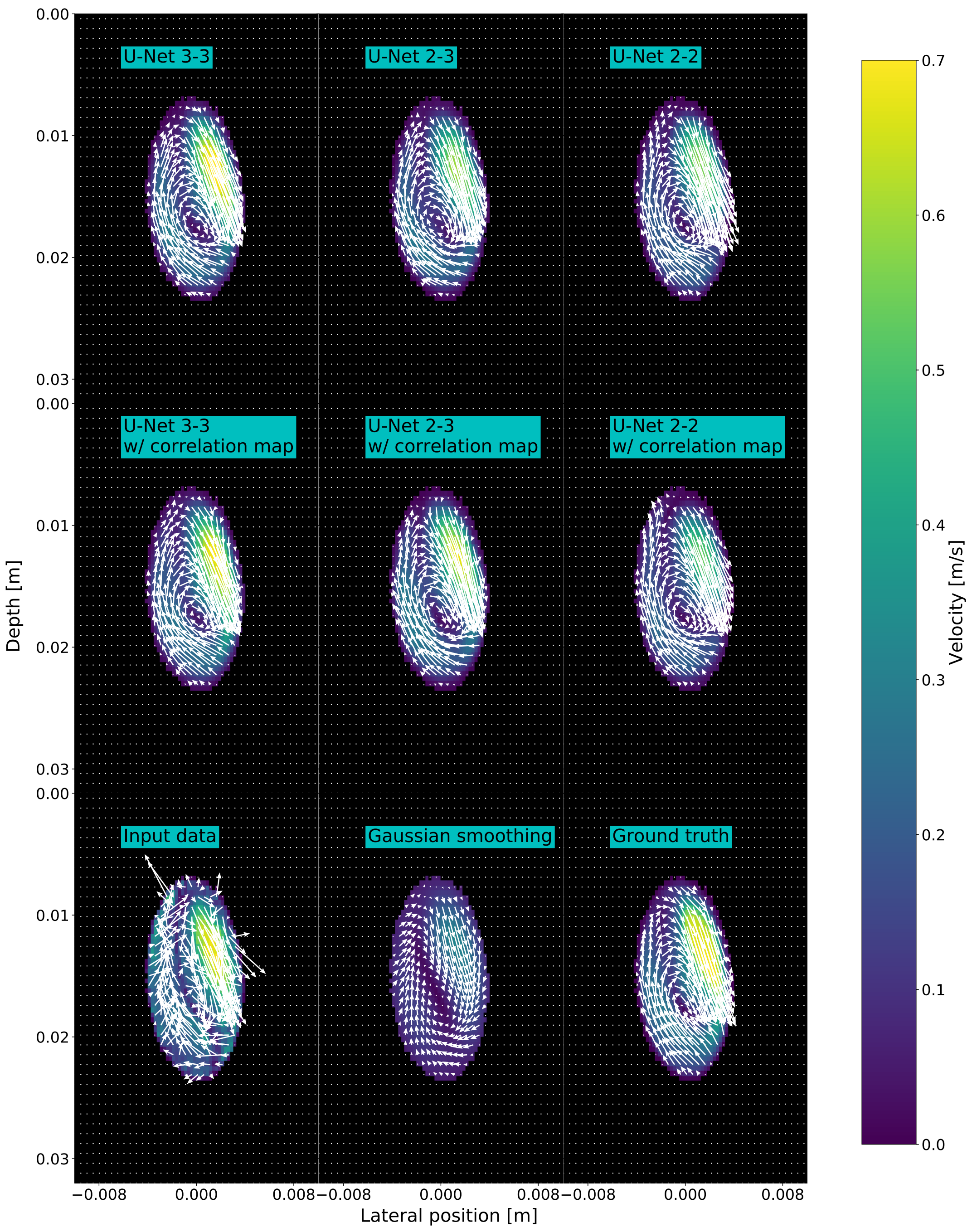


Figure 24: 2D U-Net predictions on simulated ultrasound blood velocity field data, sample 2, frame 15. The data is visualized as the (V_x, V_z) field overlaid the velocity magnitude. The results are showing one frame predicted from the same *input data* from the different models. *U-Net 3-3* denotes a 2D U-Net trained to output velocity field components (x, y, z) from input components (x, y, z). *U-Net 2-3* denotes a 2D U-Net trained to output velocity field components (x, y, z) from input components (x, z). *U-Net 2-2* denotes a 2D U-Net trained to output velocity field components (x, z) from input components (x, z). The models trained */w correlation maps* have the correlation maps corresponding to the input components concatenated at the input layer.

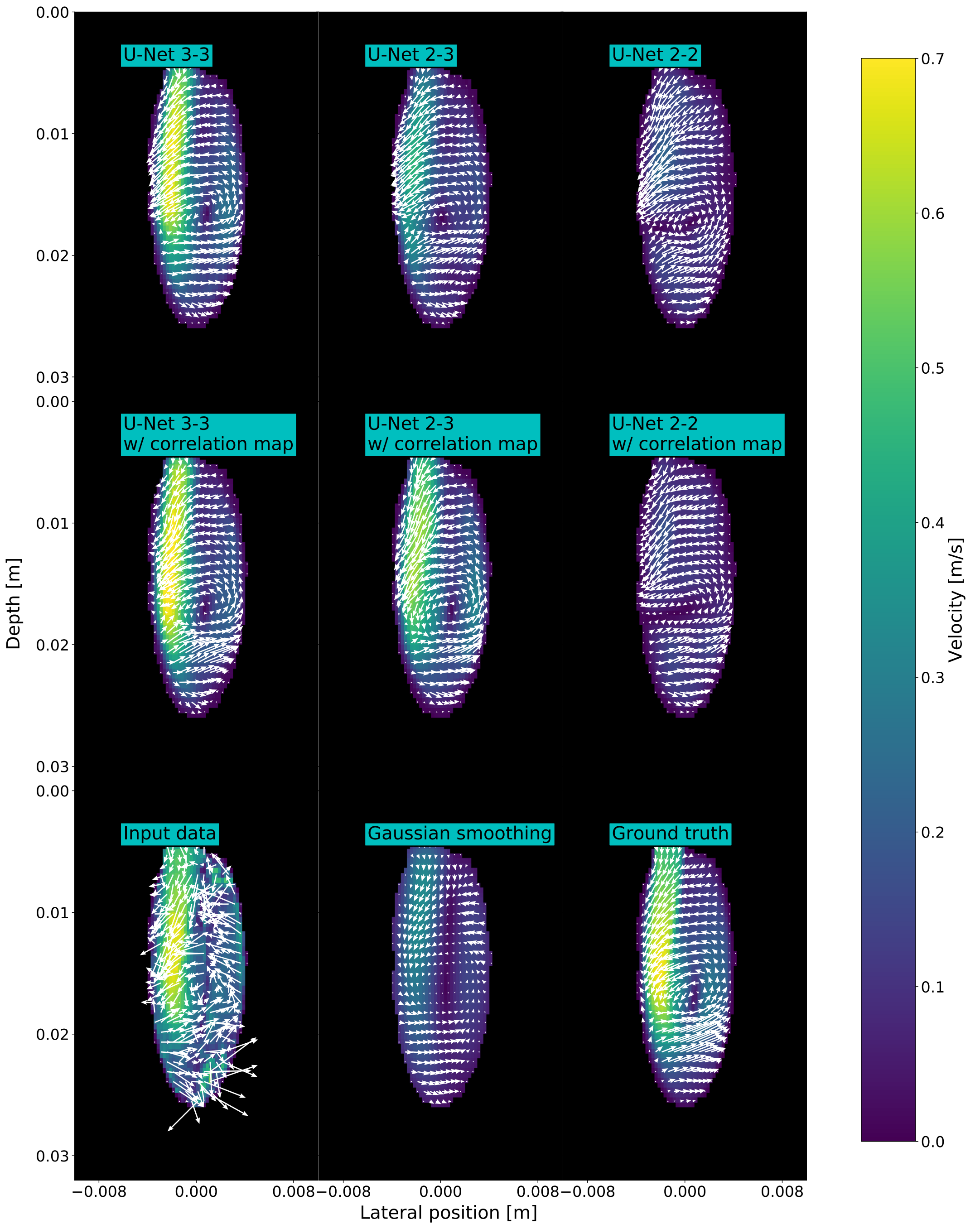


Figure 25: 2D U-Net predictions on simulated ultrasound blood velocity field data, sample 3, frame 15. The data is visualized as the (V_x, V_z) field overlaid the velocity magnitude. The results are showing one frame predicted from the same *input data* from the different models. *U-Net 3-3* denotes a 2D U-Net trained to output velocity field components (x, y, z) from input components (x, y, z). *U-Net 2-3* denotes a 2D U-Net trained to output velocity field components (x, y, z) from input components (x, z). *U-Net 2-2* denotes a 2D U-Net trained to output velocity field components (x, z) from input components (x, z). The models trained */w correlation maps* have the correlation maps corresponding to the input components concatenated at the input layer.

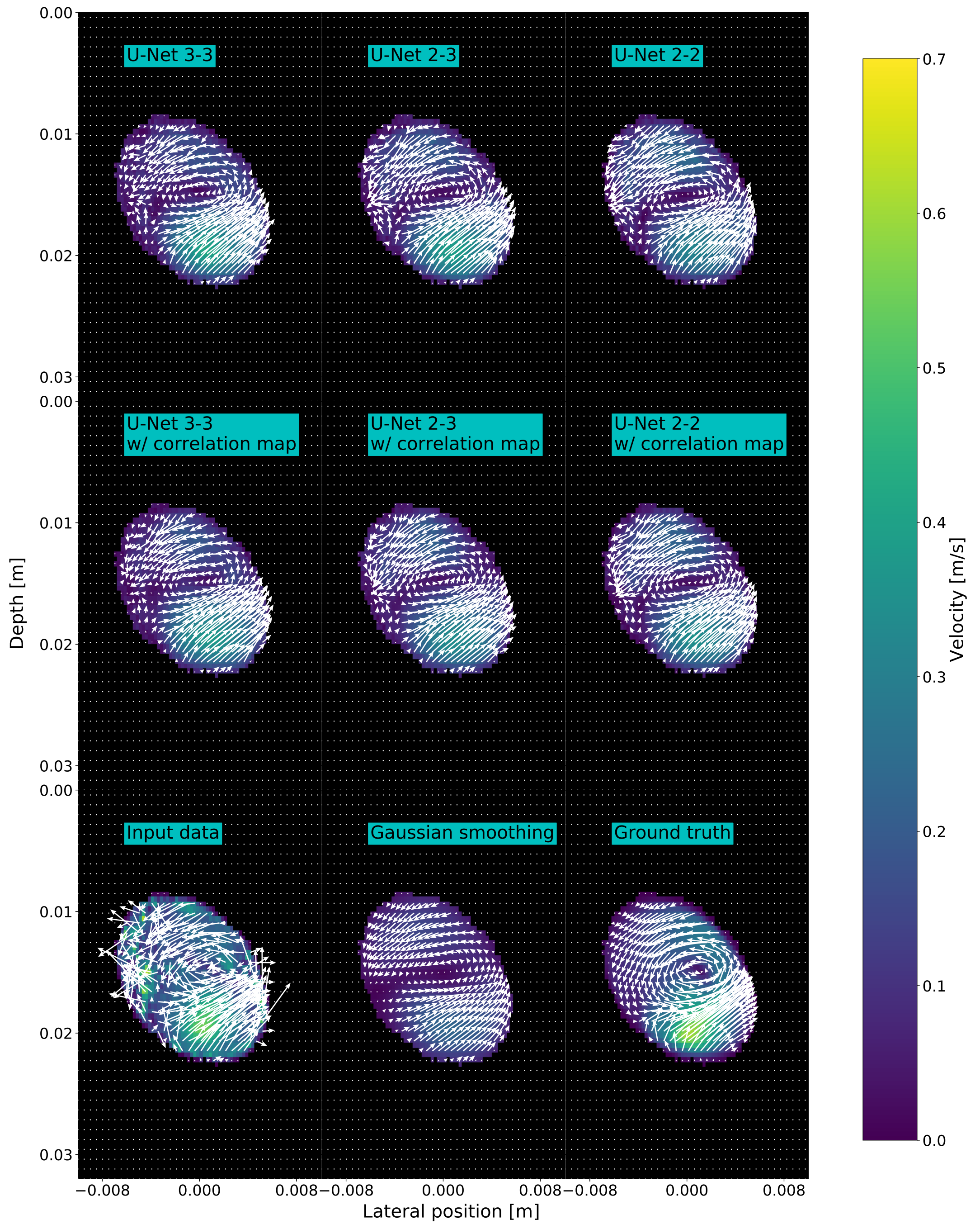


Figure 26: 3D U-Net predictions on simulated ultrasound blood velocity field data, sample 1, frame 15. The data is visualized as the (V_x, V_z) field overlaid the velocity magnitude. The data is visualized as the (V_x, V_z) field overlaid the velocity magnitude. The results are showing one frame predicted from the same *input data* from the different models. *U-Net 3-3* denotes a 3D U-Net trained to output velocity field components (x, y, z) from input components (x, y, z). *U-Net 2-3* denotes a 3D U-Net trained to output velocity field components (x, y, z) from input components (x, z). *U-Net 2-2* denotes a 3D U-Net trained to output velocity field components (x, z) from input components (x, z). The models trained */w correlation maps* have the correlation maps corresponding to the input components concatenated at the input layer.

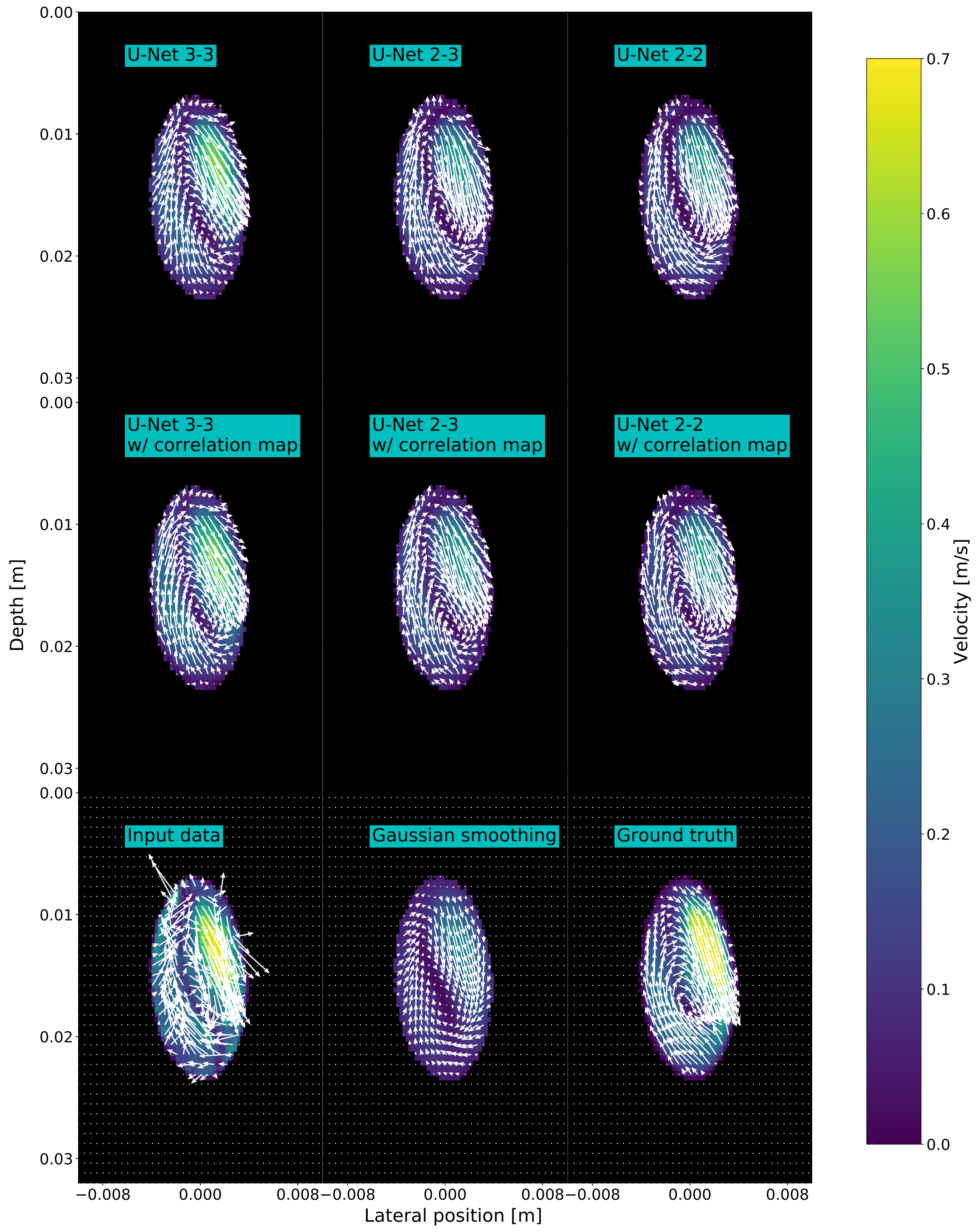


Figure 27: 3D U-Net predictions on simulated ultrasound blood velocity field data, sample 2, frame 15. The data is visualized as the (V_x, V_z) field overlaid the velocity magnitude. The results are showing one frame predicted from the same *input data* from the different models. *U-Net 3-3* denotes a 3D U-Net trained to output velocity field components (x, y, z) from input components (x, y, z). *U-Net 2-3* denotes a 3D U-Net trained to output velocity field components (x, y, z) from input components (x, z). *U-Net 2-2* denotes a 3D U-Net trained to output velocity field components (x, z) from input components (x, z). The models trained */w correlation maps* have the correlation maps corresponding to the input components concatenated at the input layer.

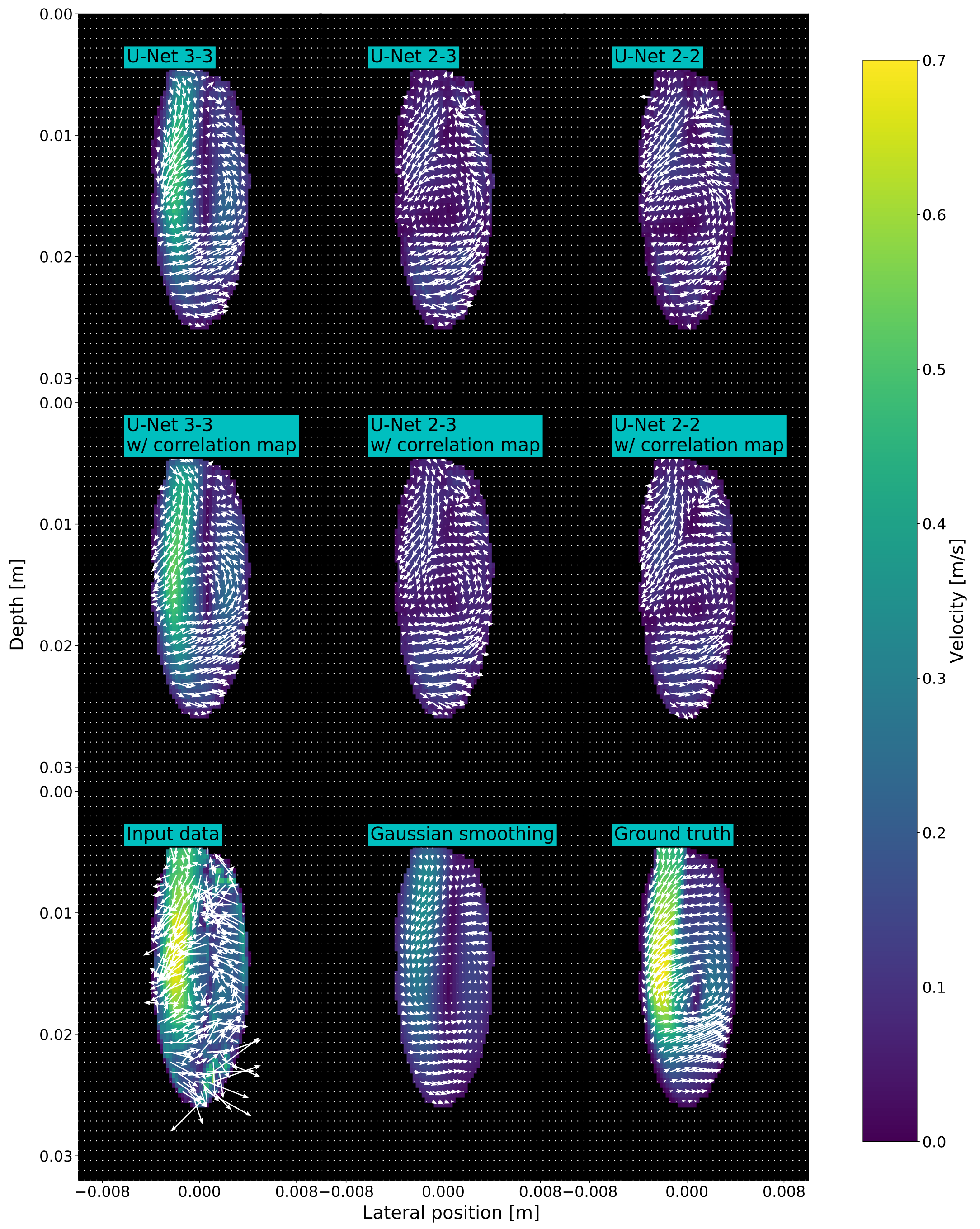
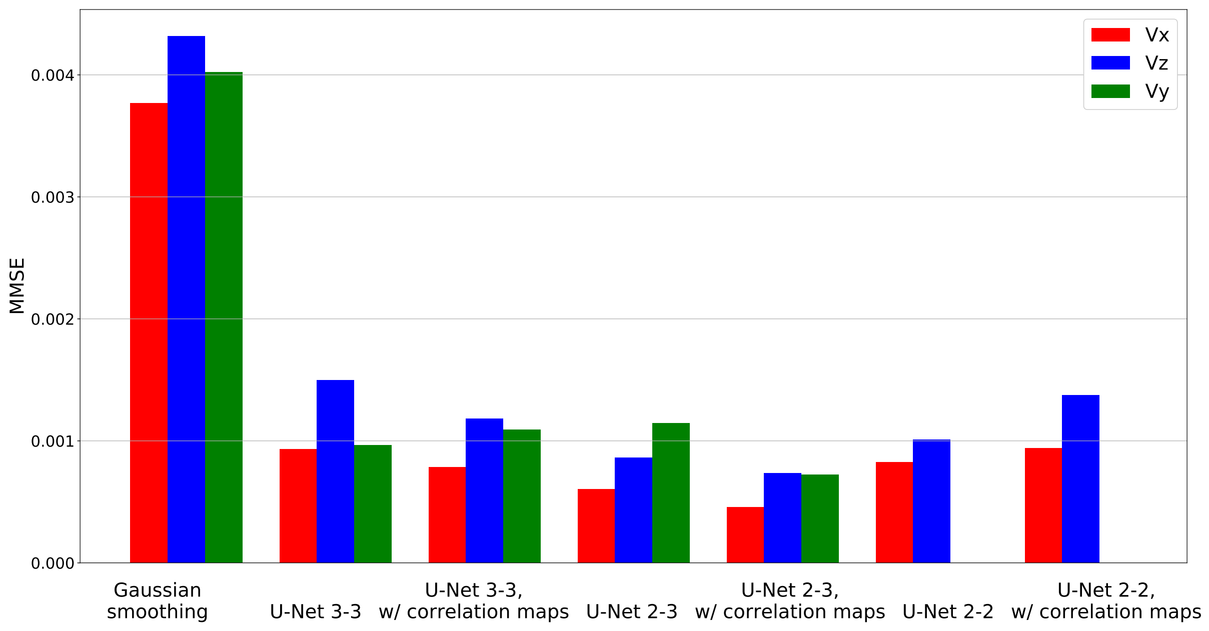
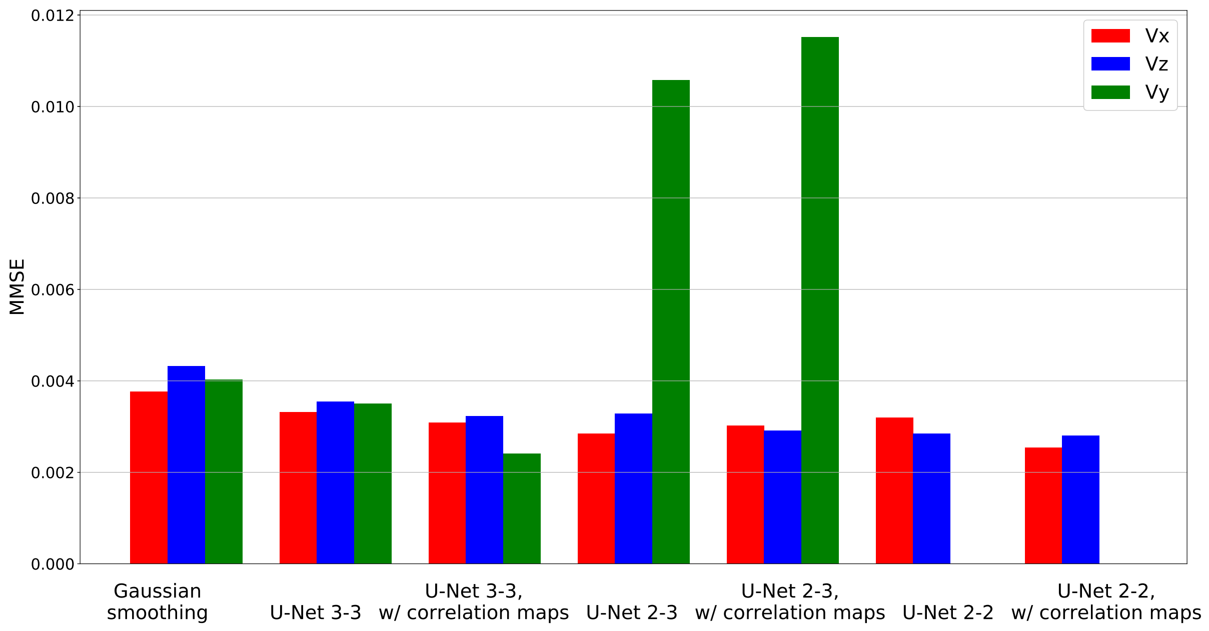


Figure 28: 3D U-Net predictions on simulated ultrasound blood velocity field data, sample 3, frame 15. The data is visualized as the (V_x, V_z) field overlaid the velocity magnitude. The results are showing one frame predicted from the same *input data* from the different models. *U-Net 3-3* denotes a 3D U-Net trained to output velocity field components (x, y, z) from input components (x, y, z). *U-Net 2-3* denotes a 3D U-Net trained to output velocity field components (x, y, z) from input components (x, z). *U-Net 2-2* denotes a 3D U-Net trained to output velocity field components (x, z) from input components (x, z). The models trained */w correlation maps* have the correlation maps corresponding to the input components concatenated at the input layer.



(a) 2D U-Net



(b) 3D U-Net

Figure 29: Masked-mean-squared-error (MMSE) of the model predictions on simulated ultrasound blood velocity field data. The *2D U-Net* has 2D convolutional layers which operate in the spatial domain while the *3D U-Net* has 3D convolutional layers which operate in both spatial and temporal domains. *U-Net 3-3* denotes a U-Net trained to output velocity field components (x, y, z) from input components (x, y, z). *U-Net 2-3* denotes a U-Net trained to output velocity field components (x, y, z) from input components (x, z). *U-Net 2-2* denotes a U-Net trained to output velocity field components (x, z) from input components (x, z). The models trained */w correlation maps* have the correlation maps corresponding to the input components concatenated at the input layer.

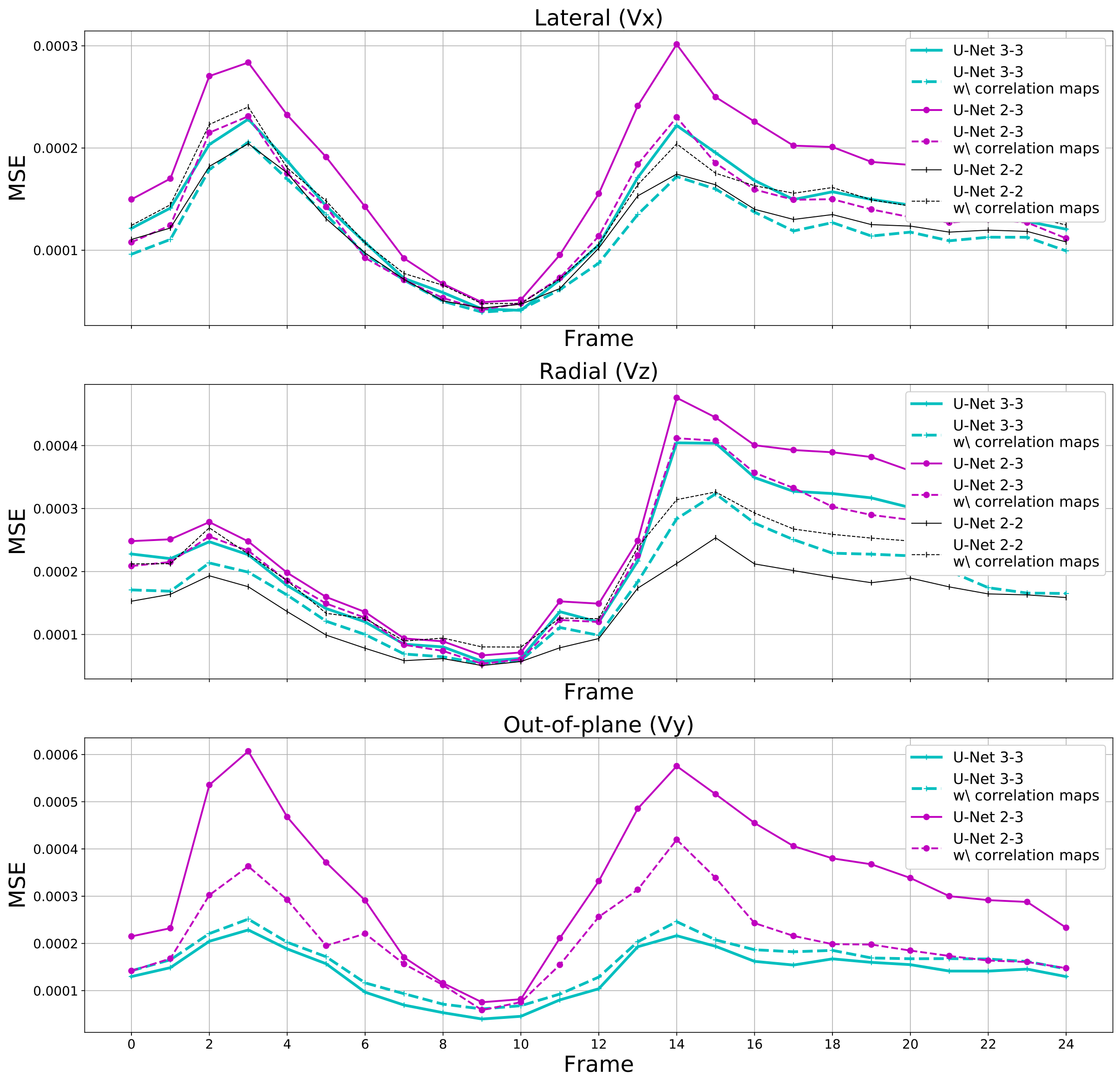


Figure 30: Mean-squared-error (MSE) of the 2D U-Net model predictions on simulated ultrasound blood velocity field data, for each frame averaged over the entire test set. *U-Net 3-3* denotes a 2D U-Net trained to output velocity field components (x, y, z) from input components (x, y, z). *U-Net 2-3* denotes a 2D U-Net trained to output velocity field components (x, y, z) from input components (x, z). *U-Net 2-2* denotes a 2D U-Net trained to output velocity field components (x, z) from input components (x, z). The models trained */w correlation maps* have the correlation maps corresponding to the input components concatenated at the input layer.

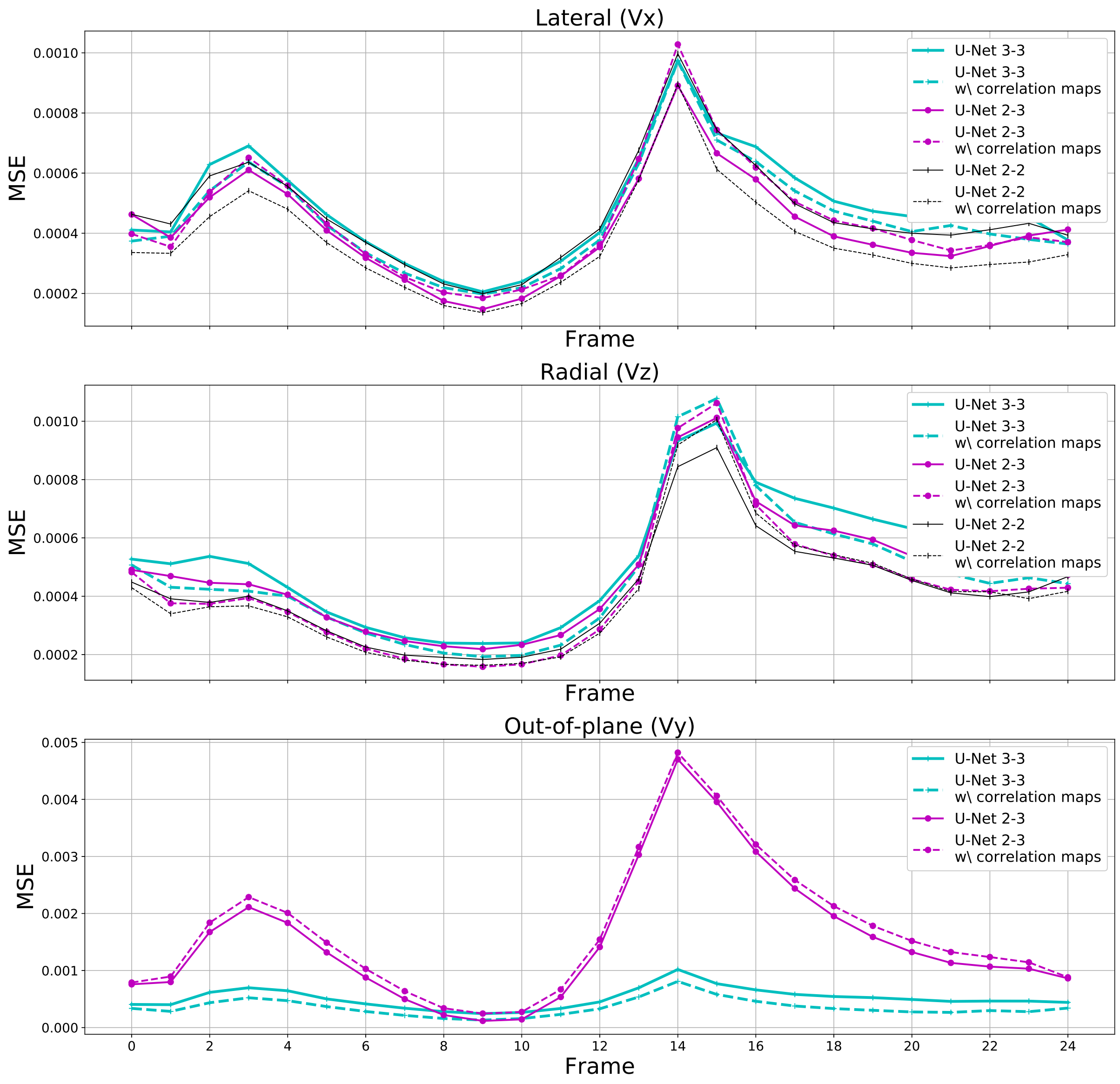
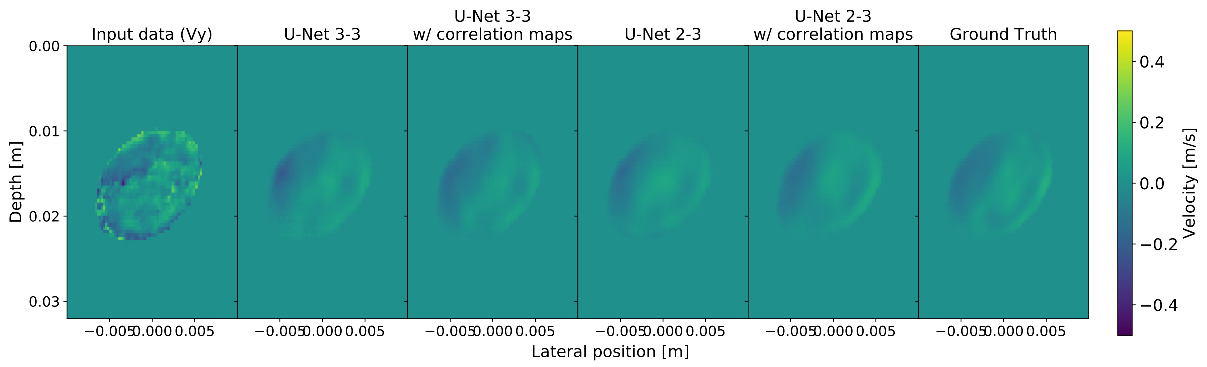
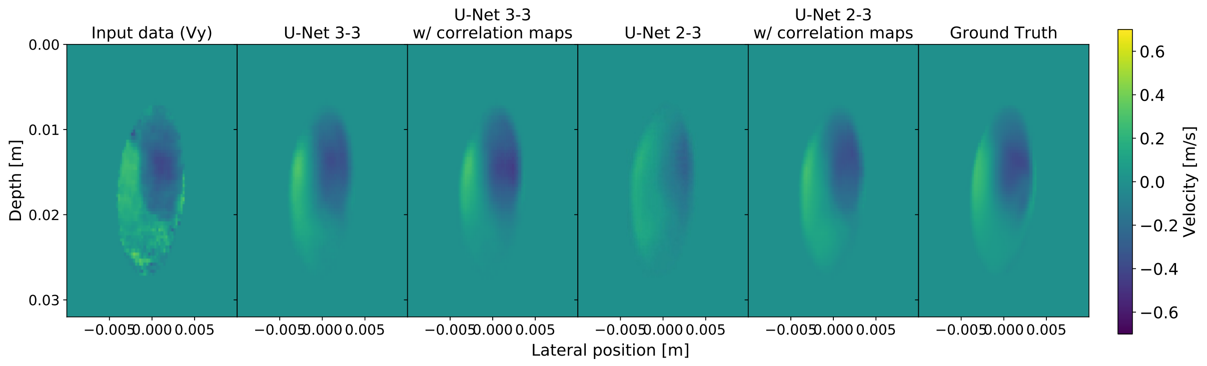


Figure 31: Mean-squared-error (MSE) of the 2D U-Net model predictions on simulated ultrasound blood velocity field data, for each frame averaged over the entire test set. *U-Net 3-3* denotes a 2D U-Net trained to output velocity field components (x, y, z) from input components (x, y, z). *U-Net 2-3* denotes a 2D U-Net trained to output velocity field components (x, y, z) from input components (x, z). *U-Net 2-2* denotes a 2D U-Net trained to output velocity field components (x, z) from input components (x, z). The models trained */w correlation maps* have the correlation maps corresponding to the input components concatenated at the input layer.

