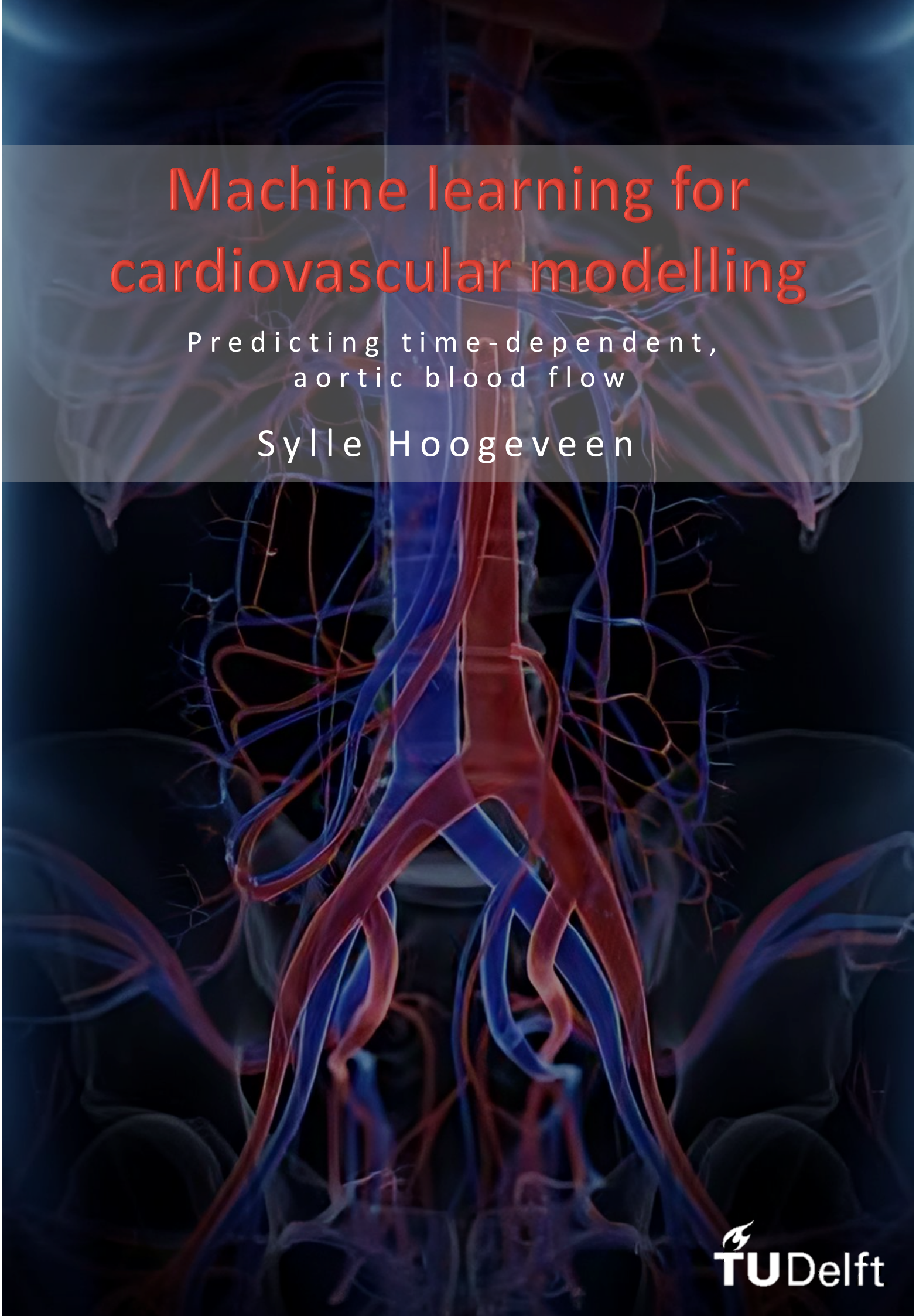


Machine learning for cardiovascular modelling

Predicting time-dependent, aortic blood flow

Sylle Hoogeveen

TU Delft

Machine learning for cardiovascular modelling

Predicting time-dependent, aortic blood flow

by

Sylle Hoogeveen

4445082

Thesis project
MSc Applied Mathematics
Computational Science and Engineering
Numerical Analysis group

Supervisor

Dr. A. Heinlein

Committee Members

Dr. M. Möller
prof. dr. ir. A.W. Heemink

To be publicly defended on Thursday October 27th at 16:00

Front image from Frankel Cardiovascular Center, University of Michigan Health
An electronic version of this thesis is available at http://repository.tudelft.nl/.

TU Delft Delft University of Technology

Faculty of Electrical Engineering, Mathematics and Computer Science

Abstract

The increase in complexity of mathematical models in an attempt to approximate reality and desire to have near real-time results have emphasized the need for fast numerical simulations. Especially in areas where classic numerical methods struggle to produce valid solutions in reasonable computational time due to their complex behaviour on multiple temporal and spatial scales, such as (cardio-vascular) fluid modelling [1], machine learning techniques can be of help. The aim of this study is to construct a single reduced order model, based on neural networks, for time-dependent incompressible blood flow through the aorta that can account for varying velocity inlet conditions, material parameters and geometries (computational domains). The objectives are to minimize the reduction of accuracy of simulated time series with OpenFoam [2], and to obtain speedup compared to the OpenFoam simulations.

In this study, aortic blood flow during one heartbeat is modelled by Navier-Stokes equations for incompressible, Newtonian fluids. Data for training and testing the neural networks was generated using the finite volume method. The network architecture consists of a convolutional encoder and decoder model for dimensionality reduction, with an additional neural network for time evolution inserted between the encoder and decoder. The network takes one time step as input and predicts five consecutive time steps ahead. Three different networks for time evolution were tested. The best performing network was tested on increasingly complex computational domains. Thereafter, the network's ability to generalize to varying inlet velocity patterns and new variations of the computational domain was tested, for which the network for time evolution was adapted to take into account varying inlet velocity patterns. The performance of the networks was evaluated on predicting one to five time steps ahead from one input time step, and using the prediction of one time step ahead recursively to obtain the prediction of one entire heartbeat.

The best performing architecture found was an encoder, recurrent neural network with long-short term memory units for time evolution, decoder model combination. The generalizability for the tested material parameter, blood viscosity, while keeping the inlet velocity pattern fixed, appeared quite good. A relative mean absolute error (MAE) of around 10% was obtained for straight, bent and bifurcated channel geometries. When testing the network on varying inlet velocity patterns, the relative MAE in the domain increased considerably to around 25%. In general, the relative errors were significantly higher in comparison to the absolute errors, and mainly increased by low velocity parts of the domain and time series. In addition, the full height of the velocity peak was not always captured and a spike in the domain MAE was seen for rapidly decreasing inlet velocities. Also, the network was unable generalize to new geometries.

Various attempts to improve the accuracy, including altering the loss function and augmenting the data, were not successful. However, many choices of data augmentations, normalization techniques and loss functions were not tested and could improve the performance. The generalizability to different computational domains could be improved by adding more geometries and more variation to the geometries in the data. Nonetheless, the prediction of one heartbeat was obtained in approximately 20s, which is equivalent to an achieved speedup of around 20 compared to OpenFoam simulations. In conclusion, this research should be seen as a basis for time-dependent blood flow predictions in the aorta, from which a multitude of different paths can be explored.

Preface

The thesis in front of you concludes my seven years of being a student at the TU Delft. Starting in 2015 as Clinical Technology student, I would not have thought I would be concluding my studies at the faculty of Electrical Engineering, Computer Science and Mathematics. Although bridging the gap in knowledge from a bachelor that mostly resembles studying Medicine to Applied Mathematics was quite challenging and required a new level of commitment to my studies, I am grateful that I did make the switch to this masters degree. The deeply rooted joy I find in solving complex puzzles and problems was definitely satisfied during this masters.

The combination of my bachelor and master degrees also led me to this research topic. I discovered my interest for machine learning during my internship. Machine learning intrigues me due to the combination of it's practical applicability to real world problems and it's lively research field, as there still is a lot to be discovered. Together with my medical interests, machine learning for cardiovascular modelling was a perfect fit. The long term goal of this project, being able to have real-time, patient specific, blood flow predictions based on the geometry of their arteries, was very motivating and I hope my research made a (small) contribution to achieving this goal.

During this thesis project I had support from many people, who I would like to thank personally. First and foremost, my supervisor Alexander, who has been super involved, motivating and enthusiastic about this project at all stages. I have never encountered a supervisor or lecturer with a quicker response time and although the feedback was often a lot and critical, it definitely brought this research to a higher level and was much appreciated. I also want to thank my brother and dad, for working their way through these pages to proof read for me, Lex, for his sharp eye on my tables and figures, together with my mom and friends for always showing interest and listening to me complain at times. A special mention also for Erik, who I have spent nearly every day with for the last couple of months, this process was a lot more fun together than it would have been alone. Last but definitely not least, I want to thank Dennis Palagin for the guidance on DelftBlue supercomputer, Matthias Möller for the smart suggestions during my literature and greenlight presentations, and together with prof. Arnold Heemink taking the time to be on my committee.

Altogether, I had an amazing time studying in Delft, during which I had the opportunity to experience so much and meet lots of great people along the way. After concluding this thesis, I am very much looking forward to exploring Central America in the upcoming month and thereafter investigating what the next step will be. Thank you for taking the time to read this research and I hope you enjoy it!

Sylle Hoogeveen

Rotterdam, October 2022

List of abbreviations

- AE - Autoencoder
- BC - Boundary Condition
- CAE - Convolutional Autoencoder
- CFD - Computational Fluid Dynamics
- CNN - Convolutional Neural Network
- DIC - Diagonal-based Incomplete Cholesky
- FVM - Finite Volume Method
- GRU - Gated Recurrent Unit
- IC - Initial Condition
- LSTM - Long-Short Term Memory
- MAE - Mean Absolute Error
- ML - Machine Learning
- MSE - Mean Square Error
- NN - Neural network
- ODE - Ordinary Differential Equation
- PCA - Principal Component Analysis
- PCG - Preconditioned Conjugate Gradient
- PDE - Partial Differential Equation
- PGD - Proper Generalized Decomposition
- PINN - Physics Informed Neural Network
- POD - Proper Orthogonal Decomposition
- PRNN - Physics Reinforced Neural Network
- RD - Reduced Basis
- RNN - Recurrent Neural Network
- ROM - Reduced Order Model
- SGD - Stocahstic Gradient Descent
- SVD - Singular Value Decomposition
- VP - Velocity Pattern

Contents

1 Introduction

1.1 Relevance

The increase in complexity of mathematical models in an attempt to approximate reality and desire to have near real-time results, for example in medical image-guidance [3] or creating digital twins [4], have emphasized the need for fast numerical simulations. Reduced order models (ROMs) aim to capture the most important features of a physical phenomenon being simulated, whilst reducing the computational load compared with full order/high-fidelity models and thus obtaining a speedup in simulation time.

Classic reduced order methods, which will briefly be discussed in chapter 2.1, generally rely on linear basis functions [5]. In contrast, the underlying dynamics are often non-linear. Also, due to discretizing, they depend on a fixed computational domain or the ability to find a parametrization. In addition, numerical methods for solving the full or reduced order model that are not completely implicit are dependent on the Courant number for convergence. Hence, the more accurate the solution (i.e. a fine mesh), the smaller the time step must become, which in turn increases simulation time.

Instead of using classical numerical methods, neural networks can be used to learn non-linear relations, produce lower dimensional representations and perform time evolution of physical phenomena governed by differential equations. Especially in areas where classic numerical methods struggle to produce valid solutions in reasonable computational time due to their complex behaviour on multiple temporal and spatial scales, such as (cardio-vascular) fluid modelling [1], machine learning (ML) techniques can be of help. Preceding work done in this direction is discussed in chapter 2.2. This thesis will continue on that basis and explore the boundaries of this approach for predicting the blood flow through the aorta during one heartbeat.

1.2 Aim and objectives

The aim of this study is to construct a single reduced order model, based on neural networks, for time-dependent incompressible blood flow through the aorta that can account for varying velocity inlet conditions, material parameters and geometries (computational domains). The objectives are to minimize the reduction of accuracy of simulated time series with OpenFoam [2], and to obtain speedup compared to the OpenFoam simulations.

However, for both classic and machine learning based reduced order models there is a trade-off between computational work and thus computation time, accuracy and generalization. To create a more accurate model, the computational load usually increases and the generalizability decreases. This leads to the research question:

"What is possible in terms of generalizability, whilst minimizing the reduction of accuracy and obtaining speedup, for time-dependent cardiovascular simulations with the use of state-of-the-art machine learning techniques?"

This question can be divided into three sub-questions:

1. Which ML network architecture is most accurate at predicting consecutive time steps of time-dependent flow?
2. Can the ML network produce accurate results for a range of physically relevant boundary conditions and material parameters?
3. How much can computational domains differ while retaining accuracy?

The answers to these questions will be sought throughout this report. As mentioned above, necessary theory and relevant preceding work is discussed in chapter 2. Thereafter, the details on the generation of data for training and testing the network, model architectures, performed tests and performance evaluation, are given in the chapter 3. The results of the tests are presented and discussed in chapter 4. Finally, the conclusions and recommendations of this thesis are given in chapter 5.

2 | Preliminaries & related work

In this chapter, the relevant theory and related research are discussed to set the framework for this study. First, reduced order models will be introduced in section 2.1. Thereafter all necessary machine learning concepts will be discussed in section 2.2 and lastly, details on cardiovascular modelling will be presented in section 2.3.

2.1 Reduced order methods

There are two approaches to constructing ROMs. The first approach is to simplify the underlying physics (known as operational based reduction methods) for example by making assumptions on certain parameters or symmetry. The second approach is discretizing the continuous equations and thereafter reducing the model, most commonly using projection-based methods [6].

Some well known projection-based methods are proper orthogonal decomposition (POD), reduced basis (RB) and proper generalized decomposition (PGD). POD relies, as the name suggests, on orthogonal decomposition and principal components. The principal components are eigenvectors of the data's covariance matrix and hence can be found through eigendecomposition of the covariance matrix or singular value decomposition (SVD) of the data matrix. The data set may originate from various sources, for example numerical simulations or physical problems. The process of finding principal components is called principal component analysis (PCA) [7]. RB methods find an approximate solution to a parameterized partial differential equation (PDE) in a lower dimension subspace. The solution is expressed as a linear combination of problem-dependent basis functions, generated from a set a solutions to the high-fidelity problem [8]. PGD is an iterative method for solving boundary value problems, where in each iteration the solution is enriched with a new mode. Taking only the most relevant PGD modes produces a ROM of the solution. The PGD method requires a variational formulation of the problem, which is most commonly obtained by the Bubnov-Galerkin method. All Galerkin methods apply linear constraints to convert a continuous operator to a discrete problem [9].

As mentioned in the introduction, the drawback of using these methods is that they usually rely on linear basis functions whilst the underlying dynamics are often non-linear and they depend on a fixed computational domain (fixed geometry) or the ability to find a parametrization. However, work has been done on using reduced order methods on moving objects inside the computational domain by using the immersed boundary method and reference meshes [10], [11]. Although much more research can be found on the use of classic ROMs, this thesis is limited to discussing them further only when used in combination with machine learning techniques.

2.2 Machine learning framework

As aforementioned, neural networks can be used to learn non-linear relations, produce lower dimension representations and perform time evolution of physical phenomena governed by differential equations. Two types of networks that could be used for creating a reduced order representation are convolutional neural networks and autoencoders with or without convolutional layers. For time evolution three types of networks, being neural networks, recurrent neural networks and again convolutional neural networks can be used. To ensure the networks take into account the underlying physics and increase the chance the networks output is physically relevant, a network can be physics informed. All the above concepts will be explained and discussed in the following sections. In addition, a short overview of other ML applications in computational fluid dynamics (CFD) is given.

2.2.1 Neural networks

A neural network (NN) is a general machine learning method that has many purposes, from regression and classification to creating embeddings for text. A NN is built by three sets of neurons: input neurons, (multiple layers of) hidden neurons and output neurons, as depicted in figure 2.1. How the neurons are connected determines the architecture of a NN. The inputs of each neuron (x_i) are multiplied by a weight (w_{ij}) and summed, the bias (b_i) is added, and the result is passed though an activation function (act) to determine the output of a neuron (x_j), as represented in equation 2.1a. The output of a neuron is then again the input of one of n neurons in the next layer, until the output layer ($\mathbf{y} \in \mathbb{R}^k$) is reached. Together this creates a mapping from input to output, as shown in equation 2.1b.

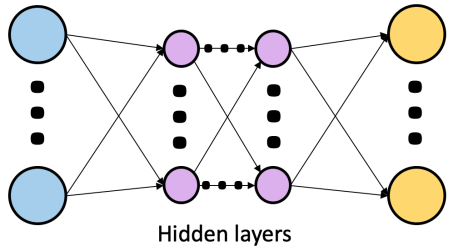


Figure 2.1: Neural Network

$$x_j = act(\sum_{i=1}^{m}(w_{ij} * x_i) + b_i) \quad \text{for } j = 1, ..., n \tag{2.1a}$$

$$\mathcal{F} : X \longrightarrow Y \quad \text{for } X \in \mathbb{R}^m, Y \in \mathbb{R}^k \tag{2.1b}$$

Choosing the architecture of a NN is a trade-off between making the NN complex enough such that the relation between input and output can be learned (prevent underfitting) and keeping the NN simple enough to be able to generalize (prevent overfitting). Generalizability refers to the ability of the NN to make predictions on input data that does not belong to the training data set, i.e. has not been seen by the network before. There are multiple techniques to improve generalizability, called regularization techniques. Some of these are early stopping of training process, parameter norm penalties such as L1 and L2 regularization and adding dropout probability to layers of neurons [12].