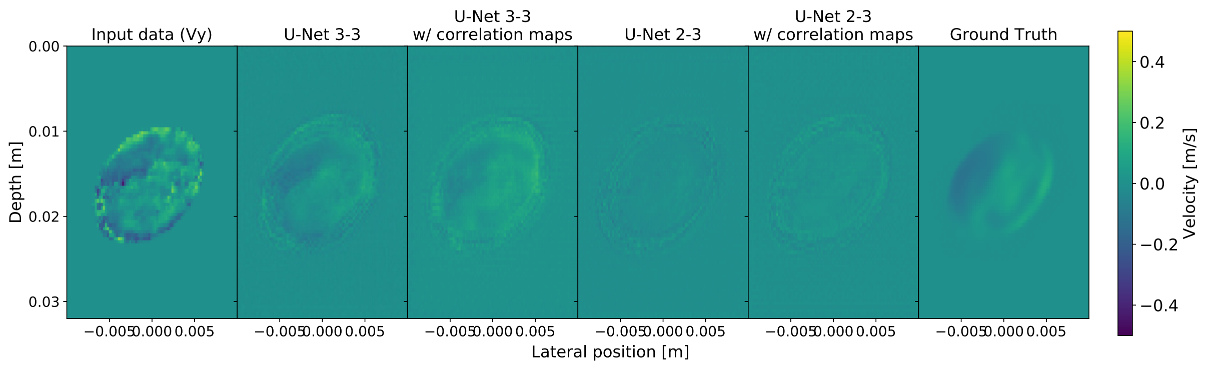


(a) 2D U-Net, Sample 1

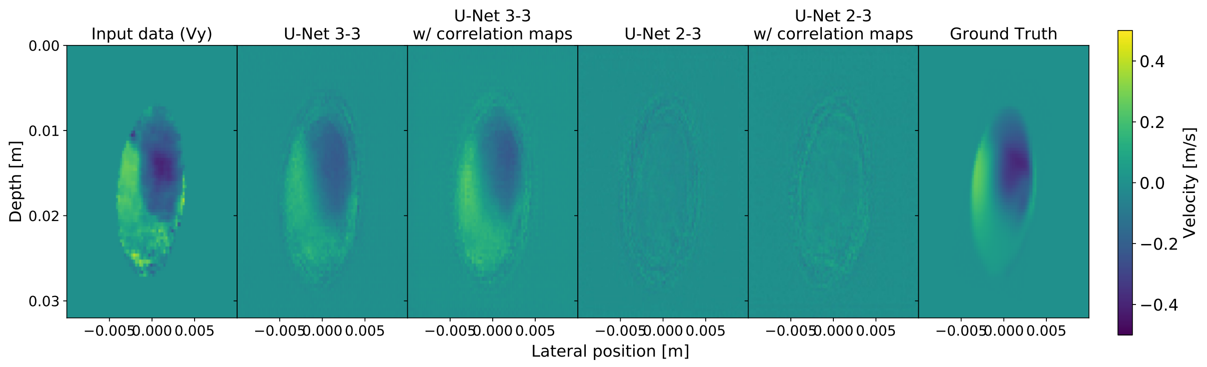


(b) 2D U-Net, Sample 2

Figure 32: Model predictions on simulated ultrasound blood velocity field data. The results are showing the out-of-plane component (V_y) of one frame predicted from the same *input data* from the different models. *U-Net 3-3* denotes a 2D U-Net trained to output velocity field components (x, y, z) from input components (x, y, z). *U-Net 2-3* denotes a 2D U-Net trained to output velocity field components (x, y, z) from input components (x, z). *U-Net 2-2* denotes a 2D U-Net trained to output velocity field components (x, z) from input components (x, z). The models trained */w correlation maps* have the correlation maps corresponding to the input components concatenated at the input layer. Only the U-Net 3-3 models have access to the V_y input data shown at the input layer. The U-Net 2-2 models only have access to the x and z components as inputs.

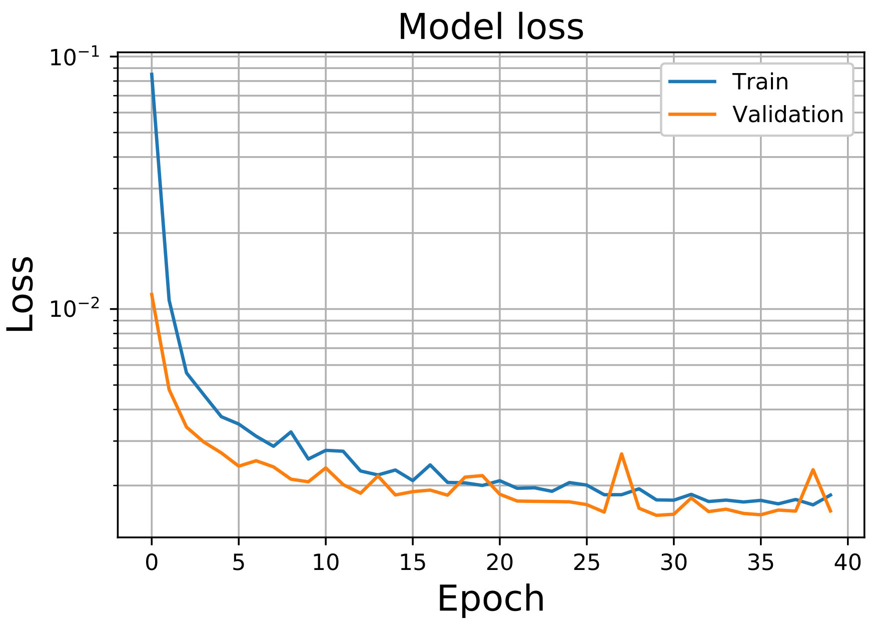


(a) 3D U-Net, Sample 1

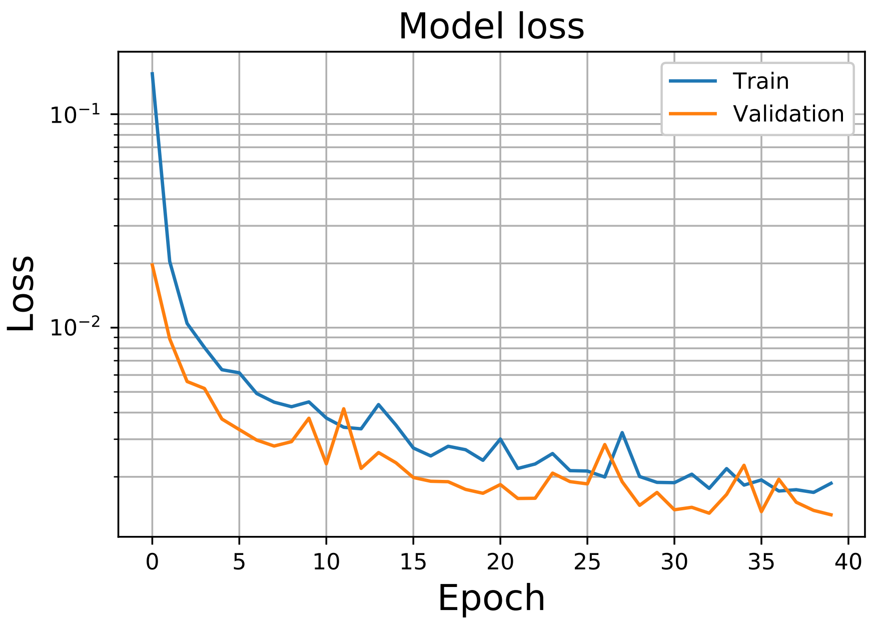


(b) 3D U-Net, Sample 2

Figure 33: Model predictions on simulated ultrasound blood velocity field data. The results are showing the out-of-plane component (V_y) of one frame predicted from the same *input data* from the different models. *U-Net 3-3* denotes a 3D U-Net trained to output velocity field components (x, y, z) from input components (x, y, z). *U-Net 2-3* denotes a 3D U-Net trained to output velocity field components (x, y, z) from input components (x, z). *U-Net 2-2* denotes a 3D U-Net trained to output velocity field components (x, z) from input components (x, z). The models trained */w correlation maps* have the correlation maps corresponding to the input components concatenated at the input layer. Only the U-Net 3-3 models have access to the V_y input data shown at the input layer. The U-Net 2-2 models only have access to the x and z components as inputs.



(a) 3D U-Net 2-3



(b) 3D U-Net 2-3 w/ correlation maps

Figure 34: Training history for the 3D U-Net 2-3 models with and without correlation map inputs. Both train and validation losses converge to a value of about 0.02 after 40 epochs.

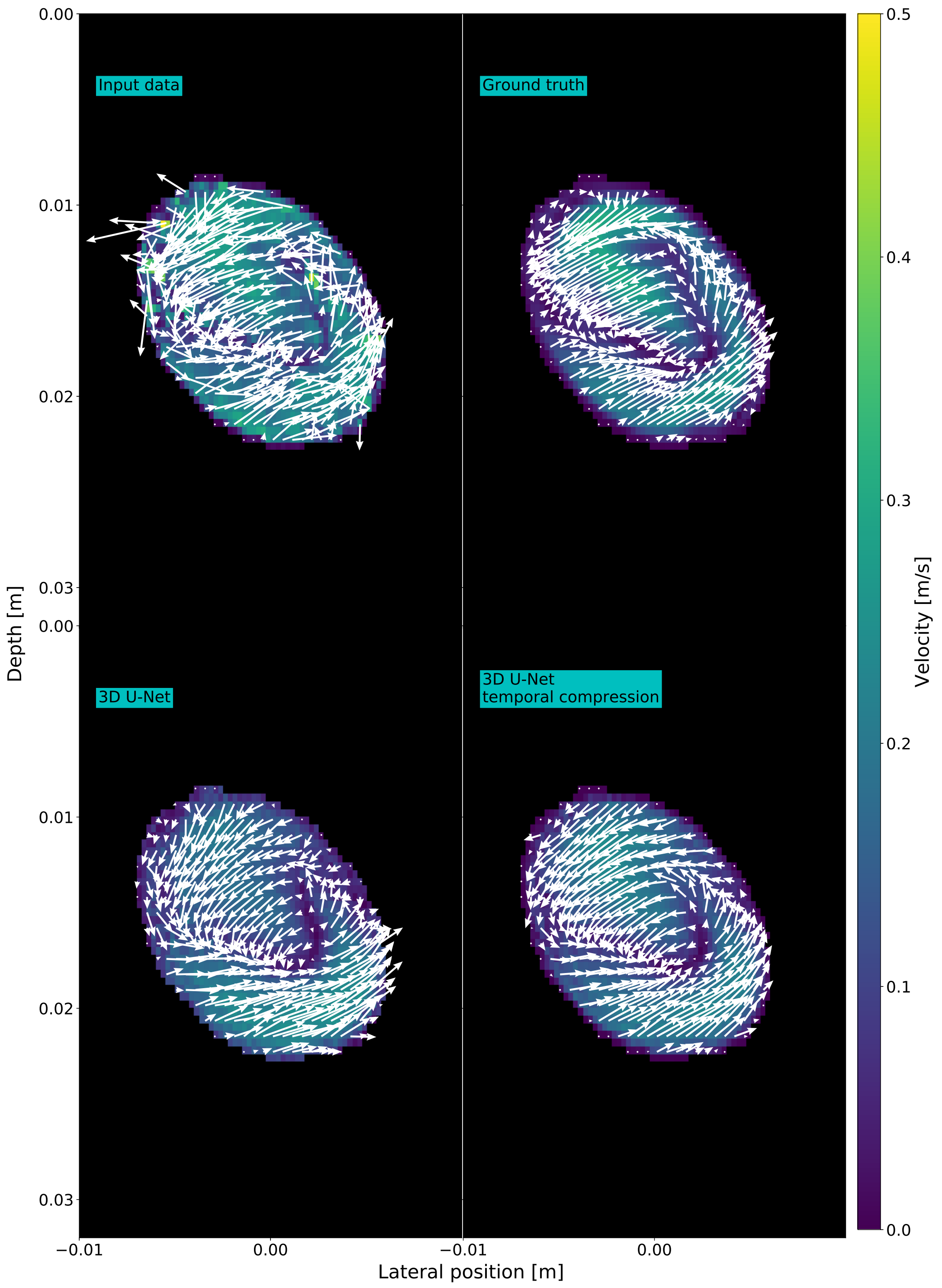


Figure 35: Sample 1, frame 20/25. Comparison of model predictions between the *3D U-Net* model only applying pooling and transposed convolutions in the spatial domain, and the *3D U-Net, temporal compression* which is also applying pooling and stridden convolutions in the time domain. The corresponding input data and the ground truth data is also shown. The data is visualized as the (V_x, V_z) field overlaid the velocity magnitude.

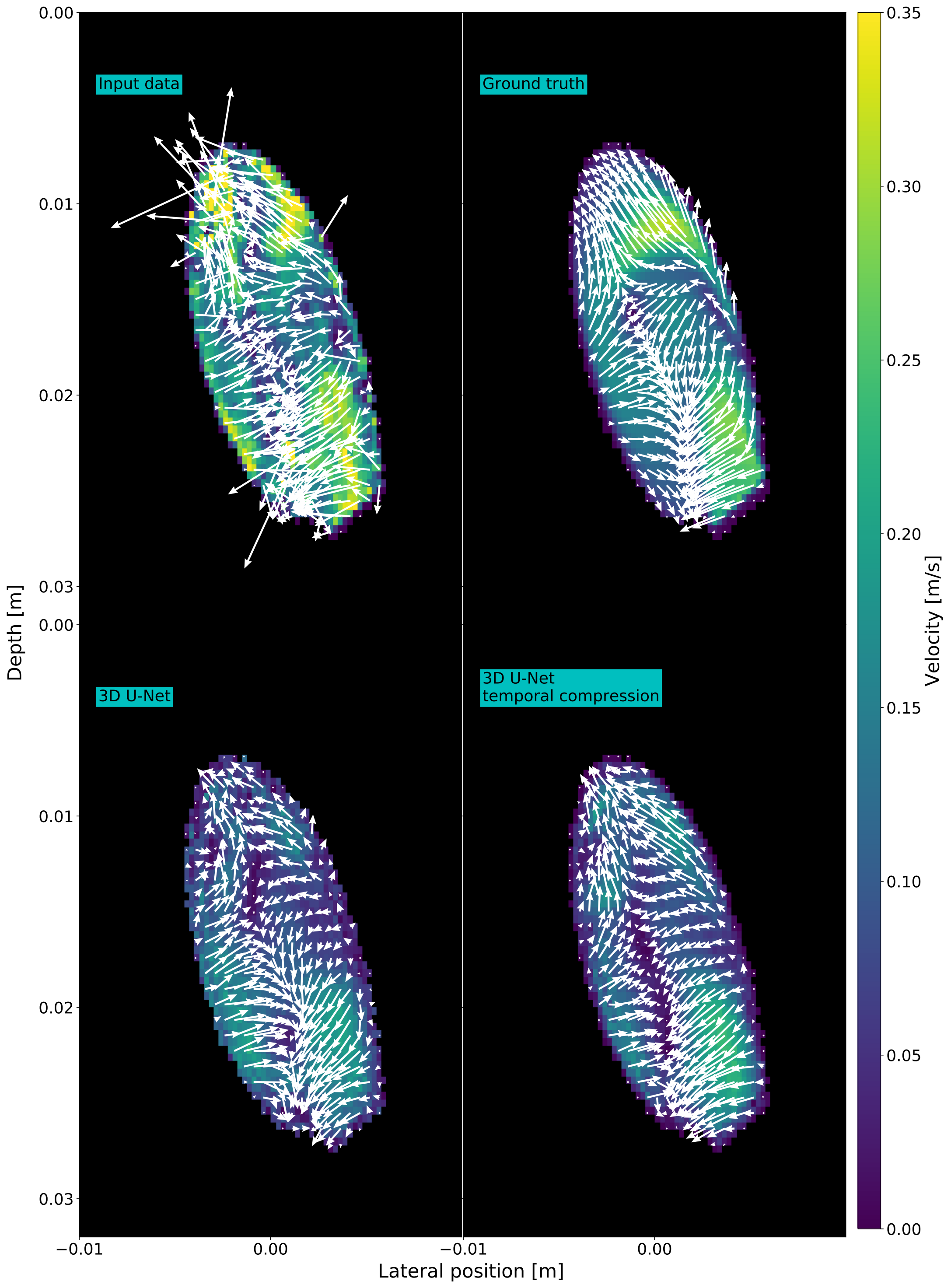


Figure 36: Sample 2, frame 15/25. Comparison of model predictions between the *3D U-Net* model only applying pooling and transposed convolutions in the spatial domain, and the *3D U-Net, temporal compression* which is also applying pooling and stridden convolutions in the time domain. The corresponding input data and the ground truth data is also shown. The data is visualized as the (V_x, V_z) field overlaid the velocity magnitude.

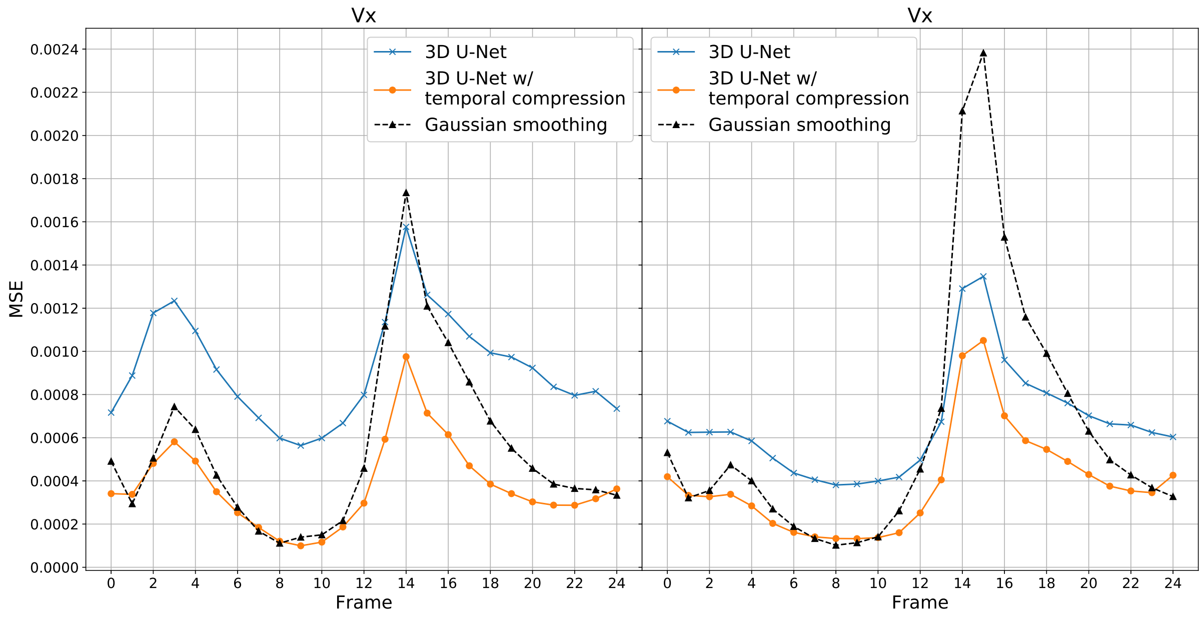
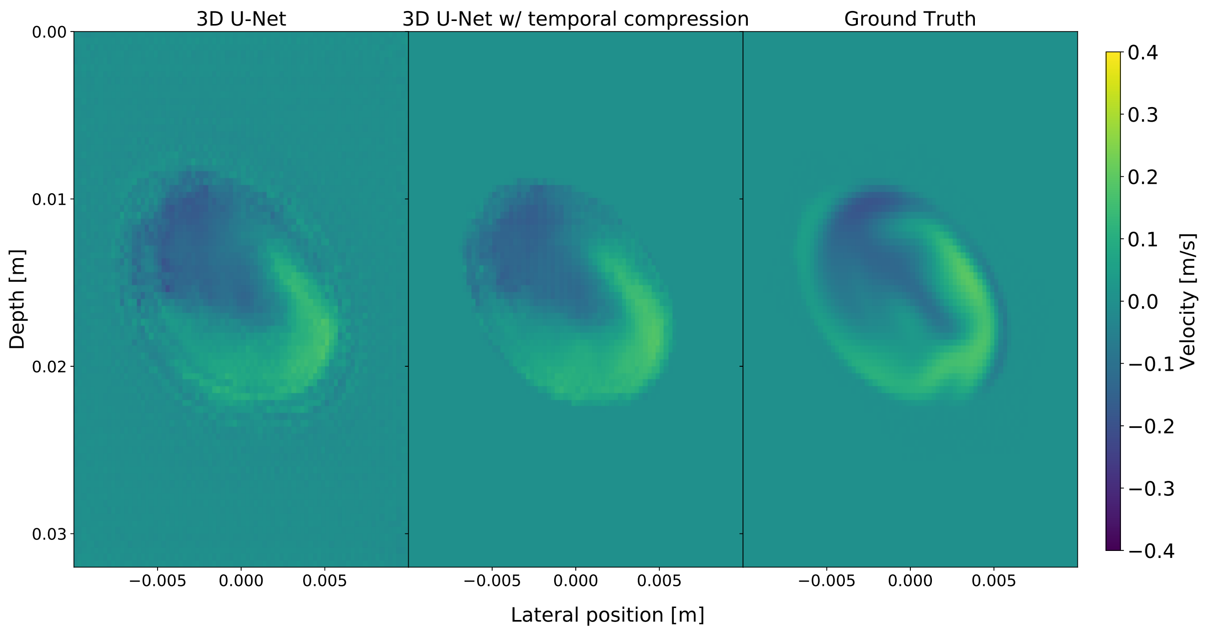
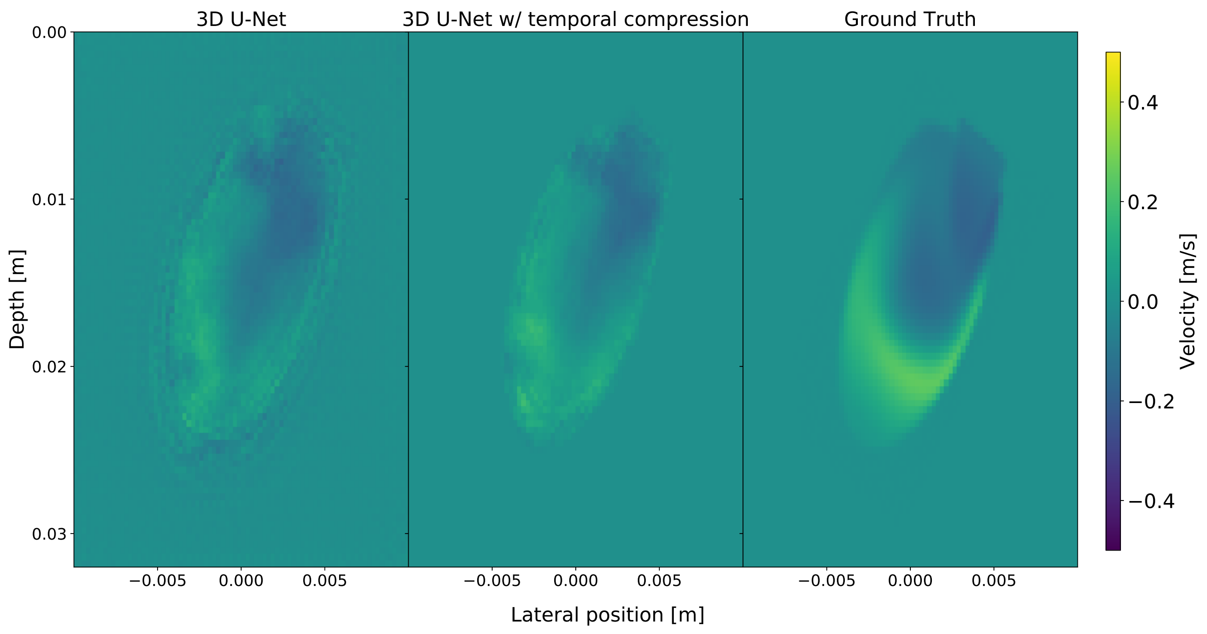


Figure 37: MSE reconstruction error computed over all model predictions, for each field component and time frame. The *3D U-Net* is only applying pooling and transposed convolutions in the spatial domain, while the *3D U-Net, temporal compression* is also applying pooling and stridden convolutions in the time domain. *Gaussian smoothing* denotes the input data smoothed with a 3D Gaussian kernel.



(a) Sample 1, frame 20/25



(b) Sample 2, frame 15/25

Figure 38: Comparison of model predictions between the *3D U-Net* model only applying pooling and transposed convolutions in the spatial domain, and the *3D U-Net, temporal compression* which is also applying pooling and stridden convolutions in the time domain. The images are showing the radial velocity component (V_z) heat maps with no segmentation mask.

4.2 In-Vivo Results

This section contains the results related to the in-vivo velocity fields. The in-vivo data set consists of 58 different measurements of the left ventricle from pediatric patients. Each measurement lasts for the duration of one cardiac cycle. 12 of the measurements are used for testing and 46 are used for training and validation.

4.2.1 B-spline Result Validation

Figure 39 shows a time frame of two of the velocity field measurements from the data set. The left ventricle is segmented manually by experts. Within the segmented area we can observe the blood velocity field, which is reconstructed from the raw data using the B-spline algorithm. The reconstructed measurements were validated by experts to verify that the B-spline interpolation algorithm provides a realistic ”ground truth” for the blood flow in this data set.

4.2.2 Fine-Tuning of Shallow vs. Deep Layers

Figure 40 and 41 show sample frames from in-vivo velocity fields reconstructed using two different iterations of the 2D U-Net, alongside the same frame reconstructed using B-spline interpolation. Figure 42 and 43 show the corresponding results for the 3D U-Net. The models are trained using transfer learning. The model iterations differ in which layers of the networks are, *fine-tuned*, i.e. trained on the in-vivo data. The remaining layers are trained on the in-silico data set. The *shallow layers* are all the layers from the input layer and up to the middle layer, which make up the layers that compress the input data. The *deep layers* are all the layers from the middle layer to the output layer, which make up the layers that reconstruct the compressed data. Figure 44 shows the mean-squared-difference from the B-spline interpolation for both models for the two sample measurements. Figure 45 shows the mean-squared-difference between the 3D U-Net predictions and the B-spline interpolation, averaged over all measurements.

4.2.3 Transfer Learning Effectiveness

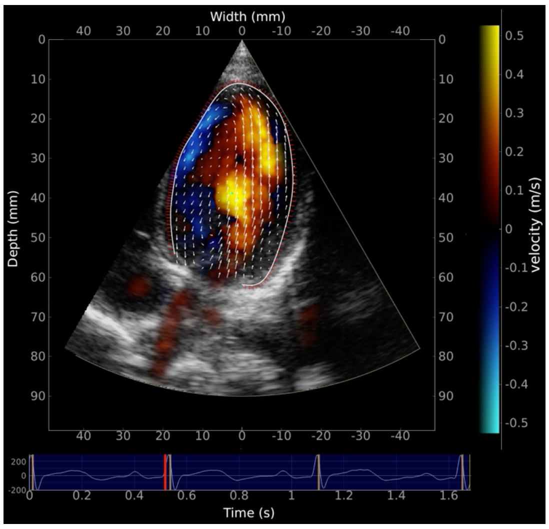
Figures 46-48 show three different instances of the results from applying different variants of transfer learning to the 2D U-Net. Figures 49-51 show similar results for the 3D U-Net. The U-Net only trained on in-silico data is the *In-silico base model*. The U-Net only trained on the in-vivo data is the *Full in-vivo model*. The transfer learning model where the deep layers are fine-tuned to the in-vivo data is the *Fine tune model*. Figures 52 shows the mean-squared-difference from the B-spline interpolation for all the three different training procedures, as well as for the input data smoothed using a 3D Gaussian kernel, for the sample in Figure 47. Note that, since the data needs to be rescaled to fit the 3D U-Net, the frames shown in figures 46-48 are chosen to align to those shown in figures 49-51 for a better comparison of the models. However, the alignment cannot be perfect, hence the illustrated input data for the 2D and 3D U-Nets are not identical. Figure 54 shows the average mean-squared-difference across all measurements and frames from the B-spline interpolation, for all the models, as well as Gaussian smoothing on both the raw data set for the 2D U-Net and the reshaped data set for the 3D U-Net.

4.2.4 2D U-Net vs. 3D U-Net

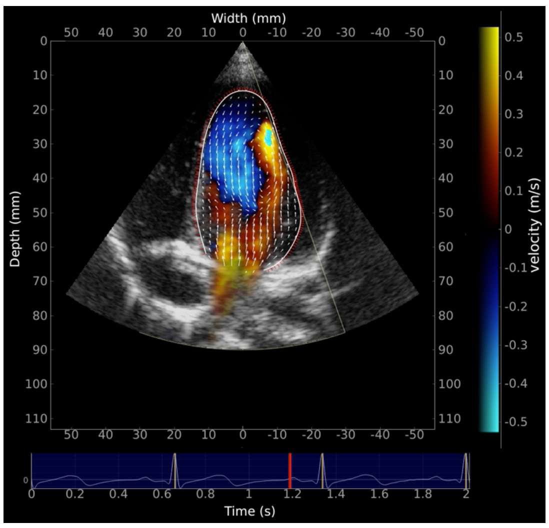
Figures 58 and 59 show comparisons between the results from the 2D U-Net and 3D U-Net models. The figures show reconstruction results from two different measurements, alongside the noisy inputs and B-spline interpolation.

4.2.5 3D U-Net vs. Gaussian Smoothing Benchmark

Figures 61-63 show reconstruction results from the 3D U-Net in comparison to the results from smoothing the noisy velocity field with a 3D Gaussian kernel. The results are shown alongside the noisy velocity field and the result from B-spline interpolation. Figure 64 shows two samples visualized as the (x,y)



(a) Sample 1



(b) Sample 2

Figure 39: Two instances of in-vivo ultrasonic measurements of the left ventricle with segmentation borders and the blood velocity field reconstructed with B-spline interpolation. The reconstructed velocity field is overlaid color doppler flow measurements and the B-Mode image showing the contours of the cardiac walls.

velocity field overlaid the velocity magnitude, illustrating the improvement from using the 3D U-Net as compared to Gaussian smoothing.

4.2.6 Smoothing the Inputs

Figures 65 and 66 show the results from applying Gaussian smoothing to the input data before inputting it to the models for two different degrees of smoothing. The *3D U-Net, smooth test data* is the 3D U-Net which is trained on noisy in-silico and in-vivo training sets, but is tested with the smoothed in-vivo test set. The *3D U-Net, smooth train & test data* is trained on the smoothed in-silico and in-vivo training sets, and tested on the smoothed in-vivo test set. *Gaussian smoothing* denotes the smoothed in-vivo test data, and *B-spline* is the result from B-spline interpolation, used as ”ground truth” to train the models. Figures 67 and 68 show quiver plots of the velocity fields predicted by the 3D U-Net with varying degrees of smoothing of the velocity field input data.

4.2.7 Compression in Three Dimensions

Figures 69-71 show a comparison between the results of a 3D U-Net model which uses pooling and transposed convolution operations in the spatial domain and a 3D U-Net model which uses pooling and transposed convolution operations in the spatial and temporal domains. The model predictions are shown along with the noisy velocity field input and the B-spline interpolation ”ground truth”. Figure 72 shows the mean-squared-difference between the model predictions and the B-spline interpolation results. The mean is computed across the entire test set, and is shown for each component and each frame.