The activation functions used in the network could be linear. However this would entail that multiple layers of a network can be collapsed back to one single layer, as it is just a complex description of a matrix vector multiplication. Hence, to learn more complex relations, non-linear activation functions are used. Commonly used non-linear activation functions are shown below [13].

$$ReLU(x) = \begin{cases} x \text{ if } x > 0 \\ 0 \text{ if } x \leq 0 \end{cases}$$

$$LeakyReLU(x) = \begin{cases} x \text{ if } x > 0 \\ ax \text{ if } x \leq 0 \end{cases}$$

$$ELU(x) = \begin{cases} x \text{ if } x > 0 \\ \alpha(e^x - 1) \text{ if } x \leq 0 \end{cases}$$

$$SELU(x) = \lambda \begin{cases} x \text{ if } x > 0 \\ \alpha(e^x - 1) \text{ if } x \leq 0 \end{cases}$$

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

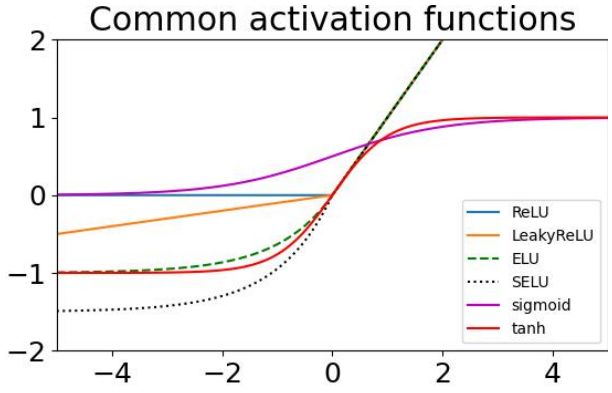


Figure 2.2: Activation functions

The training process in which the weights of the network are learned can be supervised or unsupervised. Unsupervised learning is used to discover new patterns from unlabeled data by trying to mimic the data and correcting based on the error [14]. In supervised learning, inputs are passed through the network and the output is compared to a 'ground truth' or label. This research will be mostly limited to supervised learning strategies, however in some cases (autoencoders) the label will be the input data itself, hence it is technically unsupervised learning. How the output and label are compared is determined by the loss function (L), for example the mean squared error (MSE) can be used. Hence the weights (W) and biases ($\mathbf{b}$) are adjusted, through the backpropagation algorithm [15], such that a loss function is minimized. This is represented by the optimization problem shown in equation 2.2.

$$\underset{W,\mathbf{b}}{argmin} \quad L(W, \mathbf{b}) \tag{2.2}$$

How much, and in what direction, the weights are adjusted is determined by the optimizer. For deep neural networks (NNs with many layers), Adam is the optimizer of choice due to faster running time and less memory usage than other algorithms [16]. Adam is based on stochastic gradient descent (SGD) and utilizes the concepts of momentum and adaptive learning. Here, momentum (M_t) is defined as using the previous gradients for determining current gradient (equation 2.3), such that there is less oscillation in finding the minimum compared to SGD. Thus, the learning rate can be increased, leading to faster convergence [17]. Adaptive learning refers to having a different the learning rate (λ) per parameter at each update time, using the uncentered variance (V_t) (equation 2.4) [18]. Together, they result in an update of the weights and biases determined by equation 2.5. In the following equations β and γ are rates of decay, ϵ is a small number to prevent division by zero and η is a fixed learning rate.

$$M_t = \beta M_{t-1} + (1-\beta)\frac{\partial L_t}{\partial w_{t-1}} \tag{2.3}$$

$$V_t = \gamma V_{t-1} + (1-\gamma)\left(\frac{\partial L_t}{\partial w_{t-1}}\right)^2 \tag{2.4}$$

$$w_t = w_{t-1} - \lambda * M_t \qquad \text{with } \lambda = \frac{\eta}{\sqrt{V_t}+\epsilon} \tag{2.5}$$

The most naive way of predicting future time steps with machine learning is using a neural network. With a NN, no assumption on the input data is made (such as the assumed spatial relation in convolutional neural networks) and there is no mechanism to retain information of previous input data (as in recurrent neural networks). However, this approach is still valuable to explore, as making no assumptions on the input data also means the network has all flexibility to learn undiscovered patterns. Moreover, a NN could be less complex compared to for example a recurrent neural network, which is desirable as it is more efficient computationally and memory-wise.

Using a NN for time evolution of the reduced order representation in combination with using ML techniques for reducing order was explored by Kim et al. [5]. Firstly they trained an autoencoder to create a latent space representation of smoke and fluid simulations. Thereafter, they implemented a NN to find the subsequent latent representation. As input for the NN, they used the latent representation found by the autoencoder concatenated with a control vector. The control vector is defined as the difference between known simulation input parameters of subsequent time steps. The output of the network is the difference between the current latent space representation and the next latent representation. Hence the new reduced order representation can be found by adding the output of the network to the previous reduced order representation.

2.2.2 Convolutional neural networks

A convolutional NN (CNN) is characterized by convolution kernels that slide across the input features creating feature maps. As depicted in figure 2.3 by the red square, a convolution kernel has a kernel size (in this case 3x3) on which it performs a filter operation. The filters are the weights of the network, and are learned during the training phase. Different filters uncover different relations in the input data. For example the filter (shown in stencil notation) $\left|\begin{smallmatrix}1&0&-1\\1&0&-1\\1&0&-1\end{smallmatrix}\right|$ detects vertical edges. The depth of the kernel is the amount of different filter operations it performs on the input data, and thus how many feature maps the layer creates. This is depicted in figure 2.3 by the red arrow. Using a CNN versus a NN reduces the number of parameters to be learned in a network, as the neurons share weights, i.e. the same filter is applied to every patch of input neuron to create one feature map. The kernel also has a stride, defined as the amount of spaces the kernel shifts per step [19].

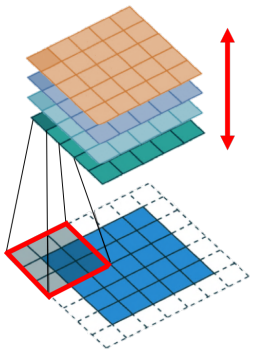


Figure 2.3: Convolution[1]

[1] Adaptation of original figure from [19]

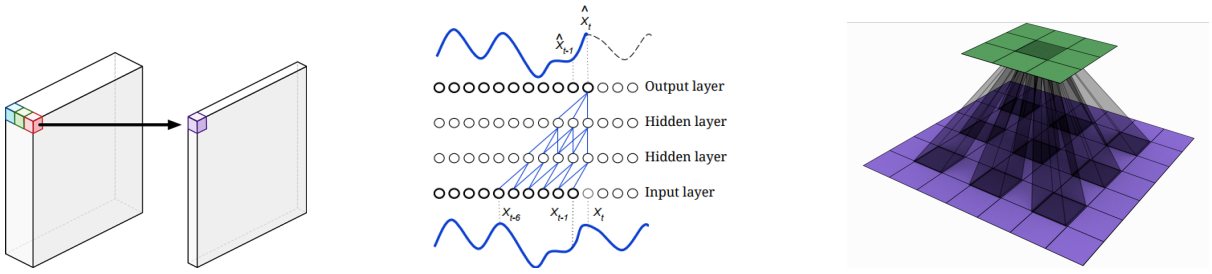
There exist multiple types of convolutions. Let $F : \mathbb{Z}^2 \to \mathbb{R}$ be a discrete function, for example an input image. Let $\Omega_r = [-r, r]^2 \cap \mathbb{Z}^2$ and let $k : \Omega_r \to \mathbb{R}$ be a discrete filter of size $(2r+1)^2$. The first type, normal convolutions, move over each channel of an input image or one feature map of the previous layer separately, as depicted in figure 2.3 and equation 2.6. In contrast, pointwise convolutions are 1x1 convolution kernels that perform a filter operation on the entire depth of the previous layer. Hence the depth of the successive layer can be controlled, as shown in figure 2.4a and equation 2.7. For time-series forecasting, causal convolutions, with or without dilatation can be used. Causal convolutions ensure only information from previous points in time are used by restricting the convolution, as can be seen in figure 2.4b and equation 2.8 [20]. Dilatation is a method to increase the receptive field whilst decreasing the number of layers and thus the number of parameters to train [21]. Dilatation can also be applied to normal convolutions. This is done by skipping over certain steps, as depicted in figure 2.4c and equation 2.9 (where l is the dilatation factor and a dilated convolution is referred to with $*_l$). One of the drawbacks of using dilatation is that it creates gridding artefacts [22].

$$(F * k)(x, y) = \sum_{dx=-r}^{r} \sum_{dy=-r}^{r} k(dx, dy) F(x - dx, y - dy) \tag{2.6}$$

$$x_{ij} = \sum_{k=1}^{n} w_k \cdot x_{ijk} \tag{2.7}$$

$$\begin{bmatrix} x_{t-n} \\ \vdots \\ x_{t-1} \end{bmatrix} \cdot \begin{bmatrix} w_1 \\ \vdots \\ w_{n-1} \end{bmatrix} = h_t \tag{2.8}$$

$$(F *_l k)(x, y) = \sum_{dx=-r}^{r} \sum_{dy=-r}^{r} k(dx, dy) F(x - l \cdot dx, y - l \cdot dy) \tag{2.9}$$



(a) Pointwise convolution[2] **(b)** Causal convolutions in 1D[3] **(c)** Dilated convolutions[4]

Figure 2.4: Different types of convolution

A CNN can consist of multiple convolutional layers, each reducing the dimension of the input by kernelsize -1 along each axis, if no padding is used. Padding is defined as adding virtual points along the edges of the input data. Further reduction of the dimension is achieved by adding pooling layers. For example, the commonly used max pooling operation returns the max value within it's pool (same as kernel). By reducing the dimension and thus the total number of weights, pooling layers reduce the computational and memory usage of the CNN. As inverse operation to pooling, upsampling can be used, which repeats the input to a supplied dimension.

A deep CNN with 3D convolutional layers to reduce dimensionality of velocity fields generated by a synthetic jet simulation was created by Lopez-Martin et al. [24]. Three dimensions in this case are two spatial and one temporal dimension. The first convolutional layers were used to find temporal-spacial relations and a reduced order representation. The final layers of the network were used for time evolution. This was achieved by adding a convolutional layer that controls the final depth dimension of the feature space to be the number of time steps to be predicted. This vector is reshaped such that it can be broken into the amount of time steps to be predicted separate vectors. These vectors are then used as input to the fully connected layers, which learn to perform the time evolution of the velocity field. Noteworthy about this approach, is that the network was retrained with a small training set of flow past a cylindrical obstacle. After which the network could successfully predict this flow pattern as well. The domain was not varied and remained a rectangle for both flow simulations.

[2] Figure from [23]
[3] Figure from: https://discuss.pytorch.org/t/causal-convolution/3456
[4] Figure from: https://medium.com/hitchhikers-guide-to-deep-learning/10-introduction-to-deep-learning-with-computer-vision-types-of-convolutions-atrous-convolutions-3cf142f77bc0

2.2.3 Autoencoders

An autoencoder (AE) is a type of NN that consists of an encoder and decoder part. The encoder part of the network tries to find an embedding or lower dimension representation (depicted in orange in figure 2.5) of the input data. This is achieved by reducing the amount of neurons in each consecutive layer (purple part in figure 2.5) until the amount of neurons is equal to the desired latent dimension size. The decoder part of the network (light blue in figure 2.5) then tries to decode the latent representation back to the full input dimensionality. This process is used in various applications, for example for denoising input data.

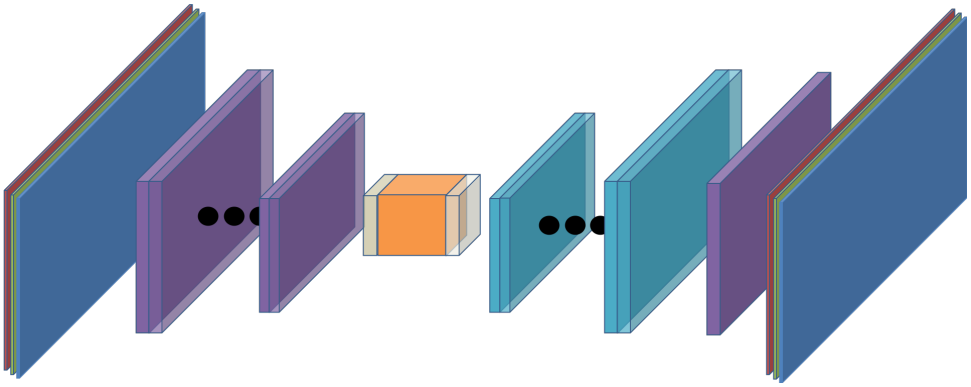


Figure 2.5: Autoencoder structure for RGB image

Also, autoencoders have been used for the purpose of creating reduced order models. Simpson et al. [25] used a simple autoencoder, consisting of layers of normal neurons, to find a reduced order representation of systems under forcing. It was noted that dynamics not captured in the training data were not captured by the produced ROM. Additionally, for high dimensionality input, a convolutional autoencoder was suggested to prevent long training time and the need for large amounts of training data.

A convolutional autoencoder (CAE) is an autoencoder network structure, using convolutional layers as explained in the section 2.2.2. Kim et al. [5] and Eichinger et al. [26] used this architecture. Opposed to other papers reviewed, Eichinger et al. studied steady state fluid flow, using a binary and signed distance function representations of the input domain and retrieving the velocity field as output. They found a speedup in the order of 100 comparing their CAE with OpenFoam simulations [26].

Kim et al. [5] used an CAE network to generate smoke and fluid simulations for graphical implementations (games, videos, etc.). Their approach led to 700x speedup compared to MantaFlow simulations while retaining accuracy (on visual inspection) for a variety of fluid behaviour. However, the ability to reconstruct physically accurate scenarios depended heavily on how closely the scenarios matched the input training data. Also, no rigorous error analysis was presented in the article as speedup was the main objective.

Due to the depth of the networks, the vanishing gradient problem can occur. The large number of steps in backpropagation through the network results in the gradient becoming very small. Hence the update of the weights becomes very small. In other words, learning halts. To prevent this, both studies implemented residual connections, allowing information to pass certain layers of the network and thus increasing the gradient. This method was introduced by He et al. [27] as residual networks (ResNets).

A (different) Kim et al. [28] proposed using a shallow masked autoencoder instead of a deep convolutional autoencoder to improve efficiency. Shallow in this case means the encoder and decoder only consisted of one hidden layer. In this network knowledge of classic numerical methods for solving PDEs/ODEs was used as follows. The hidden layer and output layer of the decoder were sparsely connected by multiplying the weight matrix of this connection by a mask matrix. The mask matrix consists of zeros and ones, and was constructed to reflect local connectivity as in the central difference scheme of the Finite Difference Method. However, using such a mask matrix makes the autoencoder computational domain specific.

2.2.4 Recurrent neural networks

Recurrent NNs (RNNs) have input (x), hidden (h) and output (y) neurons similar to a standard neural network. However, the neurons are divided in groups belonging to a certain time step. The input neurons are connected to neurons in the hidden layer belonging to the same time step. Also the neurons within the hidden layer are fully connected to hidden neurons with the same assigned time step and unidirectionally connected to hidden neurons of the next time step. The output neurons are again only connected to hidden neurons belonging to the same time step. This is depicted in figure 2.6, in which A represents a unit. RNNs can be built with different types of units: standard units, long short-term memory (LSTM) units or gated recurrent units (GRUs). The specifics of these units will be discussed in the following sections.

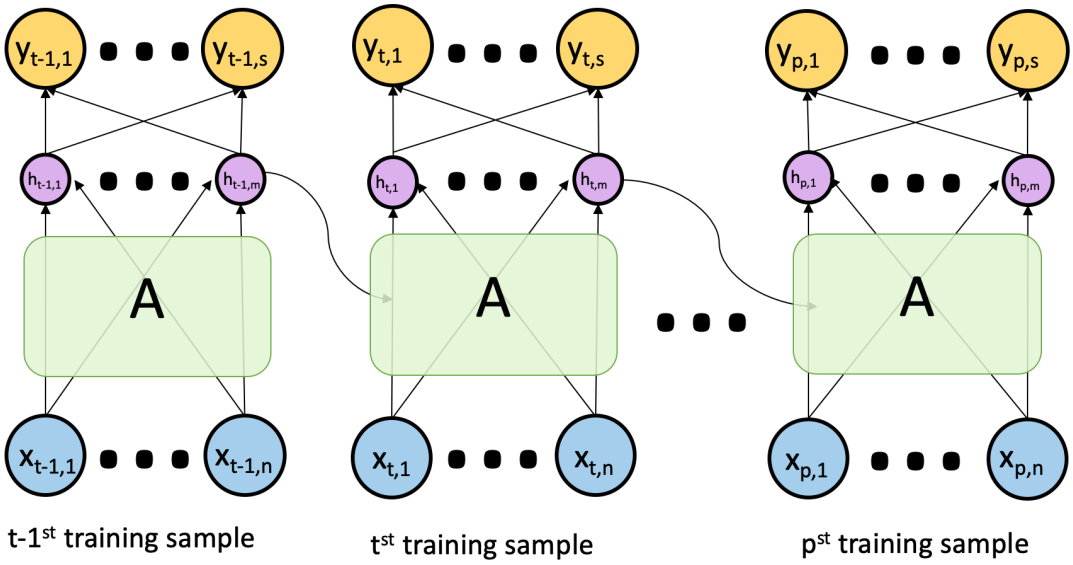


Figure 2.6: RNN with n input neurons, m hidden state variables, s output neurons and p time steps

Standard Unit

A standard unit or standard RNN cell has two inputs: x_t, the part of the sequence for the current time step, and h_{t-1} the hidden state compute from the previous part of the sequence. These are concatenated, multiplied by the weight matrix, added with the bias vector and the result is passed through an activation function, such as the hyperbolic tangent function. The outcome is a new hidden state: $h_t = act(W * x_t^\frown h_{t-1} + b)$, which will be used as input for the next time step and/or multiplied with a weight matrix V to obtain the output of the network. Note that all weight matrices are constant for each time step [29].

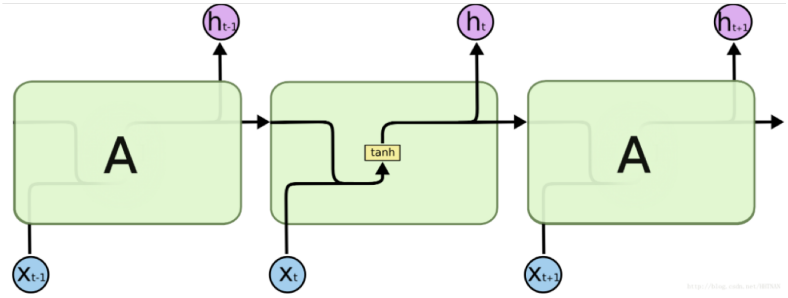


Figure 2.7: Standard unit RNN[5]

However, for standard units, the vanishing gradient problem occurs when the sequence is very long [29]. A large sequence length is similar to a deep NN, as the gradient is backpropagated many times though the network. Again, this leads to very small or even no weight updates and learning halts.

[5]figure from: https://sh-tsang.medium.com/review-empirical-evaluation-of-gated-recurrent-neural-networks-on-sequence-modeling-gru-2adb86559257

Long short-term memory unit

The LSTM unit was developed in 1997 to solve the vanishing gradient problem of the RNNs with standard units [30]. The LSTM unit has three inputs instead of two. Again the current part of the sequence x_t and hidden state of the previous parts of the sequence h_{t-1} are used. In addition, there is a second state vector which is the cell state or LSTMs memory c_{t-1}. The cell state passes through the LSTM unit with less computation, making it easier to pass through unchanged [31]. Similar to the idea of adding residual connections in deep NNs, this helps preserve the gradient. In the LSTM unit, three gates are computed from the concatenation $x = x_t \frown h_{t-1}$, each having their own weight matrix as described in equations 2.10, 2.11 and 2.12.

$$\text{Forget gate } f = \sigma(W_f x + b_f) \tag{2.10}$$
$$\text{Update gate } u = \sigma(W_u x + b_u) \tag{2.11}$$
$$\text{Output gate } o = \sigma(W_o x + b_o) \tag{2.12}$$

Here, σ is the sigmoid function which returns values between 0 and 1, killing inputs close to 0 and letting inputs close to 1 pass nearly unchanged. The new cell state is computed by multiplying the previous cell state by what was learned to forget (the forget gate) and adding this to what was learned to be remembered (see equation 2.13). The new hidden state is computed by the cell state scaled between [-1,1] by tanh function multiplied by what was learned to be exposed to the hidden state (output gate) (equation 2.14) [29]. This gives the ability to control what to forget from the cell state, what to store from the input in the cell state and what part of the cell state to expose to the hidden state at a given point in time [31]. From the hidden state again the output of the network can be computed by multiplying it with weight matrix V.

$$c_t = f * c_{t-1} + u * tanh(W_c * x + b_c) \tag{2.13}$$
$$h_t = o * tanh(c_t) \tag{2.14}$$

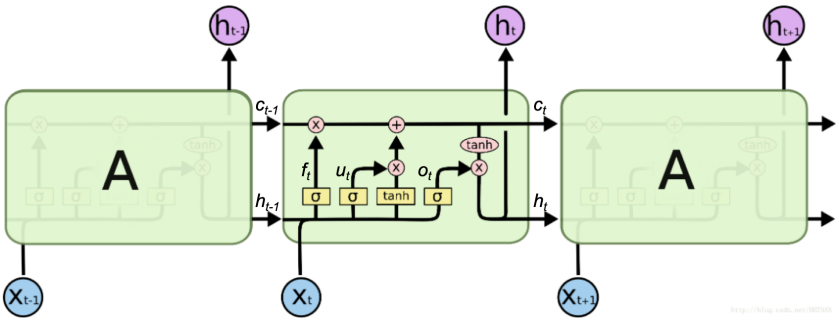


Figure 2.8: LSTM unit RNN[6]

RNNs with LSTM units have been used in combination with classical reduced order methods for temporal evolution in simulations of physical phenomena. For example, Hu et al. [32] used POD and SVD together with an LSTM network for predicting spatial-temporal distribution of floods. They reported a maintained accuracy in comparison to the full numerical model while CPU cost was reduced three orders of magnitude. Pawar et al.[33] also merged a classical reduced order approach, POD with Galerkin projection, with LSTM network for time evolution in vortex merging simulations. An improvement in accuracy by using this method was reported, compared to a purely data-driven approach. The authors suggested extending their research by introducing more variation between train and test data. Furthermore, RNNs with LSTM units have been used in combination with machine learning techniques for order reduction. The aforementioned Simpson et al. [25] trained an autoencoder for order reduction and LSTM network to predict the response of dynamical systems in the latent space based on forcing time histories.

[6]figure taken from: https://sh-tsang.medium.com/review-empirical-evaluation-of-gated-recurrent-neural-networks-on-sequence-modeling-gru-2adb86559257

Gated recurrent unit

GRUs are an evolution of LSTM units and introduced quite recently in 2014 [31]. By dropping the output gate only two gates are retained, as described in equations 2.15 and 2.16:

$$\text{Reset gate } r = \sigma(W_r x + b_r) \tag{2.15}$$

$$\text{Update gate } z = \sigma(W_z x + b_z) \tag{2.16}$$

The reset gate is similar to the forget gate in the LSTM unit and exists to decide what to forget. The update gate is to decide what should be passed to the output. First a proposed new hidden state ($\hat{h}_t$) is computed by multiplying the old hidden state with the reset gate, concatenating this with the new part of the sequence and scaling between [-1,1] with the tanh function (equation 2.17). Thereafter, the actual new hidden state is computed by taking a linear sum between the previous hidden state and the proposed new hidden state, in which the update gate determines how much of the hidden state is updated (equation 2.18) [31].

$$\hat{h}_t = tanh(W\widehat{x_t}(r_t * h_{t-1}) \tag{2.17}$$

$$h_t = (1 - z_t)h_{t-1} + z_t\hat{h}_t \tag{2.18}$$

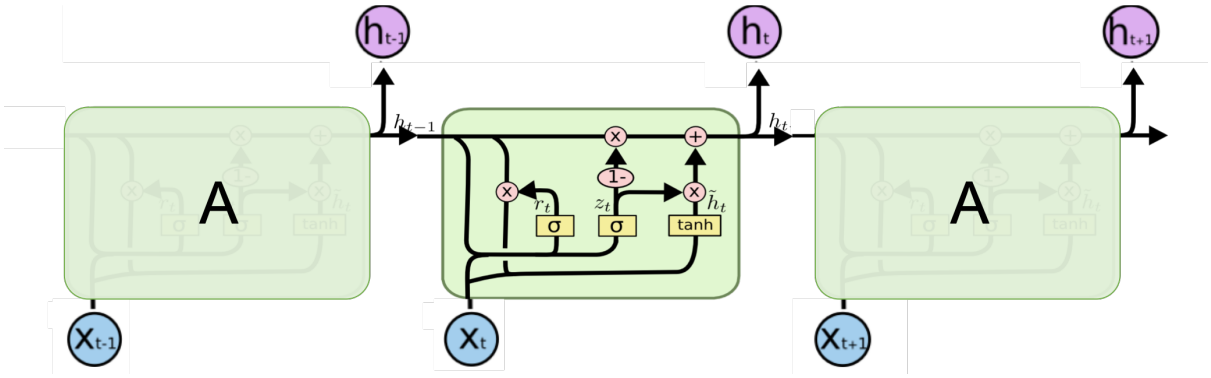


Figure 2.9: GRU RNN [7]

With only two gates, there is no mechanism to control which parts of the state is exposed to the output, but the GRU's structure is less complex. This makes the units computationally more efficient while it was proven to have comparable performance to LSTM units [34].

RNN with GRUs for time evolution have been used in combination with classical reduced order techniques. Wu et al. [35] created a latent representation of flow past a cylinder in a rectangular domain with POD-Galerkin projection and implemented a CNN with causal convolutional layers and RNNs with LSTM and GRU for comparison. The temporal convolutional network achieved better performance and needed approximately five or three times fewer parameters than the LSTM and GRU RNN, respectively.

In addition, GRU RNN has been used in the prediction of blood flow characteristics by Jamil et al. [36]. In their research, inlet velocity and percentage lumen openings (also called degree of stenosis) at eleven locations along the blood artery were used to predict velocity, pressure and wall shear stress at those positions. NNs, LSTM and GRU RNNs were compared. Although the GRU RNN displayed the best overall accuracy, it was outperformed on individual properties. NNs were found to outperform the RNN structures for predicting velocity and wall shear stress and LSTM RNN performed best for the highly varying pressure values.

[7] Adaptation of figure taken from: https://sh-tsang.medium.com/review-empirical-evaluation-of-gated-recurrent-neural-networks-on-sequence-modeling-gru-2adb86559257

2.2.5 Physics-informed neural networks

To force NNs to learn the physics instead of overfitting on the input data and thus increasing generalizability, one can think of making neural networks for solving problems involving PDEs 'physics informed'. This means information about the underlying physics of the problem is inserted into the NN, which can be done by constructing activation and loss functions specific for the underlying differential operator. All network types mentioned earlier can thus be physics informed, if the activation and/or loss function is tailored to the PDE.

An approach to making networks physics informed is by incorporating the governing equation in the loss function. This is called physics loss and for Navier-Stokes equations could take the form of equation 2.19. Using this type of loss function, possibly in addition to a data-driven loss function, results in a Physics Informed Neural Network (PINN), which was introduced by Raissi et al. [37]. Since this method was introduced, variants such as fractional PINN [38] and variational PINN [39] have been presented.

$$L_{physics} = ||\nabla \cdot \mathbf{u}||_2 + ||\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} + \frac{1}{p}\nabla p - \nu \nabla^2 \mathbf{u}||_2 \tag{2.19}$$

To ensure conservation of mass (thus non-divergence) for incompressible fluid dynamics, Kim et al. [5] introduced a stream function based loss function in the decoder part of their network. In the stream loss function as described in equation 2.20, $G(\mathbf{c})$ is the output of the network and $\mathbf{u_c}$ is a simulation sample from the data set.

$$L_G(\mathbf{c}) = \lambda_{\mathbf{u}}||\mathbf{u_c} - \nabla \times G(\mathbf{c})||_1 + \lambda_{\nabla \mathbf{u}}||\nabla \mathbf{u_c} - \nabla(\nabla \times G(\mathbf{c}))||_1 \tag{2.20}$$

Making the stream function of the model output the reconstruction target, which is divergence free by construction ($\nabla \cdot (\nabla \times G(\mathbf{c})) = 0$) and ensuring the derivatives also match. However, no discussion on comparing this loss function to purely data loss functions or physical loss was presented in the article.

Above approach of pushing a solution towards one that obeys the physical laws by modification of loss functions can be seen as soft physical constraining the network. Physical laws are not enforced at all times but are encouraged. Mohan et al. [40] took a different approach, embedding hard physical constraints in the neural network architecture. This was done by adding non-trainable layers after a CAE structure. The decoder in this network has the vector potential, by the same reasoning as in above paragraph, as output. The non-trainable layers consist of a layer enforcing (periodic) boundary conditions (BCs) with ghost cells, a layer which computes all spatial derivatives with use of finite difference stencils and a last layer to compute the curl on the vector potential to regain the velocity field. The loss function was purely data driven. No total conservation of mass was found due to the discretization error of the finite difference stencils but improvement compared to using no hard physical constraint was noted.

A combination of hard and soft physical enforcement was implemented by Sun et al. [41] in a data-free deep NN for cardiovascular modelling. In their approach the governing equations (Navier-Stokes equations) and Neumann BC are taken into the loss equation. However, initial conditions (ICs) and Dirichlet BC are enforced by constructing the NN ansatz $\hat{\mathbf{u}}$ and $\hat{p}$ with a particular solution as described in equation 2.21,

$$\begin{aligned} \hat{\mathbf{u}}(t, \mathbf{x}) &= \mathbf{u}_{par}(t, \mathbf{x}) + D(t, \mathbf{x})\tilde{\mathbf{u}}(t, \mathbf{x}) \\ \hat{p}(t, \mathbf{x}) &= p_{par}(t, \mathbf{x}) + D((t, \mathbf{x})\tilde{p}(t, \mathbf{x}) \end{aligned} \tag{2.21}$$

where $\tilde{\mathbf{u}}$ and $\tilde{p}$ are predictions from the NN, $\mathbf{u}_{par}$ and p_{par} are particular solutions satisfying the IC and BCs and D is a globally defined smooth function from internal points to the boundary in a space-time sense. That is, $\mathbf{u}_{par}(\mathbf{x}, 0) = \mathbf{u}_0(\mathbf{x})$, $p_{par}(\mathbf{x}, 0) = p_0(\mathbf{x})$, $\mathbf{u}_{par}(\mathbf{x}, t)|_{\mathbf{x} \in \partial\Omega} = \mathbf{u}^b(\mathbf{x}, t)$, $p_{par}(\mathbf{x}, t)|_{\mathbf{x} \in \partial\Omega} = p^b(\mathbf{x}, t)$ and $D(\mathbf{x}, t) = 0$ on $\partial\Omega$ for $[0, T]$ and in Ω for $t = 0$. For simple geometries $\mathbf{u}_{par}$, p_{par} and D can be constructed analytically, but for more complex problems the authors suggest using pre-trained NNs to find these functions. Only implementation of steady-state flow and Dirichlet BCs was done as proof-of-concept. When compared to a purely data-driven approach the latter was slightly more accurate.

Physics can also be injected in a time-series forecasting network. For example, by taking a reduced order representation, created either by classical methods or ML techniques, and concatenating this with computed hidden state vectors of a recurrent neural network [33].

2.2.6 Related work: machine learning & computational fluid dynamics

Machine learning has not only been used to produce reduced order representations and perform time evolution. Other attempts have been made using ML to counter increased complexity of mathematical models and create more computationally efficient algorithms in CFD. For example, the accuracy of a coarse grid CFD simulation was enhanced using ML [42]. Also, ML was used for hyperreduction, in situations were dimensionality reduction by projection-based ROMs does not lead to computation speedup. This occurs when the computation of reduced-order operators is expensive [43]. Yet another example is creating a network to recover full CFD solutions for any time instance in between saved time steps, hereby reducing the storage overhead [44]. Similarly, ML was also used to reconstruct missing or faulty flow fields in CFD simulations [45].

Besides improving the performance of numerical methods, ML has also been used in model discovery and error analysis. Examples of these use-cases are discovering wall models in turbulence modelling with reinforcement learning [46] and estimating the output error and performing mesh adaptation for CFD simulations [47].

2.3 Cardiovascular modelling

As use case this research will apply ML ROM on cardio-vascular simulations, specifically blood flow through the aorta during one heartbeat. To generate simulation data for training, validation and testing, some background knowledge on cardiovascular modelling is necessary and provided in this section. Also, a brief discussion is presented on other ML applications in cardiovascular modelling.

2.3.1 Modelling blood flow in large arteries

Blood is a non-Newtonian fluid, showing all characteristics including deformation rate dependency, viscoelasticity, yield stress and thixotropy, mainly due to presence of red blood cells [48]. However it was found that the non-Newtonian impact in big arteries, such as the aorta, is negligible [49]. Thus for the scope of this research blood is perceived as Newtonian. The flow of blood was modeled by Navier-Stokes equations as proposed in [50], adapted from 3D to 2D (equation 2.22).

$$\begin{aligned} \nabla \cdot \mathbf{u} &= 0 \\ \frac{\partial}{\partial t}(\mathbf{u}) + (\nabla \mathbf{u})\mathbf{u} + \nabla \frac{p}{\rho} - \nabla \cdot (2\nu D(\mathbf{u})) &= \mathbf{0} \end{aligned} \tag{2.22}$$

In which $\mathbf{u}$ is the velocity, p is the (mechanical) pressure, ν is the kinematic viscosity ($\nu = \frac{\mu}{\rho}$, with μ the dynamic viscosity) and D the strain-rate tensor ($D = \frac{1}{2}(\nabla \mathbf{u} + (\nabla \mathbf{u})^T)$). The dynamic viscosity of blood is reported between 3,5 and 5,5 cP, which is equal to 0,0035 and 0,0055 $\frac{Ns}{m^2}$. However, in reality, the viscosity of blood has a much larger range dependent on hemodynamic conditions [51]. Similarly, blood density differs depending on gender and moreover, body position. It was set to the average blood density of 1.060 $\frac{kg}{m^3}$ [52]. The artery walls are modeled as rigid, which is again a simplification of reality. To model artery walls more accurately, hyperelastic and viscoelastic material models can be incorporated [53].

2.3.2 Synthetic artery model problem

The aorta can roughly be divided into three segments; the ascending aorta and aortic arch (1), descending aorta (2) and bifurcation of the aorta into iliac arteries (3), as shown in figure 2.10. This research will constrict itself to modeling these segments with one inlet and one or two outlets, as shown in section 3.1.3. The bifurcation of the aorta will have the two common iliac arteries as outlets. The diameter of the aorta differs per segment, per person and per measuring method. In table 2.1 ranges of the diameter each segment in healthy, adult population (male and female) as measured in [54],[55],[56] and [57] are shown.

Aorta segment	diameter range in mm
Aortic arch (1)	22 - 36
Descending aorta (2)	20 - 30
Bifurcation aorta (3)	10 - 23
Iliac common artery (3)	6,5 - 16,5

Table 2.1: Aorta dimensions

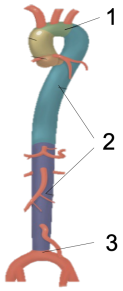


Figure 2.10: Anatomy aorta[8]

The inlet velocity during one heartbeat ranged from 0 - 71 cm/s depending on the stage of the cardiac cycle, as measured in [58]. This was used for all segments. The pressure range during one heartbeat was taken to be 80 - 120 mmHg for all segment as reported in [59]. More details on implementation of the boundary conditions can be found in section 3.1.2.

2.3.3 Related work: machine learning in cardiovascular modelling

Due to the complex nature of cardiovascular modelling and intrusiveness of velocity and pressure measurements in the human body, ML has been applied to predict velocity and pressure fields from aortic geometries [60]. Also hemodynamic parameters, such as static pressure, wall-shear-stress, secondary flow degree and specific kinematic energy, were predicted using NNs [61]. In both studies, no time dependence was taken into account. Hence, these type of networks could be used in union with a network that performs time evolution based on a velocity field instance.

[8] Adaptation of figure from: https://m.ufhealth.org/uf-health-aortic-disease-center/aorta-anatomy

3 | Methodology

In this chapter, all details of the methodology are described. It is useful to think of the general procedure as being cyclic. Each cycle starts with data generation (as described in section 3.1), then the network(s) are designed, trained and tested (as described in section 3.2), thereafter the error metrics are computed. How the performance of the networks is quantified is described in section 3.3. Lastly the results are analyzed and adjustments that need to be made to the data set(s) or network(s) are determined. After each cycle the complexity of the data and thus networks is increased. In total the cycle is repeated three times.