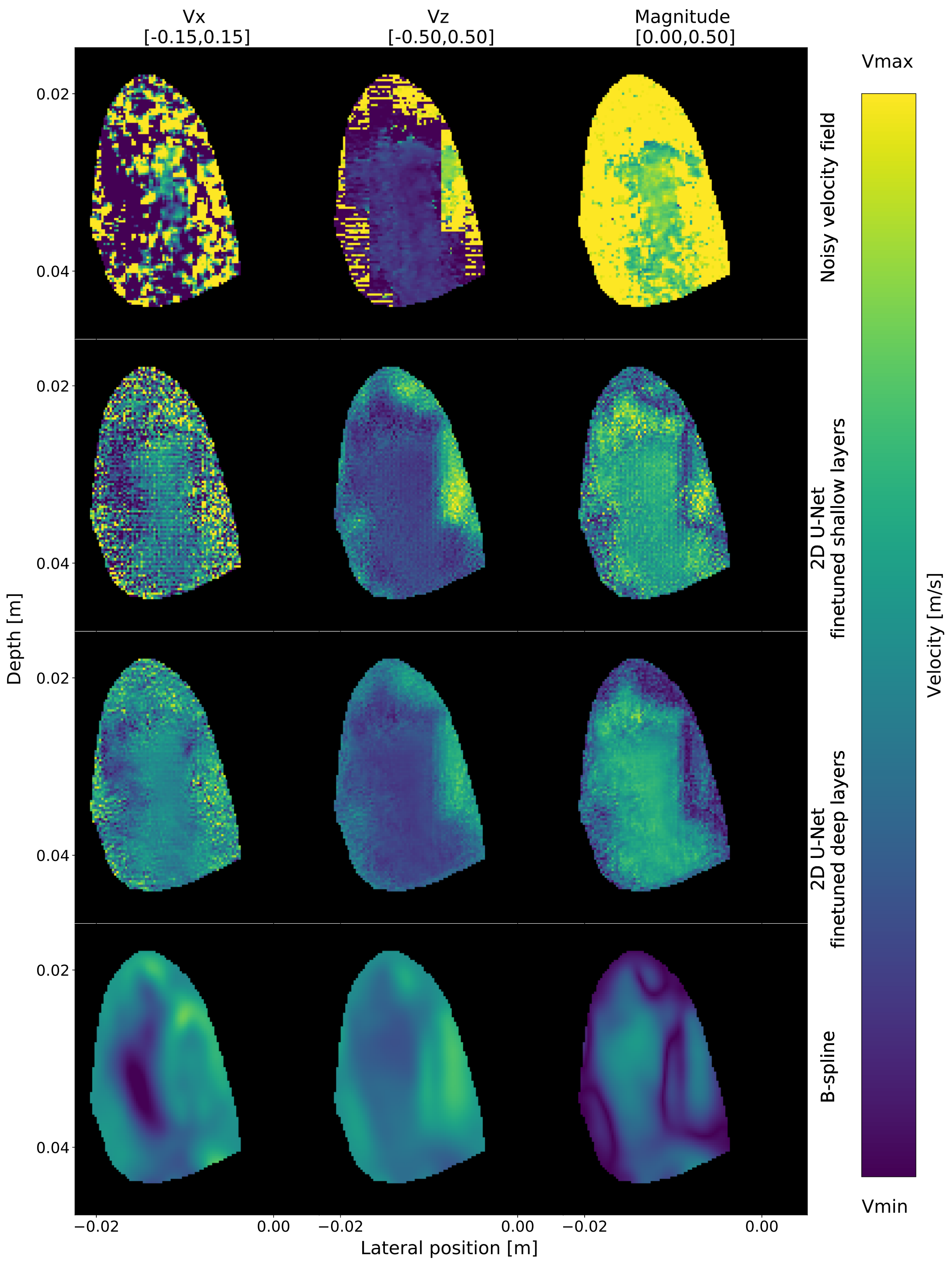


Figure 40: Measurement 9/12, frame 19/31. Results from fine tuning the shallow layers versus fine tuning the deep layers of the 2D U-Net model with in-vivo data, alongside the noisy velocity field input and the result from the B-spline interpolation. The results show the lateral (Vx) and radial (V_z) components and the velocity magnitude. Each component has a specific dynamic range (V_{min},V_{max}) shown at the top of each column.

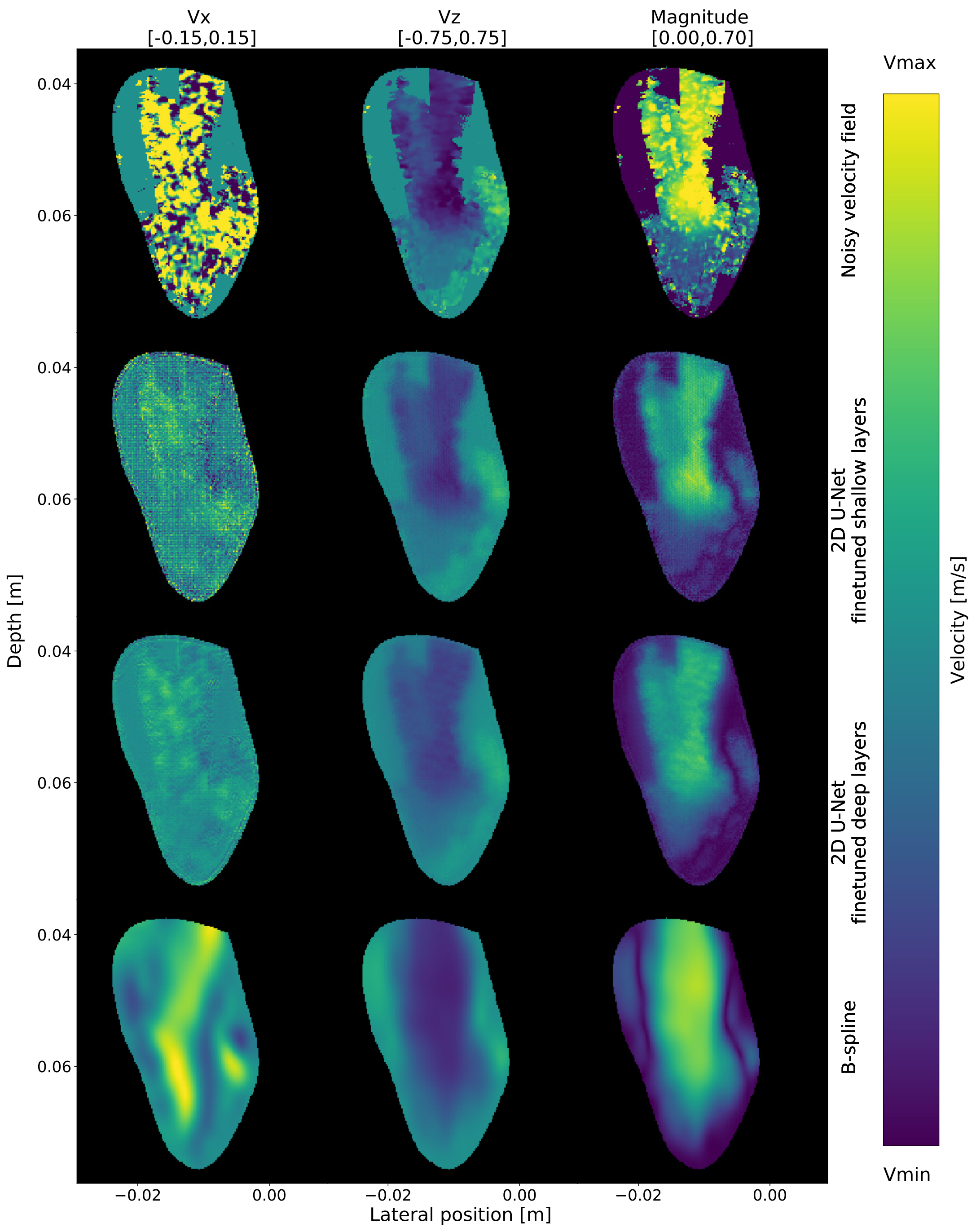


Figure 41: Measurement 12/12, frame 20/42. Results from fine tuning the shallow layers versus fine tuning the deep layers of the 2D U-Net model with in-vivo data, alongside the noisy velocity field input and the result from the B-spline interpolation. The results show the lateral (Vx) and radial (V_z) components and the velocity magnitude. Each component has a specific dynamic range (V_{min},V_{max}) shown at the top of each column.

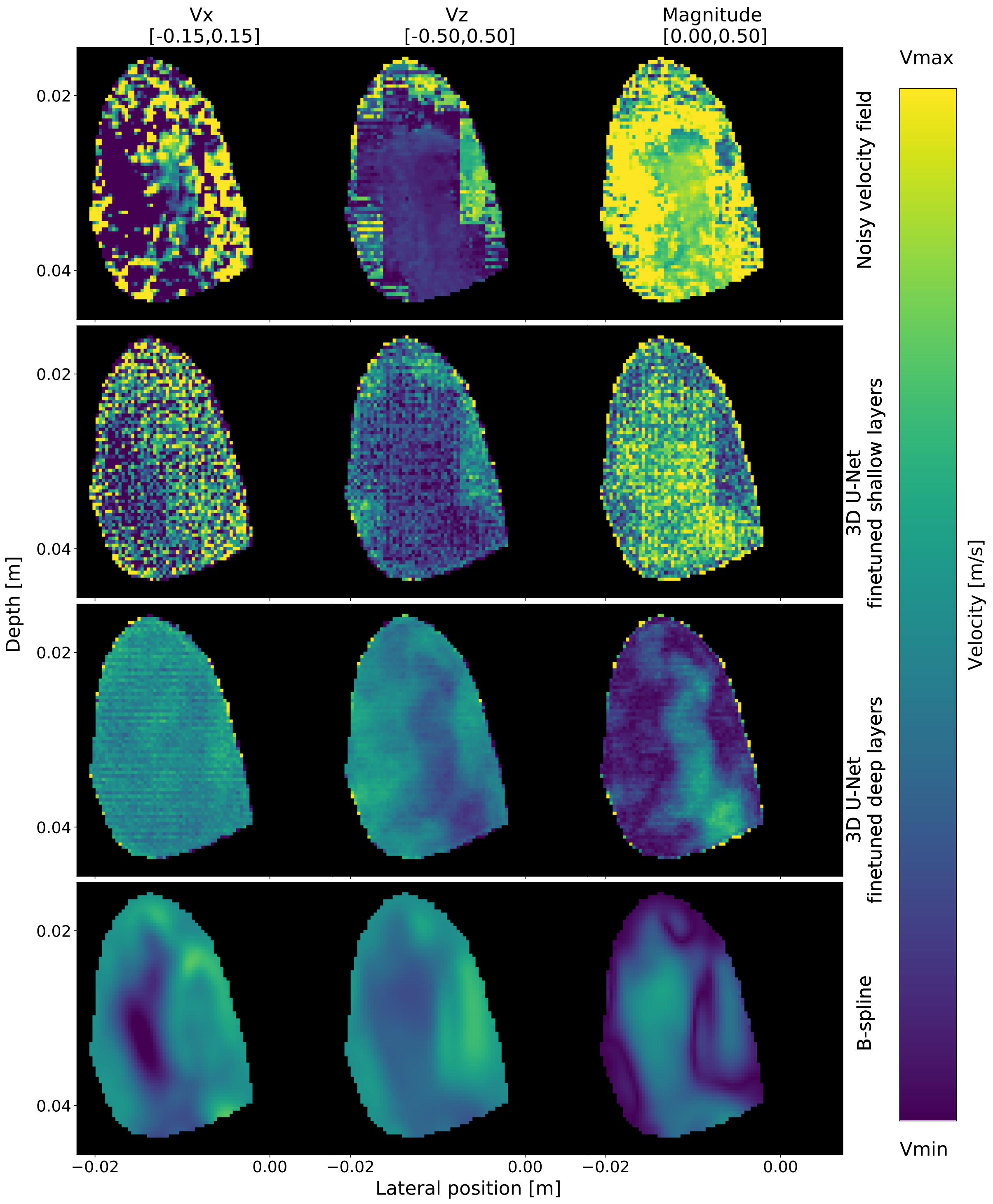


Figure 42: Measurement 9/12, frame 15/25. Results from fine tuning the shallow layers versus fine tuning the deep layers of the 3D U-Net model with in-vivo data, alongside the noisy velocity field input and the result from the B-spline interpolation. The results show the lateral (Vx) and radial (V_z) components and the velocity magnitude. Each component has a specific dynamic range (V_{min},V_{max}) shown at the top of each column.

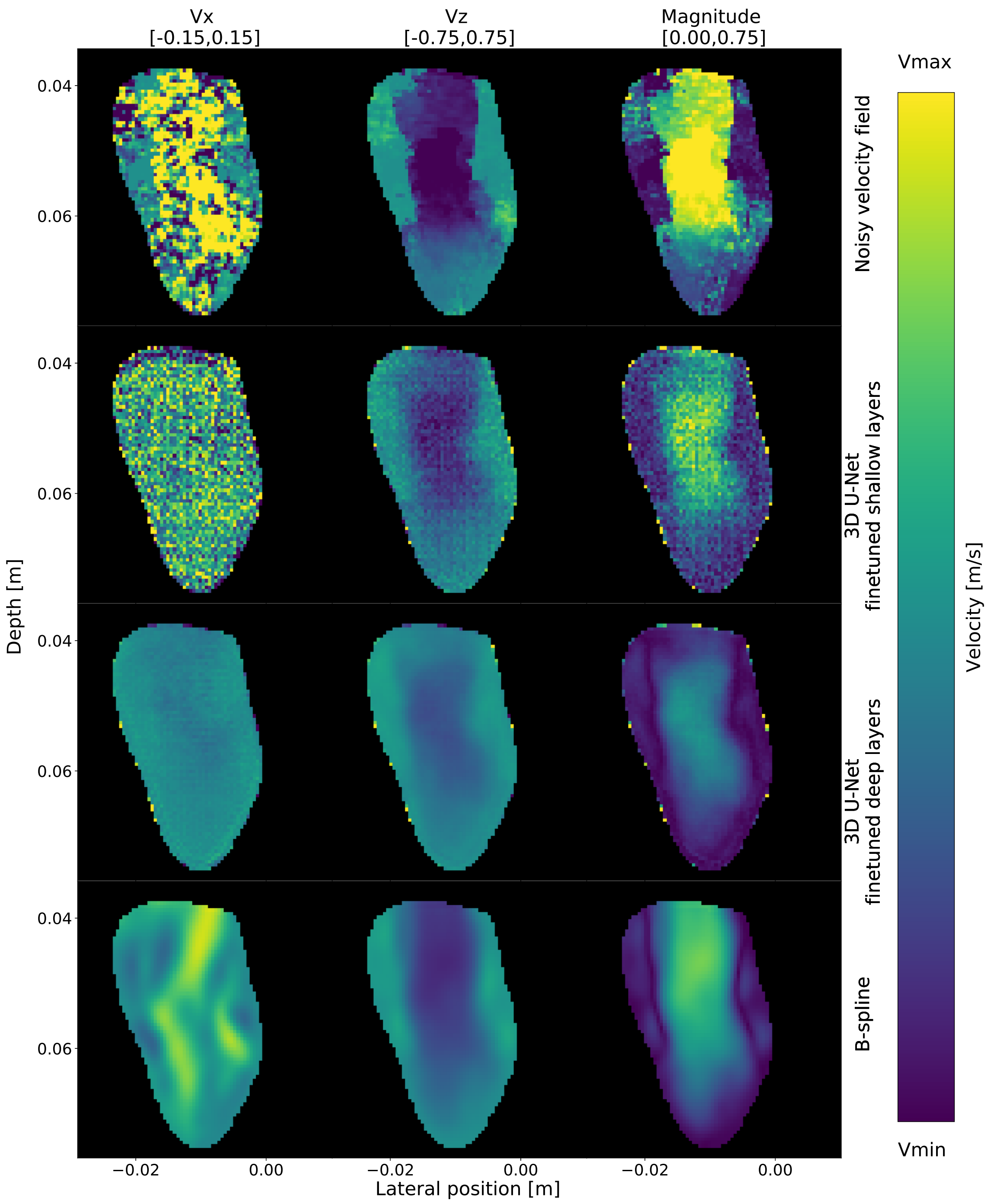
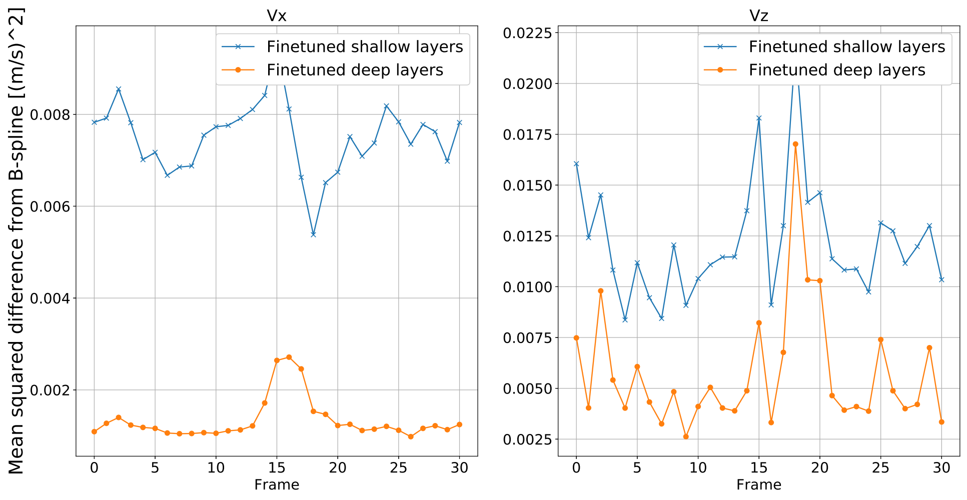
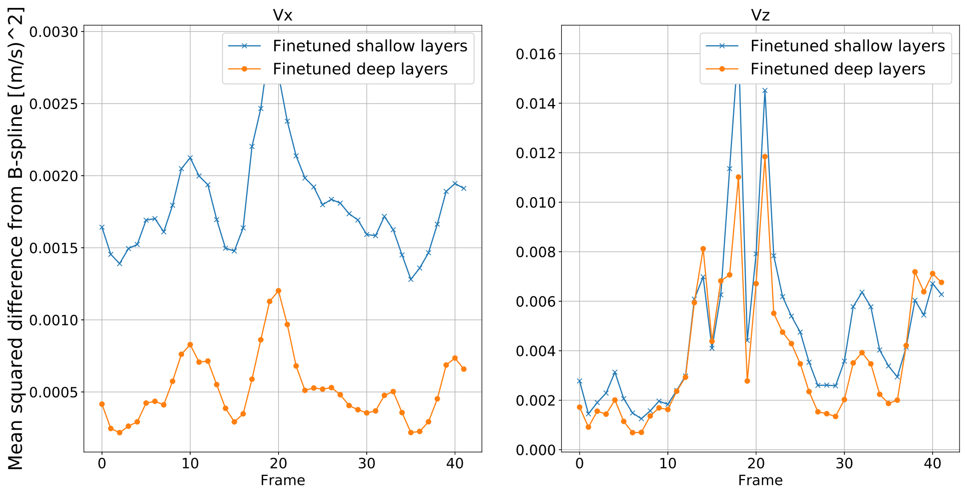


Figure 43: Measurement 12/12, frame 11/25. Results from fine tuning the shallow layers versus fine tuning the deep layers of the 3D U-Net model with in-vivo data, alongside the noisy velocity field input and the result from the B-spline interpolation. The results show the lateral (Vx) and radial (V_z) components and the velocity magnitude. Each component has a specific dynamic range (V_{min},V_{max}) shown at the top of each column.



(a) Measurement 9/12



(b) Measurement 12/12

Figure 44: Mean-squared-difference between the model predictions and the B-spline results. The graphs show the mean-squared-difference of the whole image for each frame, and for the lateral (V_x) and radial (V_z) component. *Fine-tuned shallow layers* and *Fine-tuned deep layers* denote the results of the 2D U-Net where respectively the shallow and the deep half of the network is trained to reconstruct the B-spline results from the noisy in-vivo input.

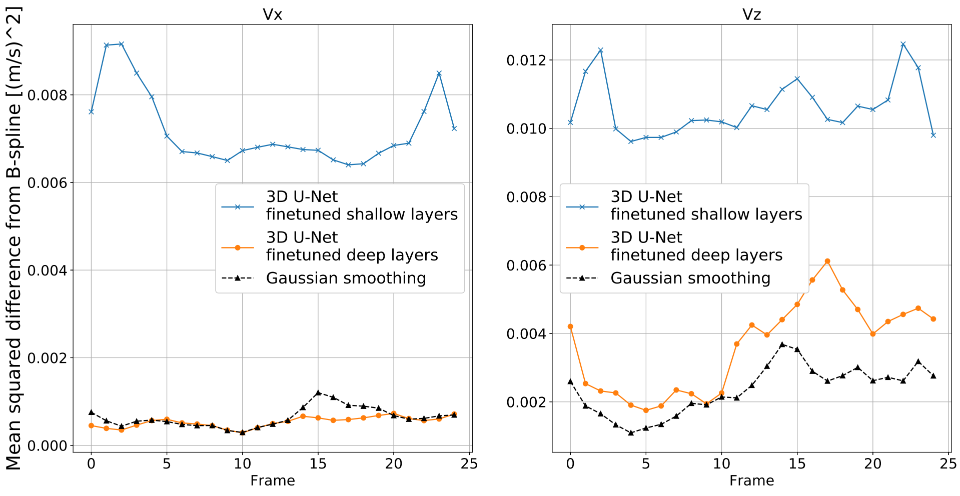


Figure 45: Mean-squared-difference between the 3D U-Net model predictions and the B-spline results. The graphs show the mean-squared-difference of the whole image for each frame, and for the lateral (V_x) and radial (V_z) component, averaged over all measurements. *Fine-tuned shallow layers* and *Fine-tuned deep layers* denote the results of the 3D U-Net where respectively the shallow and the deep half of the network is trained to reconstruct the B-spline results from the noisy in-vivo input.

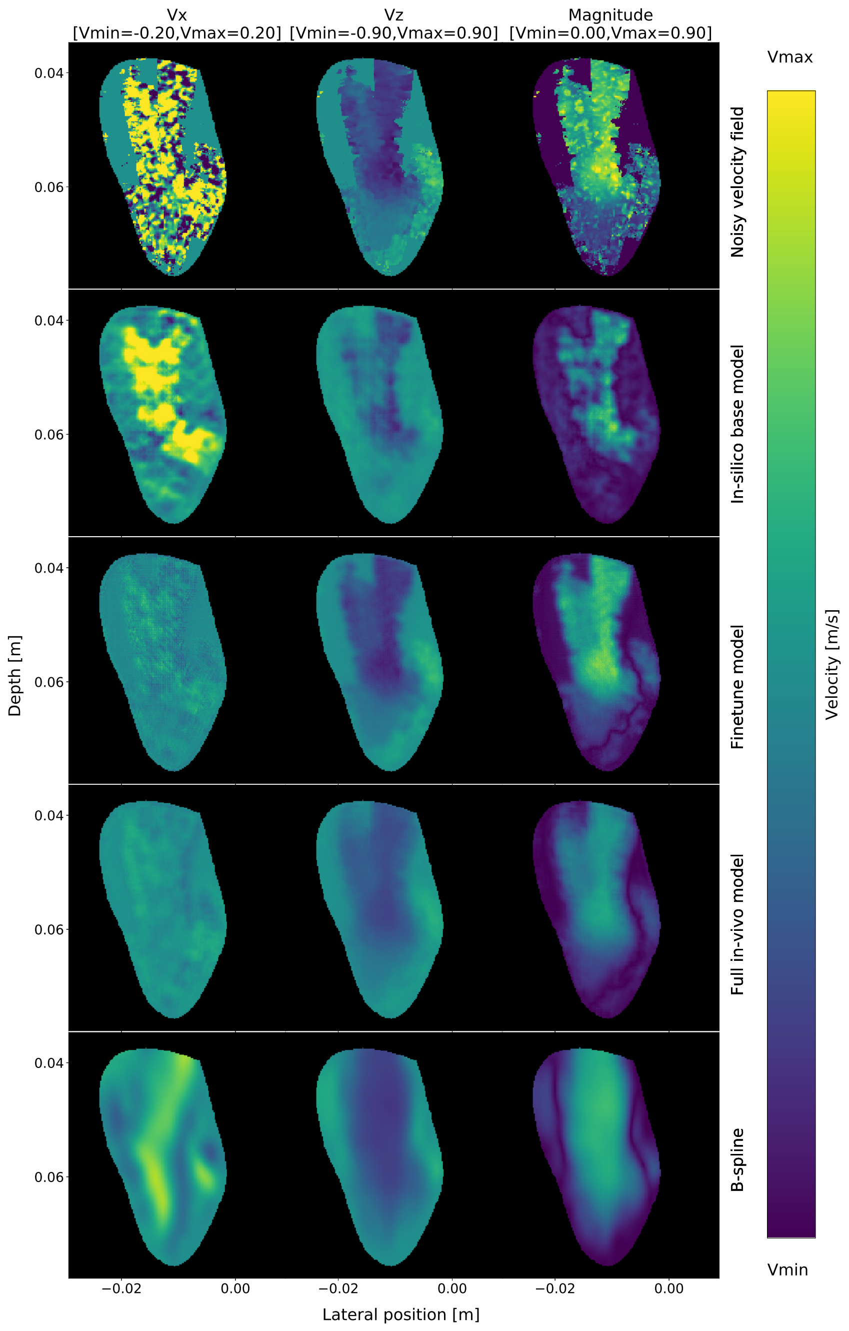


Figure 46: 2D U-Net predictions from measurement 12/12, frame 20/41. The figure shows the lateral (V_x) and radial (V_z) components of blood velocity fields, as well as the velocity magnitude. Each component has a specific dynamic range (V_{min},V_{max}). The *noisy velocity field* is given as input to a 2D U-Net trained three different ways: (1) *In-silico base model* is trained on a large data set of simulated tracking data. (2) *Full in-vivo model* is trained on a small data set of in-vivo measurements. (3) *Fine tune model*, where the shallow half of the network is trained on the simulated data and the deeper half is trained on the in-vivo data. The *B-spline* is the result from B-spline interpolation of the noisy velocity field, which is used as the "ground truth" for the in-vivo data.

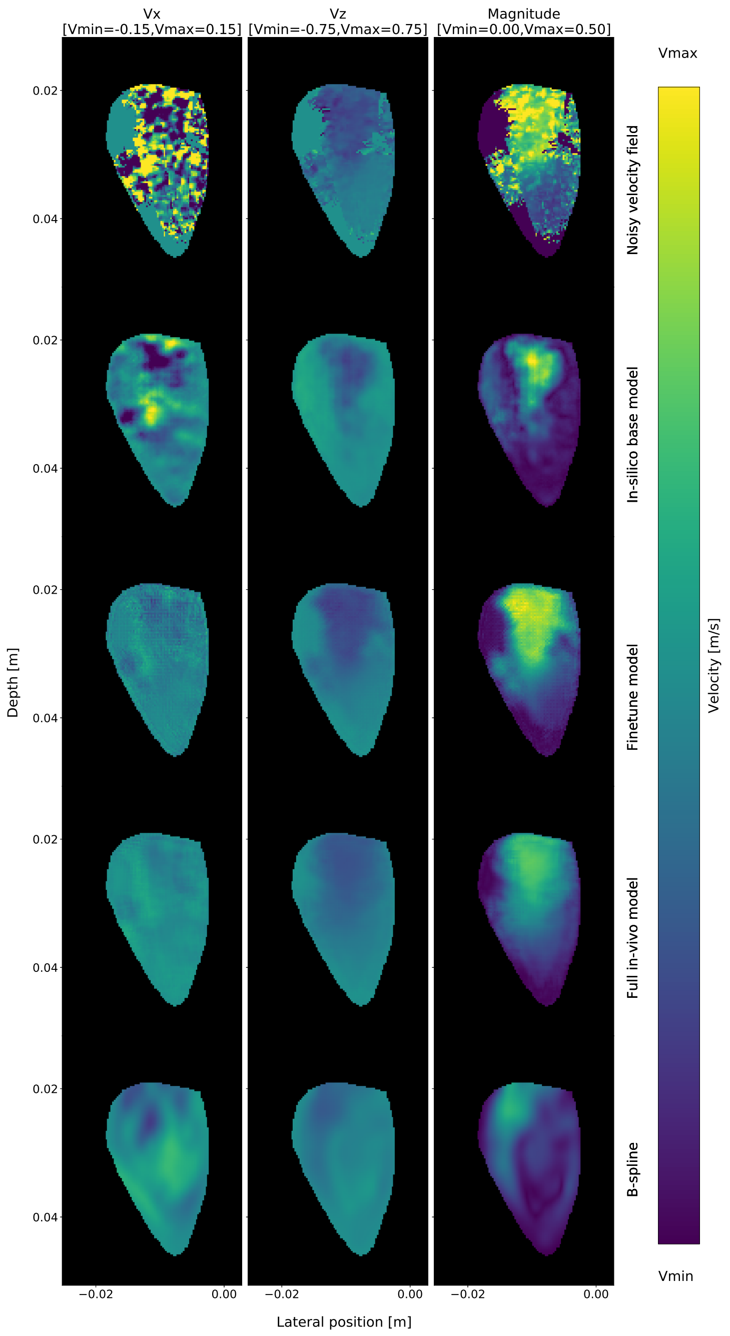


Figure 47: 2D U-Net predictions from measurement 10/12, frame 20/45. The figure shows the lateral (V_x) and radial (V_z) components of blood velocity fields, as well as the velocity magnitude. Each component has a specific dynamic range (V_{min},V_{max}). The *noisy velocity field* is given as input to a 2D U-Net trained three different ways: (1) *In-silico base model* is trained on a large data set of simulated tracking data. (2) *Full in-vivo model* is trained on a small data set of in-vivo measurements. (3) *Fine tune model*, where the shallow half of the network is trained on the simulated data and the deeper half is trained on the in-vivo data. The *B-spline* is the result from B-spline interpolation of the noisy velocity field, which is used as the "ground truth" for the in-vivo data.

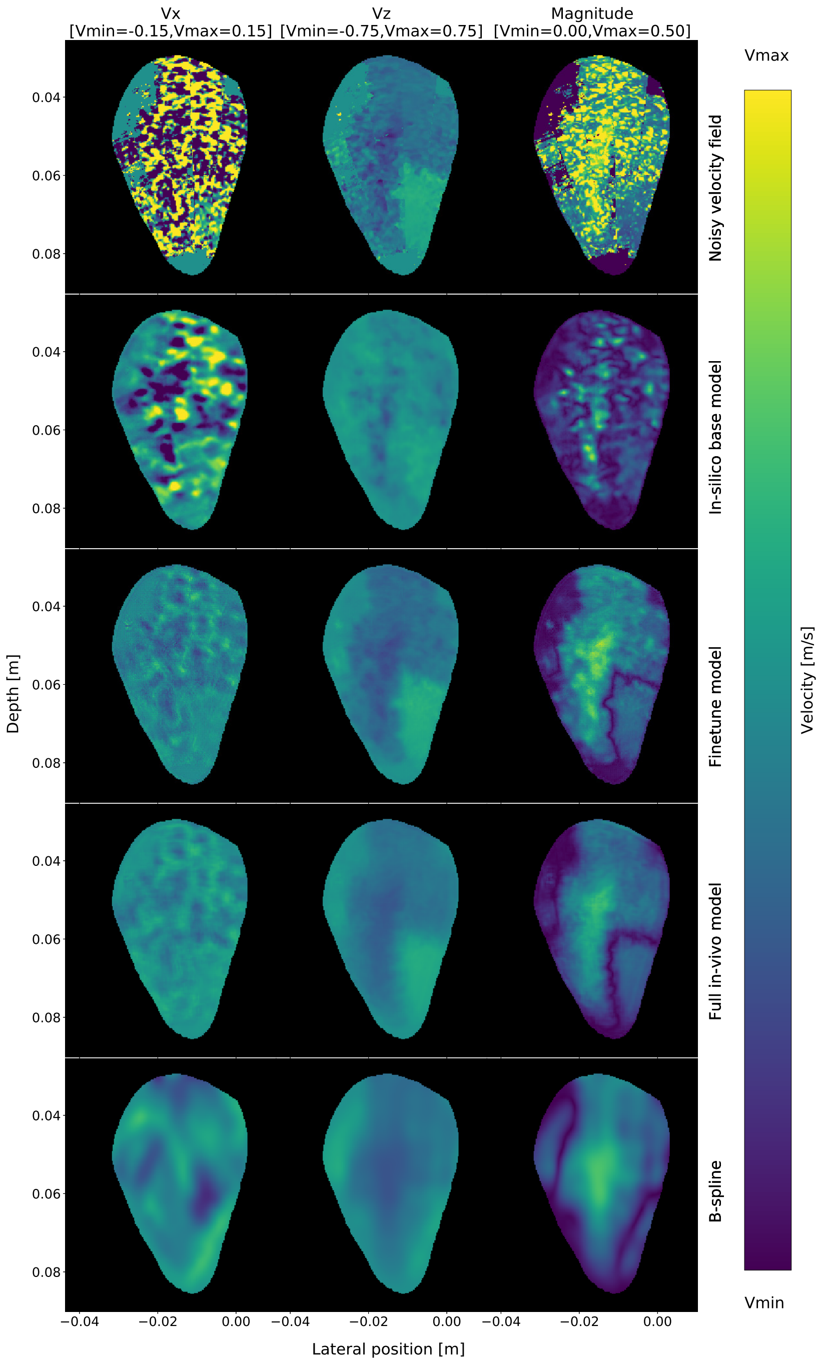


Figure 48: 2D U-Net predictions from measurement 1/12, frame 20/27. The figure shows the lateral (V_x) and radial (V_z) components of blood velocity fields, as well as the velocity magnitude. Each component has a specific dynamic range (V_{min},V_{max}). The *noisy velocity field* is given as input to a 2D U-Net trained three different ways: (1) *In-silico base model* is trained on a large data set of simulated tracking data. (2) *Full in-vivo model* is trained on a small data set of in-vivo measurements. (3) *Fine tune model*, where the shallow half of the network is trained on the simulated data and the deeper half is trained on the in-vivo data. The *B-spline* is the result from B-spline interpolation of the noisy velocity field, which is used as the ”ground truth” for the in-vivo data.

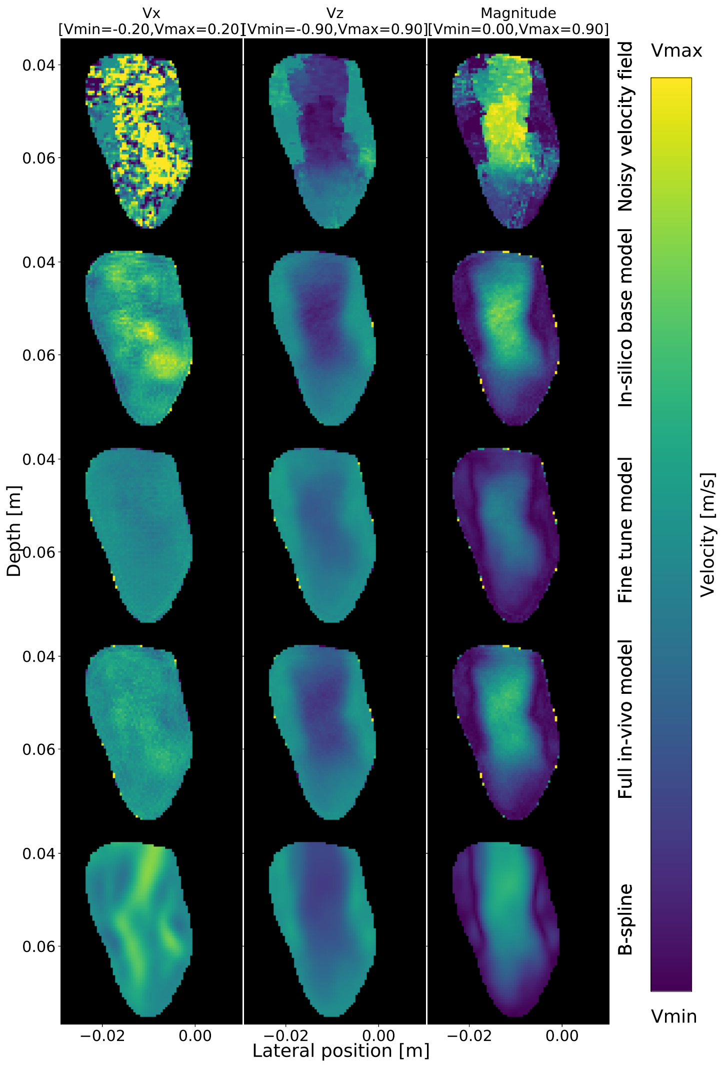


Figure 49: 3D U-Net predictions from measurement 12/12, frame 11/25. The figure shows the lateral (V_x) and radial (V_z) components of blood velocity fields, as well as the velocity magnitude. Each component has a specific dynamic range (V_{min},V_{max}). The *noisy velocity field* is given as input to a 3D U-Net trained three different ways: (1) *In-silico base model* is trained on a large data set of simulated tracking data. (2) *Full in-vivo model* is trained on a small data set of in-vivo measurements. (3) *Fine tune model*, where the shallow half of the network is trained on the simulated data and the deeper half is trained on the in-vivo data. The *B-spline* is the result from B-spline interpolation of the noisy velocity field, which is used as the "ground truth" for the in-vivo data.