In the first cycle, different network architectures for time evolution ('time networks') are tested on a data set containing variations of a straight channel. The results are shown in section 4.1. In the second cycle, the best performing network is tested on data with more complex domains, namely a bent channel and a bifurcated channel. These results are shown in section 4.2. In the third and last cycle, the bifurcation domain is taken and the inlet velocity boundary condition is varied. Also, not all domain variations are included in the training data. The results of the last cycle are referred to as 'generalizability', and the results are shown in section 4.3. Finally, some experiments concerning loss functions, data augmentation and amount of data are performed to further understand the results and to improve the performance on generalizability. This is described in section 3.2.3.

3.1 Data generation

To solve the mathematical problem stated in section 2.3 numerically, OpenFoam 9 software [2] was used. Specifically, the PimpleFoam solver [62], for incompressible, Newtonian fluids was chosen. This solver allows for dynamic time stepping and turbulence modelling. All code to generate data can be found on https://github.com/SylleH/Thesis/tree/master/Code/DataGeneration. In each section the corresponding file, implementing that section, on GitHub is mentioned in italic font.

3.1.1 Equations

The PimpleFoam solver solves the following form of the Navier-Stokes equations for incompressible, Newtonian fluids [62]:

$$\nabla \cdot \mathbf{u} = 0 \tag{3.1}$$

$$\frac{\partial}{\partial t}(\mathbf{u}) + \nabla \cdot (\mathbf{u} \otimes \mathbf{u}) - \nabla \cdot \mathbf{R} = -\nabla p_k \tag{3.2}$$

The equation 3.1, is known as the continuity equation and equation 3.2 is known as the momentum equation, in which $\mathbf{u}$ is the velocity in $\frac{m}{s}$, p_k is the kinematic pressure ($p_k = \frac{p}{\rho}$, with ρ the density) in $\frac{m^2}{s^2}$ and $\mathbf{R}$ is the stress tensor. For incompressible fluids, $\mathbf{R}$ reduced to $2\nu D(\mathbf{u})$ [63]. As $\nabla \cdot (\mathbf{u} \otimes \mathbf{u}) = (\nabla \mathbf{u})\mathbf{u}$, which is shown below, the set of equations 2.22 and the set of equations 3.1 and 3.2 are equivalent.

$$\nabla\cdot(\mathbf{u} \otimes \mathbf{u}) = \nabla \cdot \begin{bmatrix} u_x \\ u_y \end{bmatrix} [u_x \ u_y] = \nabla \cdot \begin{bmatrix} u_x^2 & u_x u_y \\ u_x u_y & u_y^2 \end{bmatrix} = \begin{bmatrix} \frac{\partial}{\partial x}(u_x^2) + \frac{\partial}{\partial y}(u_x u_y) \\ \frac{\partial}{\partial y}(u_y^2) + \frac{\partial}{\partial x}(u_x u_y) \end{bmatrix} = \begin{bmatrix} 2u_x \frac{\partial u_x}{\partial x} + u_x \frac{\partial u_y}{\partial y} + u_y \frac{\partial u_x}{\partial y} \\ 2u_y \frac{\partial u_y}{\partial y} + u_x \frac{\partial u_y}{\partial x} + u_y \frac{\partial u_x}{\partial x} \end{bmatrix}$$

$$= \begin{bmatrix} u_x \frac{\partial u_x}{\partial x} + u_y \frac{\partial u_x}{\partial y} + u_x(\frac{\partial u_x}{\partial x} + \frac{\partial u_y}{\partial y}) \\ u_y \frac{\partial u_y}{\partial y} + u_x \frac{\partial u_y}{\partial x} + u_y(\frac{\partial u_y}{\partial y} + \frac{\partial u_x}{\partial x}) \end{bmatrix} \overset{1}{=} \begin{bmatrix} u_x \frac{\partial u_x}{\partial x} + u_y \frac{\partial u_x}{\partial y} \\ u_y \frac{\partial u_y}{\partial y} + u_x \frac{\partial u_y}{\partial x} \end{bmatrix} = \begin{bmatrix} \frac{\partial u_x}{\partial x} & \frac{\partial u_x}{\partial y} \\ \frac{\partial u_y}{\partial x} & \frac{\partial u_y}{\partial y} \end{bmatrix} \begin{bmatrix} u_x \\ u_y \end{bmatrix} = (\nabla \mathbf{u})\mathbf{u}$$

1) Use $\nabla \cdot \mathbf{u} = \frac{\partial u_x}{\partial x} + \frac{\partial u_y}{\partial y} = 0$

3.1.2 Boundary conditions

For all simulation the pressure boundary conditions (BCs) at the walls, inlet and outlet are equal. The pressure BCs are all defined as zero gradient ($\frac{\partial p}{\partial \mathbf{n}} = 0$, with $\mathbf{n}$ the normal vector to the boundary), except for the outlet boundary. Thus, it is assumed that the walls are rigid and the force exerted by the wall on the fluid is equal to the force exerted by the fluid on the wall. For the inlet, the zero gradient BC entails the assumption is made that the pressure is constant at the inlet boundary. The outlet BC is a function of time, shown in figure 3.1 by the red line, created to simulate the aortic pressure during one heartbeat. A general aortic blood pressure graph [64] was sampled, converted to tabular values, exported to a csv file, as can be seen in *bloodflow_function.py*, and referred to as 'UniformFixedValue' type BC in the *openfoam/0/p* file. All files can be found in the aforementioned GitHub repository.

For the velocity inlet condition, a time dependent combination of parabolic functions was chosen to approximate the velocity pattern during one heartbeat of real measurements [58]. The blue line in figure 3.1 shows the change in velocity U_0 over time. U_0 is the center velocity, used to create a parabolic, time-dependent function for inlet boundary. This is shown in equation 3.3 with r the radius of the channel. Note that here the assumptions are made that the inlet geometry is always circular and the velocity profile is fully developed at the inlet boundary.

$$u_y|_{inlet} = U_0(-1 + \frac{x^2}{r^2}) \tag{3.3}$$

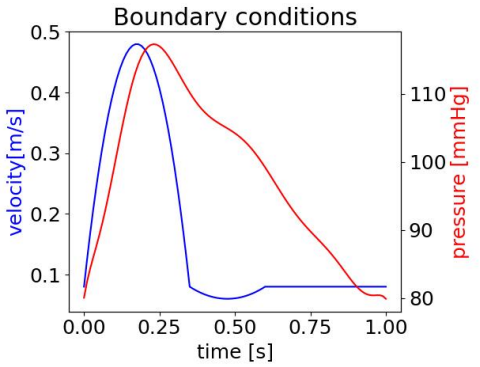


Figure 3.1: Time dependent velocity inlet and pressure outlet boundary conditions

The parabolic, time-dependent inlet condition is implemented in OpenFoam [2] as 'codedFixedValue' type BC for the inlet boundary, which is shown in the *openfoam/0/U* file. The outlet condition for the velocity is set to zero gradient, again defined as the (Neumann) BC: $\frac{\partial \mathbf{u}}{\partial \mathbf{n}} = 0$. No slip boundary conditions are implemented for all walls. This is defined as the (Dirichlet) BC: $\mathbf{u} = 0$.

In the final data set, the temporal evolution of the inlet velocity is varied for different simulations. The variation in the inlet condition was determined by six parameters; $Umax$, the height of the first parabola, $Umin$, the depth of the second parabola, $Uthree$, the height of the third parabola, tb, the duration of the first parabola, td, the duration of the second parabola and tp, the duration of the third parabola. The parameters are also shown in figure 3.2.

Firstly, five predefined velocity patterns (vp's) are created (vp0-4). Secondly, two series of random velocity pattern simulations are done by choosing a value from a predetermined range of values for each parameter (vp5-6). This is depicted in figure 3.2 and described in table 3.1 and equation 3.4. Figures 3.3 and 3.4 show the random generated velocity patterns. In green the vp's in the training set are depicted and in blue, magenta and red the vp's generated for testing are shown.

parameter	vp0	vp1	vp2	vp3	vp4	vp5	vp6
$Umax$	0,4	0,4	0,4	0,4	0,3	0,2-0,4	0,2-0,4
$Umin$	0,02	0	0	0,08	0,04	0-0,08	0-0,08
$Uthree$	0	0	0	0	0	0	0,05-0,2
tb	0,35	0,1	0,6	0,2	0,3	0,1-0,3	0,1-0,3
td	0,25	-	-	0,1	0,6	0,01-0,3	0,01-0,3
tp	-	-	-	-	-	-	0,01-0,3

Table 3.1: parameters of different inlet velocity patterns

$$U_0 = \begin{cases} 0,08 + U_{max} * (1 - \frac{(t - \frac{tb}{2})^2}{\frac{tb}{2}^2}) & t \leq tb \\ 0,08 + U_{min} * (-1 + \frac{(t - (tb + \frac{td}{2}))^2}{\frac{td}{2}^2}) & tb < t \leq tb + td \\ 0,08 + U_{three} * (1 - \frac{(t - (tb + td + \frac{tp}{2}))^2}{\frac{tp}{2}^2}) & tb + td < t \leq tb + td + tp \\ 0,08 & t > tb + td + tp \end{cases} \tag{3.4}$$

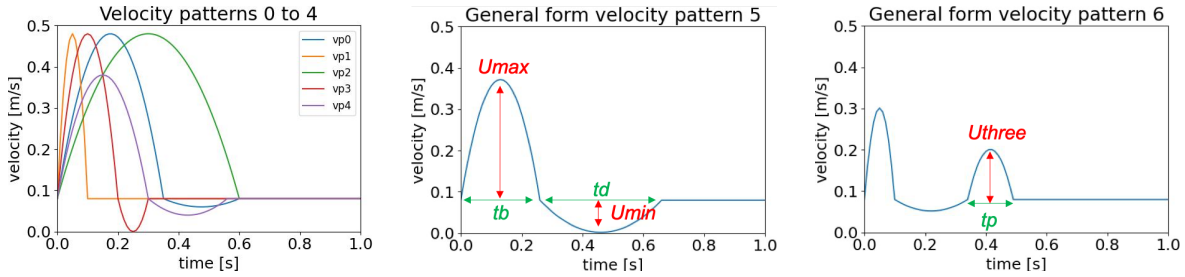


Figure 3.2: Velocity patterns from left to right: vp0-4, general form of vp5 and general form of vp6 with parameters

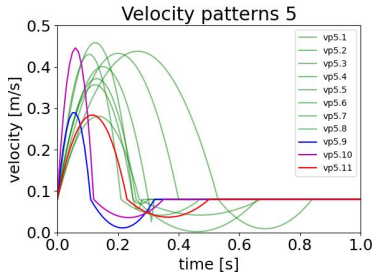


Figure 3.3: Variations of vp5

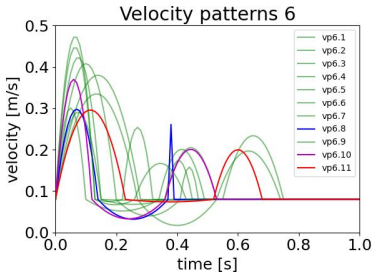


Figure 3.4: Variations of vp6

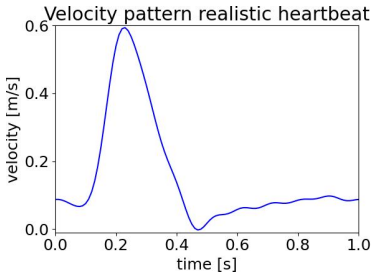


Figure 3.5: Realistic velocity pattern aorta

Finally, a more realistic velocity pattern was used to test the network. This pattern is an adaptation of the inlet flow rate as described for the aorta on HeMoLab [69, 68, 67, 66, 65]. The flow rate values were sampled, converted to velocities and Fourier analysis was used to obtain a function. The resulting inlet velocity pattern is shown in figure 3.5.

3.1.3 Geometries

Three different types of geometries, shown in figure 3.6, are created to approximate the three aorta segments described in section 2.3. The geometries are varied in channel width and if applicable in secondary outlet width. The meshes are created using the GMSH python library [70] and by implementing the following steps:

1. Define the mesh size as 2mm (straight), 2.5mm (bend) and 1.5mm (bifurcation)
2. Define the 2D mesh by defining points, lines and a surface in the xy-plane to obtain the desired geometry
3. Extrude the surface in the z-direction by one cell (this is necessary as OpenFoam [2] only allows for 3D meshes)
4. Add all surfaces to appropriate Physical Groups and name these groups, such that it is clear which part of the mesh is which in OpenFoam [2] (for example, the inlet and outlet surface)
5. Save the mesh as .msh2 file (this is compatible with gmshToFoam function)
6. In OpenFoam [2], run gmshToFoam on mesh file
7. Set front and back patches to 'empty' type in *constant/polyMesh/boundary* file (this is necessary to run 2D simulations)
8. Set inflow and outflow boundaries to patch type and set other remaining boundaries to wall type

The code to create the meshes can be found in *create_mesh.py* and all OpenFoam[2] commands are listed in the *Allrun* script, which are both available in the aforementioned GitHub repository.

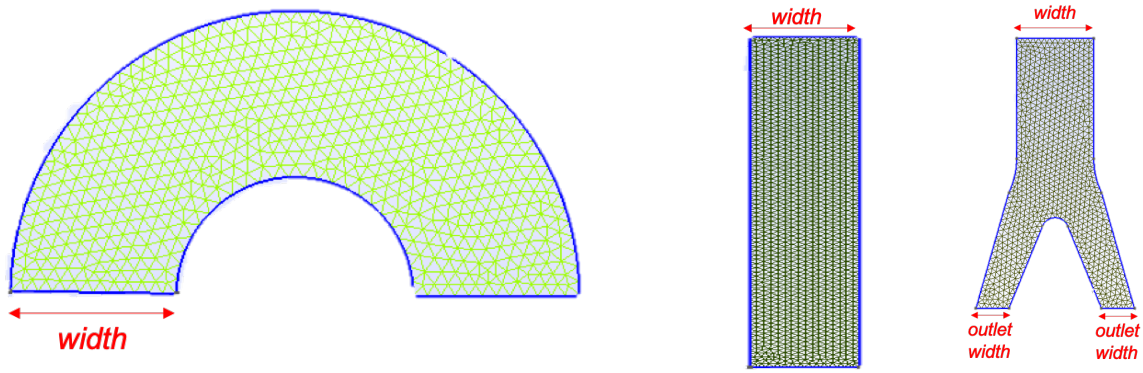


Figure 3.6: Meshes of three geometries representing three aortic segments. From left to right: aortic arch = channel bend, descending aorta = channel straight, and bifurcation aorta = channel bifurcation

3.1.4 Schemes

To represent and solve the equations numerically, the Finite Volume Method (FVM) is used. The FVM is based on integration of the governing differential equation, in this research the momentum equation described in equation 3.2, over a 3D control volume (small volume around a point in the unstructured mesh). The divergence terms in the equation are converted to surface integrals using Gauss's theorem, and the terms are evaluated as fluxes at the boundary of each volume. These steps are shown for a general conservation law in equations 3.5 to 3.8 . The method is conservative, meaning it produces a solution that obeys the physical conservation laws (such as conservation of momentum, mass or energy).

$$\frac{\partial \mathbf{u}}{\partial t} + \nabla \cdot \mathbf{f}(\mathbf{u}) = \mathbf{0} \tag{3.5}$$

$$\int_{v_i} \frac{\partial \mathbf{u}}{\partial t} dv + \int_{v_i} \nabla \cdot \mathbf{f}(\mathbf{u}) dv = \mathbf{0} \tag{3.6}$$

$$v_i \frac{\partial \bar{\mathbf{u}}_i}{\partial t} + \int_{S_i} \mathbf{f}(\mathbf{u}) \cdot \mathbf{n} dS = \mathbf{0} \tag{3.7}$$

$$\frac{d\bar{\mathbf{u}}_i}{dt} + \frac{1}{v_i} \int_{S_i} \mathbf{f}(\mathbf{u}) \cdot \mathbf{n} dS = \mathbf{0} \tag{3.8}$$

To discretize the equations, various numerical schemes are used for different terms in the equations and are specified in the *openfoam/system/fvSchemes* file. There are many options, which can be found in OpenFoam documentation [71]. In this research, the numerical scheme used for the time derivative is the Euler implicit method and for the gradient, divergence and Laplacian terms Gauss integration is used. As the fluxes are evaluated at the boundaries of the control volume, an interpolation scheme is needed for transforming cell-centre quantities to surface centres. This was set as linear for all terms with the exception of the divergence term. There, linear-upwind was chosen [72]. In addition, the Laplacian term needs a surface-normal gradient scheme, which was set to 'corrected' [73]. Details on the use of the FVM and these numerical schemes for the momentum equation are specified in appendix A.

3.1.5 Solver

As mentioned before, PimpleFoam from Openfoam 9 [2] was chosen as the solver, which implements the PIMPLE algorithm. This is a combination of the Pressure Implicit with Splitting Operators (PISO) [74] and Semi-Implicit Method for Pressure Linked Equations (SIMPLE) [75] algorithms. However, the number of outer correctors for the PIMPLE algorithm was set to 1, thus the algorithm acts as the PISO algorithm. The PISO algorithm is characterized by decoupling the operations on pressure from those on velocity in the solving process. More details on the PIMPLE algorithm can be found in appendix B. For solving the linear equations obtained after applying the discretization schemes, iterative methods are used. The preconditioned conjugate gradient (PCG) with simplified Diagonal-based Incomplete Cholesky (DIC) as preconditioner was used to solve the pressure equations and the symmetric Gauss-Seidel method was used to solve the velocity equations. These options are specified in the *openfoam/system/fvSolution file*.

PimpeFoam allows for dynamic time stepping and the implementation of turbulence models if necessary. However, for these simulations, either a time step of 0.001s or 0.0001s was sufficient for ensuring convergence (retaining a maximum Courant number below 1) and the flow did not become turbulent (Reynold's numbers below 3000 [76]) by design. This was determined by the use of equations 3.9 and 3.10 [77].

$$v_{critical} = \frac{R\mu}{\rho d} \tag{3.9}$$

$$v_{effective} = \frac{v_{max}}{2} \tag{3.10}$$

In the equation above $v_{critical}$ is the critical velocity in $\frac{m}{s}$ at which blood flow becomes turbulent, R is the experimentally found maximum Reynold's number at which flow is not turbulent, μ is the dynamic viscosity in $\frac{Ns}{m^2}$, ρ the density in $\frac{kg}{m^3}$ and d the diameter in m of the channel. Simulations were run for 3s, thus 3 heartbeats, of which the last second was used as input data. Hence the flow pattern was fully developed. The output of the simulations was saved for every 0.01s, which resulted in 101 unique images per simulation. These settings can be found in the *openfoam/system/controlDict* file.

3.1.6 Image creation

The PimpleFoam solver's output was stored as VTK (Visualization Toolkit [78]) data with the "foamToVTK -allPatches" command and thereafter converted to PNG image files by the *VTKtoPNG.py* script. The script combines the value of the velocity on nodes of the mesh (stored in the VTK data) with the node coordinates (stored in the GMSH mesh file) to obtain the velocity value at (Cartesian) coordinates. The velocity values at the coordinates are interpolated with cubic interpolation and mapped onto a pixel grid with resolution 384x515.

The velocity values are converted using a gray scale color map. This entails the values are normalized between the minimum and maximum velocity (0 and 0,7 $\frac{m}{s}$, respectively). The PNG image files are the output of the data generation pipeline.

3.1.7 Data generation pipeline

Combining all described steps in this section (3.1), an automated data pipeline was created such that input for the NNs can be generated swiftly. The pipeline requires as input which scenario to run (a choice between 'bend', 'straight' or 'bifurcation') and geometry parameters, that is, channel width and secondary channel outlet width, if applicable. In addition, the material parameter blood viscosity is given as input. It is chosen from [3.5, 4.0, 4.5, 5.0, 5.5]$*10^{-6}$ ($\frac{Ns}{m^2}$) for all scenarios. All geometry input parameters are shown in table 3.2 as well as the velocity pattern (vp), and the total number of simulations (in the last columns).

scenario	channel width (mm)	outlet width (mm)	vp	# sim
straight	[20.0, 22.0, 24.0, 26.0, 28.0, 30.0]	-	vp0	30
bend	[22.0, 24.0, 26.0, 28.0,30.0, 32.0, 34.0, 36.0]	-	vp0	40
bifurcation	[14.0, 16.0, 18.0, 20.0, 22.0]	[6.5, 8.5, 10.5, 12.5]	vp0	100
bifurcation varied vp	[14.0, 16.0, 18.0, 20.0, 22.0]	[6.5, 8.5, 10.5, 12.5]	vp0 - vp6	70

Table 3.2: Input parameters for data generation pipeline

To summarize, the geometry input parameters are used iteratively to create an unstructured mesh (and set the correct radius for parabolic velocity inlet BC). The pressure outlet(s) values for each time step (stored in a csv file) together with the GMSH mesh file are loaded into the OpenFoam Docker image. In the Docker image, the GMSH mesh is converted to a Foam polymesh and the OpenFoam files are adjusted as described in previous sections. Next, the PimpleFoam solver is run. The output of the solver is stored in VTK files and thereafter converted to PNG image files (one for each time step). The PNG image files are the outputs of the data generation pipeline and used as inputs for the NNs. An example of the output of the pipeline for each scenario is shown in figure 3.8. The whole data pipeline is depicted in figure 3.7.

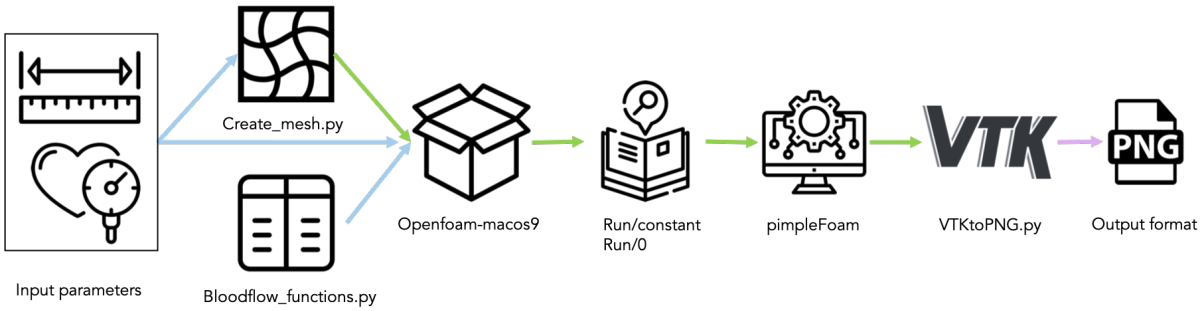


Figure 3.7: Data Generation Pipeline, icons from: Flaticons.com

Figure 3.8: Output of three scenarios. From left to right: bend, straight, and bifurcation

3.2 Machine learning methodology

Taking into account the possibilities for different network architectures discussed in chapter 2 and given the high dimensionality of the input data and desired flexibility in the computational domain, a convolutional encoder-decoder model was chosen for dimensionality reduction and expansion. Three general network architectures in this category are proposed: an encoder-decoder model with a NN for time evolution and an encoder-decoder with a RNN using GRUs and one using LSTM units for time evolution. Also, for incorporating information on the varying inlet velocity pattern in the final phase, three time network variations are tested. The specific network architectures are discussed in section 3.2.1. The training protocol and tests are described in sections 3.2.2 and 3.2.3.

3.2.1 Network architectures

To obtain the final architecture, the encoder and decoder models were first built similar to the design in [5]. Different numbers of filters per layer and dimensions of the latent space were tested to evaluate the influence of these hyperparameters. Then, a separate NN was inserted in between in the encoder and decoder to take one step ahead in time. Training the encoder-decoder part of the model and time NN part separately yielded no usable results. The encoder and decoder models were able to encode and decode the flow fields to a latent space of dimension 150. However, the time NN was not able to perform any time evolution, even when changing number of nodes per layer, number of layers and adding known inlet velocity and outlet pressure in the latent space. This was concluded from the fact that the decoded output of the neural network was an average over all time steps. This result can be explained as the encoder and decoder model have no indication of time. Hence, the hypothesis is that geometry and flow variables are mixed in the latent space, which implies trying to propagate only the flow variables in time (since the geometry variables are constant), is not possible. This was confirmed by manually altering the encoded features in the latent space and checking what this changed when decoded. Examples of the results of these experiment are shown in appendix C.

Training the network as a whole yielded viable results. It's hyperparameters were tuned using the Hyperband algorithm [79]. All hyperparameters, options, and results from the hypertuning algorithm are shown in table 3.3. However, this resulted in an increase in the number of parameters in the order of 10 and small decrease in the overall loss. Thus the hypertuned architecture was not used at this stage. The chosen hyperparameters are noted in the last column.

hyperparameter	**range**	**step size**	**result**	**chosen**
filters	8 - 32	8	24	16
kernel size	2 - 4	1	3	3
conv2d layers before skip	2 - 5	1	3	3
conv2d - concatenate - maxpooling repeats	2 - 6	2	2	4
latent dimension	25 - 150	25	75	150
NN layers	2 - 5	1	2	2
nodes per layer NN	250 - 1.000	250	1.000	512

Table 3.3: Hyperparameters

Unfortunately, there is no way to ensure which part of the this network is responsible for encoding, decoding and time propagation. Hence this network architecture has limited use, it can only predict one time step ahead for a straight channel time series. It does however indicate the network structure is complex enough to learn the relation between consecutive time steps through a latent encoding of size 150.

The final architecture consists of an encoder, a 'time network' which was either a reused NN (the output of the NN was reused as input to the NN to predict further time steps ahead), or a RNN to predict multiple time steps ahead, and the decoder model (implemented as multiple networks sharing weights) to decode the latent vectors produced by the time network. One velocity field image was provided as input and predictions were made five time steps ahead, as depicted in figure 3.9. This structure ensures that most time dependent variations happen in the reused or recurrent network. The resolution of the input image was 192x256. This was chosen as it is the closest value of the halved original resolution (384x515), which is divisible by two multiple times. The resolution was halved to reduce the number of parameters in the network.

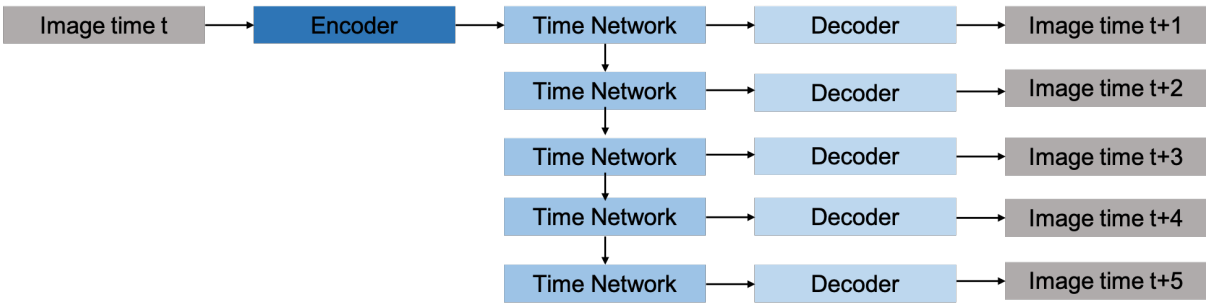


Figure 3.9: General architecture

The encoder and decoder models are shown in figure 3.10 (below). It may be noted that encoder and decoder models are symmetric except for the type of skip connection. The encoder has concatenation, whilst the decoder has addition skip connections. The concatenation skip connection allow for reusing earlier representations, hence keeping more information, whilst the main advantage of addition skip connections is that they keep the number of features fixed. The legend on the right shows what the colors represent. The convolutional 2D layers are described by s, k and f parameters, being stride, kernel size and number of filters, respectively. The max pooling and up sampling layers show the pool size in between the brackets. These parameters and type of layers are explained in section 2.2.2.

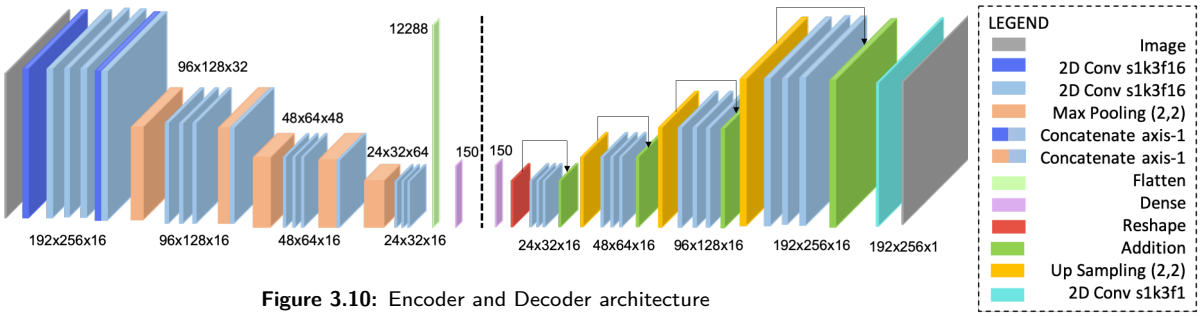


Figure 3.10: Encoder and Decoder architecture

The tested options for the time network, inserted between the encoder and decoder, are a reused NN, and LSTM and GRU RNNs. These are depicted in 3.11 (below). The numbers after the letter n indicate the number of nodes in the layer. In all time networks a dropout layer, as explained in section 2.2.1, with a dropout probability of 0.1 was implemented. The RepeatVector5 layer before the LSTM and GRU layers copies the output of the encoder five times, which is used as input to the LSTM and GRU layer for the five different time steps. This is necessary for these layers to return five predictions, as explained in section 2.2.4. As mentioned before, the performance of the total network with different time networks is compared first, of which the results are shown in 4.1.

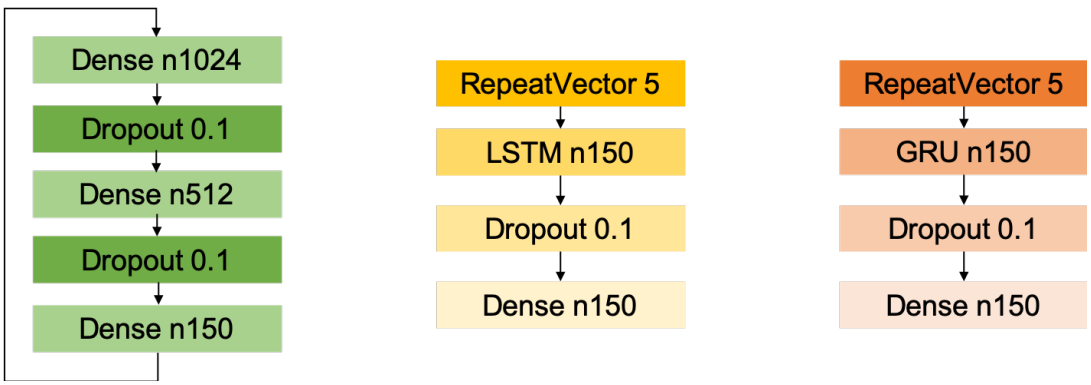


Figure 3.11: Time Network architectures, from left to right: NN, RNN with LSTM units, RNN with GRUs

In the last phase, the time network is altered to incorporate information on the inlet velocity pattern. This is implemented in three different ways. Firstly, the inlet velocity for the predicted time step is given as extra input, additional to the latent representation produced by the encoder. This variant of the time network will be referred to as the velocity variant. Secondly, the difference between the current inlet velocity and inlet velocity of the predicted time step is used as extra input. This variant will be referred to as the difference variant. Thirdly, the difference between inlet velocities is also used as extra input, but the output of the time network is added to the input. This forces the network to learn the difference between old and new latent representations instead of the new latent representation directly. This variant of the time network will be referred to as the delta variant. In all variations the number of nodes in the hidden layer was equal to the number input nodes, being 151. All the variations are shown in figure 3.12 and the results on comparing the variants of the time network are shown in section 4.3.1.

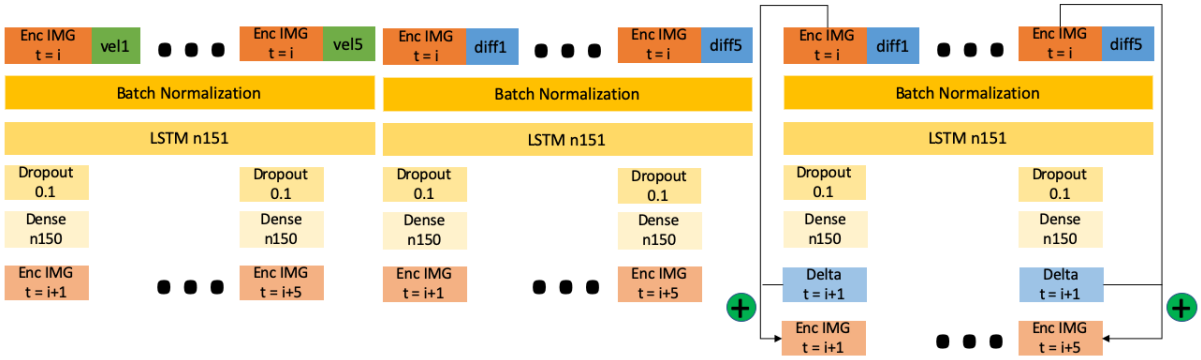


Figure 3.12: Time network variations, from left to right: velocity variant, difference variant and delta variant

3.2.2 Training and tests

As described at the start of this chapter, the general procedure is cyclic and consists of three cycles. In the first cycle, the best time network architecture is chosen. In the second cycle that network is tested on more complex geometries and the last cycle the generalizablity to new geometries and inlet velocity patterns is tested. As described in section 3.1, the simulated time series contain 101 time steps. As the final architecture uses one image as input and five images as output, 96 input-output groups could be made from one time series. All training was performed with 90/10 training/validation split.

In the first cycle, all three time networks in the final architecture are trained on the straight channel data set containing 29 different simulation. The data set is extended by data augmentation, that is, shifting the channel through the domain and flipping the image around the y-axis, resulting in 27.840 unique input-output groups. In the second cycle, a data set containing bent channels and a data set containing bifurcated channels are used to train the best performing architecture separately. The bent data set is extended only by flipping, resulting in 7.488 unique input-output groups for training. The bifurcation data set is not flipped nor shifted and contained a total of 9.504 unique input-output groups in the training set. One time series was excluded from training and reserved for testing.

In the final cycle, testing generalizability of the NNs, random geometries from the total (training, validation and test) bifurcation data set are combined with an inlet velocity pattern (see section 3.1.2). This gives a new training/validation data set consisting of fourteen geometries with vp0, ten with vp1, nine with vp2, nine with vp4, eight with vp5 and eight with vp6. As vp5 and vp6 are random variations of a pattern, each unique variation occurs only once in the training data. For testing, three new variations of vp5 and vp6 are simulated on geometries present in the training set and a closer approximation to the real-life inlet velocity pattern (again, see section 3.1.2) is simulated on a known geometry as well. Besides this, three vp's that were present in the training set were simulated on geometries not present in the training set to test geometry generalizability.

On the test time series the performance is analysed by making predictions on individual input images (the task the network is trained on), as well as predicting one whole heartbeat from t=0 as the only input image. This was done by taking the prediction one time step ahead and feeding this back into the network 95 times, giving the first 96 time steps. Then, the heartbeat is finalized with the last predictions of two, three, four and five time steps ahead. Results on using other time step sizes to predict one heartbeat can be found in appendix D. The test data set was never shifted nor flipped.

Early stopping was implemented for the training with patience set to 10, together with a maximum amount of epochs of 250. The batch size was 18. As optimizer Adam (AMSGrad variant) [80] was used. All training was done on the DelftBlue [81] supercomputer, using one Tesla V100S-PCIE-32GB GPU.

3.2.3 Optimization and analysis of the network

To optimize the performance of the network for generalizabilty and analyze the behaviour of the network further, multiple experiments were done. Firstly, two paths were explored to increase the performance, being alteration of the loss function and transformation of the data. As the data contains large differences between high velocity and low velocity within one image, between time steps and between time series, there is a difference in size of the error between these cases as well. Moreover, by using the standard MSE as loss function these differences are amplified because the velocity values are below one. Squaring values below one entails small differences become even smaller compared to larger differences. Hence, different loss functions to counter this are tested. The loss functions tested are MSE + MAE and using the arctan function to transform absolute error before computing the mean (MatanE). MAE stands for mean absolute error and is also given in equation 3.13. The loss functions are shown in equations 3.11 and 3.12 in the before mentioned order.

$$MSE + MAE = \frac{1}{n}\sum_{i=0}^{n}(y_i - \hat{y}_i)^2 + \frac{1}{n}\sum_{i=0}^{n}|y_i - \hat{y}_i| \tag{3.11}$$

$$MatanE = \frac{1}{n}\sum_{i=0}^{n} tanh(|y_i - \hat{y}_i|) \tag{3.12}$$

The difficulties arising from high and low velocity differences could also be tackled by transforming the input data. Histogram equalization of the input images could be an option following from this line of thought. This transformation scales the image across the whole range of values such that the cumulative distribution function is roughly linear, see [82].