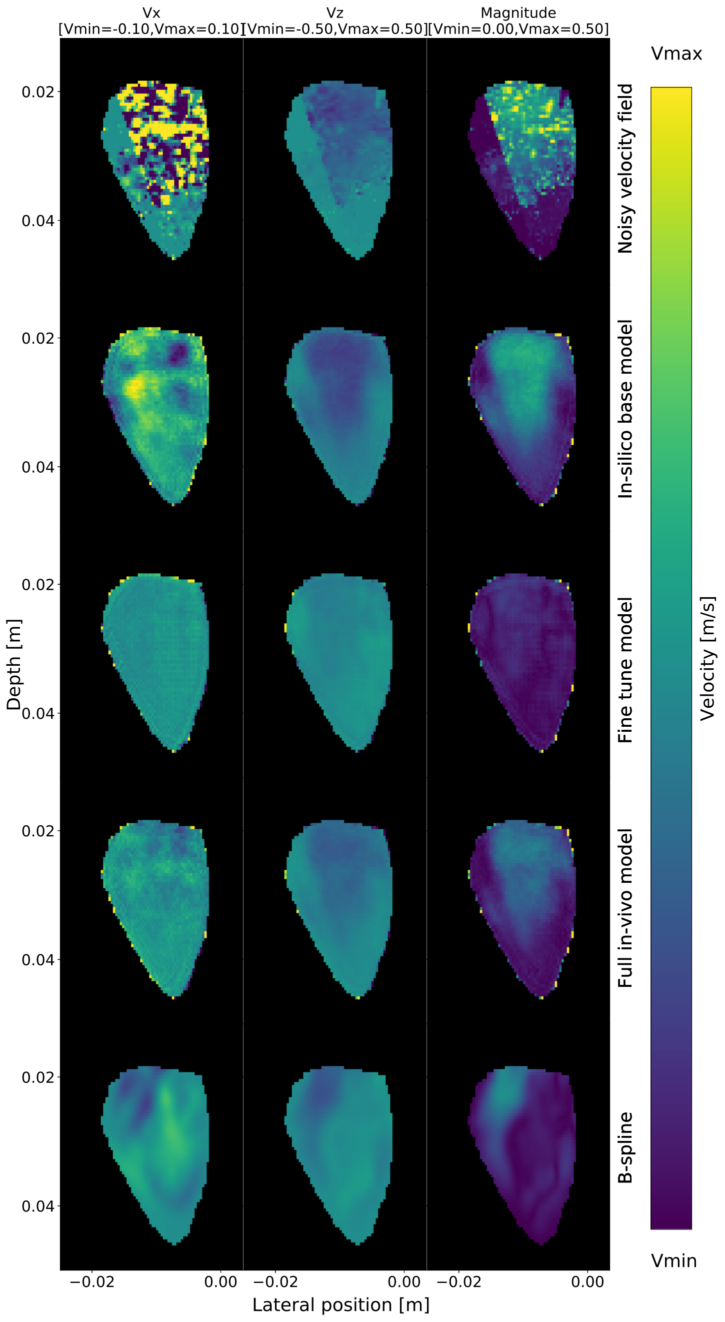


Figure 50: 3D U-Net predictions from measurement 10/12, frame 10/25. The figure shows the lateral (V_x) and radial (V_z) components of blood velocity fields, as well as the velocity magnitude. Each component has a specific dynamic range (V_{min},V_{max}). The *noisy velocity field* is given as input to a 3D U-Net trained three different ways: (1) *In-silico base model* is trained on a large data set of simulated tracking data. (2) *Full in-vivo model* is trained on a small data set of in-vivo measurements. (3) *Fine tune model*, where the shallow half of the network is trained on the simulated data and the deeper half is trained on the in-vivo data. The *B-spline* is the result from B-spline interpolation of the noisy velocity field, which is used as the "ground truth" for the in-vivo data.

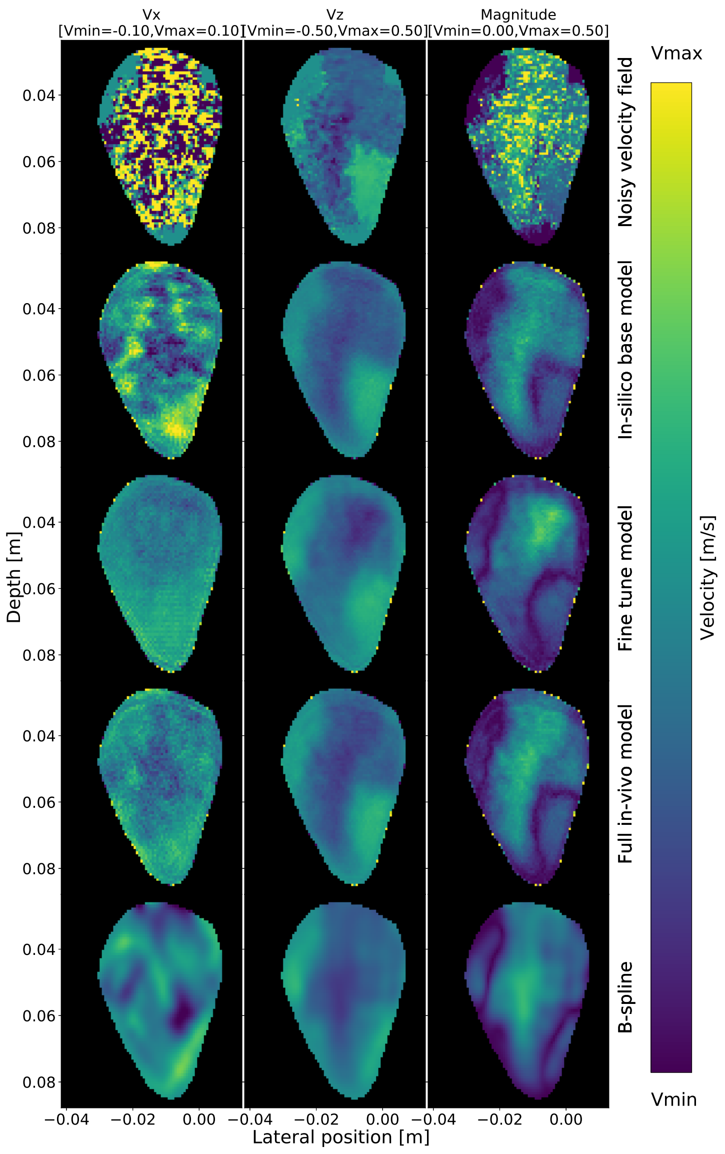
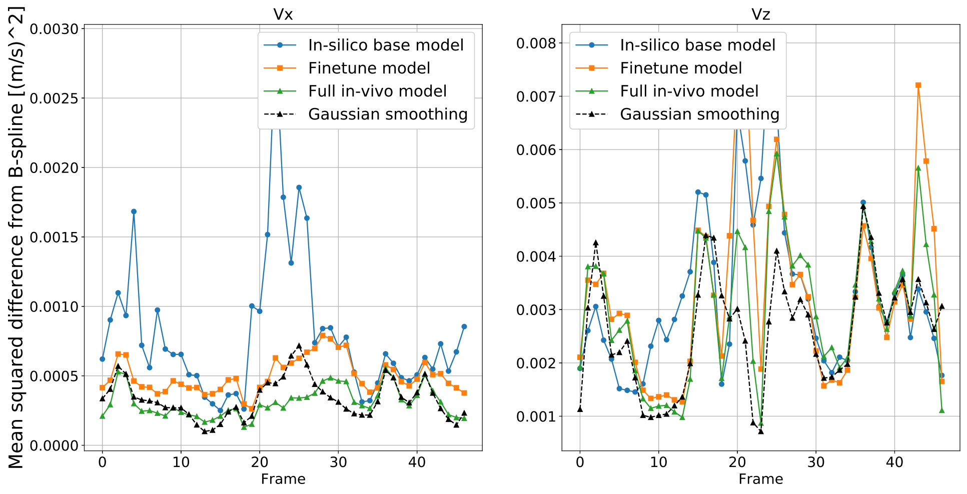
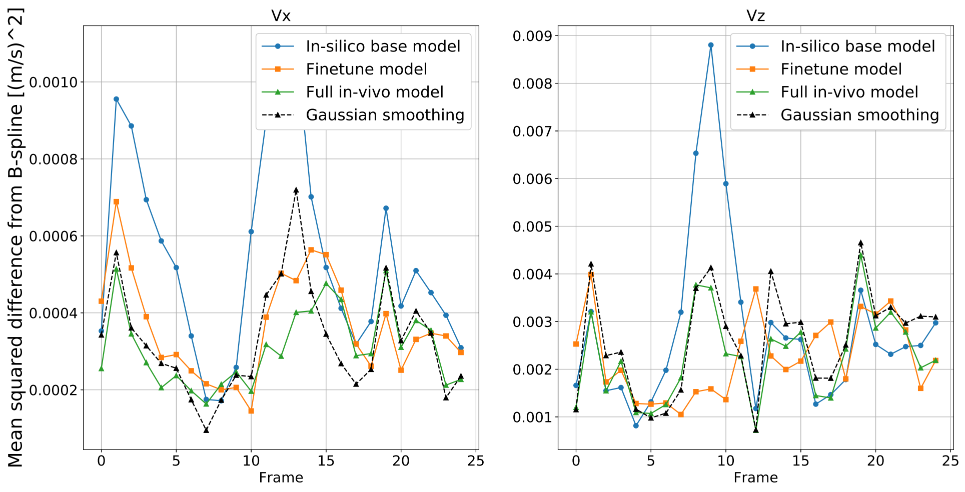


Figure 51: 3D U-Net predictions from measurement 1/12, frame 18/25. The figure shows the lateral (V_x) and radial (V_z) components of blood velocity fields, as well as the velocity magnitude. Each component has a specific dynamic range (V_{min},V_{max}). The *noisy velocity field* is given as input to a 3D U-Net trained three different ways: (1) *In-silico base model* is trained on a large data set of simulated tracking data. (2) *Full in-vivo model* is trained on a small data set of in-vivo measurements. (3) *Fine tune model*, where the shallow half of the network is trained on the simulated data and the deeper half is trained on the in-vivo data. The *B-spline* is the result from B-spline interpolation of the noisy velocity field, which is used as the "ground truth" for the in-vivo data.

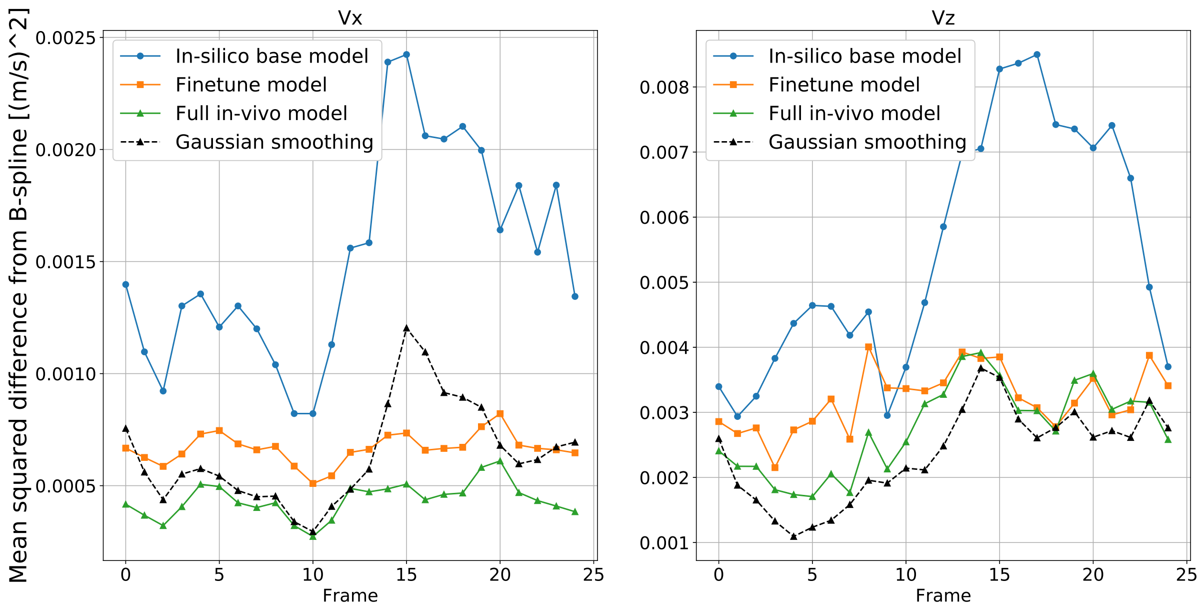


(a) 2D U-Net

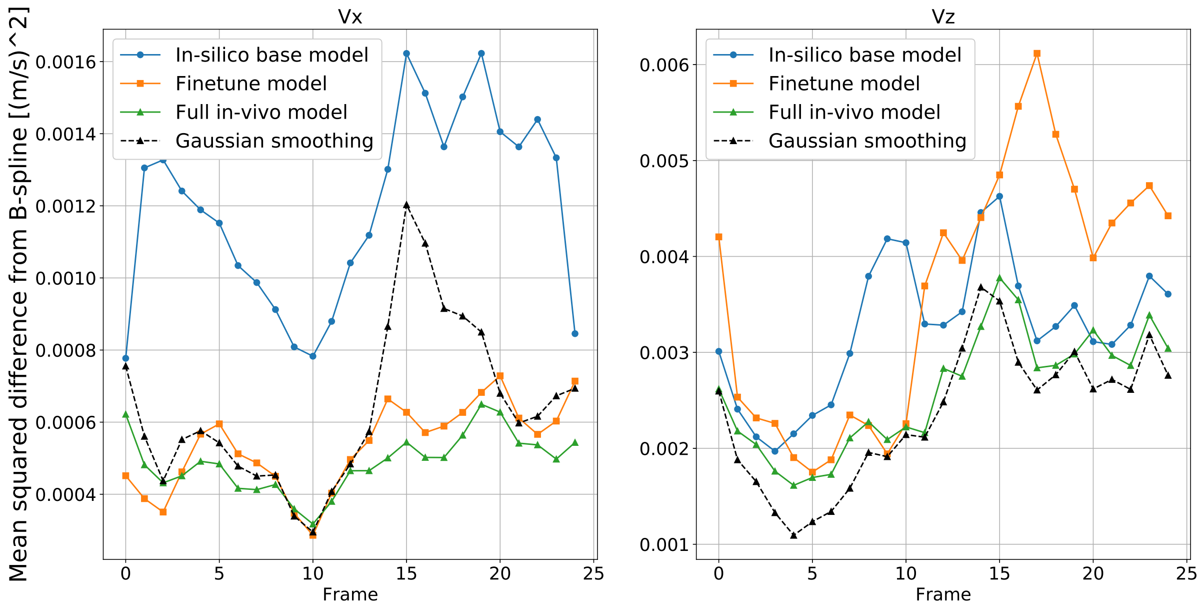


(b) 3D U-Net

Figure 52: Mean-squared-difference between the model predictions and the B-spline results for measurement 10/12. The graphs show the mean-squared-difference of the whole image for each frame, and for the lateral (V_x) and radial (V_z) component. *Gaussian smoothing* denotes the mean-squared-difference between the B-spline results and the results of Gaussian smoothing applied to the noisy velocity field, which can be used as a benchmark for the proposed models.



(a) 2D U-Net



(b) 3D U-Net

Figure 53: Mean-squared-difference between the predictions and the B-spline results over all measurements. The graphs show the mean-squared-difference computed over all images of all the measurements in the test set for each frame, and for the lateral (V_x) and radial (V_z) component. *Gaussian smoothing* denotes the mean-squared-difference between the B-spline results and the results of Gaussian smoothing applied to the noisy velocity field. Since the measurements used to train the 2D U-Net have different frame rates the mean-squared-difference computed between each measurement and the B-spline interpolation is resampled to 25 frames before the average metric over all the measurements are computed.

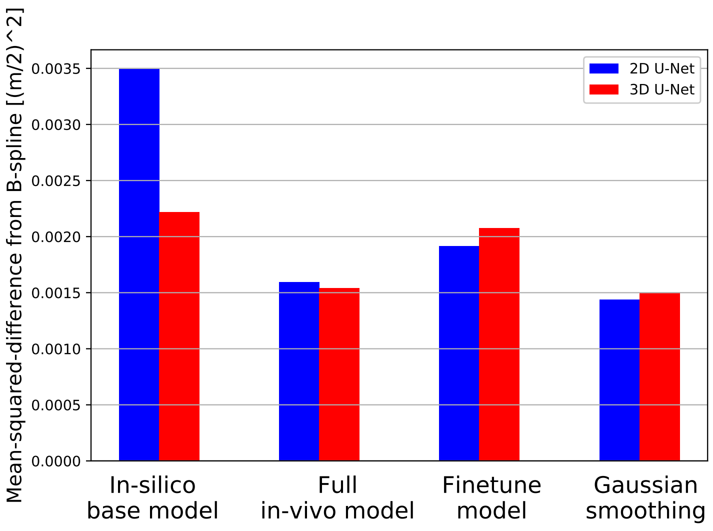


Figure 54: Mean-squared-difference between the model predictions and the B-spline interpolation for all frames and measurements. *In-silico base model* is trained solely on in-silico data, the *Full in-vivo model* is trained solely on in-vivo data and the *Fine tune model* has the shallow half of the network trained on in-silico data and the deep half trained on in-vivo data. *Gaussian smoothing* denotes smoothing by a 3D Gaussian kernel.

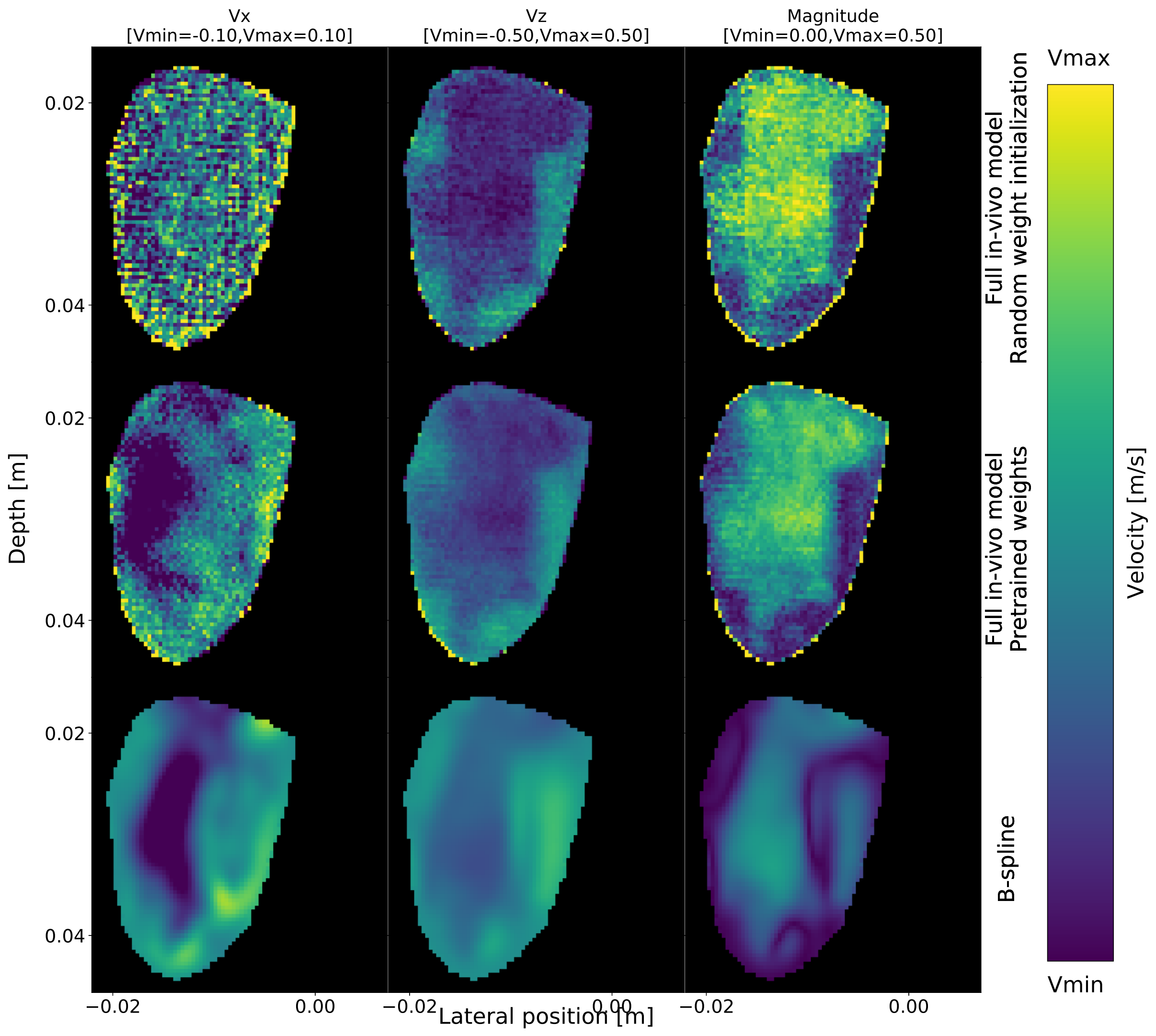


Figure 55: 3D U-Net predictions from measurement 9/12, frame 8/25. The figure shows the lateral (V_x) and radial (V_z) components of blood velocity fields, as well as the velocity magnitude. Each component has a specific dynamic range (V_{min},V_{max}). The *Full in-vivo model* has all the weights trained on the in-vivo data. The figure shows the difference between using a model which is pre-trained on the in-silico data, as opposed to a model which has random weight initialization. The *B-spline* is the result from B-spline interpolation of the noisy velocity field, which is used as the "ground truth" for the in-vivo data.

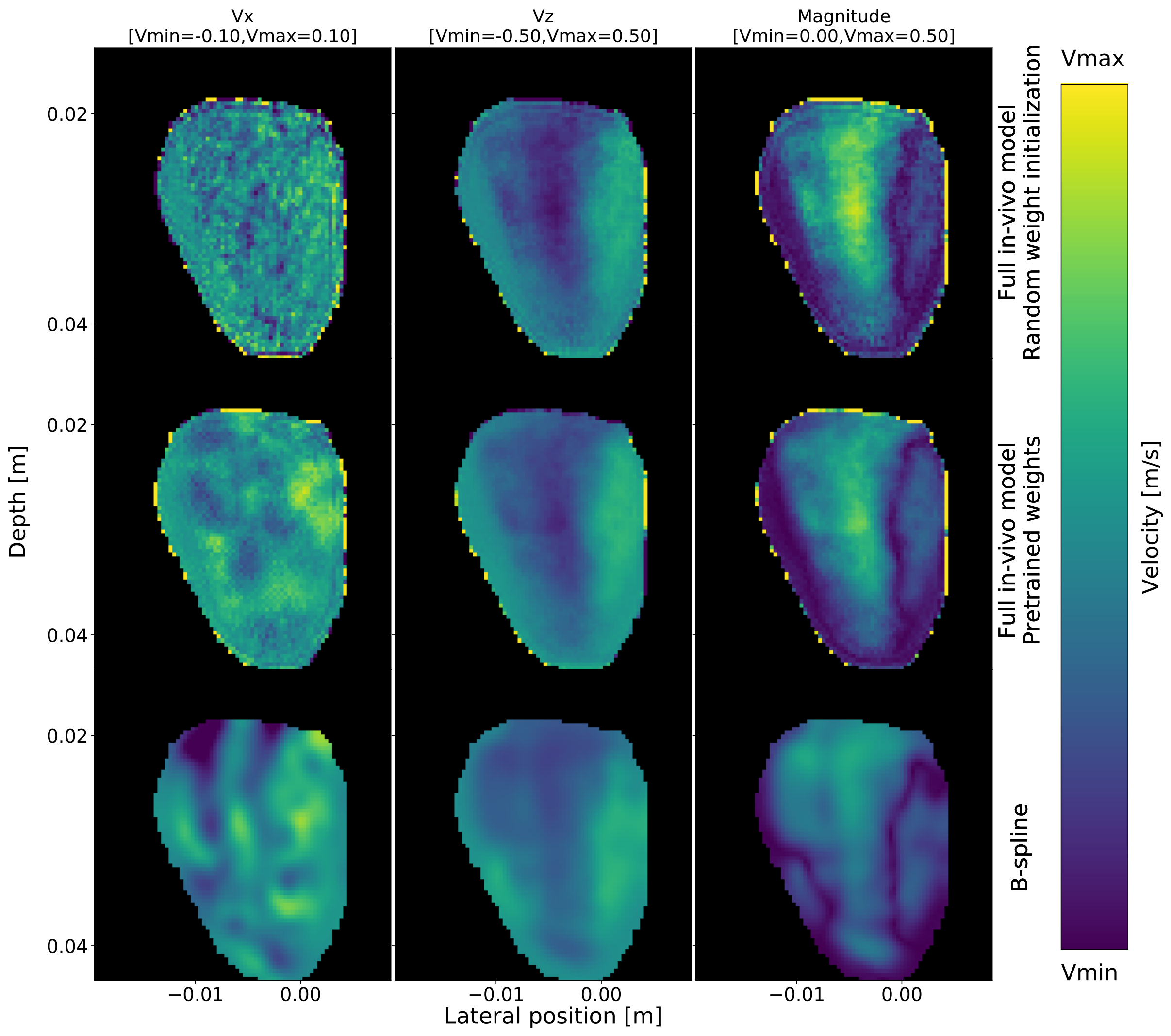


Figure 56: 3D U-Net predictions from measurement 3/12, frame 20/25. The figure shows the lateral (V_x) and radial (V_z) components of blood velocity fields, as well as the velocity magnitude. Each component has a specific dynamic range (V_{min},V_{max}). The *Full in-vivo model* has all the weights trained on the in-vivo data. The figure shows the difference between using a model which is pre-trained on the in-silico data, as opposed to a model which has random weight initialization. The *B-spline* is the result from B-spline interpolation of the noisy velocity field, which is used as the "ground truth" for the in-vivo data.

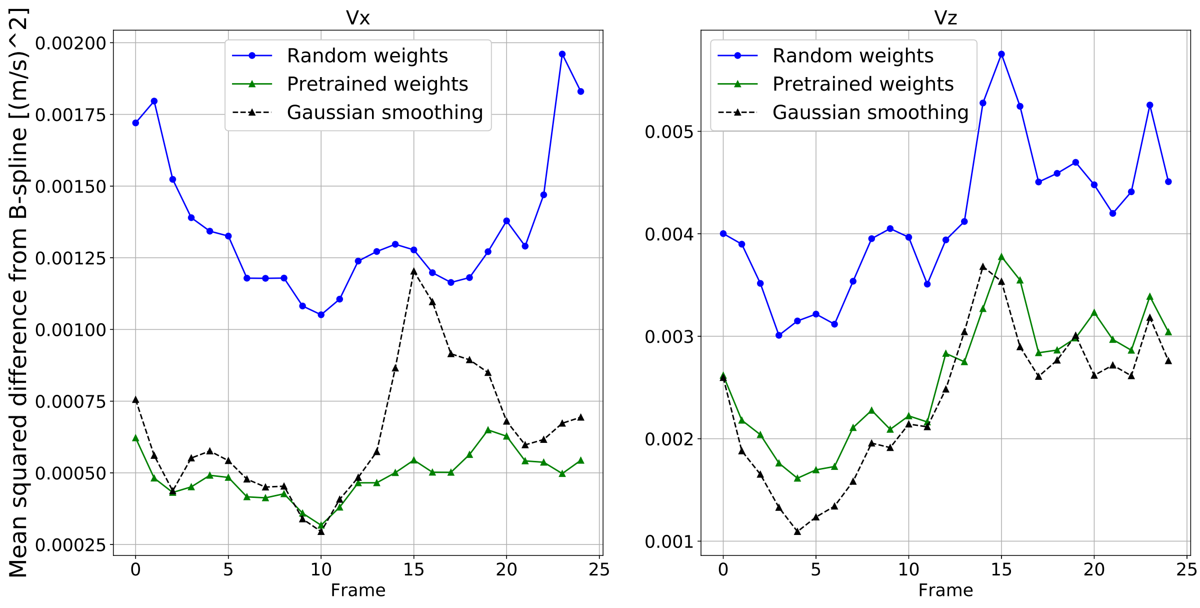


Figure 57: Mean-squared-difference between the 3D U-Net model predictions and the B-spline results over all measurements. The figure shows the difference between using a model which is pre-trained on the in-silico data, as opposed to a model which has random weight initialization. The graphs show the mean-squared-difference computed over all images of all the measurements in the test set for each frame, and for the lateral (V_x) and radial (V_z) component. *Gaussian smoothing* denotes the mean-squared-difference between the B-spline results and the results of Gaussian smoothing applied to the noisy velocity field.

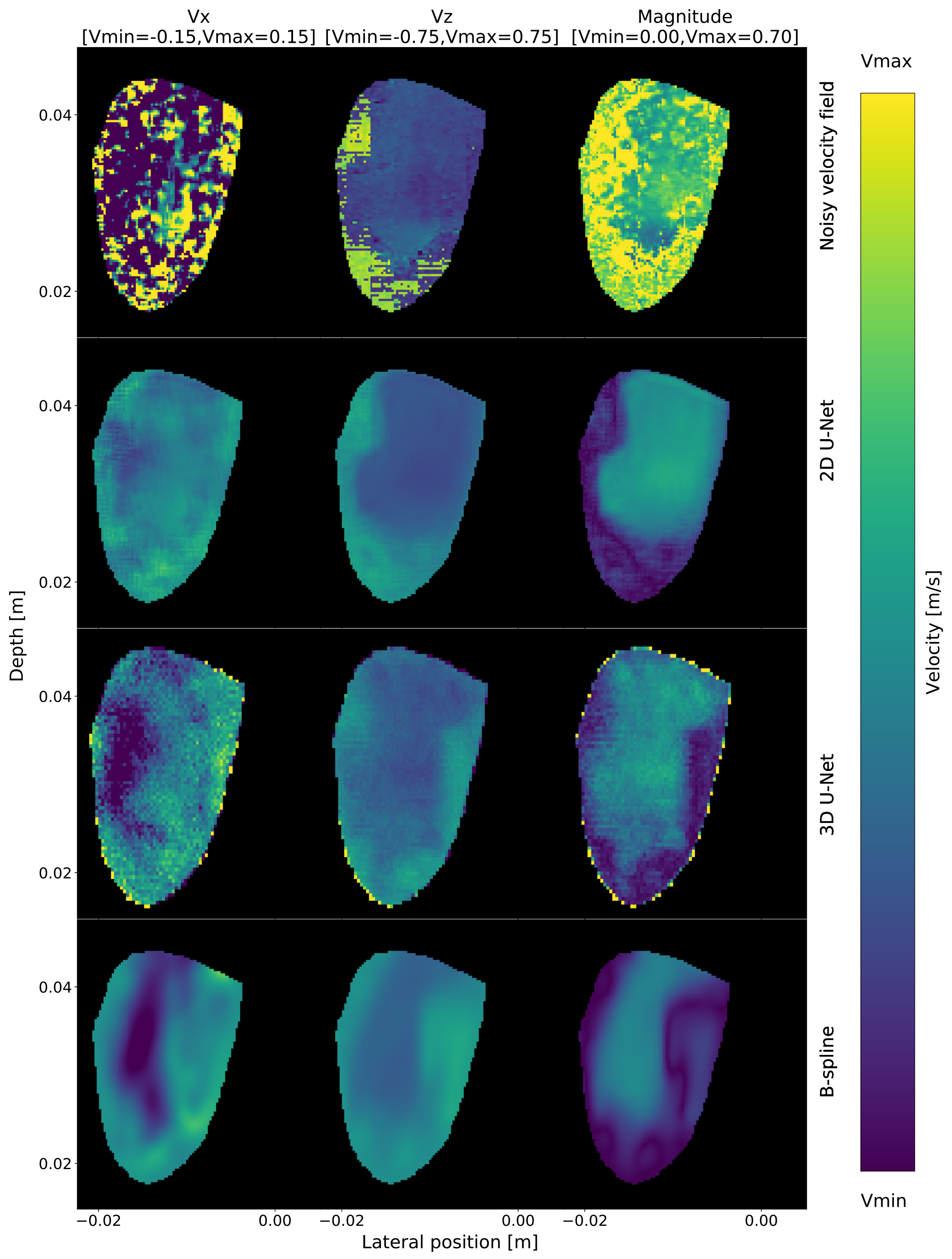


Figure 58: Predictions from measurement 9/12, frame 15/25. The figure shows the lateral (V_x) and radial (V_z) components of blood velocity fields, as well as the velocity magnitude. Each component has a specific dynamic range (V_{min},V_{max}). The *noisy velocity field* is given as an input to the *2D U-Net* and *3D U-Net* models, producing the results shown. The *B-spline* is the result from B-spline interpolation of the noisy velocity field, which is used as the "ground truth" for the in-vivo data.

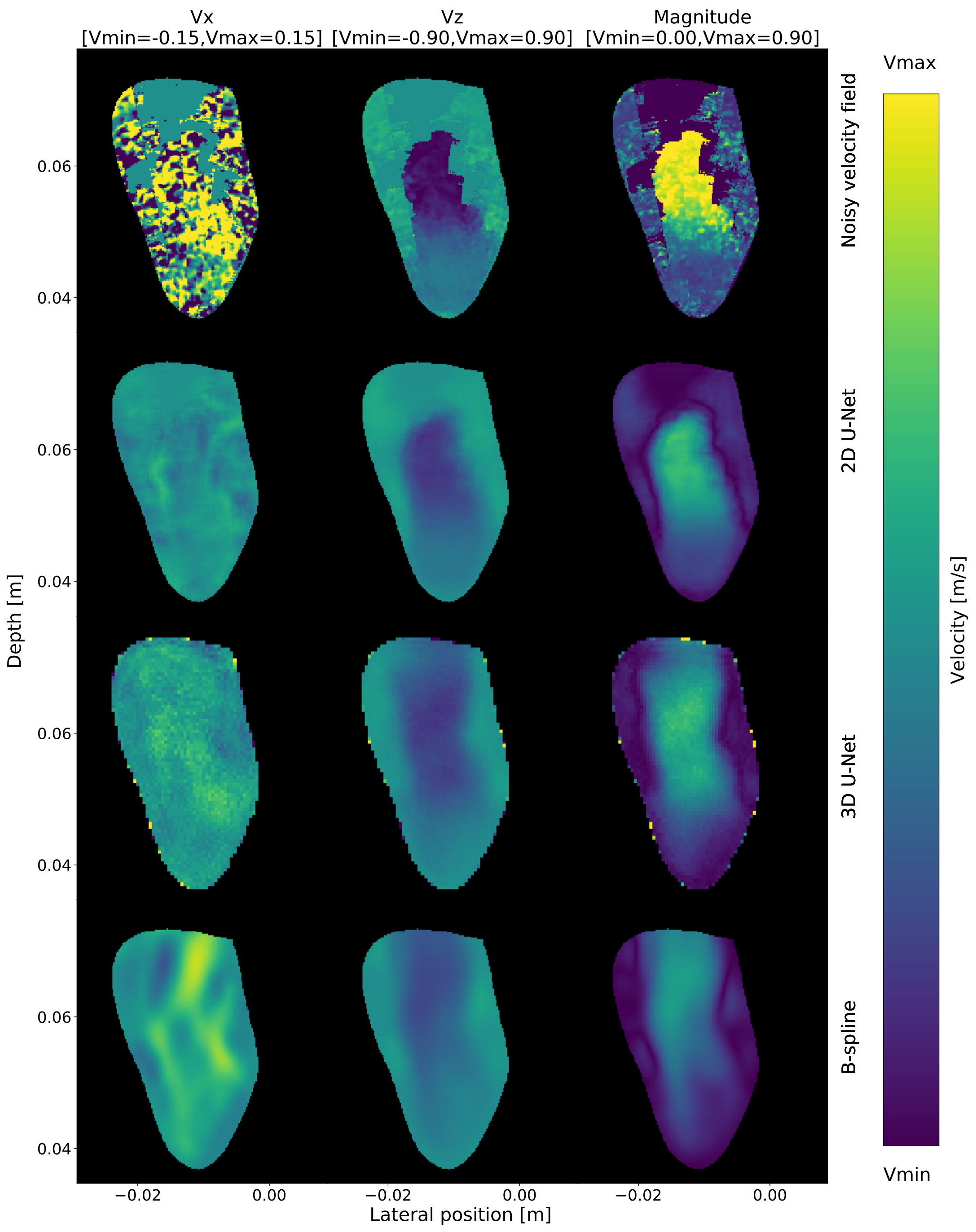


Figure 59: Predictions from measurement 12/12, frame 11/25. The figure shows the lateral (V_x) and radial (V_z) components of blood velocity fields, as well as the velocity magnitude. Each component has a specific dynamic range (V_{min},V_{max}). The *noisy velocity field* is given as an input to the *2D U-Net* and *3D U-Net* models, producing the results shown. The *B-spline* is the result from B-spline interpolation of the noisy velocity field, which is used as the "ground truth" for the in-vivo data.