Also, simply adding more time series to the data set could improve performance. This was tested by incrementally training the network on more data and examining the error versus amount of data. Besides this, adding more information to the network by letting it predict more than five time steps ahead might increase performance. This was tested by training the network to predict ten time steps ahead.

In terms of stability, predicting one entire heartbeat by using the prediction of one time step ahead already gives an indication of how the error builds up over repeated predictions. To extend this idea, the prediction of one time step ahead can be used to predict multiple heartbeats, whilst monitoring the error per additional prediction.

3.3 Error measures

To quantify the performance of the network architectures, the boundaries and domain errors are investigated separately. The boundary is defined as the pixels at the very edge of the domain and is one pixel wide. The domain is defined as all the remaining pixels inside the domain. The domain error metrics are root mean square error (RMSE), mean absolute error (MAE) and maximum error (ME). The RMSE is used instead of MSE for interpretability, as the dimension is velocity instead of velocity squared. The RMSE and MAE are calculated for each image and thereafter averaged over all images in the test time series. The maximum error is also calculated for each image and before the maximum over all images in the time series is taken. The boundary error metrics measure the flow on the boundary, i.e., violation of the no slip boundary condition. This is measured as maximum over all pixels (ME) and all images and as mean over all pixels (MAE) and all images, calculated in the same way as for the domain. All errors (with the exception of the RMSE) were computed absolute and relatively to the true value (i.e., the value of the input image) and are represented in equations 3.13,3.14,3.15, 3.16 and 3.17. In all equations y is the true value, $\hat{y}$ is the predicted value, and n is either the number of pixels on the boundary or the number of pixels in the domain.

$$MAE = \frac{1}{n}\sum_{i=0}^{n}|y_i - \hat{y}_i| \tag{3.13}$$

$$ME = max(|y_i - \hat{y}_i|) \tag{3.14}$$

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=0}^{n}(y_i - \hat{y}_i)^2} \tag{3.15}$$

$$RMAE = \frac{1}{n}\sum_{i=0}^{n}\left(\frac{|y_i - \hat{y}_i|}{y_i}\right) \tag{3.16}$$

$$RME = max\left(\frac{|y_i - \hat{y}_i|}{y_i}\right) \tag{3.17}$$

In addition, the conservation of flow was measured by computing the in and out flow from the in- and outlet velocities for each image, adding all in and out flow for one time series and thereafter taking the difference between total in and out flow. This was done for the input time series and the predicted time series, after which the difference was computed to give the error.

4 Results

As described in chapter 3, three cycles were passed. In the first cycle, three different time networks were tested, of which the results are shown in section 4.1. In the second cycle, the best performing architecture was tested on more complicated domains. These results are shown in section 4.2. Thirdly, three variations of the time network were implemented and the generalizability of the network was tested, with the results shown in section 4.3. Lastly, the network was optimized and further analyzed, of which the results can be found in section 4.4.

The error metrics, as described in section 3.3, are presented throughout this chapter in multiple ways. The tables, for example table 4.1, show all error metrics for the predictions of individual input images (the task the network was trained on), in columns 't+1' until 't+5'. Besides these error metrics, the errors made on predicting one whole heartbeat, as described in section 3.2.2, are presented in the column 'one beat'. In addition to the tables, various figures are shown to highlight specific findings.

4.1 Networks for time evolution

The same encoder-decoder model was tested with three different networks for time evolution. First of all, a reused NN, secondly an RNN with LSTM units and lastly an RNN with GRUs, as described in section 3.2.1, was evaluated. The resulting errors are shown in the tables 4.1, 4.2 and 4.3. The red cells mark the worst performing time network and the green cells mark the best performing time network in the comparison on that error metric and prediction. Thus, the tables should be viewed together and every cell is either red, green or white in one of the three tables.

			t+1	t+2	t+3	t+4	t+5	One beat
RELATIVE	AVG	boundary	0,36477	0,29985	0,29752	0,27985	0,28926	0,38589
		domain	0,07459	0,07388	0,07338	0,07670	0,07796	0,17432
	MAX	boundary	15,0	8,0	9,0	5,0	7,0	23,0
		domain	19,0	19,0	18,0	18,0	17,0	76,0
ABSOLUTE	AVG	boundary	0,00331	0,00263	0,00247	0,00216	0,00214	0,00348
		domain	0,00301	0,00312	0,00309	0,00326	0,00325	0,00995
	MAX	boundary	0,04902	0,02941	0,04118	0,03137	0,03725	0,04902
		domain	0,07451	0,07059	0,07843	0,07843	0,07843	0,21569
	domain RMSE		0,00618	0,00635	0,00630	0,00660	0,00666	0,01665
net flow error			0,000117	0,000104	0,000126	0,000103	0,000121	0,000124

Table 4.1: Error metrics for the encoder-decoder model with the reused NN for time evolution. Columns t+1 to t+5 show the error metrics for individual input images and the last column shows the error metrics for one whole heartbeat.

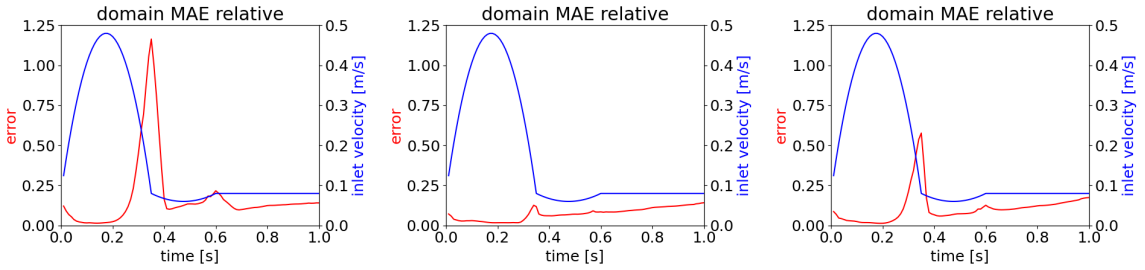
			t+1	t+2	t+3	t+4	t+5	One beat
RELATIVE	AVG	boundary	0,18307	0,17199	0,16679	0,16669	0,17314	0,19807
		domain	0,04775	0,04424	0,04295	0,04273	0,04362	0,07376
	MAX	boundary	7,0	6,0	6,0	6,0	5,0	7,0
		domain	18,0	18,0	18,0	17,0	17,0	20,0
ABSOLUTE	AVG	boundary	0,00160	0,00141	0,00130	0,00127	0,00129	0,00157
		domain	0,00191	0,00174	0,00163	0,00158	0,00162	0,00347
	MAX	boundary	0,02745	0,02157	0,02353	0,03137	0,04510	0,02353
		domain	0,05098	0,05294	0,05490	0,05294	0,05490	0,07843
	domain RMSE		0,00393	0,00362	0,00346	0,00339	0,00344	0,00900
net flow error			7,14E-05	7,46E-05	7,05E-05	6,91E-05	6,76E-05	0,000095

Table 4.2: Error metrics for the encoder-decoder model with the RNN containing LSTM units for time evolution. Columns t+1 to t+5 show the error metrics for individual input images and the last column shows the error metrics for one whole heartbeat..

			t+1	t+2	t+3	t+4	t+5	One beat
RELATIVE	AVG	boundary	0,34480	0,29026	0,27702	0,27002	0,27217	0,30068
		domain	0,05214	0,05030	0,05020	0,05104	0,05245	0,10887
	MAX	boundary	14,0	12,0	12,0	13,0	13,0	11,0
		domain	17,0	18,0	18,0	17,0	17,0	37,0
ABSOLUTE	AVG	boundary	0,00213	0,00203	0,00207	0,00207	0,00213	0,00178
		domain	0,00208	0,00195	0,00190	0,00189	0,00192	0,00617
	MAX	boundary	0,03529	0,03137	0,03137	0,03137	0,03137	0,03137
		domain	0,12745	0,07647	0,07059	0,06471	0,08824	0,11569
	domain RMSE		0,00422	0,00401	0,00393	0,00392	0,00395	0,01122
net flow error			0,000099	0,000098	0,000098	0,000100	0,000098	0,000111

Table 4.3: Error metrics for the encoder-decoder model with the RNN containing GRUs for time evolution. Columns t+1 to t+5 show the error metrics for individual input images and the last column shows the error metrics for one whole heartbeat.

As can be seen in tables 4.1, 4.2 and 4.3, the encoder-decoder model combined with the RNN containing LSTM units for time evolution performs best overall, although the model with GRUs has comparable performance on some error metrics. The model with the reused NN as time network has the largest or second largest errors in nearly all cases. The absolute errors for all networks are small, in contrast with the large relative errors, which indicates the most significant errors are made when the blood velocity is low. This is confirmed by looking at the relative MAE in the domain per time step when predicting one whole heartbeat, shown in figure 4.1.



(a) reused NN **(b)** RNN with LSTM units **(c)** RNN with GRUs

Figure 4.1: Comparison of the realtive domain MAE per time step (from left to right: reused NN, RNN with LSTM units and RNN with GRUs as time network). The red line shows the relative MAE in the domain and the blue line the inlet velocity in [m/s] versus time in [s].

Also, for all networks, it is noted that the maximum error on the boundary as well as in the domain is large compared to the average error. Further investigation of the location of these maximum errors uncovers that they occur in the same area for all predictions and networks, namely the corners of the domain. This can be seen in figure 4.2 for the prediction of five time steps ahead. The orange dots represent the location of the relative ME in the domain and the blue dots the location of relative ME on the boundary for the 96 predictions of five time steps ahead. The size of the dots represent how often the relative ME occurs on that exact location. An image is added in the background to show the domain outline of the straight channel from the test time series.

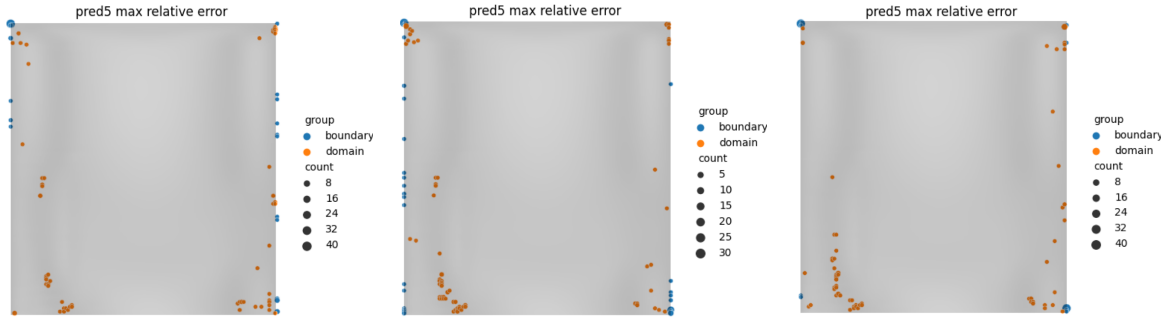


Figure 4.2: Relative ME locations of the different time networks for the predictions of five time steps ahead (from left to right: reused NN, RNN with LSTM units and RNN with GRUs)

4.2 Bent and bifurcated channels

The best overall performing architecture, using the LSTM network for time evolution, was trained on the more complex domains, being bent and bifurcated channels, separately. Training the network on all domains in one data set was also tested. This gave no viable results.

			t+1	t+2	t+3	t+4	t+5	One beat
RELATIVE	AVG	boundary	0,42746	0,40887	0,40167	0,40250	0,40365	0,42041
		domain	0,03665	0,03421	0,03371	0,03425	0,03502	0,06909
	MAX	boundary	43,0	37,0	36,0	36,0	36,0	43,0
		domain	22,0	18,0	17,0	16,0	15,0	37,0
ABSOLUTE	AVG	boundary	0,01502	0,01437	0,01426	0,01426	0,01429	0,01504
		domain	0,00164	0,00152	0,00146	0,00152	0,00153	0,00510
	MAX	boundary	0,11176	0,10784	0,10980	0,11373	0,10980	0,11176
		domain	0,05098	0,04118	0,03922	0,03922	0,03922	0,12157
	domain RMSE		0,002946	0,002713	0,002633	0,002698	0,002726	0,00841
net flow error			3,41E-06	5,59E-06	0,000024	3,29E-05	3,46E-05	6,84E-05

Table 4.4: Error metrics for the encoder-decoder model with LSTM network for time evolution on the bent channel data set. Columns t+1 to t+5 show the error metrics for individual input images and the last column shows the error metrics for one whole heartbeat.

			t+1	t+2	t+3	t+4	t+5	One beat
RELATIVE	AVG	boundary	0,27063	0,25478	0,25038	0,24826	0,24748	0,27806
		domain	0,04298	0,04175	0,04141	0,04189	0,04317	0,15201
	MAX	boundary	24,0	28,0	26,0	26,0	27,0	24,0
		domain	23,0	20,0	21,0	21,0	22,0	40,0
ABSOLUTE	AVG	boundary	0,01187	0,01136	0,01123	0,01116	0,01111	0,01163
		domain	0,00125	0,00122	0,00121	0,00122	0,00120	0,00389
	MAX	boundary	0,18235	0,15490	0,14118	0,13529	0,13333	0,17647
		domain	0,04510	0,04118	0,04118	0,04118	0,04314	0,08627
	domain RMSE		0,00272	0,00268	0,00264	0,00264	0,00264	0,00840
net flow error			3,51E-06	6,59E-06	5,46E-06	6,06E-06	7,86E-06	4,97E-06

Table 4.5: Error metrics for the encoder-decoder model with LSTM network for time evolution on the bifurcated channel data set. Columns t+1 to t+5 show the error metrics for individual input images and the last column shows the error metrics for one whole heartbeat.

Again, what is noted from tables 4.4 and 4.5 is that the maximum errors are large compared to the average errors and the relative errors are large compared to the absolute errors. The latter illustrates once more the low velocity areas are predicted worse than high velocity areas. This is supported by the location of the maximum errors, as they occurred near the boundary of the domain, which is shown in appendix E. It is also noteworthy that the model has more difficulty predicting the boundary values of the bent channel accurately. When comparing the errors in predicting one whole heartbeat and the errors made predicting on individual images, the error only increases around a factor 3 for the relative MAE in the domain whilst the relative boundary MAE increases even less. This shows the network is complex enough to learn the flow over time. Visually, this is confirmed, as shown in figure 4.3 for the bent channel and figure 4.4 for the bifurcated channel. However, as only one inlet velocity pattern is used, the network could be overfitted on that specific inlet velocity pattern.

Throughout the results section, certain time steps during the prediction of one heartbeat are presented as shown in the figures below. The left column shows the inlet velocity pattern, with the time step marked by a dot, the second column shows the true velocity field, the third column shows the predicted velocity field, then the color scale for the velocity field images is shown and in the last column the difference between the true and predicted images (the absolute error) is shown together with it's color scale.

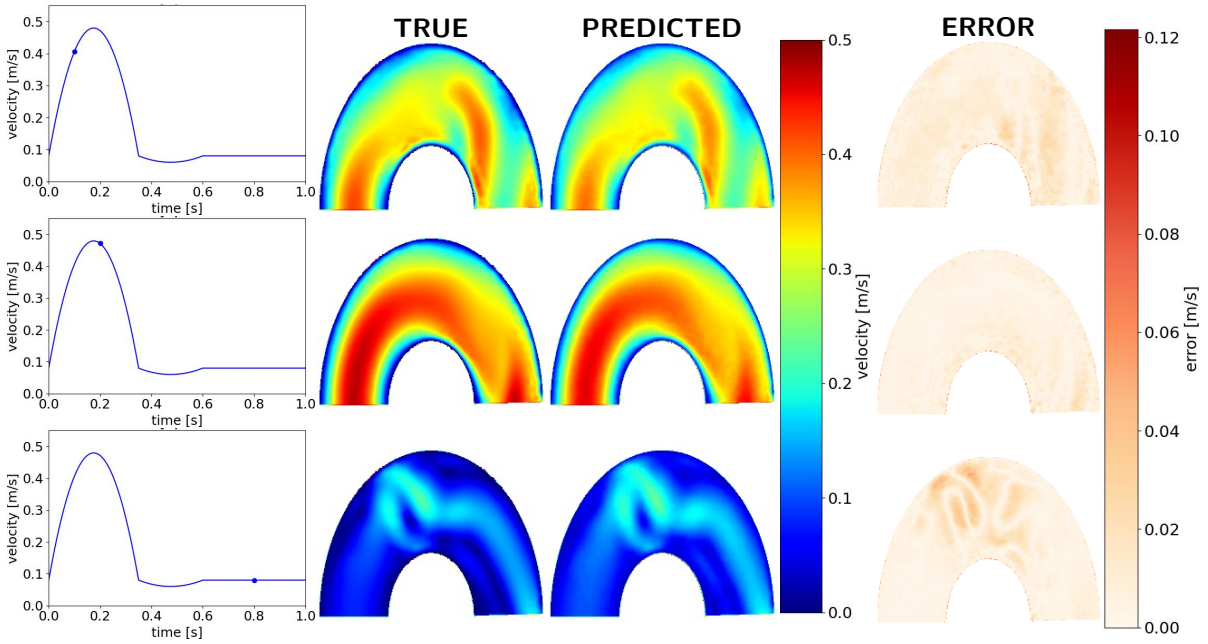


Figure 4.3: Three time steps (from top to bottom t = 0.1, t = 0.2, and t = 0.8) in predicting one heartbeat for the bent channel. The first column shows the inlet velocity with the moment in time marked, the second column the true velocity fields, the third column the predicted velocity fields and the last column shows the difference between true and predicted values.

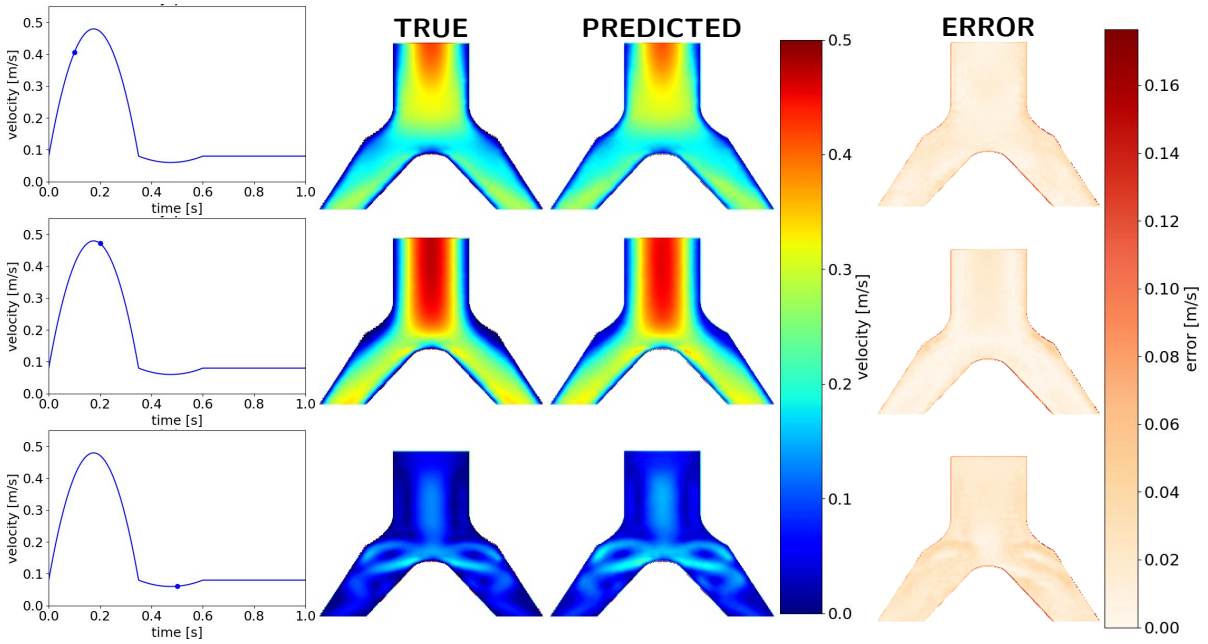


Figure 4.4: Three time steps (from top to bottom t = 0.1, t = 0.2, and t = 0.5) in predicting one heartbeat for the bifurcated channel. The first column shows the inlet velocity with the moment in time marked, the second column the true velocity fields, the third column the predicted velocity fields and the last column shows the difference between true and predicted values.

4.3 Generalizabilty of the neural networks

The previous network could be overfitted, as only one inlet velocity pattern is used and all geometries are present in the training data set. Hence, the generalizability of the network is tested in various ways. Firstly by introducing more inlet velocity patterns and testing it on a more realistic reconstruction of the inlet velocity of the aorta during a heartbeat, of which the results are shown in section 4.3.2. Secondly, the network was tested on geometries excluded from the training data set, meaning the specific combination of channel width and outlet width did not occur in the training data, of which the results are shown in section 4.3.3. First, the results are presented in section 4.3.1 of three ways to adjust the time network, to take into account the variation of velocity pattern (vp), as described in section 3.2.1.

4.3.1 Time network variations

Tables 4.6, 4.7 and 4.8 show the average error metrics over the three test time series, vp5.9, vp5.10 and vp5.11, shown in figure 3.3, for the three variations, velocity, difference and delta, described in section 3.2.1, of the time network. Again, the red and green cells mark the highest and lowest errors, respectively.

			t+1	t+2	t+3	t+4	t+5	One beat
RELATIVE	AVG	boundary	0,26913	0,25667	0,25600	0,26019	0,26515	0,28774
		domain	0,13750	0,14037	0,14622	0,15451	0,16430	0,41834
	MAX	boundary	34,7	35,7	34,0	39,0	46,7	40,3
		domain	17,3	19,3	22,3	29,0	39,7	64,0
ABSOLUTE	AVG	boundary	0,01527	0,01478	0,01451	0,01468	0,01482	0,01641
		domain	0,00369	0,00409	0,00450	0,00496	0,00536	0,01418
	MAX	boundary	0,15830	0,14641	0,14732	0,15190	0,16013	0,15007
		domain	0,06954	0,08235	0,10889	0,14092	0,17203	0,29373
	domain RMSE		0,00675	0,00735	0,00799	0,00869	0,00931	0,02479
net flow error			0,000014	0,000044	0,000057	0,000068	0,000081	0,000075

Table 4.6: Error metrics for the velocity variant of the time network, averaged over three variations of vp5. Columns t+1 to t+5 show the average error metrics for individual input images and the last column shows the average error metrics for one whole heartbeat.

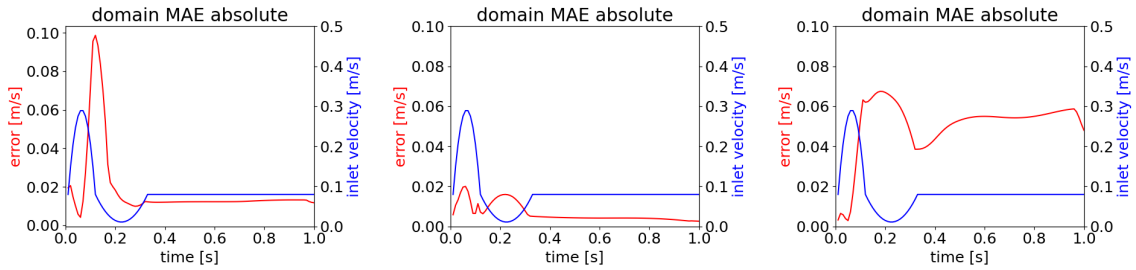
			t+1	t+2	t+3	t+4	t+5	One beat
RELATIVE	AVG	boundary	0,25126	0,24407	0,24222	0,24374	0,24639	0,24342
		domain	0,13186	0,13911	0,14764	0,15335	0,16062	0,26533
	MAX	boundary	34,3	36,0	42,7	34,7	35,7	27,3
		domain	22,7	23,3	23,7	23,3	23,7	24,7
ABSOLUTE	AVG	boundary	0,01414	0,01371	0,01352	0,01354	0,01366	0,01385
		domain	0,00301	0,00341	0,00386	0,00424	0,00462	0,00994
	MAX	boundary	0,16379	0,14092	0,15647	0,16837	0,17660	0,15281
		domain	0,06314	0,07229	0,07778	0,08693	0,10797	0,15647
	domain RMSE		0,00572	0,00635	0,00699	0,00755	0,00815	0,01814
net flow error			0,000040	0,000020	0,000021	0,000033	0,000044	0,000158

Table 4.7: Error metrics for the difference variant of the time network, averaged over three variations of vp5. Columns t+1 to t+5 show the average error metrics for individual input images and the last column shows the average error metrics for one whole heartbeat.

			t+1	t+2	t+3	t+4	t+5	One beat
RELATIVE	AVG	boundary	0,20907	0,21131	0,21183	0,21199	0,21257	0,30708
		domain	0,10592	0,12168	0,13701	0,14830	0,15850	1,11154
	MAX	boundary	42,3	42,0	37,7	36,7	36,0	41,0
		domain	16,3	18,3	19,0	21,3	23,0	75,7
ABSOLUTE	AVG	boundary	0,01469	0,01470	0,01478	0,01491	0,01502	0,02086
		domain	0,00289	0,00358	0,00419	0,00460	0,00494	0,03577
	MAX	boundary	0,18941	0,18575	0,18118	0,18026	0,18026	0,21412
		domain	0,05765	0,06863	0,08235	0,09150	0,09699	0,26170
	domain RMSE		0,00523	0,00626	0,00716	0,00779	0,00832	0,05790
net flow error			0,000035	0,000045	0,000051	0,000057	0,000061	0,000603

Table 4.8: Error metrics for the delta variant of the time network, averaged over three variations of vp5. Columns t+1 to t+5 show the average error metrics for individual input images and the last column shows the average error metrics for one whole heartbeat.

All variants are given the same amount of extra information but the information is supplied in different ways. As can be seen in the tables, this results in comparable performance on the prediction of one to five time steps ahead. The difference variant of the time network performs best, especially when looking at the whole heartbeat prediction. It is noteworthy that, although the delta network performs better on nearly all metrics compared to the velocity network, and even better on the relative average metrics compared to the difference network, it performs much worse when predicting one whole heartbeat. The delta network grossly overestimates the low velocity part of the heartbeat. Whilst the velocity network recovers from overestimating the velocity much better, it still overestimates the velocity more than the difference network. As there is no correction mechanism, the overestimated velocity is propagated during the rest of the heartbeat, resulting in higher errors. This is shown in figure 4.5 for vp5.9. Hence, the following tests on generalizability and network optimization will be done with the difference variant of the time network.



(a) velocity variant **(b)** difference variant **(c)** delta variant

Figure 4.5: Comparison of the absolute domain MAE per time step on vp5.9 (from left to right: velocity, difference and delta variant of the time network). The red line shows the absolute MAE in the domain in [m/s] and the blue line the inlet velocity in [m/s] versus time in [s].

In addition, the hypothesis that the previous network was overfitted on one inlet velocity pattern is supported by the results presented in this section. The error for the prediction of one to five time steps ahead increased from around 4% to about 15% and the error on predicting one heartbeat nearly doubled from 15% to 26% (domain relative MAE). Although, it should be noted, the size of the training data set decreased whilst the variation increased. This could also have contributed to the reduction in performance.

4.3.2 Velocity pattern variation

Besides the three variations of velocity pattern 5, the difference variant of the time network was also tested on three variations of velocity pattern 6, being vp6.8, vp6.10 and vp6.11, see figure 3.4. The average over these velocity patterns is shown in table 4.9.

			t+1	t+2	t+3	t+4	t+5	One beat
RELATIVE	AVG	boundary	0,25968	0,25551	0,25370	0,25626	0,25925	0,26182
		domain	0,13114	0,14338	0,15729	0,16859	0,17892	0,46395
	MAX	boundary	33,7	34,0	39,3	44,0	47,0	27,7
		domain	29,3	27,0	26,3	27,0	29,3	33,3
ABSOLUTE	AVG	boundary	0,01401	0,01373	0,01352	0,01360	0,01375	0,01424
		domain	0,00340	0,00413	0,00499	0,00570	0,00625	0,01675
	MAX	boundary	0,16654	0,14732	0,15007	0,16196	0,17020	0,16196
		domain	0,12810	0,12627	0,12444	0,12353	0,12353	0,17843
	domain RMSE		0,00627	0,00740	0,00866	0,00974	0,01059	0,02773
net flow error			0,000034	0,000013	0,000014	0,000027	0,000042	0,000261

Table 4.9: Error metrics for the difference variant of the time network, averaged over three variations of vp6. Columns t+1 to t+5 show the average error metrics for individual input images and the last column shows the average error metrics for one whole heartbeat.

Compared to the results of the difference network on the variations of velocity pattern 5, it can be noted that the average relative domain error has significantly increased from 26% to 46%, whilst the average absolute MAE only increased from around 0,010 to 0,015. This was mainly caused by the prediction of vp6.8, which has a spike as second velocity peak, see figure 3.4. The network overestimated the velocity after the spike, creating a relative domain MAE of 85% during one heartbeat. If the prediction of vp6.8 is excluded, an average relative MAE of 27% was found on the remaining two variations of vp6. The full results are shown in appendix F.

Again, it can be seen the network's prediction is worse in low velocity areas, as it is stagnant at the end of the heartbeat, even if there is a low velocity peak. An example of this behaviour is shown in figure 4.6 at t = 0.48 (the third row). In addition, the network does not always capture the full height of the velocity peak. An example of this behaviour is shown for velocity pattern 5.10 in figure 4.7 at t = 0.06 (the first row). It can also be seen in figures 4.8 and 4.9, that the MAE in the domain spikes during the descent of the velocity peaks. The spike in the error when the inlet velocity is decreasing originates from the model overestimating the velocity, as can be seen in figures 4.6 and 4.7 at t = 0.12 (the second row).

This could be caused by the information which is passed to the RNN, being the difference in inlet velocity between time steps, in combination with the use of a batch normalization layer. The differences are negative when the inlet velocity is decreasing. Thus, when normalizing the data between 0 and 1 in the batch normalization layer, negative differences become a value close to 0. Hence, the model might give more importance to areas with increasing velocity.

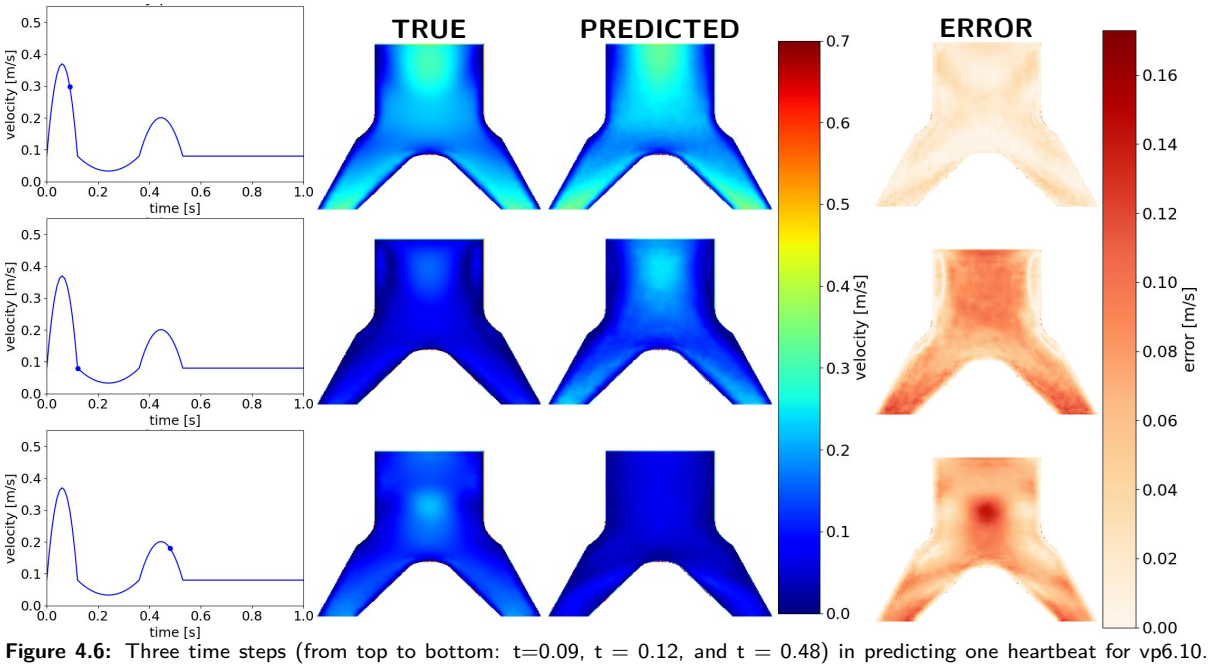


Figure 4.6: Three time steps (from top to bottom: t=0.09, t = 0.12, and t = 0.48) in predicting one heartbeat for vp6.10. The first column shows the inlet velocity with the moment in time marked, the second column the true velocity fields, the third column the predicted velocity fields and the last column shows the difference between true and predicted values.

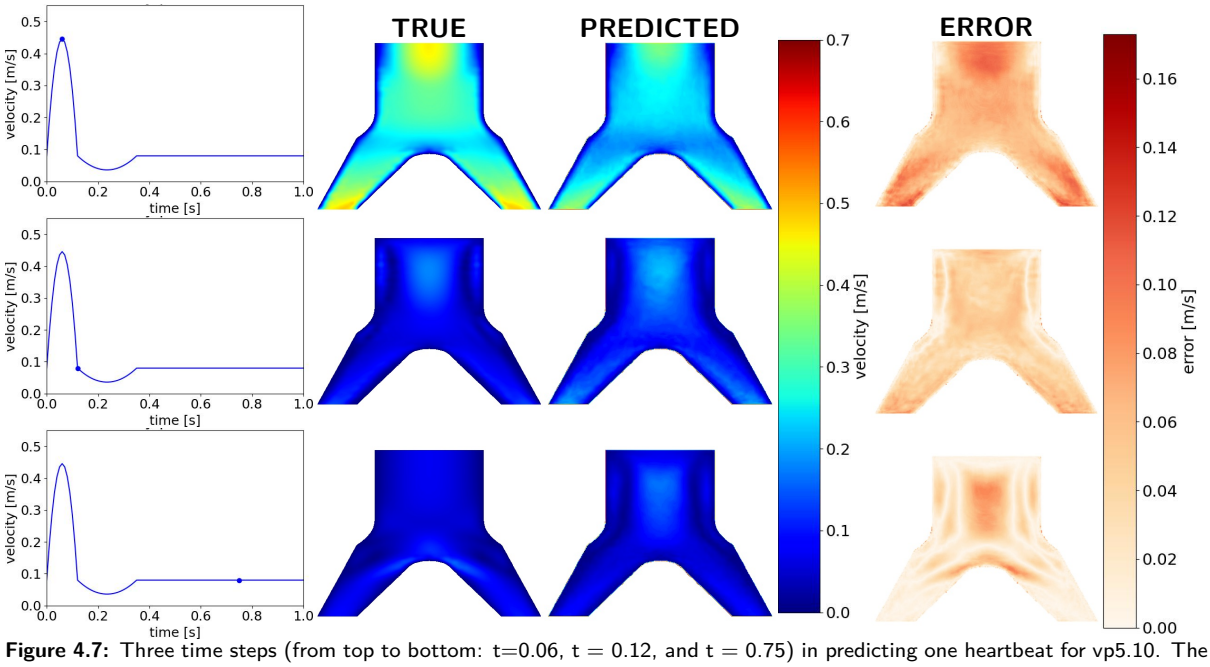


Figure 4.7: Three time steps (from top to bottom: t=0.06, t = 0.12, and t = 0.75) in predicting one heartbeat for vp5.10. The first column shows the moment in the time series, the second column the true velocity fields, the third column the predicted velocity fields and the last column shows the difference between true and predicted values.

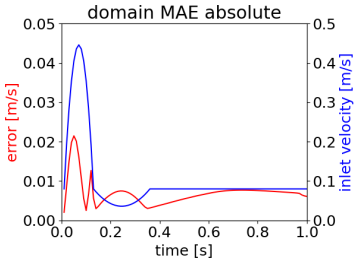


Figure 4.8: Absolute MAE in the domain for vp5.10. The red line shows the absolute error in [m/s] and the blue line the inlet velocity in [m/s] versus time in [s].