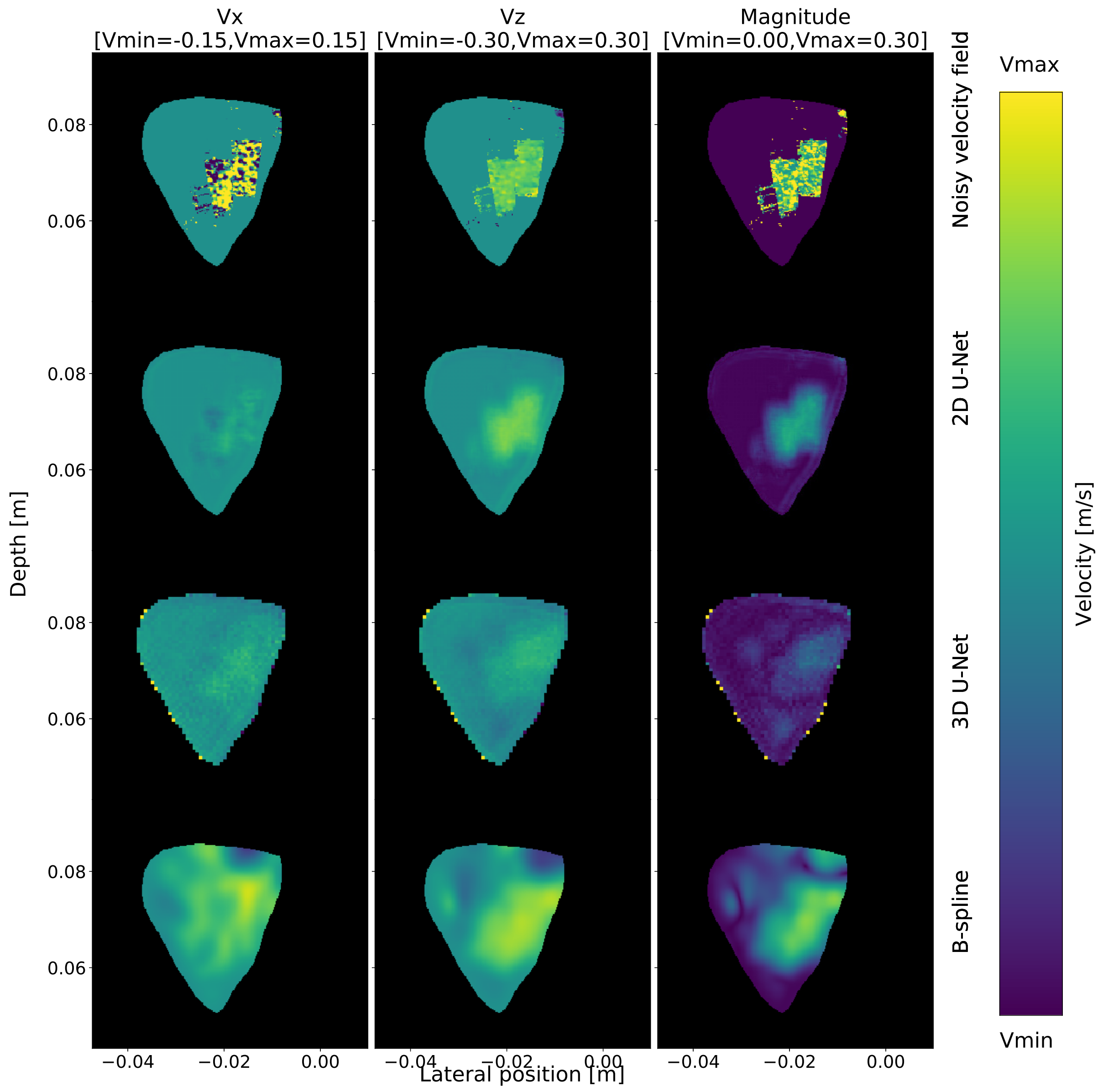


Figure 60: Predictions from measurement 2/12, frame 10/25. The figure shows the lateral (V_x) and radial (V_z) components of blood velocity fields, as well as the velocity magnitude. Each component has a specific dynamic range (V_{min},V_{max}). The *noisy velocity field* is given as an input to the *2D U-Net* and *3D U-Net* models, producing the results shown. The *B-spline* is the result from B-spline interpolation of the noisy velocity field, which is used as the "ground truth" for the in-vivo data.

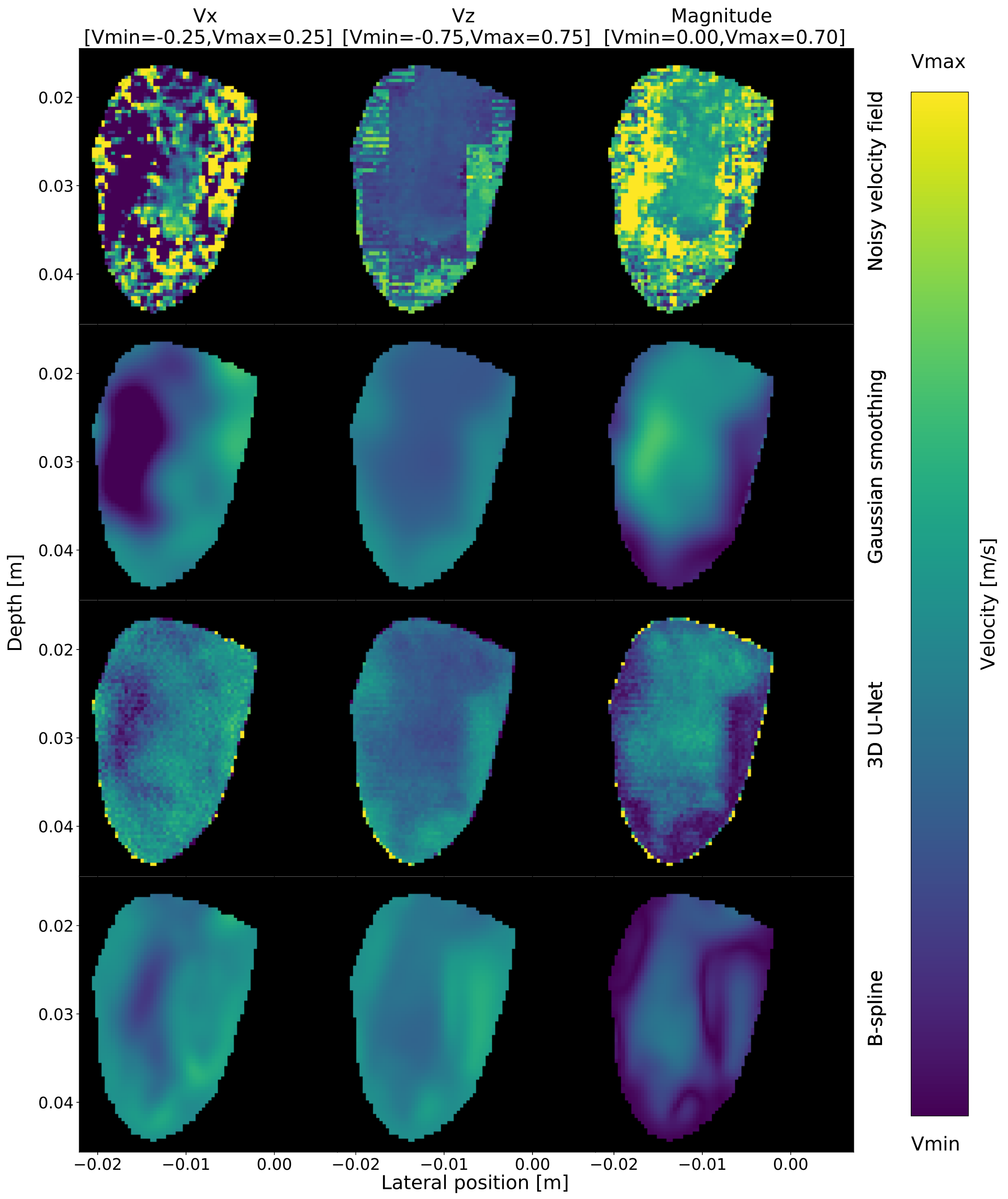


Figure 61: Predictions from measurement 9/12, frame 15/25. The figure shows the lateral (V_x) and radial (V_z) components of blood velocity fields, as well as the velocity magnitude. Each component has a specific dynamic range (V_{min},V_{max}). The *noisy velocity field* is given as an input to *3D U-Net* model, producing the results shown. The results are compared to the same noisy velocity field smoothed with a 3D Gaussian kernel. The *B-spline* is the result from B-spline interpolation of the noisy velocity field, which is used as the "ground truth" for the in-vivo data.

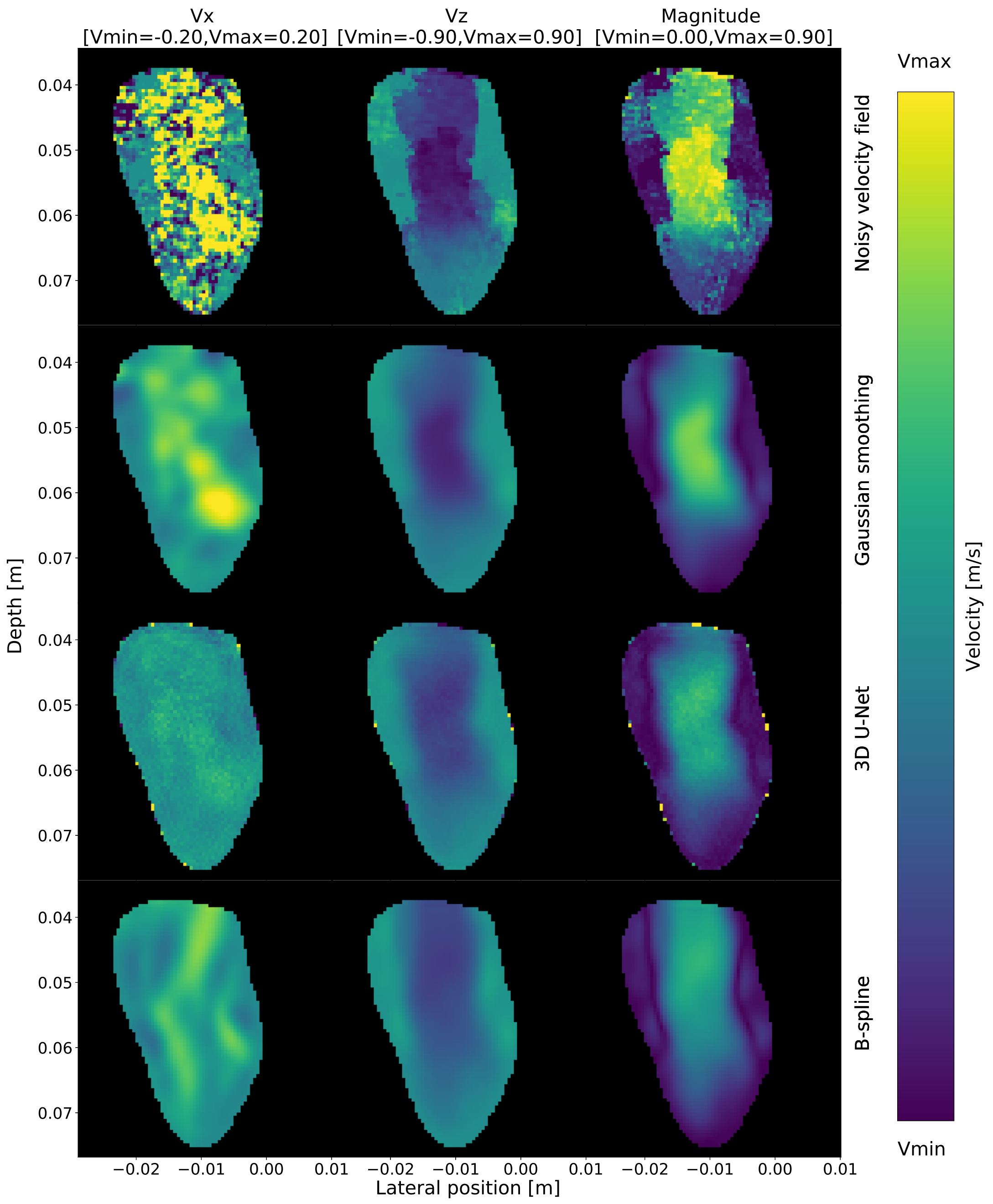


Figure 62: Predictions from measurement 12/12, frame 11/25. The figure shows the lateral (V_x) and radial (V_z) components of blood velocity fields, as well as the velocity magnitude. Each component has a specific dynamic range (V_{min},V_{max}). The *noisy velocity field* is given as an input to *3D U-Net* model, producing the results shown. The results are compared to the same noisy velocity field smoothed with a 3D Gaussian kernel. The *B-spline* is the result from B-spline interpolation of the noisy velocity field, which is used as the "ground truth" for the in-vivo data.

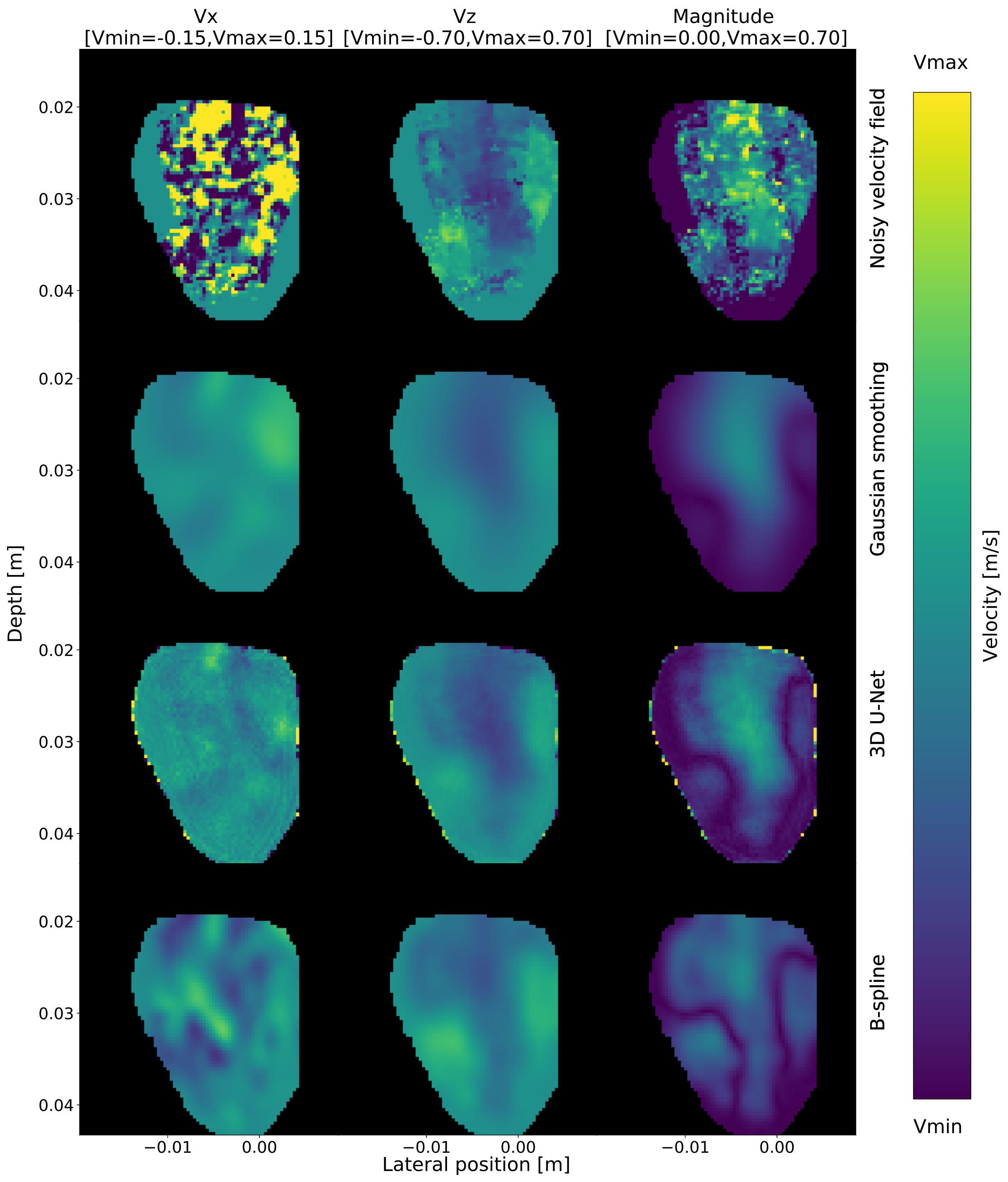
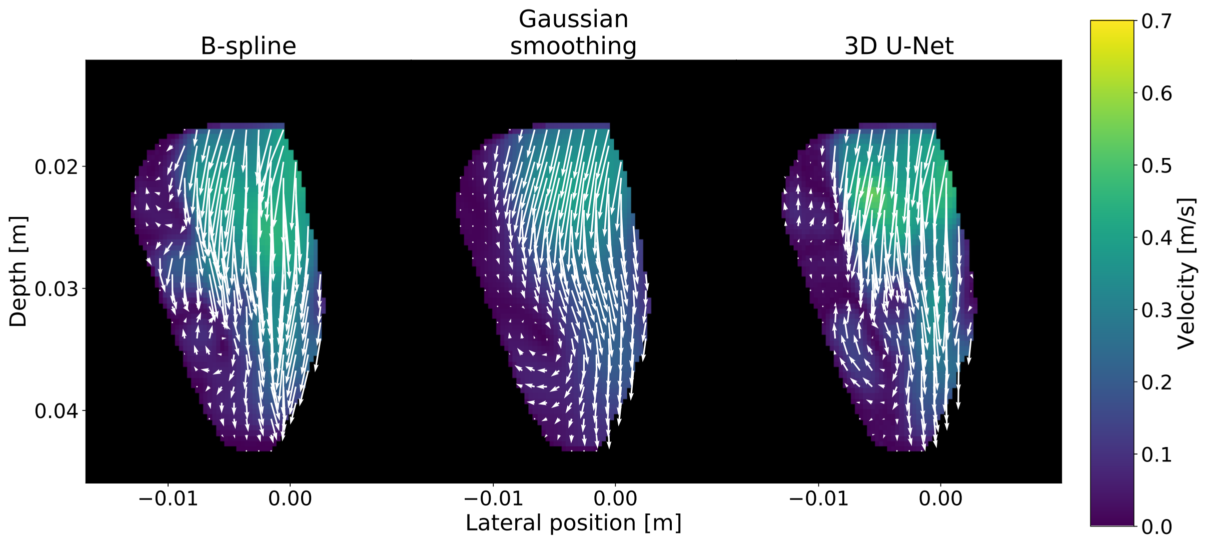
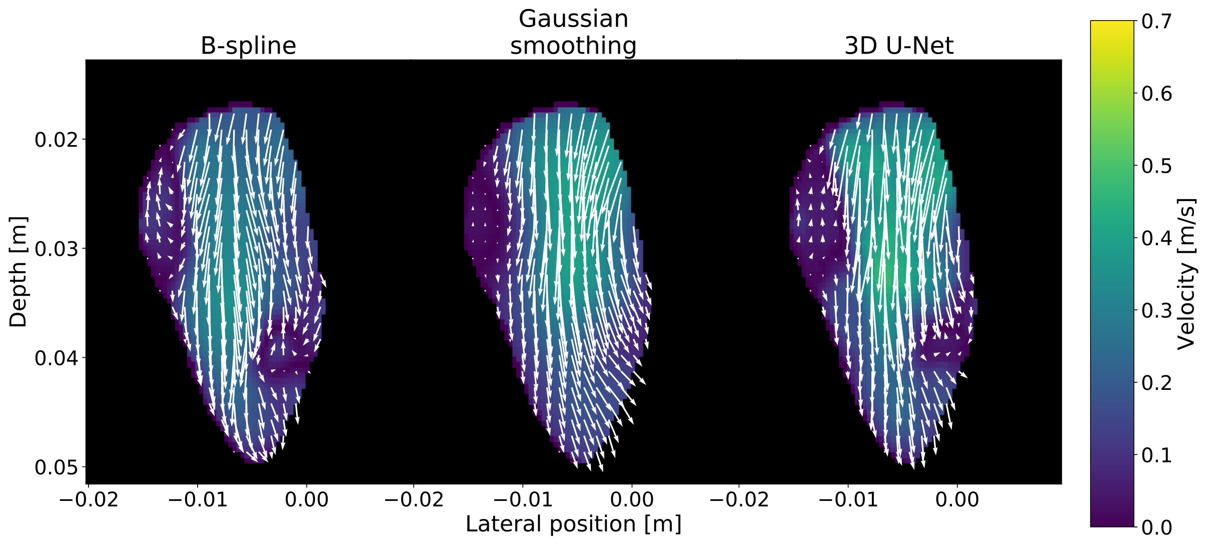


Figure 63: Predictions from measurement 3/12, frame 18/25. The figure shows the lateral (V_x) and radial (V_z) components of blood velocity fields, as well as the velocity magnitude. Each component has a specific dynamic range (V_{min},V_{max}). The *noisy velocity field* is given as an input to *3D U-Net* model, producing the results shown. The results are compared to the same noisy velocity field smoothed with a 3D Gaussian kernel. The *B-spline* is the result from B-spline interpolation of the noisy velocity field, which is used as the "ground truth" for the in-vivo data.



(a) Measurement 8/12, frame 18/25



(b) Measurement 7/12, frame 18/25

Figure 64: Comparison of model predictions from the *3D U-Net* model, alongside the results from *Gaussian smoothing* and *B-spline* interpolation. The results are visualized as the (x,z) velocity field overlaid the velocity magnitude.

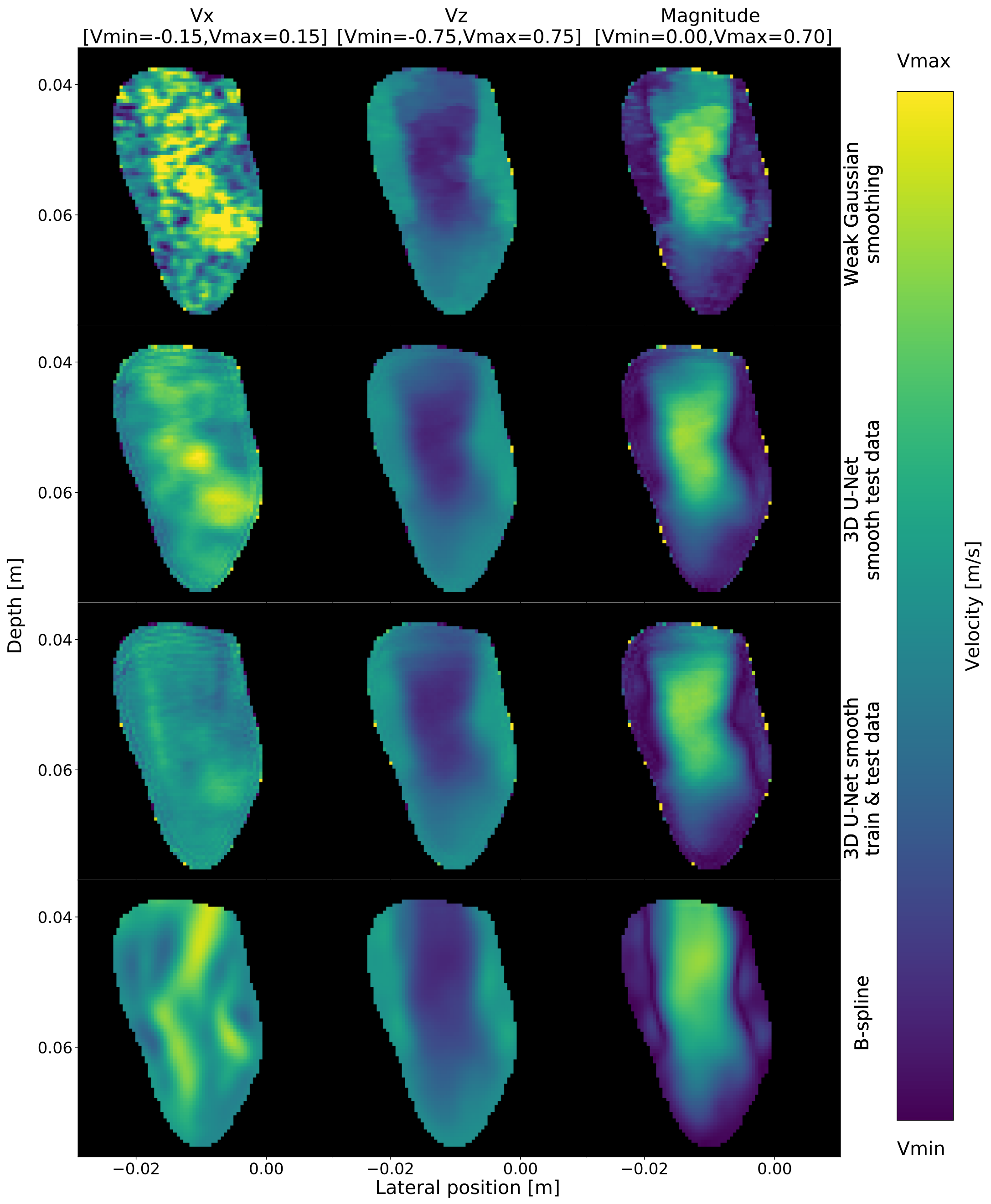


Figure 65: Predictions from measurement 12/12, frame 11/25. The figure shows the lateral (V_x) and radial (V_z) components of blood velocity fields, as well as the velocity magnitude. Each component has a specific dynamic range (V_{min},V_{max}). The *Weak Gaussian Smoothing* denotes the field smoothed with a 3D Gaussian kernel with standard deviations $1 \times 1 \times \frac{1}{2}$. This smoothed input is given as an input to a 3D U-Net model, giving the results dubbed *3D U-Net with smooth test data*. The same smoothed test data is given as input to a 3D U-Net trained on the smoothed training set, giving the results dubbed *3D U-Net with smooth train & test data*. The *B-spline* is the result from B-spline interpolation of the noisy velocity field, which is used as the ”ground truth” for the in-vivo data.

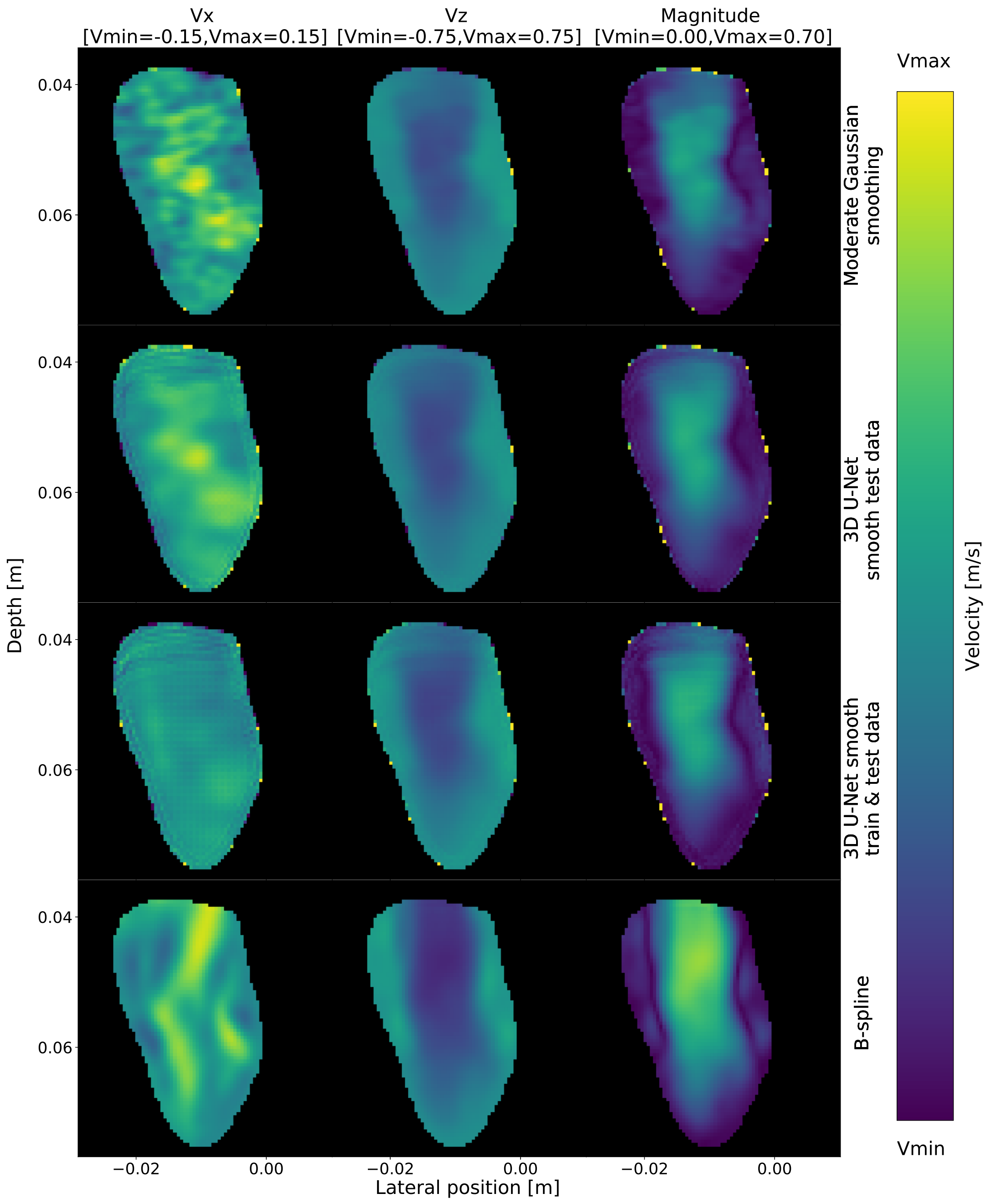


Figure 66: Predictions from measurement 9/12, frame 15/25. The figure shows the lateral (V_x) and radial (V_z) components of blood velocity fields, as well as the velocity magnitude. Each component has a specific dynamic range (V_{min},V_{max}). The *Moderate Gaussian Smoothing* denotes the field smoothed with a 3D Gaussian kernel with standard deviations $2 \times 2 \times 1$. This smoothed input is given as an input to a 3D U-Net model, giving the results dubbed *3D U-Net with smooth test data*. The same smoothed test data is given as input to a 3D U-Net trained on the smoothed training set, giving the results dubbed *3D U-Net with smooth train & test data*. The *B-spline* is the result from B-spline interpolation of the noisy velocity field, which is used as the "ground truth" for the in-vivo data.

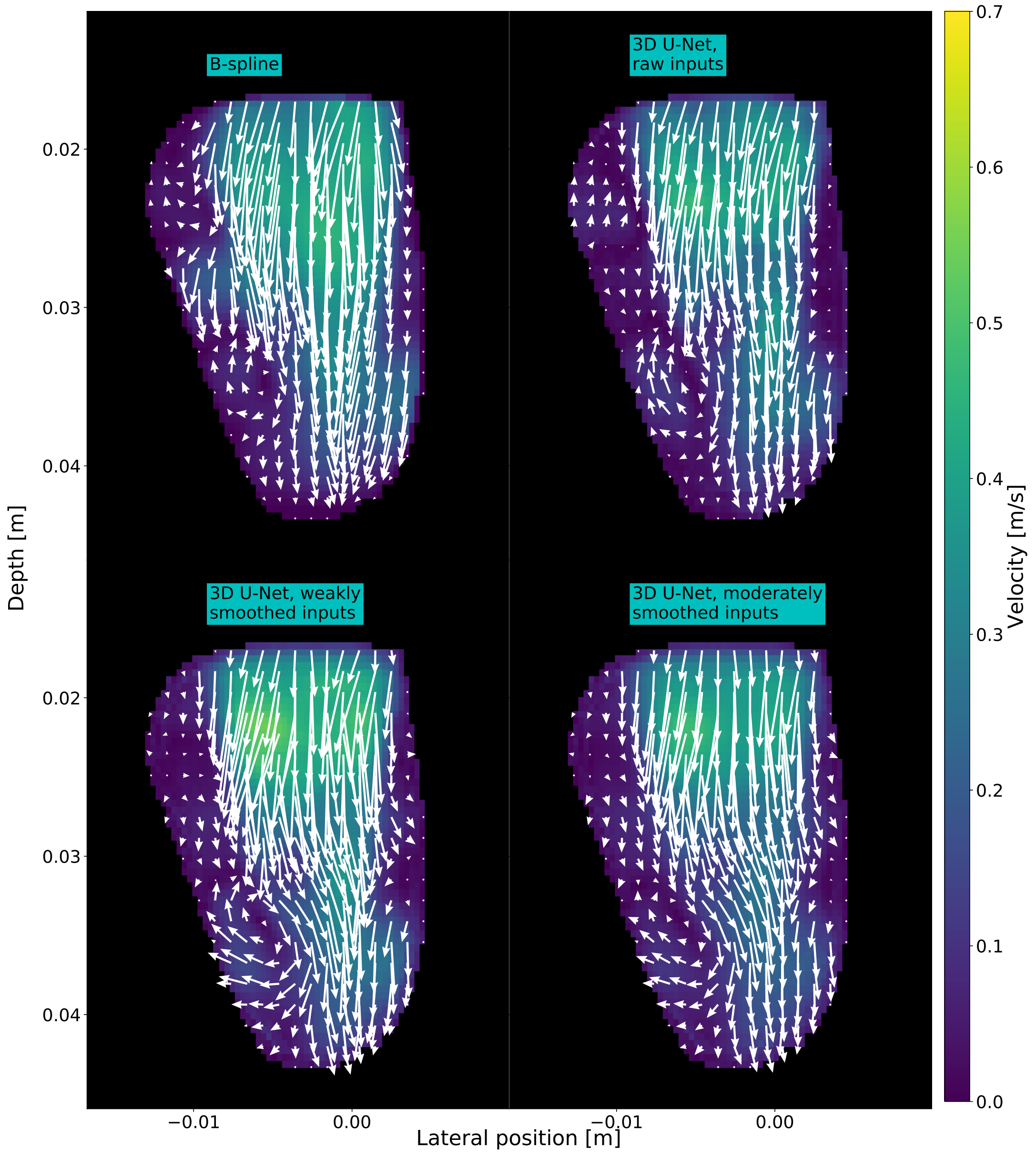


Figure 67: Measurement 8/12, frame 18/25. 3D U-Net predictions from Gaussian smoothed data, alongside the results from *B-spline* interpolation. The *raw input* is the raw test data with no smoothing applied. The *weakly smoothed input* is the raw data smoothed with a Gaussian kernel with standard deviations $1 \times 1 \times \frac{1}{2}$. The *moderately smoothed input* is the raw data smoothed with a Gaussian kernel with standard deviations $2 \times 2 \times 1$. The results are visualized as the (x,z) velocity field overlaid the velocity magnitude.

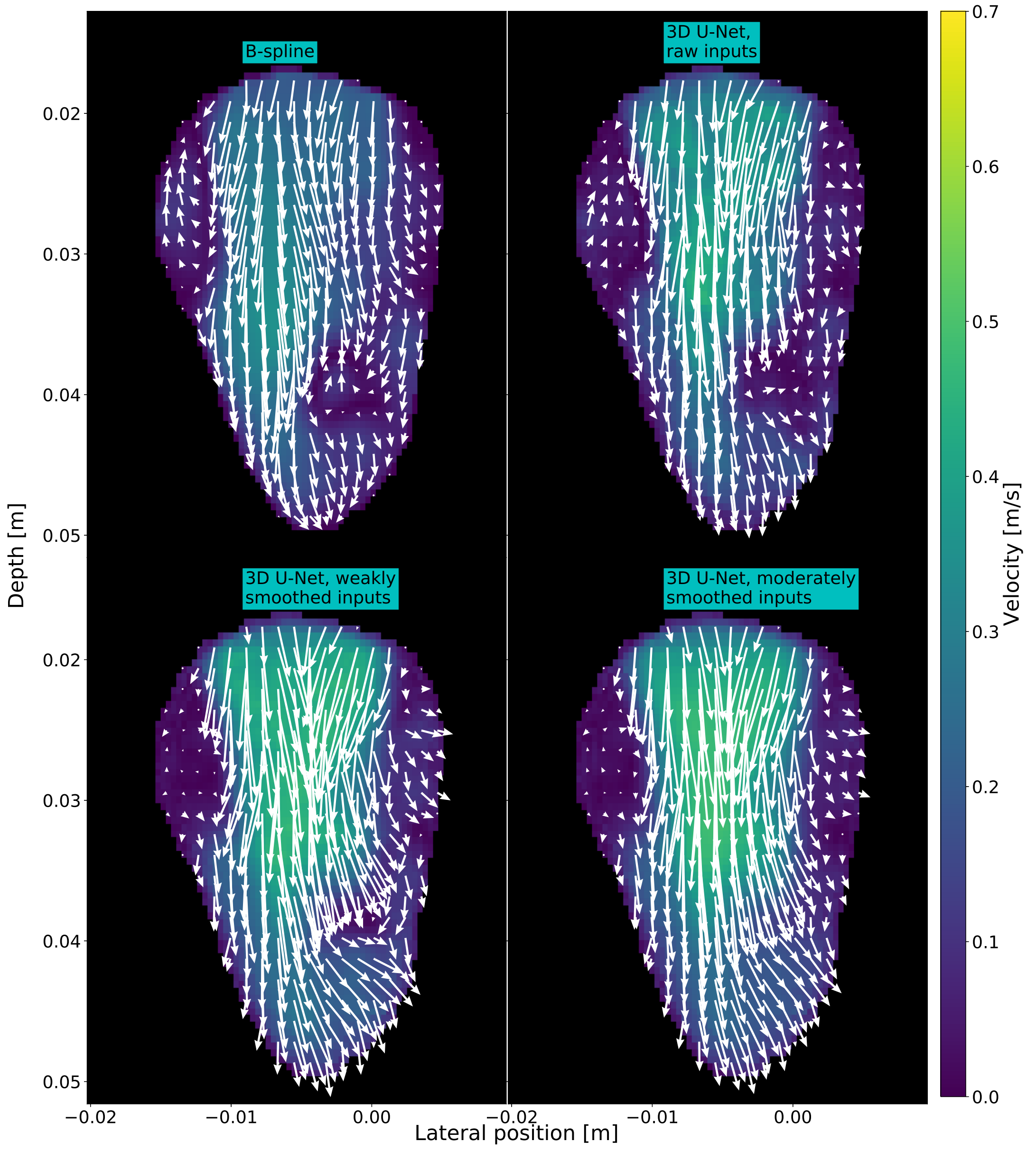


Figure 68: Measurement 7/12, frame 18/25. 3D U-Net predictions from Gaussian smoothed data, alongside the results from *B-spline* interpolation. The *raw input* is the raw test data with no smoothing applied. The *weakly smoothed input* is the raw data smoothed with a Gaussian kernel with standard deviations $1 \times 1 \times \frac{1}{2}$. The *moderately smoothed input* is the raw data smoothed with a Gaussian kernel with standard deviations $2 \times 2 \times 1$. The results are visualized as the (x,z) velocity field overlaid the velocity magnitude.

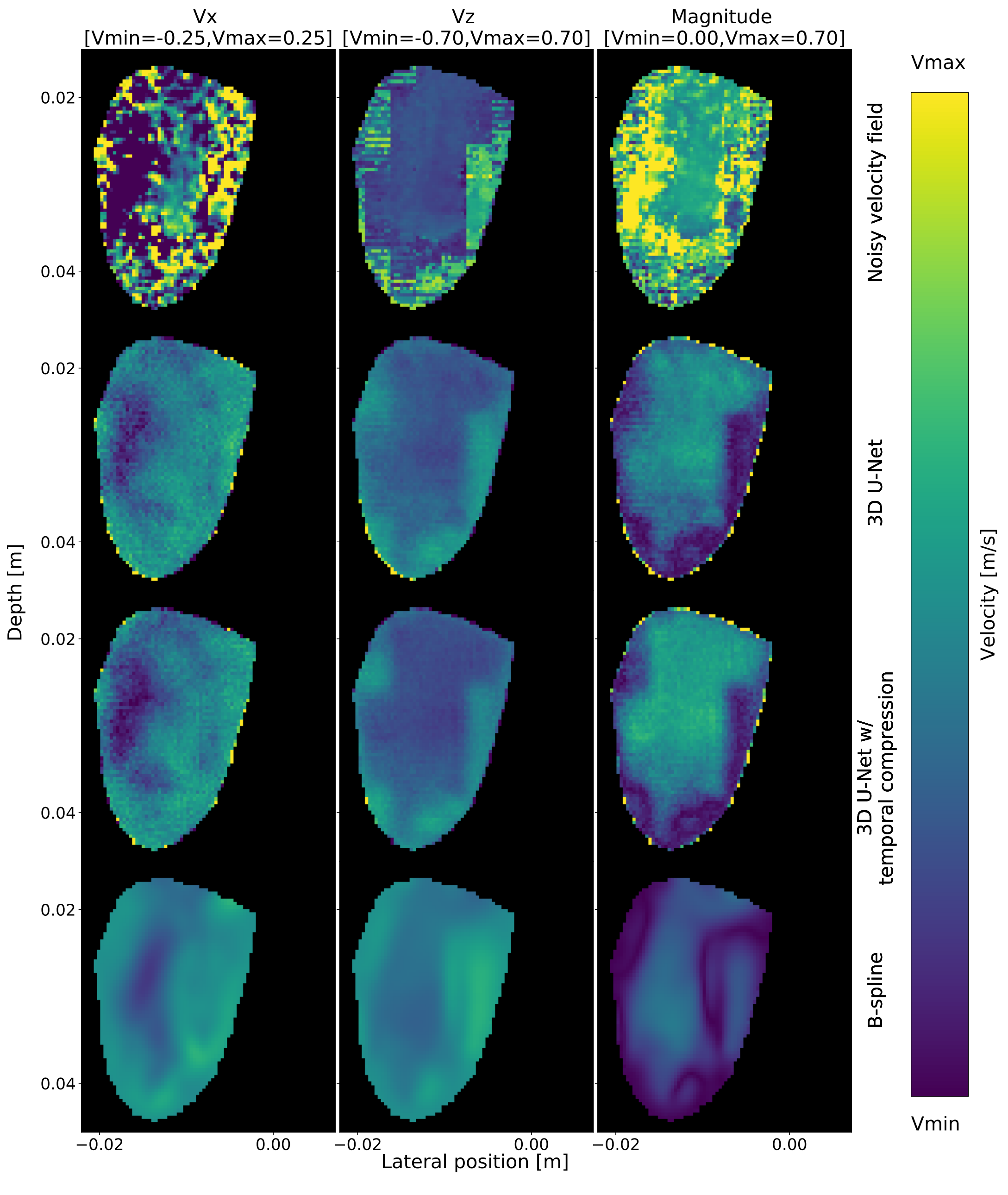


Figure 69: Predictions from measurement 9/12, frame 15/25. The figure shows the lateral (V_x) and radial (V_z) components of blood velocity fields, as well as the velocity magnitude. Each component has a specific dynamic range (V_{min},V_{max}). The *noisy velocity field* is given as an input to the 3D U-Net models. The model dubbed *3D U-Net* uses pooling and transposed convolutions along both spatial directions. The *3D U-Net with temporal compression* additionally uses pooling and transposed convolutions in the temporal direction. The *B-spline* is the result from B-spline interpolation of the noisy velocity field, which is used as the "ground truth" for the in-vivo data.

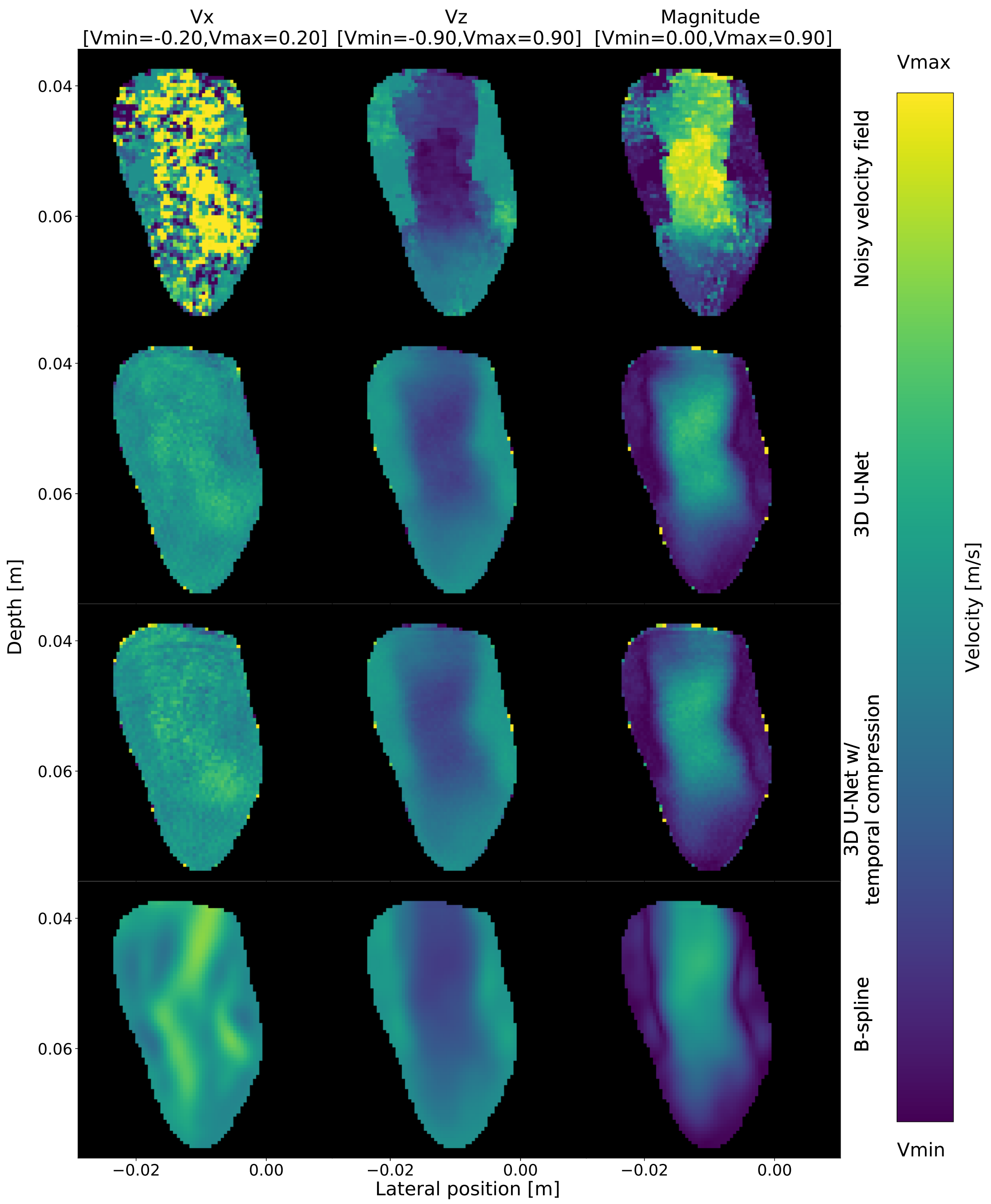


Figure 70: Predictions from measurement 12/12, frame 11/25. The figure shows the lateral (V_x) and radial (V_z) components of blood velocity fields, as well as the velocity magnitude. Each component has a specific dynamic range (V_{min},V_{max}). The *noisy velocity field* is given as an input to the 3D U-Net models. The model dubbed *3D U-Net* uses pooling and transposed convolutions along both spatial directions. The *3D U-Net with temporal compression* additionally uses pooling and transposed convolutions in the temporal direction. The *B-spline* is the result from B-spline interpolation of the noisy velocity field, which is used as the "ground truth" for the in-vivo data.

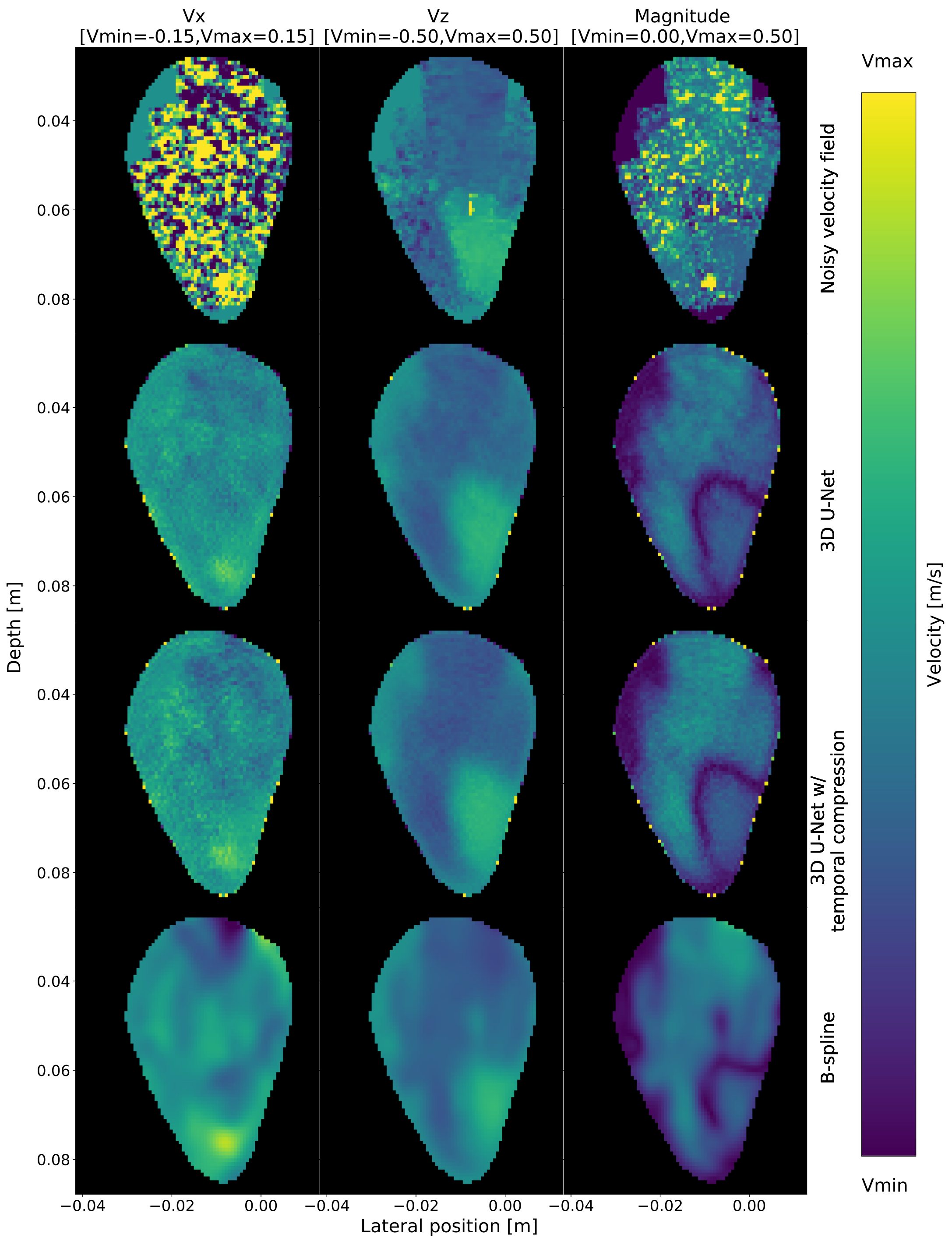


Figure 71: Predictions from measurement 1/12, frame 20/25. The figure shows the lateral (V_x) and radial (V_z) components of blood velocity fields, as well as the velocity magnitude. Each component has a specific dynamic range (V_{min},V_{max}). The *noisy velocity field* is given as an input to the 3D U-Net models. The model dubbed *3D U-Net* uses pooling and transposed convolutions along both spatial directions. The *3D U-Net with temporal compression* additionally uses pooling and transposed convolutions in the temporal direction. The *B-spline* is the result from B-spline interpolation of the noisy velocity field, which is used as the "ground truth" for the in-vivo data.

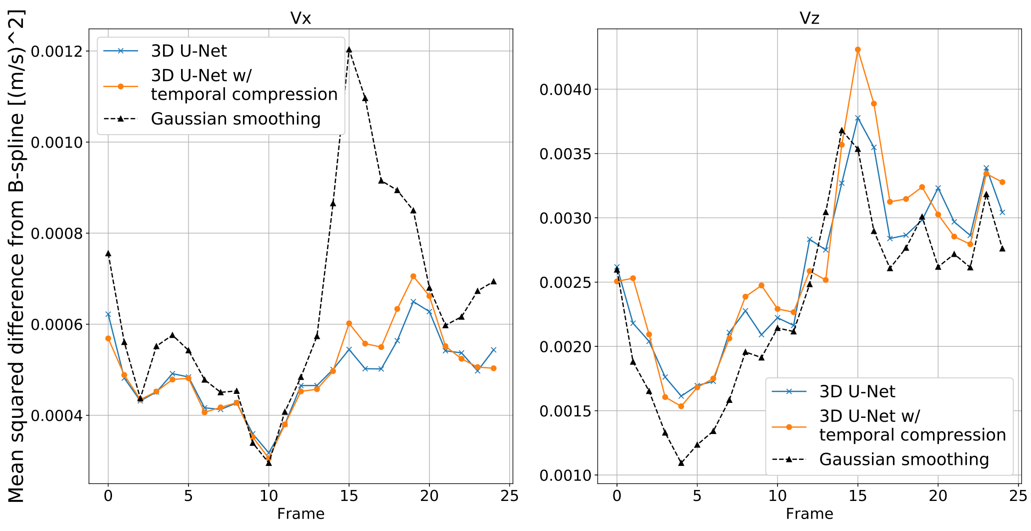


Figure 72: Mean-squared-difference between the model predictions and the B-spline results over the entire in-vivo test set. The graphs show the mean-squared-difference computed over all images of all the measurements for each frame, and for the lateral (V_x) and radial (V_z) component. *Gaussian smoothing* denotes the mean-squared-difference between the B-spline results and the results of Gaussian smoothing applied to the noisy velocity field.

4.3 GAN Results

This section contains the results from using a GAN to perform reconstruction of ultrasound blood velocity field measurements. The network is evaluated on reconstruction of the in-silico data set, as well as on the denoising of the MNIST handwritten numbers, added white Gaussian noise. This is done for model verification.

4.3.1 Reconstruction of In-Silico Velocity Data

Figures 73 and 74 show the generator denoising prediction for two samples of the in-silico data set. The *adversarial prediction* is the prediction by the generator which is trained with an adversarial loss using the proposed GAN structure, while the *direct prediction* shows the prediction by the same generator trained directly with a MSE loss. Both iterations of the generator are trained on approximately the same amount of data. Figure 75 shows the MSE reconstruction error between the model predictions and the ground truth, for each frame and velocity component. Figure 76 shows the training history of the GAN trained for 2000 epochs.

4.3.2 Denoising of the MNIST Data Set

Figure 77 shows the generator denoising prediction for two samples of the MNIST handwritten digits data set. The *adversarial prediction* is the prediction by the generator which is trained with an adversarial loss using the proposed GAN structure, while the *direct prediction* shows the prediction by the same generator trained directly with a MSE loss. Both generator models are trained such that they both see approximately the same amount of data. Figure 78 shows the training history of the GAN trained for 2000 epochs. The MSE reconstruction error for the MNIST test set was computed to 0.0373 for the adversarial loss generator and 0.018 for the direct loss generator.

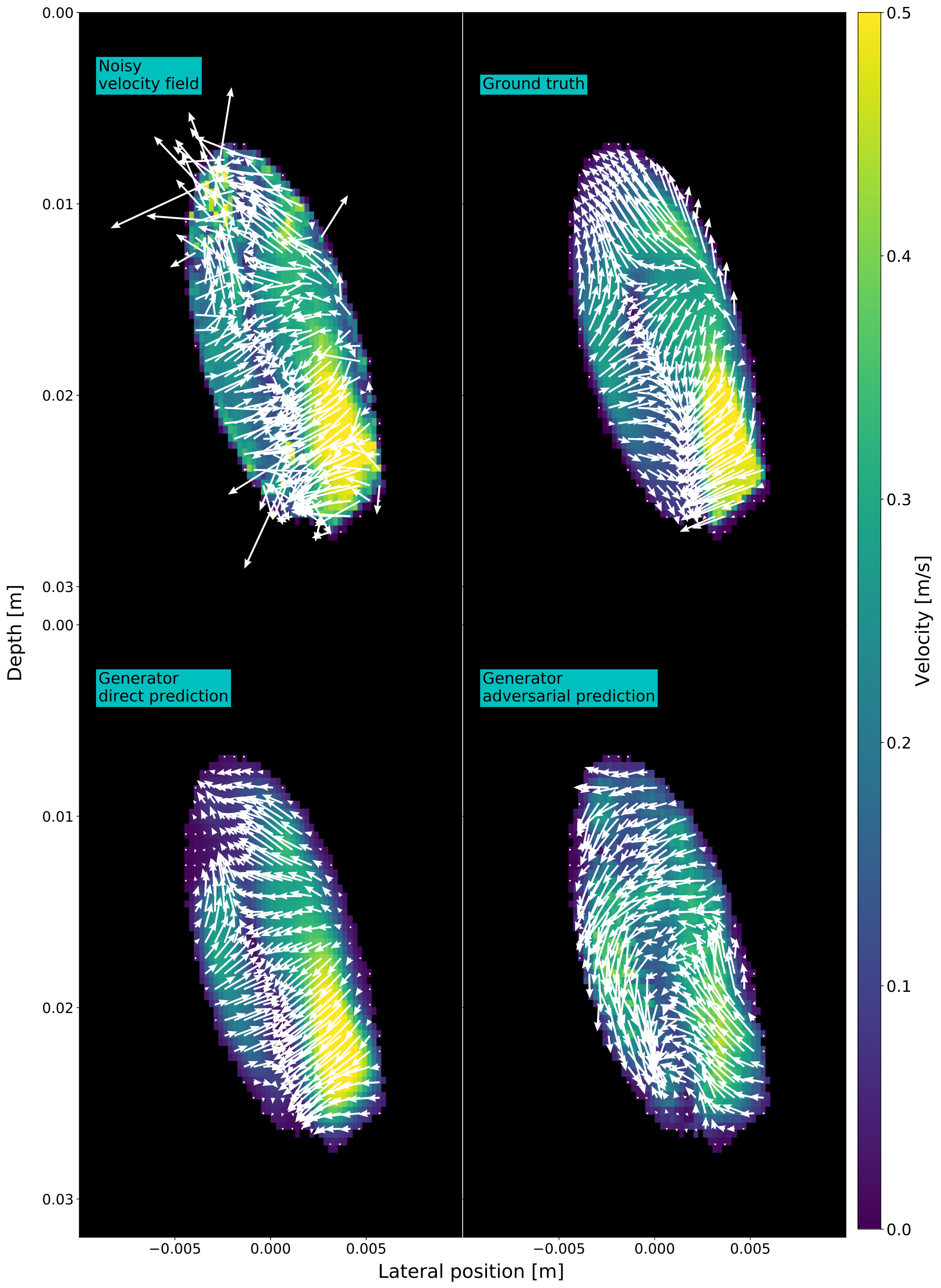


Figure 73: Sample 1, frame 15/25. Generator reconstruction prediction of the in-silico ultrasound blood velocity field data, alongside the input and ground truth, after adversarial training and after direct training with a MSE loss function. The data is visualized as the (x, z) field overlaid the velocity magnitude.

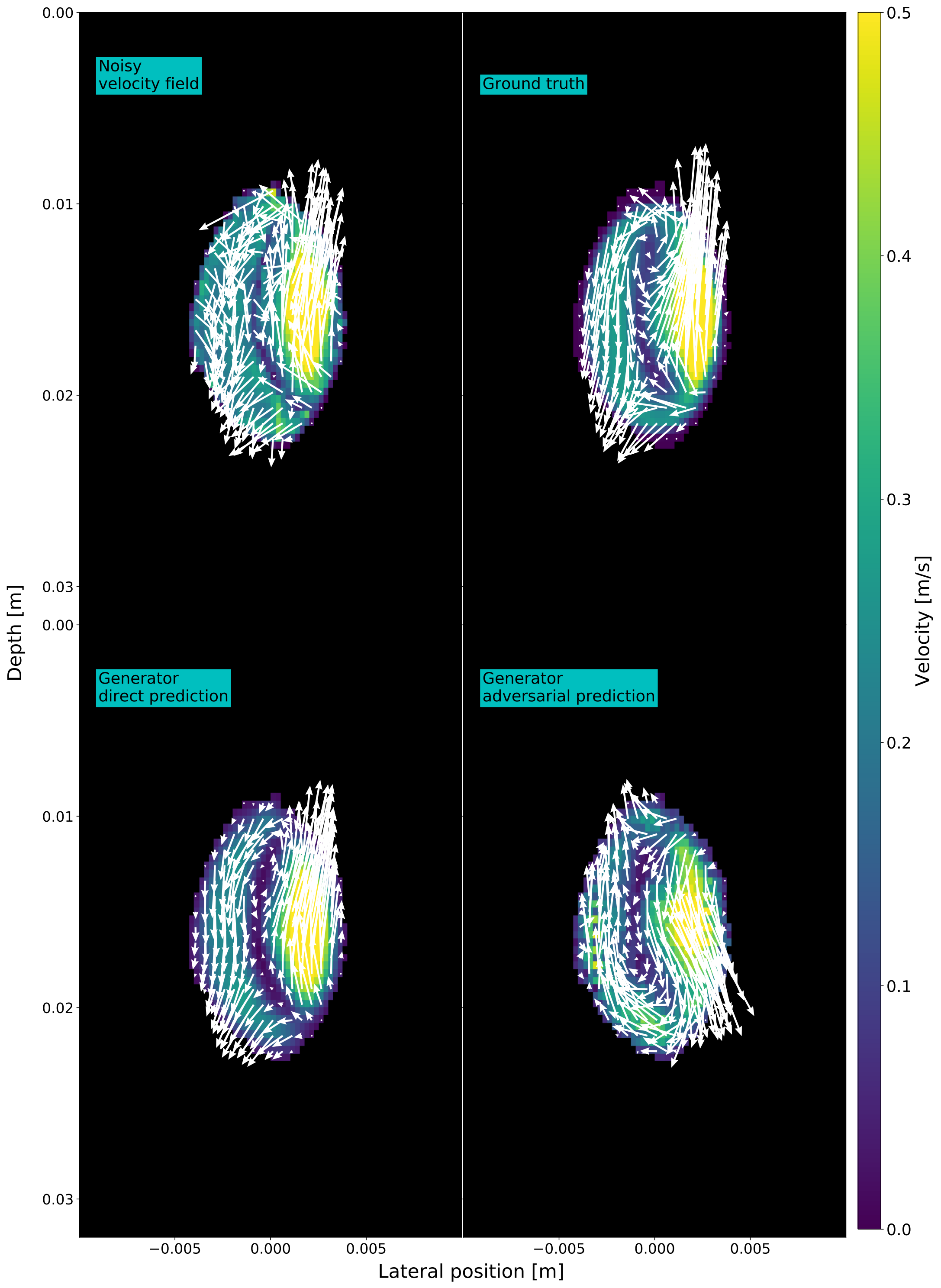


Figure 74: Sample 2, frame 15/25. Generator reconstruction prediction of the in-silico ultrasound blood velocity field data, alongside the input and ground truth, after adversarial training and after direct training with a MSE loss function. The data is visualized as the (x, z) field overlaid the velocity magnitude.

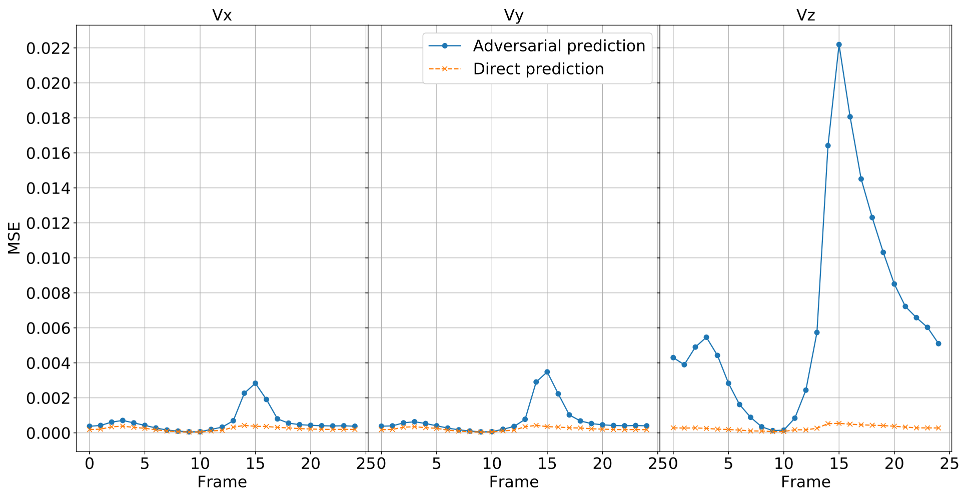


Figure 75: Average MSE for each frame of the in-silico velocity test data. The adversarial prediction is predicted by the generator trained with an adversarial loss, while the direct prediction is from the same generator trained with a MSE loss function. The MSE is averaged over all data samples and shown for each time frame and velocity component.

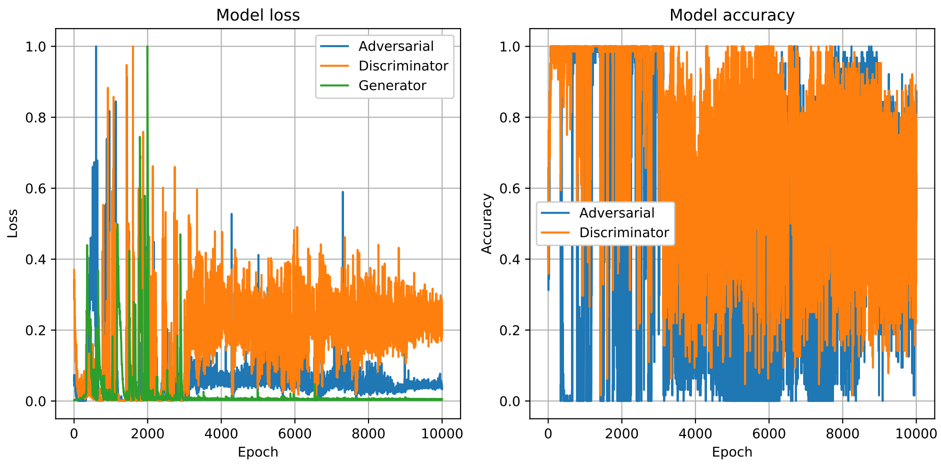
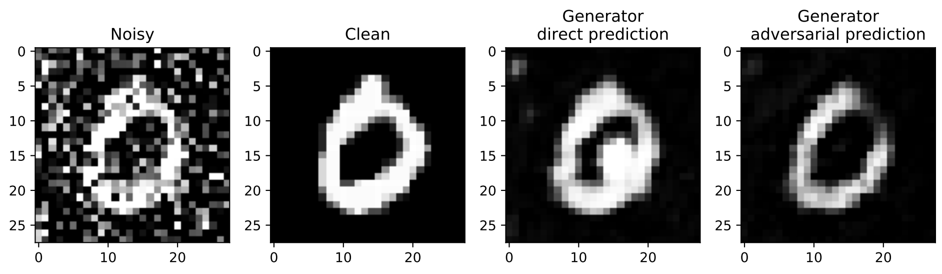
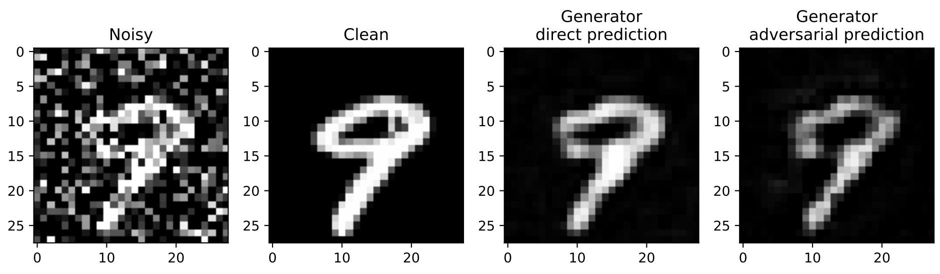


Figure 76: Training history for the GAN trained to reconstruct in-silico blood velocity fields. The discriminator loss and accuracy are computed after an update of the discriminator weights during an epoch. The adversarial loss and accuracy are computed after an update of the generator weights during an epoch. The generator loss is the MSE reconstruction loss of the generator predictions. The loss curves are normalized.



(a) Sample 1



(b) Sample 2

Figure 77: Generator denoising prediction of the MNIST handwritten digits data set, alongside the input and ground truth. The adversarial prediction is by the generator trained with an adversarial loss, while the direct prediction is from the generator trained directly with a MSE loss function.

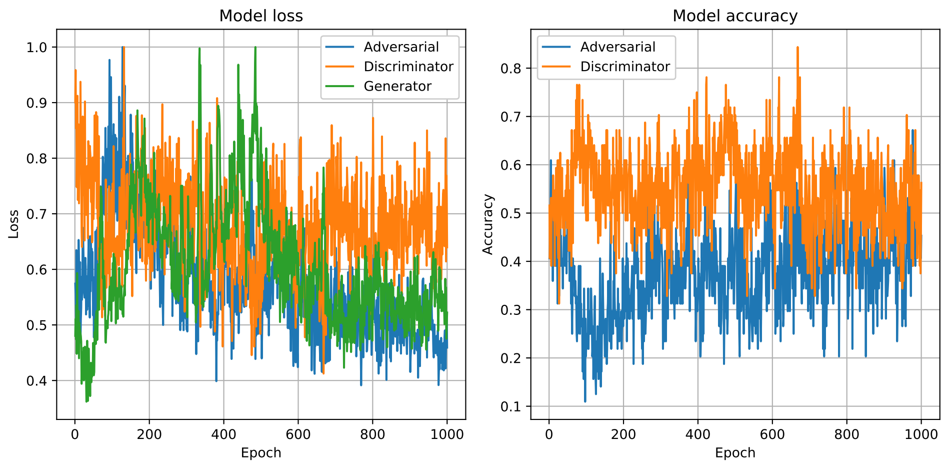


Figure 78: Training history for the GAN trained to denoise the MNIST handwritten digits data set. The discriminator loss and accuracy are computed after an update of the discriminator weights during an epoch. The adversarial loss and accuracy are computed after an update of the generator weights during an epoch. The generator loss is the MSE reconstruction loss of the generator predictions. The loss curves are normalized.

5 Discussion

5.1 In-Silico Results Discussion

This section discusses the results shown in section 4.1.

5.1.1 Exploring the Model Capabilities

The proposed model architecture comes in two different variations, one which performs convolutional operations in the spatial dimensions (*2D U-Net*) and one which performs convolutional operations in both spatial and the temporal dimension (*3D U-Net*). Additionally, it is explored how changing the amount of velocity field components at the input and output layers, and also whether or not adding correlation maps as additional input features, alters the performance of the model. Figures 23-25 show three data samples predicted with varying amounts of input and output features, predicted by the 2D U-Net. Figures 26 - 28 show similar predictions by the 3D U-Net. First of all we can notice that all the varying training procedures produce results which are very close to the ground truth, and also significantly better than the Gaussian smoothing benchmark. In Figures 23 and 26 we can see that the vortex in the middle of the object is accurately reconstructed by the neural network, while the Gaussian smoothing dampens the magnitude in the center of the vortex. The same effect can be observed in Figures 24 and 27, as well as in Figures 25 and 28, where the central vortices are better visualized by the neural networks. This shows that the proposed model is better at reconstructing details in turbulent flow patterns, at least for this generated data set.

However, it is important to keep in mind that even though the shown predictions are from the test set, not the training set, there still might occur some sort of overfitting, as the train and test data are both generated from the same CFD phantom. Even though all slices in the test and training data sets are unique, they may be very similar. Hence it is hard to tell from this data set alone whether or not the model has learned the true dynamics of flow, or if it simply has learned the flow dynamics of the specific CFD phantom it was trained on. This is further addressed when the testing is performed with an in-vivo data set, as discussed in 5.2. When looking at the quantitative results shown in Figure 29 we can see that the 2D U-Net outperforms the Gaussian smoothing benchmark by a relatively large margin, but for the 3D U-Net the improvement is less prominent. This is the case even though the qualitative results for the 3D U-Net show a significant improvement over the Gaussian smoothing, in the same way the 2D U-Net does. The reduced improvement in the reconstruction error metric is hence not visible in the frame wise visualization of the flow fields. This might indicate that the increased error is coming from the estimation inaccuracies in time. If both the 2D and 3D architectures are achieving good results for this reconstruction problem, then the 2D U-Net might have an advantage because the data temporally fixed to one frame. The 3D U-Net, on the other hand, might give slight distortions temporally due to its 3D convolutions. To summarize, from Figure 29 it might seem like there is a big performance difference between the 2D and 3D architectures, but this is probably due to the MSE and MMSE metrics being biased to give slightly lower values to the 2D U-Net predictions. Intuitively, since flow varies in time, introducing 3D convolutions should be beneficial for learning. However, this hypothesis might only be properly tested with a harder reconstruction problem, as for the in-vivo data. This is further discussed in 5.2.4.

For the 2D U-Net, when changing the number of input and output components, there is little observable variation in the results. In Figures 23 and 24 we cannot see any real difference between the predictions when dropping the out-of-plane (y) component on the input, and on the both the input and output. Both the velocity magnitude and the radial-lateral (x,z) velocity field is more or less the same. In Figure 25 we can see from the velocity magnitude that there is a high out-of-plane component in the upper left part of the object. For the U-Net 2-2 prediction, where there is no y component the magnitude is low in this area, and for the U-Net 2-3 prediction it is weaker than for the U-Net 3-3 prediction. This shows that removing the y component from the input weakens its prediction at the output. The reconstruction of the y component is further discussed in 5.1.2. As the (x, z) field is more or less the same when varying the input/output components, this shows that as long as the model receives the x and z components at the input layer, it is able to reconstruct them well. The performance does not seem to increase when it is also receiving the y component. For the 3D U-Net, Figures 26-28 show very similar results. The exception to what was said for the 2D U-Net is that the y component seems to be more poorly estimated for the 3D architecture. This is further discussed in 5.1.2.

When looking at the qualitative results it is hard to tell whether or not including the correlation maps at the input improves the performance of the model. In some singular cases, such as for the U-Net 2-3 model predictions shown in Figure 23, it might look like the radial velocity of the inner rings of the center vortex is slightly better reconstructed using the correlation map, as opposed to without. However, if we look to the U-Net 2-2 model in the same figure we can see that the opposite is happening, i.e. the model without correlation maps reconstructs the vortex better. These performance differences are nonetheless minimal, and it is therefore not possible to determine whether or not the correlation maps give an increase in overall performance. Looking at the quantitative results in Figure 29(a) we can see that the correlation maps seem to give an overall improvement on performance for the 2D U-Net 3-3 and 2-3 models, but not for the 2-2 model. In Figure 29(b) we can see that the correlation maps give a performance increase for the 3D U-Net 3-3 and 2-2 models, but not for the 2-3 model. The frame-by-frame results shown in Figures 30 and 31 show that the slight performance differences are consistent for each frame, showing that the performance differences are the same throughout the entirety of the measurements. This shows that there is seemingly a performance difference between the models with and without the correlation maps in terms of reconstruction error, but which one of the models that are doing the best does not seem to follow any noticeable pattern. Thus we can conclude that the slight performance difference we see is not due to the correlation maps, but rather due to the random weight initialization which happens before the training phase when the models are created. This conclusion tells us that the information contained in the correlation maps is redundant in the eyes of the neural network and does not improve learning. This is at least how it would seem, but a more thorough analysis of what information the network actually extracts is beyond the scope of this thesis.

5.1.2 Reconstruction of the Out-of-Plane Component

Most ultrasound technology being used today is based on acquisition of 2D measurements. This limits the possibility of flow visualization as information of out-of-plane flow cannot be captured directly. Traditional color flow imaging can only measure the radial velocity, while blood speckle tracking is limited to the dimensionality of the B-mode image. Without 3D acquisition equipment it is therefore necessary to reconstruct the y component from the other two components. Figure 32 shows two samples of the y component predicted by the 2D U-Net 3-3, which is given the all the three input components of the noisy input data, and the 2D U-Net 2-3, which is only given the x and z components. Here we can see that the U-Net 2-2 models perform similarity well to the U-Net 3-3 models, as the main features of the ground truth are captured without explicit knowledge about the y component. From Figure 30 we can see that, on average, the U-Net 2-2 models perform worse in terms of reconstruction error than the U-Net 3-3 models. This seems reasonable as reconstructing the y component should be an easier task with explicit knowledge. However, it is apparent that there is enough information about the missing component embedded in the other two such that the neural network is able to learn how to perform a decent reconstruction.

Note that in Figure 32(a) it seems like the negative velocities are more accurately estimated by the U-Net 2-2 with correlation maps than without. This is substantiated by the quantitative results in Figure 30 where we can see that the U-Net 2-2 with correlation maps does perform quite a lot better than without. There might be a real advantage to using correlation maps when trying to reconstruct a missing component, as there might be non-redundant information about the missing component embedded in the correlation maps which is beneficial for learning. As discussed in 5.1.1, there cannot be seen any overall advantage by using the correlation maps, as no visual improvement or any clear pattern in reduction of reconstruction error can be seen. However, there is a possibility that the improved performance is not simply due to random weight initialization for the U-Net 2-2 model in particular. The reason for this is that the reduction in reconstruction error for the y component for the U-Net 2-2 with correlation maps, seen in Figure 30, seems to be substantially greater than for the other model pairs. However, since the U-Net 2-3 with correlation maps also performs better on both the x and z components this may still indicate that the U-Net 2-3 without correlation maps is just particularly unlucky with its initial weights. In this case it would make sense for it not only to underperform on the y component, but also the other two components. In conclusion, we still cannot say for sure whether or not the correlation maps give an additional benefit when reconstructing the y component from the x and z components. In order to unravel a more definite answer we need to investigate this more in depth.

When it comes to the 3D U-Net it seems like the 2-3 model is unable to make predictions of the y component. In Figure 33 we can see that while the 3-3 models manage to reconstruct the y component as expected, the 2-3 models only output near-zero fields. In Figure 31 we can see that the average reconstruction error for the U-Net 2-3 is very large for the y component in comparison to the other two components. This shows that the model has not been able to learn how to reconstruct the missing component, and thus outputs mostly zeros as this is the second best option to minimize the loss. Since the data set used to train the 3D U-Net is generated exactly the same way as the data set used to train the 2D U-Net, and since the 2D U-Net is in fact able to reconstruct the missing component, there is no reason to conclude that the x and z components in the data set do not contain enough information for the 3D U-Net to be able to reconstruct the y component. Instead, this has to be due to the differences in the model structure. Since the 3D architecture has about 2.8 times more trainable parameters than the 2D architecture, it is natural to assume it would need more data in order to achieve the same performance as the 2D architecture. Therefore it is trained with 2.8 times more data to account for the increase in parameters, but there still might be a possibility it would need even more data to learn properly. However, Figure 34 shows that the U-Net 2-3 models show no signs of overfitting on the training data as both the validation and training losses have similar behaviors. If the models were overfitting on the training set we would expect the validation loss to increase relative to the training loss as the number of epochs increase. Thus the models are not underachieving on the task of reconstructing the y component because of overfitting. Nonetheless, deep neural networks always benefit from more data, hence it could be an idea to test the performance when the amount of data is increased substantially more, maybe instead by a factor of 10. However, due to hardware limitations, this could only be done if the data generation and training procedures described in 3.5 and 3.6.1 are altered to do joint data generation and training. Otherwise, we would exhaust the memory, as the current data set size is already pushing the memory capacity. To summarize, there is no obvious reason as to why the 3D U-Net is not able to reconstruct the y component from the x and z components while the 2D U-Net is able to do so, but it might get clearer if we are able to train the 3D U-Net with substantially more data and see whether or not this changes the performance.

5.1.3 Compression in Three Dimensions

As described in section 2.4.2, the U-Net architecture is based on cascading pooling layers which reduce the number of data points in the images, and followingly cascading transposed convolutional layers which increase the number of data points again. The pooling layers are often interpreted to do a form of compression of the data into a latent space representation, while the transposed convolutional layers perform a decoding of the latent space information to produce the output. The intuition behind this is that the information in the data consists of both rough and fine features. The U-Net deconstructs the data into its set of features and combines them in optimal ways to produce the desired output. So far we have only explored model architectures which uses this compression/decoding scheme in the spatial domain. The 3D architecture introduces 3D convolutional layers, but does not do any compression/decoding along the temporal axis. Only the pure convolutional layers operate on the time domain as well as the spatial domain. Based on the intuition, it may be an advantage to also try to perform the compression/decoding scheme in the time domain as well as the spatial domain, as we might be able to encode more information about the temporal dependencies in the flow.

Figure 35 and 36 show two sample predictions from the test data of the original 3D U-Net model, and the variation which performs compression and decoding along both spatial and temporal axes. We cannot see any significant difference between the model predictions for any of the samples shown when visualized this way. However, from Figure 37 we can see a significant improvement in reconstruction error by the 3D U-Net with temporal compression. Not only does the 3D U-Net with temporal compression improve from the model with only spatial compression, but it also beats the Gaussian smoothing benchmarks for basically all frames on average. By observing the results shown in Figure 38 we can see where this improvement comes from. Here we can see that the 3D U-Net with no temporal compression generates blurry and distorted contours, while the 3D U-Net with temporal compression generates contours which are of much higher quality in comparison to the ground truth. This might confirm our intuition that the additional temporal compression encodes more temporal information about the flow; when temporal compression is applied, the neural network is better able to learn the temporal variations in the object walls and in the flow such that its predictions about the object's placement in time is more accurate. However, making such an assumption is only based on intuition and does not necessarily have to be true. When "upgrading" the model to use temporal compression, the number of learnable parameters increases from about 5.4 million to 5.6 million because of the added dimension in the transposed convolutional layers. Hence the increased total number of parameters alone could be the source of the increased performance, without the intuitive explanation necessarily being true. This statement is difficult to justify properly, but nonetheless, the proposed model variation does improve the performance, no matter what the true explanation is.

5.2 In-Vivo Results Discussion

This section discusses the results shown in section 4.2.

5.2.1 Validation challenges of the "ground truth"

As there does not exist any ground truth data for the noisy in-vivo ultrasound blood flow measurements it is not possible to do coherent supervised learning on the data. A workaround is to use B-spline interpolation as approximate labels instead, as mentioned in 3.6.3. The B-spline interpolation results of the in-vivo data set were validated in a ultrasound blood flow visualization software, as shown in Figure 39. The results were deemed realistic by field experts who performed the validation. In addition to this, the method has been validated by in-silico tests to give accurate a satisfactory reconstruction of flow fields for clinical use. However, the method is likely to not give perfect results, thus the comparison between the neural network predictions and the B-spline interpolation "ground truth" should be done while still keeping in mind that the B-spline interpolation does not necessarily give a perfect image of the blood flow fields, but rather a very realistic image.

There are two ways of reasoning for this approach. One way is to remember that the neural networks are in fact trained through coherent supervised learning with simulated data. When the networks are then further trained on the B-spline interpolation results, this could be interpreted as combining the domain knowledge from the simulation and from the B-spline interpolation algorithm. This could potentially enhance the overall performance since the network is learning from a bigger, more diverse data set. The other way of reasoning is that if we could manage to create a model which reconstructs the B-spline interpolation results with high accuracy, this would be beneficial in terms of computation time. When using GPU accelerated computing, the B-spline interpolation algorithm takes somewhere around a minute to complete, while the prediction with the neural network merely takes a couple of seconds. Hence there is a potential for vastly decreasing computation time, which would be very useful for real-time measurements.

5.2.2 Fine-Tuning of Shallow vs. Deep Layers

When using the network based transfer learning approach where half the network layers are fine-tuned to fit the target domain in-vivo data, it is beneficial to fine-tune the deeper half of the network. This can be seen from the qualitative comparison of fine-tuning the deep and shallow layers of the 2D U-Net (Figures 40-41), and similarly for the 3D U-Net (Figures 42-43). Here we see that the model with fine-tuned shallow layers creates fields which appear significantly more noisy than the model with fine-tuned deep layers. This is especially apparent in the lateral component. This coincides with the intuition behind this kind of transfer learning; the shallow layers represent the rougher features of the data, which will be similar to the rough features of the source domain data. The differences between the source and target domains are mainly in the finer details, hence fitting the deeper layers is beneficial as the deeper layers encode the finer features of the target domain. The quantitative results shown in Figures 44 and 45 similarly show that the MSE reconstruction error is less for the models with fine-tuned deep layers. However, it is important to notice that even though the models with fine-tuned deep layers are better, they are not performing too well in terms of reconstructing the B-spline interpolation results. This is especially the case for the lateral component, which tends to almost vanish completely. This will be discussed further in the upcoming sections.

5.2.3 Transfer Learning Effectiveness

There are different approaches to doing transfer learning. One is by fine-tuning a part of the model to the target domain (the in-vivo data), while leaving the rest fit to the source domain (the in-silico data), as described in 5.2.2. From this point on we will consider the model where the deeper half of the U-Net is fine-tuned, and will be referred to as the *Fine-tune model*. The other way of doing transfer learning which is considered in this thesis is by fitting all the weights of the entire model to the target domain, but initializing the model weights to the ones learned from the source domain. This is referred to as the *Full in-vivo model*. Additionally, we consider using the model trained on the source domain without doing any training on the target domain. This is referred to as the *In-silico base model*.

Figures 46-48 show three different samples of the predictions of the in-vivo test set by the different 2D U-Net model variations, while 49-51 show the same predictions by the equivalent 3D U-Net model variations. Firstly, it can be seen that the In-silico base model, for both 2D and 3D model architectures, seems to struggle with denoising the input fields. We can observe that the model predicts speckles of high velocities, especially in the lateral component, which most of the time does not resemble the B-spline interpolation results. This is likely to be because the lateral component tends to be severely noisy, more noisy than the in-silico data from the source domain it is trained on. This might point towards the fact that the performance of the model could benefit from modifying the in-silico data closer resemble the characteristics of the in-vivo data. In particular, it could be beneficial to generate the in-silico data such that the lateral component always is more noisy than the radial component. This would give a more realistic data set as the radial signal tends to be significantly stronger than the lateral signal. This effect is not taken into account at this time since the generated data is rotated randomly after adding noise, hence omitting the information about which component is which. See 3.5 for a complete description of the data generation.

There are also other characteristics in the in-vivo data which are not taken into account in the in-silico data, which might affect the performance. The biggest difference is the large cut-offs which can be seen in the noisy input data, most prominent in the noisy velocity field shown in Figure 46. These cut-offs are mainly due to antialiasing which is performed after the doppler measurement, setting aliased regions to zero. This is not taken into account while generating the source domain data set. In Figure 46 we can observe that these regions are not handled very well in terms of reconstruction, especially not by the 2D U-Net models. Instead, these areas are often left untouched. The antialiasing could potentially be taken into account in the source domain data generation, maybe aiding to mitigate this effect. Possible improvements to these issues are discussed in 5.4.1.

We can see that the transfer learning models also struggle with the reconstruction of the lateral component. Both the Fine-tune model and the Full in-vivo model tend to underestimate the lateral component. This can again be connected to the underlying source domain data not representing the target domain well enough. It is interesting to note that the transfer learning models underestimate the velocities in the lateral component, while the In-silico base model overestimates them. This can be explained by the fact that the transfer learning models are trying to minimize the MSE loss from the B-spline interpolation results. As the target domain data is both sparse and too different from the source domain data, the networks might see it more optimal to underestimate the velocities as this would on average give a smaller MSE reconstruction error. Figure 53(a) shows the reconstruction error from the B-spline interpolation averaged over all measurements in the test set, while Figure 53(b) shows similar results from the 3D U-Net. The trend in these graphs is that the in-silico model is shown to have a relatively large MSE in the beginning (systole) and in the middle (diastole) of the cardiac cycle. This is especially the case for the lateral component. Figure 53(b) shows that the reconstruction error in the lateral component is on average significantly higher than for the transfer learning models, as well as the Gaussian smoothing benchmark. This is due to the large amounts of noise still left in the lateral component by the In-silico base model.

From the quantitative results it seems like the transfer learning models are reconstructing the lateral component quite well, however this might be caused by the fact that the MSE metric will be lower for near-zero estimations than for erroneous estimations with greater magnitude. If we observe the predictions of the Fine-tune model and the Full in-vivo model in Figure 47 we can see that the Fine-tune model predicts near-zero fields for both components. Yet, Figure 52 shows that it is doing much better than the In-silico base model, although Figure 47 shows that neither the In-silico base model nor the Fine-tune model provide predictions that resemble the B-spline interpolation results. This shows that MSE metric does not fully capture the quality of the results. It does however show that the Full in-vivo model tends to do better than both the other models, for example shown in the averaged results in Figure 53. This coincides with what we see in the quantitative results.

In the result shown in Figure 51 the Full in-vivo model reconstructs the radial component well, and is marginally better at reconstructing the lateral component as compared to the Fine-tune model. Although the lateral component is still fairly different from the one reconstructed by the B-spline interpolation, we can observe similar patterns in the transitions of velocities. The positive velocities in the B-spline interpolation around the edges can also be found in the Full in-vivo model prediction. The large spot of negative velocity in the middle of the lateral image is not found in the model prediction, however we can observe a large pool of negative velocities smaller in magnitude in the center of the prediction. In total this gives an impression of the Full in-vivo model prediction being able to roughly capture the ”ground truth” lateral component. We can see the same effect in the other two 3D U-Net comparisons, shown in Figures 49 and 50. Here, the lateral component predictions by the Full in-vivo models are not very accurate, but they give a closer impression of the ”ground truth” than the other two models. The lateral component predictions made by the 2D U-Net models, shown in Figures 46-48, are poorer than the ones done by the 3D U-Net models for all the different training variations, although the radial components seem to be better for the Full in-vivo model as compared to the other two. In general it seems like the 2D U-Net performs worse than the 3D U-Net.

To summarize the performance we can look at Figure 54. Here we see that in terms of MSE reconstruction error, the Full in-vivo model seems to perform the best out of the three models. This fits the qualitative results which seemed to be in favour of the Full in-vivo model as well. From this bar chart alone it seems like only the in-silico model benefits from using a 3D architecture as opposed to a 2D architecture, however the qualitative analysis shows that a 3D architecture is favorable, which is further discussed in 5.2.4. It is also worth noticing that the Full in-vivo model does not seem to perform any better than the Gaussian smoothing benchmark. See section 5.2.5 for a further analysis of this. In conclusion, the Full in-vivo model seems to be the best choice of model. The following sections will therefore focus on this model.

When the Full in-vivo model seems to outperform the other two models, one can begin to doubt whether using the in-silico data set in training gives any benefit at all. To analyse this, the Full in-vivo model is compared to a model using the same architecture, trained fully on the in-vivo data set using randomly initialized weights. Figures 55 and 56 show comparisons between the Full in-vivo 3D U-Net model with weights pretrained on the in-silico data set, and an identical model with randomly initialized weights. In both examples we see that the models perform similarly well on reconstructing the radial component, but the model with pretrained weights drastically outperforms the model with randomly initialized weights in reconstructing the lateral component. The model with randomly initialized weights only outputs noise in the lateral component, while the model with pretrained weights produces meaningful predictions. The result in Figure 55 is particularly good, as the main features of the lateral component are captured. Figure 56 shows a less accurate prediction of the lateral component, but the features are still roughly captured, albeit slightly underestimated. These results coincide with the quantitative results shown in Figure 57, where the model with pretrained weights outperforms the one with randomly initialized weights in terms of MSE, for both components. This shows that applying transfer learning to this problem does in fact have a positive effect. Hence there is information in the in-silico data which is beneficial for reconstruction of the in-vivo data. Although the results might not prove to be as good as the state of the art, this still shows that the proposed transfer learning method works in concept.

5.2.4 2D U-Net vs. 3D U-Net

When analyzing the qualitative results from comparing the 2D and 3D U-Net architectures it is clear that the 3D U-Net architecture is making predictions which are closer to the B-spline interpolation results. In Figure 58, when focusing on the radial component and magnitude, we see that the 2D U-Net erroneously smoothens the input data into something which closely resembles the input field, but the B-spline interpolation result shows that the "truth" is in fact almost a mirror image of what the model is predicting. The 3D U-Net, on the other hand, overcomes this issue and makes a prediction which more closely resembles the B-spline interpolation result. The lateral component is also underestimated more by the 2D U-Net, while the 3D U-Net prediction is closer in terms of correctly predicting the magnitude of the lateral component. Figure 59 shows a data sample which has some cut-offs in the upper half. The 3D U-Net manages to reconstruct the cut-off region and correctly fills in negative velocities in the radial component. It also correctly fills in positive velocities in the lateral component, albeit underestimated and noisy. The 2D U-Net does not manage to do fill in the missing data, as we can see that the cut-offs are left more or less untouched. We can see similar results in Figure 60 where there are severe cut-off effects. Here, the 2D U-Net does some smoothing of the region which contains data, but does not do anything to fill in the cut-off area around. The 3D U-Net prediction, on the other hand, contains information of the entire field. The prediction cannot be said to be very good for any of the components in this case, as both components are underestimated, but the model is at least trying to do something.

The performance difference in terms of cut-off reconstruction makes sense as the 3D U-Net should be better suited to reconstruct temporal information. Since the cut-off regions will vary with each time frame, reconstruction of these areas could be inferred from adjacent frames. The 2D U-Net cannot handle such temporal information and relies solely on the standalone static frame to perform reconstruction. We can tie this to the Navier-Stokes equations, as discussed in 2.2.2. In order to fully describe a flow system we need both the equations stated in 1 and 2. 2 only describes spatial relations, but 1 also includes a temporal gradient. If we choose to believe that the networks are learning features which somehow relate to the Navier-Stokes equations, then it is not sufficient giving the networks spatial information only. This seems to be confirmed by the results shown, as the 2D U-Net does not manage to reconstruct the missing areas, while the 3D U-Net does so to some extent. This also coincides with the findings by Bo et. al., as discussed in 2.4.6.

If we look at the quantitative results summarized in Figure 54, there does not seem to be any significant performance difference between the 2D and 3D model architectures, except for the In-silico base model, where the 3D U-Net clearly performs the best. The same can be seen in Figure 53, where the 2D U-Net only scores slightly higher than the 3D U-Net on MSE for each frame. The reason for this has likely to do with the limitations of the MSE metric. In the cut-off regions where the 2D U-Net predicts only zeros, the MSE score will not necessarily be higher than that of the 3D U-Net which is filling in velocity data in these regions. This is because if the filled in data is inaccurate or distorted, this might result in a high MSE score which is comparable to that of a MSE score computed from an empty region. Hence, on average, we can expect the MSE scores of from the empty regions and the filled in regions to not be too different from each other.

5.2.5 3D U-Net vs. Gaussian Smoothing Benchmark

In 5.2.4 it was shown that the 3D U-Net gives better results than the 2D U-Net as it allows for the use of temporal information, as well as spatial information. However, it is worth analyzing how the 3D U-Net results are in comparison to the Gaussian smoothing benchmark. The Gaussian smoothing uses a 3D kernel which, like the 3D U-Net, operates on both spatial and temporal dimensions. The results in Figures 61-63 show instances of the time frames predicted with the 3D U-Net, in comparison to the data smoothened with a 3D Gaussian kernel. In Figure 61 we see that the 3D U-Net reconstructs the radial component in a way that gives a sharper image than that of that of the Gaussian smoothing. In the lateral component the 3D U-Net prediction also more closely resembles the B-spline interpolation, as the field smoothened with a Gaussian kernel struggles with dampening the large region of negative velocities on the left side. Figure 62 shows a frame where the reconstruction of the radial component is very similar between the two approaches. The magnitude is slightly more accurately estimated by the 3D U-Net, but it struggles with the lateral component. The pattern estimated by Gaussian smoothing is distorted in comparison to the B-spline interpolation result, but it is closer than the 3D U-Net prediction in terms of predicting higher velocities. In Figure 63 we can observe that the radial component is better estimated by the 3D U-Net. We can also see that the lateral component is less blurry for the 3D U-Net prediction, however both the 3D U-Net prediction and the Gaussian smoothing result are somewhat underestimated and distorted. In this case, the 3D U-Net lateral prediction is more similar to the Gaussian smoothing result than the B-spline interpolation. This might be a sign that the filtering operations performed by the neural network are similar in character to the Gaussian smoothing filter, however providing a more detailed result than the Gaussian smoothing. By looking at the quantitative results in Figure 53(b) (for the Full in-vivo model) we can see that the 3D U-Net scores similarly well on mean-squared-difference from the B-spline results as the Gaussian smoothing. It tends to do better for the lateral component around frames 15-20, which correspond to diastolic relaxation. This may indicate that the neural network benefits from higher signal-to-noise ratios in comparison to the Gaussian smoothing, at least for the lateral component, as the velocity signal tends to be greater during diastole.

Figure 64 shows quiver plots of two frames reconstructed with B-spline interpolation, Gaussian smoothing and the 3D U-Net. In Figure 64(a) we can observe a vortex in the middle left of the ventricle. The vortex is visible in all the three fields shown, however it is highlighted more prominently in the 3D U-Net prediction. In this case, it is actually even more prominent than in the B-spline interpolation result. In Figure 64(b) we can observe a small vortex in the lower right part of the ventricle. This vortex is visible in both the B-spline interpolation result and in the 3D U-Net prediction, but not in the result from

Gaussian smoothing. It is apparent that the vortex is so small and low in magnitude that the Gaussian kernel smoothens it out completely, making an impression of laminar flow through the whole ventricle. This shows that the gain in reconstruction of sharp transitions in the velocity field which comes from applying the 3D U-Net, as opposed to Gaussian smoothing, can be beneficial for detecting turbulent flow patterns in the blood flow.

5.2.6 Smoothing the Inputs

It was mentioned in 5.2.3 that the in-vivo data set, especially the lateral component, might be too noisy for the neural network to handle. Hence it has a tendency to underestimate the lateral component, as this would be a minimal cost solution when erroneous predictions of large magnitude would cause a large MSE loss. Therefore it is of interest to see what happens if we smoothen the raw velocity fields to some extent before giving them to the neural network. Figure 65 shows the same sample as Figure 62, but this time with weakly smoothened input fields. Figure 66 shows the same sample, but with moderately smoothened input fields. In Figure 62 the lateral component is underestimated by the 3D U-Net, but Figure 65 shows that smoothing the input field before performing inference with the neural network does have a positive effect when it comes to reconstructing the velocity magnitude of the lateral component. When the model is additionally trained with a smoothened training set the added benefit is lost. This might be because too much information is lost in the training set as a result of the smoothing, hence making it harder to tune the model parameters based on the new training data. Figure 66 shows that when moderate smoothing is applied, we achieve similar results as with weak smoothing, only with lower magnitude. This shows that there is a trade-off between getting a decent lateral estimate and underestimating the magnitude when increasing the severity of smoothing of the input fields. If we compare the results of smoothened test data in Figures 65 and 66 to the results of Gaussian smoothing in Figure 62 we can see that the model predictions are in fact more similar to the Gaussian smoothing results than the B-spline interpolation results. This shows that even though smoothing the input velocity fields seems to increase reconstruction performance of the lateral component, it is not necessarily better than the Gaussian smoothing benchmark in this regard. However, with the added benefit of better reconstruction of the radial component, as discussed in 5.2.5, the approach of using the proposed neural network model should grant an overall benefit over the Gaussian smoothing benchmark.

Figures 67 and 68 show quiver plots of two samples where the effect of smoothing the input velocity fields is visualized. In Figure 67 the lateral magnitude of the vortex in the lower left part of the ventricle is enhanced when weakly smoothing the inputs, giving a further improvement of the visualization of the vortex. However, when moderate smoothing is applied, we can observe that the radial magnitude is dampened in comparison to the model which has no input smoothing, giving a poorer visualization of the vortex. In Figure 68 the small vortex in the lower right part of the ventricle is less visible when weak input smoothing is applied, and it disappears completely when moderate input smoothing is applied. This again shows that smoothing the velocity fields before performing inference with the neural network can have a positive effect on reconstructing the lateral component, but the technique is also prone to dampening important information in the fields.

5.2.7 Compression in Three Dimensions

As discussed in 5.1.3 there was an added benefit when using a 3D U-Net architecture with pooling and transposed convolutions (i.e. compression and decompression) in both spatial and the temporal dimension, in terms of a smaller MSE loss for the in-silico data set. Figures 69-71 show three samples comparing the model architectures with and without compression and decompression in the temporal dimension. We cannot observe any notable difference between the model predictions in any of the samples shown. Figure 72 shows the mean-squared-difference from the B-spline interpolation results, averaged over all samples in the test set. We cannot see any significant performance difference from these graphs either, as both the loss curves of each model variation are very similar. Hence it seems like there is no improvement in predictive performance of changing the model architecture in the proposed way. This differs from the in-silico results discussed in 5.1.3. However, from the in-silico results there were not seen any clear performance difference in the quality of the reconstructed fields either, there was only a quantitative improvement. It was suggested that this improvement might come from a more accurate temporal estimation, which is not visible frame by frame. As the MSE loss is overall significantly higher for the in-vivo data than for the in-silico data, this might mean that such a performance difference we saw in the in-silico data is too small relative to the overall observable loss in the in-vivo data.

5.3 GAN Results Discussion

This section discusses the results shown in section 4.3.

5.3.1 In-Silico Data Reconstruction

As discussed in 5.2.4, one issue which might limit the performance of the proposed neural network is the loss function. It was suggested that a reason the model tends to underestimate the reconstruction of noisy fields is that the optimal action when an accurate reconstruction is difficult to achieve is to output low velocities, as this is likely to make the MSE loss lower than predicting erroneous high velocities. There is of course a possibility to change the loss function into something else, like the mean-absolute-error loss. The problem would however still persist as the loss is still being computed pixel-wise, hence being prone underachieve when exposed to distorted predictions. By introducing the GAN this problem could possibly solve this problem, as the GAN is trained in a completely different way. Through optimization of the adversarial loss, the generator network, which performs the reconstruction, is trained specifically to create realistic data, with no explicit way of being told how evaluate ones prediction as realistic.

Figures 73 and 74 show two reconstruction predictions of a GAN generator, in comparison to the same network trained with a MSE loss. It is clear that the GAN predictions are very wrong, which is confirmed by looking at the MSE loss curve shown in Figure 75. However, it is interesting to note that even though the predictions are clearly wrong, they do have characteristics of which can seem realistic. The GAN prediction shown in Figure 73 is not as accurate as the direct prediction with MSE loss, as it overestimates the vorticity, giving a wrong impression of the flow dynamics. Figure 74 shows that the GAN is predicting a flow moving clockwise, while the ground truth is in fact a counter-clockwise flow. However, both samples show predictions which at first glance seem realistic, as they capture known flow characteristics, such as vorticity. This shows that the GAN is able to generate flow fields which contain the dynamics which are needed to describe turbulent flow patterns. It seems like the GAN is able to learn and reproduce such patterns, but at it is not able to relate them well enough to the inputs. Figure 76 shows the training history for the GAN. We can see that both the accuracy and loss metrics are oscillating a lot, which indicate that the model is struggling to converge to a Nash equilibrium. This shows that the model is not learning appropriately from the data, and that there is potential for improvement. If the model were to be improved such that the prediction errors were corrected, then it might be able to handle the in-vivo data better as it could avoid making underestimated predictions for substantially noisy data. The GAN shows itself to be able to make quite bold predictions, but whether or not it is able to make bold predictions which are of high quality needs further investigation. Possible improvements of the GAN are discussed in 5.4.2.

5.3.2 Model Verification using the MNIST Handwritten Numbers

As the GAN approach did not provide satisfactory results on the in-silico data it is of interest to see how it performs on an easier problem, such as denoising the MNIST handwritten numbers data set. From Figure 77 we can see that the GAN performs at least as well as just the generator trained directly with a MSE loss function on an equal amount of data. Figure 78 shows the training history for the MNIST data set. In comparison to the training history for the ultrasound data set, shown in Figure 76, we can see that the training phase of the GAN is more stable with the MNIST data set. The loss and accuracy curves oscillate less, and they seem closer to reach convergence. The directly trained network has a tendency to leave big artifacts in the reconstructed image, most prominent in Sample 1. The generator trained with an adversarial loss avoids this problem, but instead has a tendency to underestimate the contours of the numbers. It is hard to tell which one of the models is achieving the best, but this example yet again illustrates how the two models seem to perform the task differently, thus yielding quite different looking results. This different approach the GAN is taking might in the end be beneficial for this denoising problem, as well as the ultrasound data reconstruction problem, if the method is further enhanced to the specific problems. Both these problems are qualitative, in the way that the achieved performance can only truly be appreciated by individuals observing the results visually.

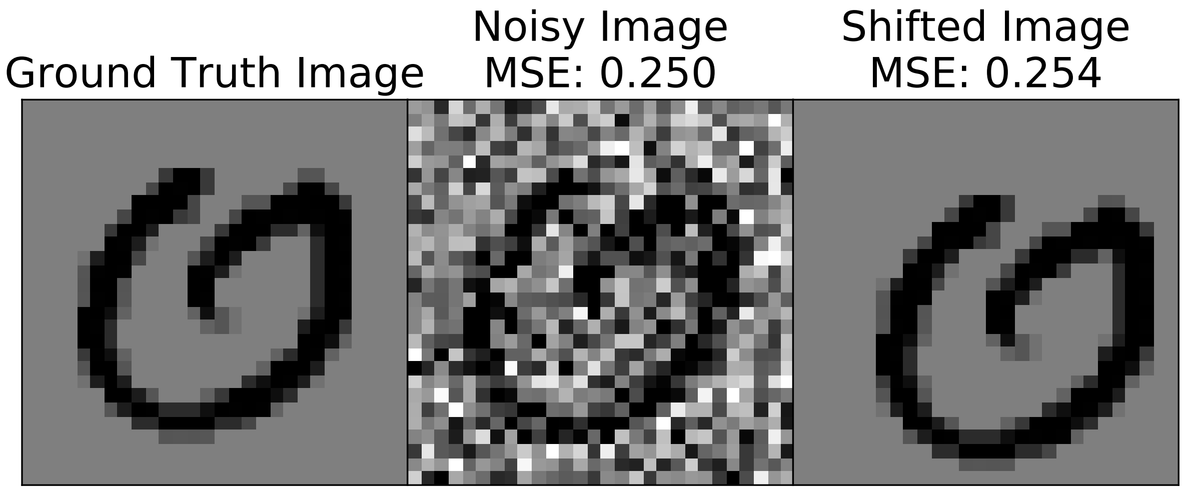
There is no single metric which properly can capture the performance of image reconstruction. This explains why in recent years we have seen that GANs vastly outperform other ML and non-ML based methods in image processing. [50] The GAN is trained in a way which does not give the models explicit knowledge about why they are wrong or right, instead the models try to learn this evaluation metric themselves directly from the data. This tends to be beneficial for image reconstruction, as opposed to using an explicit performance metric like the MSE. Figure 79 illustrates the limits of the MSE metric. For both samples shown, the noisy images score better than the shifted images using the MSE metric, even though it is clear that the shifted images are preferable in terms of interpreting the numbers. This is because the MSE evaluates the error pixel for pixel. For 79(a) the number in the shifted image are shifted in the direction of the lower right corner in such a way that the contours of the shifted number are overlaid in-between the contours of the number in the ground truth image. This gives a large reconstruction error for the pixels in the number contours, which ultimately results in a larger error than the error originating from all the noise in the noisy image. In the case of the ultrasound data, some distortion of the velocity fields in space or time would not necessarily be detrimental for the clinician's interpretation of the data, but it could have a large impact on the MSE, or other similar metrics.

5.4 Further work

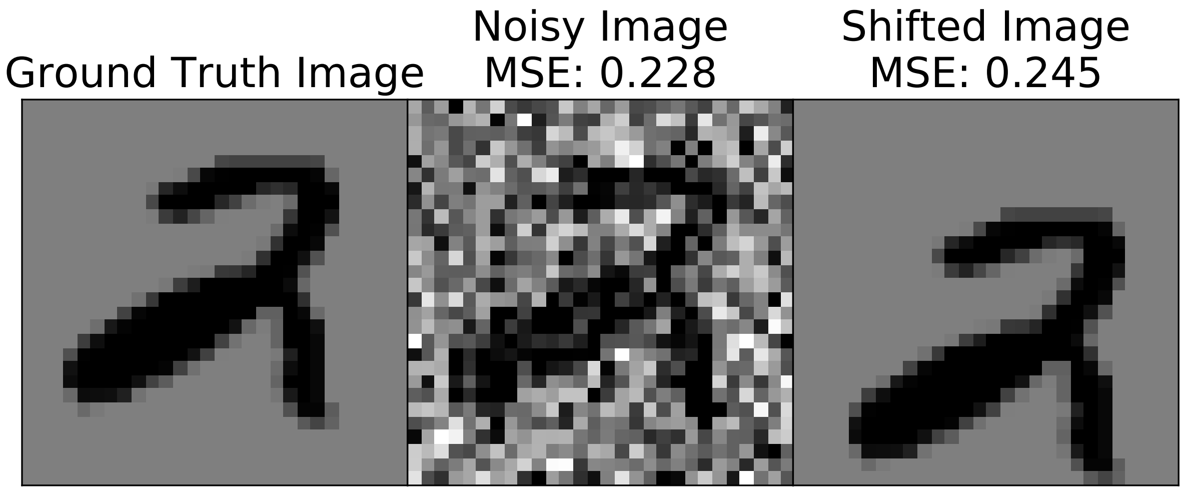
5.4.1 A Better In-silico Data Set

One improvement would be to increase the realism in the in-silico data set. In 5.2 we saw that the transfer from the source domain task to the target domain task was difficult because of the differences between the domains, especially the reduction in lateral quality and the introduction of missing data from aliasing. The in-silico data set was generated by using BST in all spatial directions. This was done because when random rotation and slicing is performed, the information about which component corresponds to the beam direction is lost. Therefore it would not make sense using doppler measurements on the rotated FUSK data, as the beam direction used in the doppler measurement will not correspond to the true radial direction in the rotated FUSK data.

One solution could be to simulate aliasing effects by introducing cut-offs to the data which would be similar to the missing data regions generated by the antialiasing algorithm. This would have to be done intelligently to produce realistic aliasing effects, but could ultimately benefit the information transfer to the in-vivo target domain and improve the performance of reconstructing missing data regions. Another way to further close the gap between the source and target domain would be to limit the sample space of possible rotations to configurations which would result in data closer in resemblance to a left ventricle. Per now, the in-silico data set contains configurations which give velocity fields far from the ones we would expect to see in real ultrasound measurements form the left ventricle. This is because slices that would correspond to realistic beam directions are equally likely as those which would not be possible to achieve with an ultrasound examination of a patient. Figure 80 shows some samples of unrealistic slices. These slices still show realistic flow fields, and may resemble flows in other areas, such as in arteries. However, it might be better to limit the use of the algorithm to do intraventricular flow reconstruction initially before trying to make a more general model. Limiting the rotation of the FUSK object might also make it easier to incorporate doppler measurements in the VFI simulation. If the radial axis is



(a) Sample 1



(b) Sample 2

Figure 79: Two samples from the MNIST handwritten numbers data set. Each sample is modified in two different ways: adding white Gaussian noise and shifting it randomly by a few pixels. The MSE is computed between the modified images and the ground truth. Even though the numbers in the shifted images are clearly more interpretable than in the noisy ones, the MSE scores for the noisy images are lower.

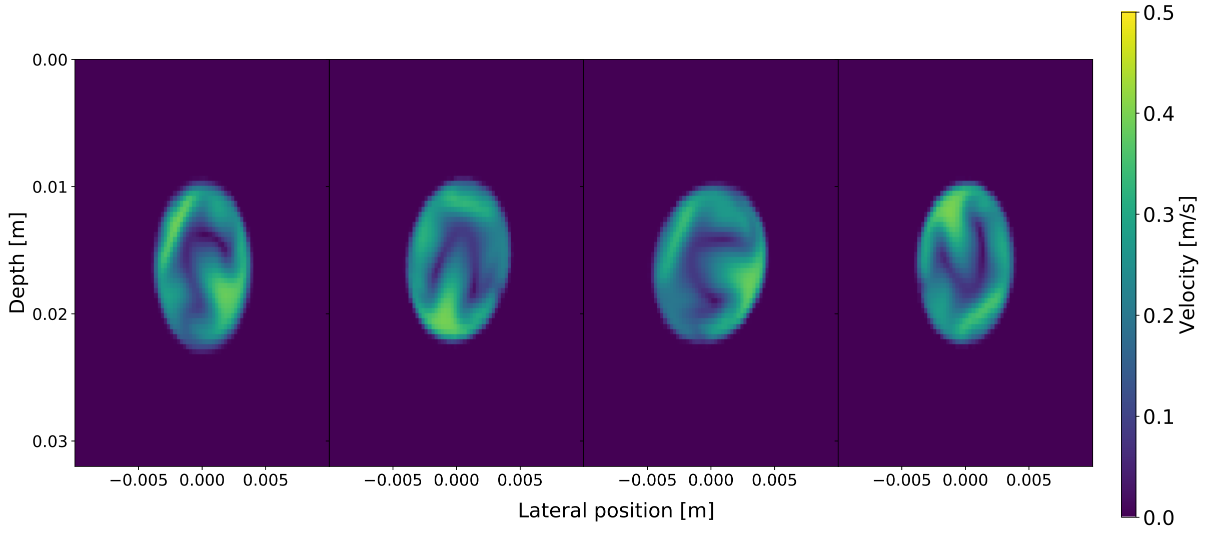


Figure 80: Four samples of the CFD phantom ground truth velocity magnitude from the in-silico data set. All these slices are far from apical, and have little resemblance to the left ventricle.

sufficiently static during the rotation, then the offset between the beam direction of the doppler measurement and the radial component of the data object might be small enough for it to make sense to perform doppler measurements. This could allow us to use the more realistic configuration of doppler measurement in combination with BST, as opposed to just BST. Finally, it is important to note that deep neural networks always benefit from more training data, hence combining different simulations and VFI techniques into a large data set should give an increase in performance.

5.4.2 Regularization of the Generator Loss Function

The GAN shows itself capable of producing realistic looking fields, but they do not look like the ground truth fields corresponding to the generator's inputs. First of all, the GAN is not converging properly, which is a problem that could be resolved with further hyperparameter tuning and improvements in the training procedure and model architectures. It could benefit from a proper systematic hyperparameter tuning, as it is very sensitive to small changes in the hyperparameters. In addition, there is a lot of potential for trying different improvements which are suggested by Salimans et. al. [51].

However, the main challenge with the GAN seems to be that the generator's predictions differ too much from the ground truth, as it is making drastic changes to the input data. This is because of the way it is trained: when the generator is trained its prediction is evaluated by the discriminator. The discriminator does not know anything about the ground truth corresponding to the generator's prediction, and bases its evaluation only on the data it has previously been trained on. Hence it will be able to judge whether or not a prediction seems real or not, but not if the prediction is related to the input image. This is instead left to the generator, which we hope does not change the input data too much. However, we see that this is exactly what it is doing, being able to turn the velocity field around completely as seen in Figure 74. We might be able to mitigate this problem by regularization, as shown in Figure 81. This involves adding a reconstruction loss term to the loss function, for instance the MSE loss. This would still make the discriminator enforce generation of realistic looking fields, but also penalize the generator if its predictions deviate too much from the corresponding inputs.

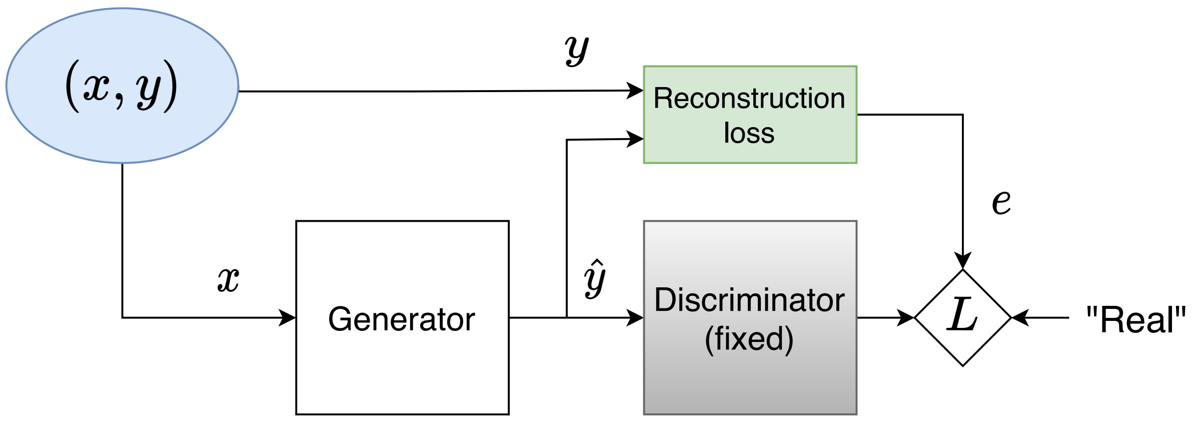


Figure 81: Proposed change in the generator's training procedure where regularization is applied by adding a reconstruction loss cost e to the loss function L. This Figure should be seen in relation to Figure 22 in 3.7.2.

5.4.3 U-Net with Prior Regularization

All of the models proposed in this thesis are based entirely on machine learning. This is done under the assumption that all relevant information can be extracted directly from the data, however this might be too ambitious. Instead, we might take inspiration from the way deep learning has recently been used in *optical flow* (OF). OF is a video processing technique which aims to estimate the velocity of each pixel in the video frames. Xiang et. al. [59] used a CNN structure similar to the U-Net called the *FlowNet*, with the motivation of speeding up velocity estimation as compared to traditional methods. In the paper they also state that similar attempts to train the FlowNet have overemphasized the power of the model, and neglected the advancements in OF prior to the introduction of Deep Learning. In their proposed model, they insert handcrafted algorithms into the networks loss function, which act as regularization terms based on prior knowledge. We can take inspiration from this and incorporate some of the state of the art techniques described in 2.3.5 into the model. An idea to a modification of the U-Net for reconstruction of blood flow measurements is shown in Figure 82. The architecture is based on the FlowNet structure proposed by Xiang et. al. This new architecture differs from the one previously discussed in this thesis as it introduces two modifications:

- **Multistage loss function.** The multistage loss function computes the total loss as a weighted sum of losses from different compression levels. This makes the model less vulnerable to small prediction errors at the output layer and instead makes it better at capturing the overall features in the flow.
- **Loss functions with physical regularization terms.** The loss functions would not only ensure a good data fit, but they could also impose constraints from fluid dynamics through regularization. This enforces the model to provide more realistic field predictions, and helps fill in the information gap in areas with missing data.

When Xiang et. al. imposed a similar architecture using regularization priors they showed that this produced more accurate flow fields while still improving the computation time as compared to the standard methods. We might see similar improvements when using this approach for the problem at hand. We have already discussed that the model particularly struggles with reconstructing areas with missing data. We also discussed that the loss function was a limitation on performance as it does not capture the bigger picture of the flow, but only evaluates each data point individually. The physical regularization may also benefit the GAN approach in a similar way by including extra regularization terms to the reconstruction loss. In conclusion, although the model proposed in this thesis poses some

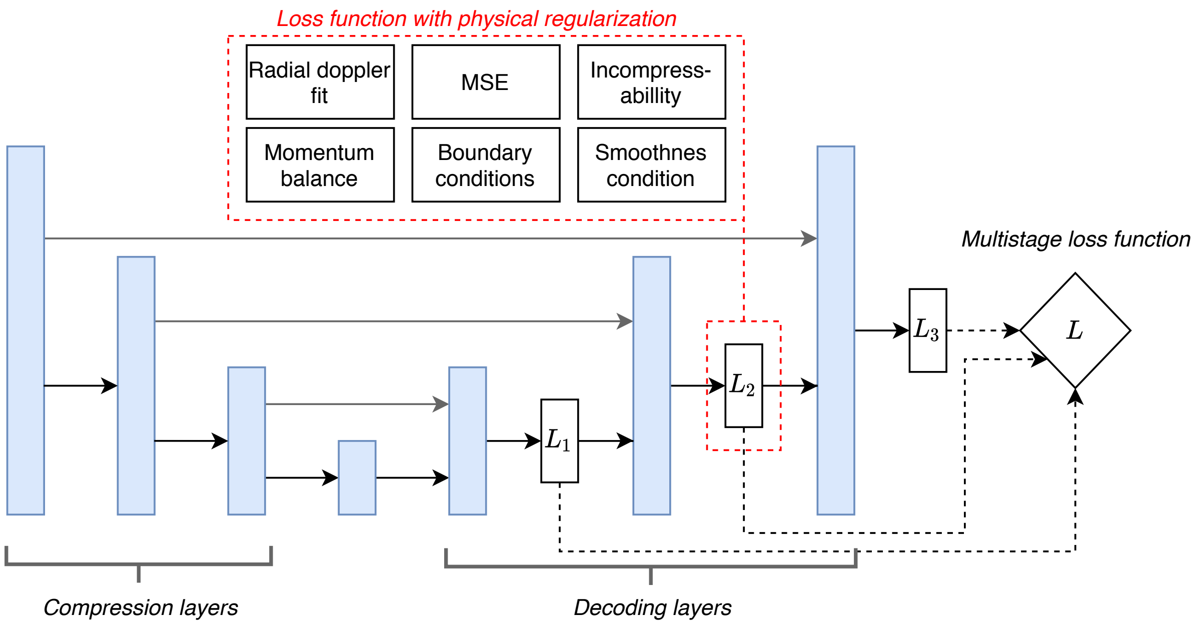


Figure 82: A possible improvement of the U-Net architecture for blood velocity field reconstruction. The architecture introduces a multistage regularized loss function, as compared to the architecture described in 3.6.1. The regularization terms are examples of physical constraints one could impose on the model. The conceptual architecture is inspired by Figure 2 from [59].

challenges, there are still ways of improving it. Given some more development, we might uncover the full potential of the CNN for blood velocity field reconstruction.

6 Conclusion

In this thesis, a Deep Learning based method for the reconstruction of VFI measurements is proposed. We firstly train a residual CNN based on the U-Net for reconstruction of simulated ultrasound blood flow measurements. We then use this pre-trained model to do transfer learning on a dataset of real ultrasound blood flow measurements in the left ventricle of pediatric patients. Different variants of the model are compared, most notably one trained on static field frames using 2D convolutions (2D U-Net) and one trained on time series of field images using 3D convolutions (3D U-Net). It is shown that both models perform substantially better than a Gaussian smoothing benchmark on the simulated data, as both models are able to recreate higher order dynamics in the flow which are lost when applying Gaussian smoothing. When tested on real data, the 2D U-Net struggles with reconstructing areas with missing data. As these areas vary over time, the 3D U-Net performs much better and can fill in the missing areas. This shows that including temporal information in the model makes the model better at reconstructing properties in the flow, which coincides with our fundamental understanding that temporal information is needed to fully describe a flow system. Additionally, the 3D U-Net is able to reconstruct flow fields with higher resolution than the Gaussian smoothing benchmark, ultimately being beneficial for visualizing important characteristics in the flow. However, the model struggles with producing a good estimate of the lateral component, as it is biased to predict underestimated measurements. This is shown to be related to the fact that the real measurements are more noisy than the simulated training data, as smoothing the real data to some extent before giving it to the model strengthens the lateral estimate, but can also weaken the overall velocity magnitude if applied too generously. The problem is also postulated to be related to the restrictions imposed by the MMSE loss function, which can be unforgiving to displacement and distortion errors. Hence a GAN based on the same U-Net model is developed to see if the introduction of an implicit loss function can increase the predictive capabilities of the model. However, as GANs are notoriously difficult to train, no definitive answer is found as the GAN is not able perform reasonable reconstruction of the data. Although the GAN fails at the task, it is able to create fields which look realistic when viewed unrelated to the input data. It is suspected that given more fine tuning and development, the GAN approach could potentially achieve satisfactory results and maybe improve the problem of underestimation. Overall it seems like CNN models have a lot of potential for fast and accurate reconstruction of VFI measurements, and should be researched further in order to better overcome the challenges which are introduced when using this approach.

References

[1] Haidong Wang et al. "Global, regional, and national life expectancy, all-cause mortality, and cause-specific mortality for 249 causes of death, 1980–2015: a systematic analysis for the Global Burden of Disease Study 2015". eng. In: *The Lancet* 388.10053 (2016), pp. 1459–1544. ISSN: 0140-6736.

[2] Henry Mcgill C., C Mcmahan Alex, and Samuel Gidding S. "Preventing Heart Disease in the 21st Century: Implications of the Pathobiological Determinants of Atherosclerosis in Youth (PDAY) Study". In: *Circulation* 117.9 (2008), pp. 1216–1227. ISSN: 0009-7322.

[3] Martin J O'Donnell et al. "Global and regional effects of potentially modifiable risk factors associated with acute stroke in 32 countries (INTERSTROKE): a case-control study". eng. In: *The Lancet* 388.10046 (2016), pp. 761–775. ISSN: 0140-6736.

[4] Gianni Pedrizzetti et al. "The Vortex — an Early Predictor of Cardiovascular Outcome?" In: *Nature Reviews Cardiology* 11.9 (2014), pp. 545–553.

[5] Takeji Saito et al. "VISUALIZATION OF FLOW DYNAMICS FROM PULMONARY VEINS TO LEFT ATRIUM AND LEFT VENTRICLE USING PHASE-RESOLVED 3D CINE PHASE CONTRAST MRI (4D-FLOW)". eng. In: *Journal of the American College of Cardiology* 61.10 (2013), E951–E951. ISSN: 0735-1097.

[6] Peter Lam. *What to know about MRI scans.* URL: `https://www.medicalnewstoday.com/articles/146309` (visited on 06/09/2020).

[7] *Cardiac CT, Ultrasound, MRI and More: Which Tests Are Used When for CAD?* URL: `https://www.gehealthcare.com/feature-article/cardiac-ct-ultrasound-mri-and-more-which-tests-are-used-when-for-cad` (visited on 06/09/2020).

[8] *How Much Does an Ultrasound Machine Cost? (2020).* 2020. URL: `https://lbnmedical.com/ultrasound-price-guide/` (visited on 06/07/2020).

[9] Lacie Glover. *Why does an MRI Cost So Darn Much?* URL: `https://money.com/why-does-mri-cost-so-much/` (visited on 06/07/2020).

[10] *VScan Extend Handheld Ultrasound.* URL: `https://handheldultrasound.gehealthcare.com/vscan-extend/#overview` (visited on 06/09/2020).

[11] *Vivid E95 Premium 4D Cardiac Ultrasound System.* URL: `https://www.gehealthcare.com/products/ultrasound/vivid/vivid-e95` (visited on 06/09/2020).

[12] M Wigen et al. "4-D Intracardiac Ultrasound Vector Flow Imaging-Feasibility and Comparison to Phase-Contrast MRI". In: *IEEE Transactions on Medical Imaging* 37.12 (2018), pp. 2619–29.

[13] Kondo Claude Assi et al. "Intraventricular vector flow mapping—a doppler-based regularized problem with automatic model selection". eng. In: *Physics in Medicine and Biology* 62.17 (2017), pp. 7131–7147. ISSN: 0031-9155.

[14] Tim Appenzeller. "The AI revolution in science". In: *Science* (2017). ISSN: 0036-8075.

[15] N Kriegeskorte. “Deep neural networks: a new framework for modeling biological vision and brain information processing”. English. In: *Perception* 45.s2 (2016), pp. 73–73. ISSN: 0301-0066.

[16] Erik Smistad. *Medical image segmentation for improved surgical navigation.* eng. Trondheim, 2015.

[17] Andrew Gilbert et al. “Automated Left Ventricle Dimension Measurement in 2D Cardiac Ultrasound via an Anatomically Meaningful CNN Approach”. eng. In: *arXiv.org* 11798 (2019). ISSN: 03029743. URL: `http://search.proquest.com/docview/2312686187/`.

[18] Ortal Senouf et al. “High Frame-Rate Cardiac Ultrasound Imaging with Deep Learning”. In: *Lecture Notes in Computer Science* 11070 (2018).

[19] Joseph Woo. “A short History of the development of Ultrasound in Obstetrics and Gynecology”. In: (2006). URL: `http://www.ob-ultrasound.net/history1.html` (visited on 12/11/2019).

[20] Bjørn Angelsen. *Medisinsk museum: PEDOF.* Norwegian. URL: `https://www.ntnu.no/medisinskmuseum/ultralyd/pedof` (visited on 05/30/2020).

[21] *Medisinsk museum: Hjertebank.* Norwegian. URL: `https://www.ntnu.no/medisinskmuseum/ultralyd/hjertebank` (visited on 05/30/2020).

[22] Ingvild Kinn Ekroll et al. “Combined Vector Velocity and Spectral Doppler Imaging for Improved Imaging of Complex Blood Flow in the Carotid Arteries”. English. In: *Ultrasound in Medicine & Biology* 40.7 (2014), p. 1629. ISSN: 0301-5629.

[23] LN Bohs et al. “Speckle tracking for multi-dimensional flow estimation”. In: *Ultrasonics* 38.1 (2000), pp. 369–375.

[24] Thor Amund Hagen. *Snart kan ultralyd bli like vanlig som stetoskopet hos fastlegen.* Norwegian. 2020. URL: `https://www.nrk.no/trondelag/snart-kan-ultralyd-bli-like-vanlig-som-stetoskopet-hos-fastlegen-1.14882636` (visited on 05/30/2020).

[25] Lindsay M. Biga et al. *Anatomy & Physiology: Heart Anatomy.* URL: `https://open.oregonstate.education/aandp/chapter/19-1-heart-anatomy/` (visited on 05/29/2020).

[26] Lindsay M. Biga et al. *Anatomy & Physiology: Cardiac Cycle.* URL: `https://open.oregonstate.education/aandp/chapter/19-3-cardiac-cycle/` (visited on 05/29/2020).

[27] Joris Van Cauwenberge et al. “Assessing the performance of ultrafast vector flow imaging in the neonatal heart via multiphysics modeling and In vitro experiments”. eng. In: *IEEE TRANSACTIONS ON ULTRASONICS FERROELECTRICS AND FREQUENCY CONTROL* (2016). ISSN: 0885-3010. URL: `https://biblio.ugent.be/publication/8508295`.

[28] Sigurd Wifstad, Thomas Grønli, and Lasse Løvstakken. “Reconstruction of Ultrasound Blood Velocity Fields based on Deep Learning (Project thesis)”. 2019.

[29] Yunus A. Cengel and John M. Cimbala. *Fluid Mechanics - Fundamentals and Applications.* 3rd ed. McGraw Hill Education (India) Private Limited, 2014.

[30] TG Bjåstad. “High frame rate ultrasound imaging using parallel beamforming”. PhD thesis. NTNU, 2009.

[31] Hans Trop. “Clutter Rejection Filters in Color Flow Imaging: A Theoretical Approach”. In: *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control* 44 (1997).

[32] Sverre Holm. “Medisinsk ultralydavbildning”. In: *Fysikkens verden* 1 (1999).

[33] Linda G Shapiro. *Computer vision.* eng. Upper Saddle River, N.J, 2001.

[34] Hartmut Prautzsch. *Bézier and B-spline techniques.* eng. Berlin, Germany ; 2002.

[35] Alberto Gomez et al. “4D Blood Flow Reconstruction Over the Entire Ventricle From Wall Motion and Blood Velocity Derived From Ultrasound Data”. eng. In: *IEEE Transactions on Medical Imaging* 34.11 (2015), pp. 2298–2308. ISSN: 0278-0062.

[36] Thomas Grønli et al. “A fast 4D B-spline framework for model-based reconstruction and regularization in vector flow imaging”. In: *IEEE Ultrasonics Symposium (IUS)* (2018).

[37] Martín Abadi et al. "TensorFlow: A system for large-scale machine learning". eng. In: *arXiv.org* (2016). ISSN: 2331-8422. URL: `http://search.proquest.com/docview/2079694224/`.

[38] Kurt Hornik. "Approximation capabilities of multilayer feedforward networks". eng. In: *Neural Networks* 4.2 (1991), pp. 251–257. ISSN: 0893-6080.

[39] Michael A. Nielsen. *Neural Networks and Deep Learning.* Determination Press, 2015.

[40] Rafael C. Gonzales and Richard E. Woods. *Digital Image Processing.* 4th ed. Pearson, 2017.

[41] *CS231n COnvolutional Neural Networks for Visual Recognition.* URL: `https://cs231n.github.io/neural-networks-1/#actfun` (visited on 06/01/2020).

[42] Djork-Arné Clever, Thomas Unterthiner, and Sepp Hochreiter. "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)". In: *Arxiv.org* (2015).

[43] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *Arxiv.org* (2015).

[44] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". eng. In: *arXiv.org* (2015). ISSN: 2331-8422. URL: `http://search.proquest.com/docview/2081710836/`.

[45] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation". In: *Lecture Notes in Computer Science (including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 9351 (2015), pp. 234–241.

[46] D Hwang et al. "Improving the Accuracy of Simultaneously Reconstructed Activity and Attenuation Maps Using Deep Learning". In: *Journal Of Nuclear Medicine* 59.10 (2018), pp. 1624–1629.

[47] Siming Yan et al. "Calcium Removal From Cardiac CT Images Using Deep Convolutional Neural Network". In: *Arxiv.org* (2018).

[48] Sebastian Ruder. "An overview of gradient descent optimization algorithms". eng. In: *arXiv.org* (2017). URL: `http://search.proquest.com/docview/2076187906/`.

[49] C. Tan et al. "A survey on deep transfer learning". In: vol. 11141. Springer Verlag, 2018, pp. 270–279. ISBN: 9783030014230.

[50] I.J. Goodfellow et al. "Generative adversarial nets". In: vol. 3. January. Neural information processing systems foundation, 2014, pp. 2672–2680.

[51] Tim Salimans et al. "Improved Techniques for Training GANs". eng. In: *arXiv.org* (2016). URL: `http://search.proquest.com/docview/2079145728/`.

[52] Tero Karras, Samuli Laine, and Timo Aila. "A Style-Based Generator Architecture for Generative Adversarial Networks". In: (2018).

[53] Alec Radford, Luke Metz, and Soumith Chintala. "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks". eng. In: *arXiv.org* (2016). ISSN: 2331-8422. URL: `http://search.proquest.com/docview/2078303239/`.

[54] Bo Liu, Haibo Huang, and Lu Xi-Yun. "Deep learning methods for super-resolution reconstruction of turbulent flows". eng. In: *Physics of Fluids* 32.2 (2020). ISSN: 1070-6631. URL: `http://search.proquest.com/docview/2353777310/`.

[55] Tianyuan Liu et al. "Deep Learning for Nanofluid Field Reconstruction in Experimental Analysis". eng. In: *IEEE Access* 8 (2020), pp. 64692–64706. ISSN: 2169-3536.

[56] Sangseung Lee and Donghyun You. "Prediction of laminar vortex shedding over a cylinder using deep learning". eng. In: *arXiv.org* (2017). ISSN: 2331-8422. URL: `http://search.proquest.com/docview/2076873667/`.

[57] T Hergum et al. "Fast Ultrasound Imaging Simulation in K-space". In: *IEEE Trans Ultrason Ferroelectr Freq Control* 6.56 (2009), pp. 1159–67.

[58] Yann LeCun, Corinna Cortes, and Christopher J.C. Burges. *THE MNIST DATABASE of handwritten digits.* URL: `http://yann.lecun.com/exdb/mnist/` (visited on 06/04/2020).

[59] Xuezhi Xiang et al. “Deep Optical Flow Supervised Learning With Prior Assumptions”. eng. In: *IEEE Access* 6.99 (2018), pp. 43222–43232. ISSN: 2169-3536.

[60] Adrian Rosebrock. *Why is my validation loss lower than my training loss?* URL: `https://www.pyimagesearch.com/2019/10/14/why-is-my-validation-loss-lower-than-my-training-loss` (visited on 11/29/2019).

Table 4: FUSK configuration.

Parameter	**Value**
Center frequency f_0	4.5 MHz
Pulse bandwidth	0.3 MHz
Focal point	18.5 mm
Aperture (Tx and Rx)	19 mm $\times$ 16 mm
Apodization (Tx and Rx)	Tukey window
Antialiasing filter window	Hamming
Antialiasing filter length	3

Table 5: Speckle tracking configuration.

	In-silico	**In-vivo**
Density (x,y,z) [points/m]	2000, 2000, 2000	2000, (0), 2000
ROI origo (x,y,z) [m]	-0.01, -0.01, -0.0185	(From raw data)
ROI extent (x,y,z) [m]	0.02, 0.02, 0.037	(From raw data)
Tracking direction	Forward	Two-way estimate
Correlation estimator	SSD	SSD
v_{max} (x,y,z) [m/s]	1, 1, 1	1, (0), 1.5
v_{min} (x,y,z) [m/s]	0.3, 0.3, 0.3	0.3, (0), 0.3
Block match kernel size (x,y,z) [mm]	1, 1, 1	7, (0), 7
Block match kernel dimensions	10, 10, 10	10, (0), 10
Doppler method	None	Doppler + Dealiasing
Doppler kernel size (x,y,z) [mm]	None	7, (0), 7
Doppler kernel dimensions	None	10, (0), 10
Dealiasing limit (0,1)	None	0.85

A FUSK configuration

Table 4 shows the most relevant parameters used in FUSK.

B Blood speckle tracking configuration

Table 5 shows the configuration used for the tracking performed on the in-silico and the in-vivo data.

C Explanation of bias in training curves

In the loss curves presented in this thesis we see that the validation loss is always less than the training loss. At first glance this seems wrong as the model is optimizing its parameters to the training set and not the validation set. Hence we would expect the training loss to be lower than the validation loss, not vice versa. There may be several reasons to why the validation loss is lower than the training loss. The most apparent reason is that when training a network using regularization, such as dropout layers, the training loss will be larger than it would otherwise. Since regularization is not applied on the validation set, the validation loss will tend to be lower, as long as the model has not overfit to the data. Additionally, training loss is computed during each epoch while validation loss is computed after each epoch. This means that on average the validation loss will be computed 1/2 epoch after the training loss [60]. Figure 83 shows the training history before and after these effects are corrected. Here we see that when the corrections are made the results indeed look more reasonable. Note that the validation loss is not bigger than the training loss after the corrections. This is because the nature of the training and validation sets are very similar as they are both generated in the same way.

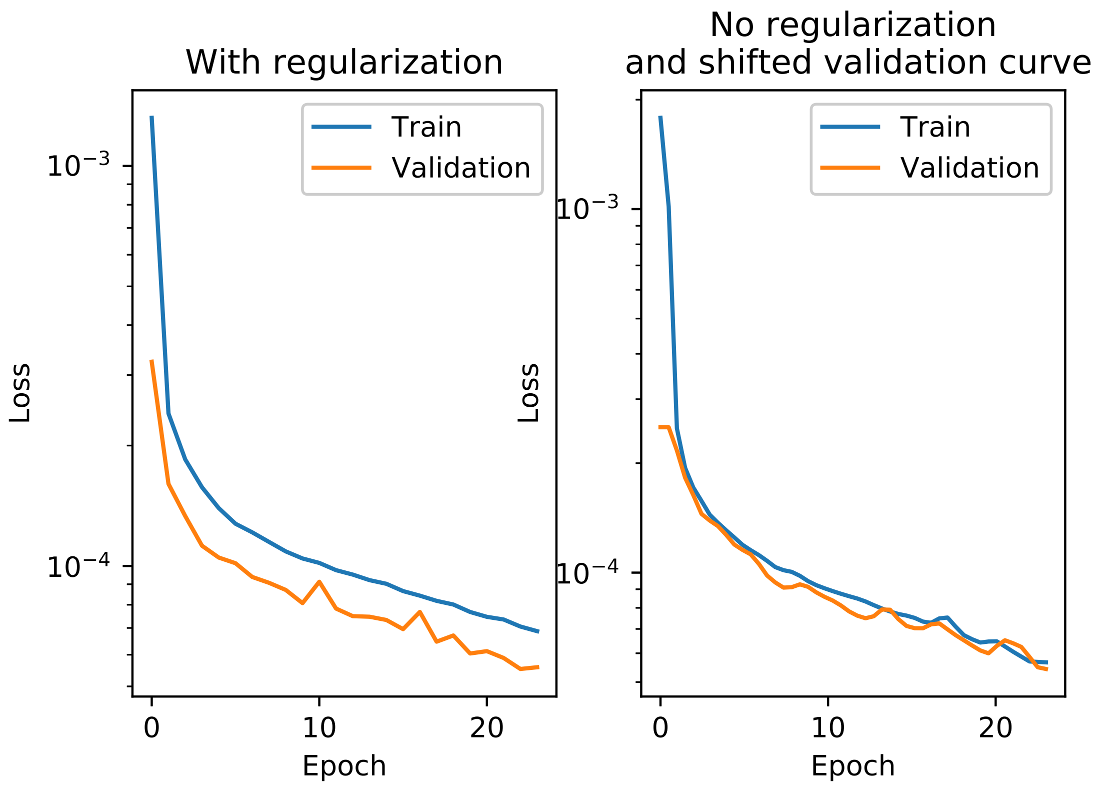


Figure 83: Training history for the U-Net trained on simulated ultrasound data. The plot on the left shows training with dropout regularization. The plot on the right shows training without regularization, and with the validation curve shifted 1/2 epoch to the right.

Sigurd Vangen Wifstad

Graduate thesis June 2020

NTNU
Norwegian University of
Science and Technology