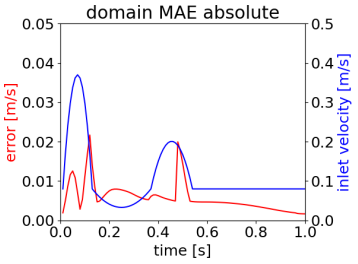


Figure 4.9: Absolute MAE in the domain for vp6.10. The red line shows the absolute error in [m/s] and the blue line the inlet velocity in [m/s] versus time in [s].

Finally, the inlet pattern of a realistic heartbeat, described in section 3.1.2, was used to test the model. As shown in table 4.10, the network shows reasonable performance on the task it was trained on, i.e., predicting five time steps ahead, with a relative MAE between 11% and 13% in the domain. However, again a big increase to 28% relative MAE in the domain was found when predicting one whole heartbeat. This performance is comparable to the performance on variations of velocity pattern 5. Also, for this velocity pattern the same behaviour, capturing the duration but not the total height of the velocity peak and performing worse on low velocity areas, as for variations of vp5 and vp6, was encountered. However, the underestimation of the velocity peak is worse than previously encountered. This is shown in figure 4.11. Nonetheless, this does indicates that the network is not overfitted on the training inlet velocity patterns. Moreover, the error does not spike during the descent of the velocity peak, which can be seen in figure 4.10. The absence of the error spike during inlet velocity descent and the gross underestimation of the peak velocity could be explained by the more gradual ascent and descent of the velocity peak, compared to the parabolic inlet velocity patterns. The differences during ascent are smaller compared to the parabolic inlet patterns, which might lead to less increase of the velocity field. The differences during descent are smaller negative values, leading to larger values after normalization, compared to the parabolic inlet patterns.

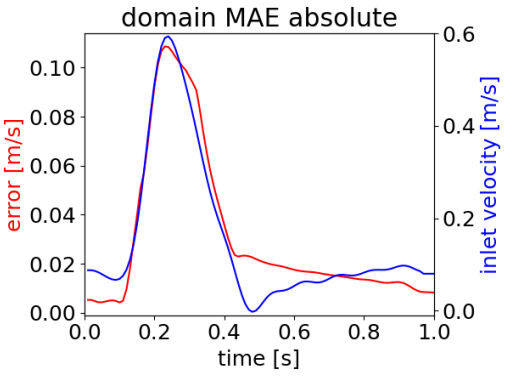


Figure 4.10: Absolute MAE in the domain for realistic inlet velocity pattern. The red line shows the absolute MAE in the domain in [m/s] and the blue line the inlet velocity in [m/s] versus time in [s].

			t+1	t+2	t+3	t+4	t+5	One beat
RELATIVE	AVG	boundary	0,40117	0,34966	0,34635	0,35531	0,36485	0,32377
		domain	0,11805	0,12059	0,12600	0,13014	0,13374	0,28873
	MAX	boundary	45,0	35,0	35,0	36,0	42,0	37,0
		domain	19,0	20,0	20,0	22,0	21,0	34,0
ABSOLUTE	AVG	boundary	0,01426	0,01319	0,01251	0,01254	0,01278	0,01231
		domain	0,00815	0,00903	0,00973	0,01020	0,01059	0,03132
	MAX	boundary	0,26902	0,14549	0,16471	0,16745	0,15647	0,37333
		domain	0,12902	0,12902	0,13726	0,15373	0,18941	0,40078
	domain RMSE		0,01399	0,01527	0,01629	0,01702	0,01770	0,05275
net flow error			0,000092	0,000117	0,000126	0,000129	0,000131	0,000403

Table 4.10: Error metrics for the difference variant of the time network on realistic velocity pattern. Columns t+1 to t+5 show the error metrics for individual input images and the last column shows the average error metrics for one whole heartbeat.

In general, it can be noted that low velocity areas in an image, low velocity parts of the time series and entire time series with low velocities are predicted worse than their high velocity counterparts. This can be explained by the fact that, in high velocity areas, the pixel intensities are also around one order of magnitude higher, thus the loss in these areas is relatively higher. Moreover, the MSE was used as loss function for all training, which amplifies this behaviour due to squaring values below one.

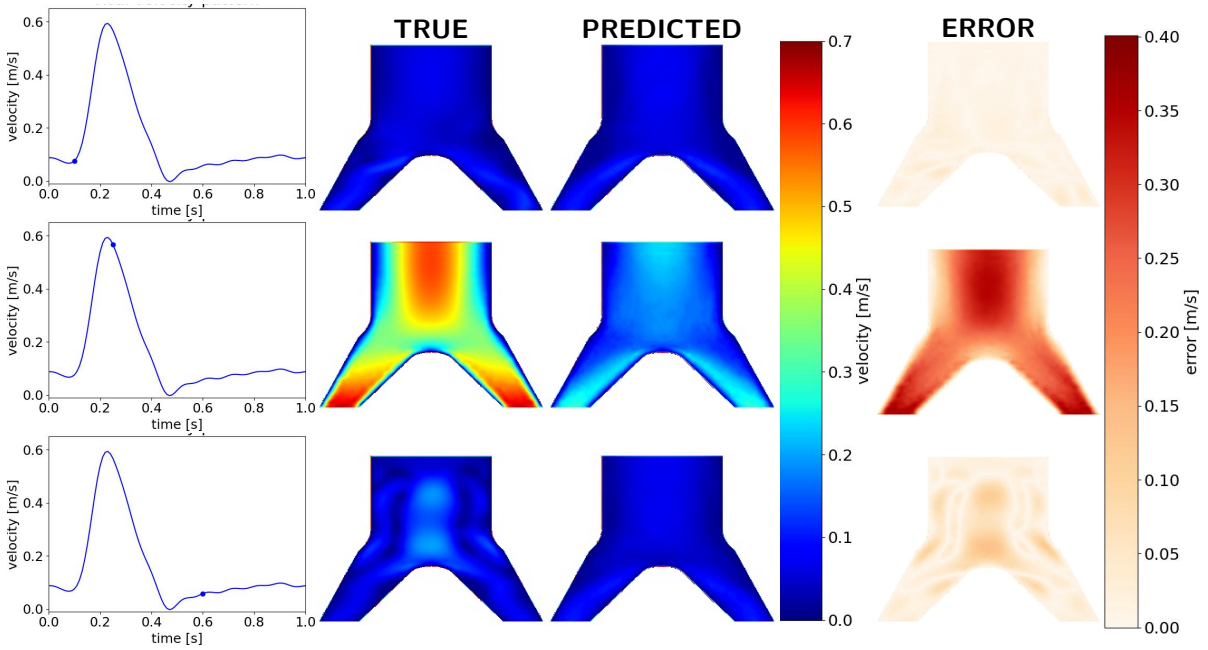


Figure 4.11: Three time steps (from top to bottom: t=0.10, t = 0.25, and t = 0.60) in predicting one heartbeat for realistic velocity pattern. The first column shows the moment in the time series, the second column the true velocity fields, the third column the predicted velocity fields and the last column shows the difference between true and predicted values.

4.3.3 Geometry variation

When making predictions on three geometries that were not present in the training data set, it can be seen the network cannot generalize to new geometries at all. The new geometries are all interpolations of existing geometries in the training data set. The domain is predicted incorrectly, more specifically, the network chooses the domain in the training set that is closest to the new geometry. Evidently, this could be related to the fixed number of geometric variations of the domain in the training data. With five different channel widths and four different outlet widths possible, there are only 20 possible geometries. Hence, the network could have learned to encode these possibilities, instead of encoding actual domain parameters, i.e. overfitting on the training domains. In addition, the velocity field is entirely off in all predictions. Examples of these findings are shown in figure 4.12, in which the top row shows the true values and the bottom row the predicted values.

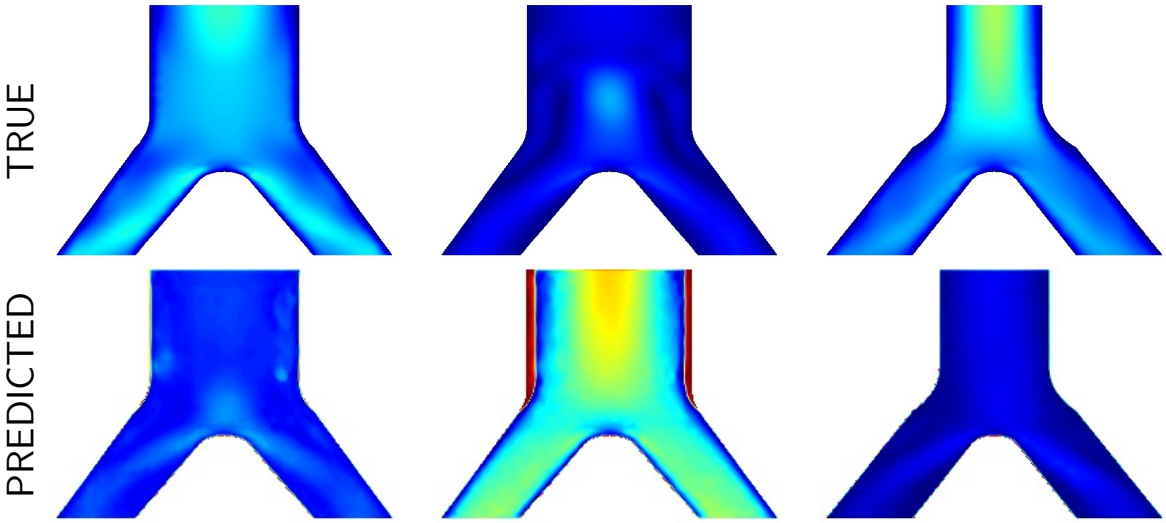


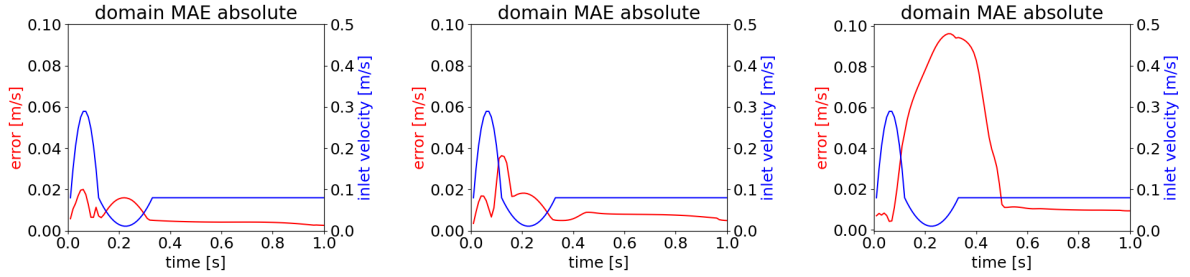
Figure 4.12: Three new geometries (from left to right) during various moments in the time series. The top row shows the true velocity field and the bottom row shows the corresponding predicted velocity field.

4.4 Network optimization and analysis

In an attempt to improve the performance of the network, various alternatives were tested. The variation of the loss function, of which results are shown in section 4.4.1, and data augmentation, of which results are shown in section 4.4.2, could counter the difference in importance between high an low velocities learned by the network. In addition, it was tested whether increasing the data set size and increasing the number of predicted time steps would improve the performance. The results of these experiments are also shown in section 4.4.2. Lastly, the stability of the model was tested by predicting three heartbeats, of which results are shown in section 4.4.3.

4.4.1 Loss function variation

As mentioned in the section 3.2.3, various loss functions were tested to increase the performance on low velocity areas. From the two options MSE+MAE and MatanE, only the MatanE loss function gave viable results. The error metrics for the MatanE loss function are shown below in table 4.11. The absolute MAE in the domain per time step can be compared for the different loss functions in figure 4.13. The full results of the MSE+MAE loss function can be found in appendix G.



(a) MSE loss function **(b)** MatanE loss function **(c)** MSE+MAE loss function

Figure 4.13: Absolute MAE in the domain using various loss functions (from left to right: MSE, MatanE and MSE+MAE) for vp5.9. The red line shows the absolute error in [m/s] and the blue line the inlet velocity in [m/s] versus time in [s].

			t+1	t+2	t+3	t+4	t+5	One beat
RELATIVE	AVG	boundary	0,42649	0,44269	0,44350	0,44598	0,44602	0,42768
		domain	0,10660	0,11494	0,12118	0,12733	0,13363	0,37920
	MAX	boundary	66,7	68,7	68,7	73,7	77,7	69,3
		domain	21,0	22,7	25,0	24,3	21,7	42,3
ABSOLUTE	AVG	boundary	0,03375	0,03533	0,03557	0,03622	0,03652	0,03414
		domain	0,00296	0,00337	0,00373	0,00407	0,00443	0,01241
	MAX	boundary	0,40810	0,37791	0,37791	0,39621	0,41177	0,37425
		domain	0,07046	0,08418	0,09242	0,10248	0,10248	0,16562
	domain RMSE		0,00546	0,00609	0,00663	0,00713	0,00770	0,02251
net flow error			0,000070	0,000071	0,000081	0,000090	0,000100	0,000119

Table 4.11: Error metrics for the difference variant of the time network using the MatanE loss function, averaged over three variations of vp5. Columns t+1 to t+5 show the average error metrics for individual input images and the last column shows the average error metrics for one whole heartbeat.

Although no general improvement in the error metrics can be seen, the relative domain MAE decreased very slightly from 13% - 16% to 10% - 13% for predicting one to five time steps ahead, indicating similar performance on low velocity areas. Interestingly, for predicting the entire heartbeat, the increased attention to small errors results in underestimating the flow in high velocity areas. The network tends to predict a low velocity fields for all time steps. An example of this behaviour can be seen in figure 4.14.

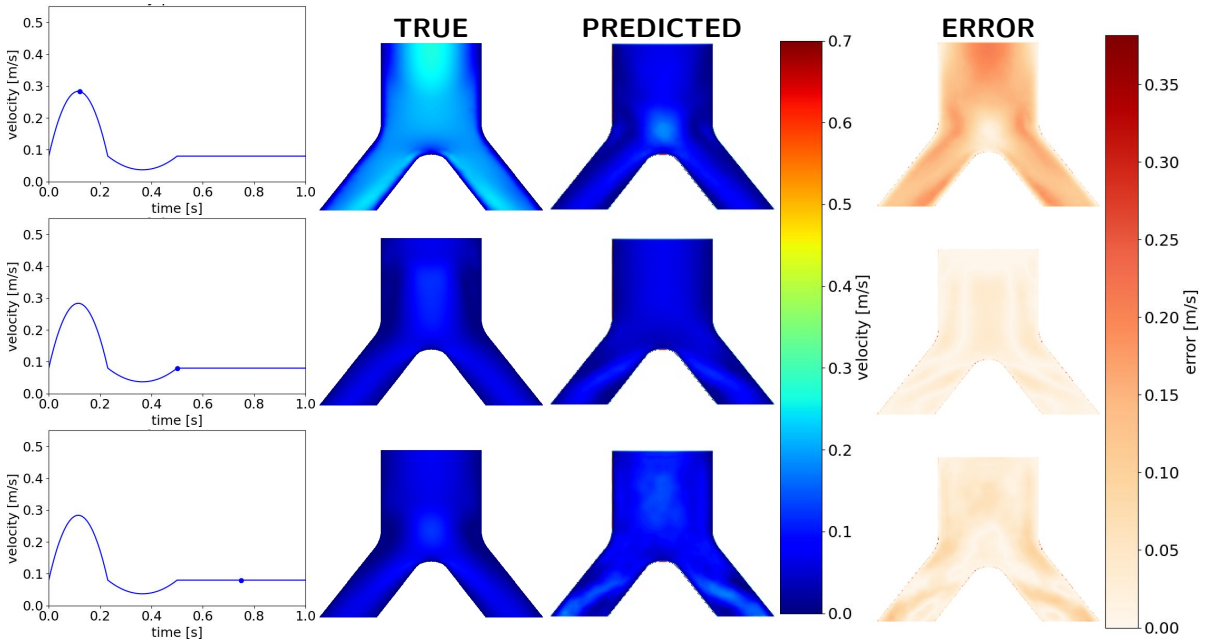


Figure 4.14: Three time steps (from left to right: t=0.12, t = 0.50, and t = 0.75) in predicting one heartbeat for vp5.11 using loss function MatanE. The first column shows the moment in the time series, the second column the true velocity fields, the third column the predicted velocity fields and the last column shows the difference between true and predicted values.

4.4.2 Data augmentation and network alteration

First the results of histogram equalization, in which the contrast is enhanced by spreading the pixel intensities within one image, are shown in table 4.12 and figure 4.15. Histogram equalization should increase the importance of low velocity areas in the domain, compared to no histogram equalization.

			t+1	t+2	t+3	t+4	t+5	One beat
RELATIVE	AVG	boundary	0,22286	0,21259	0,20939	0,21087	0,21363	0,40107
		domain	0,15102	0,15523	0,15931	0,16306	0,16871	1,60866
	MAX	boundary	54,3	40,3	39,7	44,3	46,3	74,3
		domain	21,0	21,3	23,0	23,7	24,3	116,3
ABSOLUTE	AVG	boundary	0,01746	0,01615	0,01569	0,01585	0,01609	0,02184
		domain	0,00481	0,00525	0,00560	0,00579	0,00598	0,06780
	MAX	boundary	0,18484	0,17477	0,16928	0,16837	0,16928	0,23699
		domain	0,07686	0,08784	0,08693	0,09608	0,10706	0,35961
	domain RMSE		0,00871	0,00941	0,00995	0,01025	0,01056	0,11497
net flow error			0,000116	0,000143	0,000156	0,000164	0,000171	0,001275

Table 4.12: Error metrics when using histogram equalization as data augmentation, averaged over three variations of vp5. Columns t+1 to t+5 show the average error metrics for individual input images and the last column shows the average error metrics for one whole heartbeat.

In table 4.12 it can be seen that the error metrics on individual input-output sets (the error metrics on t+1 up to and including t+5) are quite comparable to the error metrics of the model without histogram equalization. The average error on the boundary is even decreased by 5%, which was expected as this is a low velocity area for all time steps. However, when one entire beat is predicted, the error metrics are much larger. On visual inspection, the velocity increases rapidly during the incline of the inlet velocity peak, but fails to decrease again and stagnates on predicting high velocity in the centre of the channel and low velocity at the boundaries. It seems the relation between areas within one image is learned, i.e. high velocity in the centre of the channel and low velocity at the boundaries. In contrast, the relation between consecutive time steps seems lost. This is shown in figure 4.15.

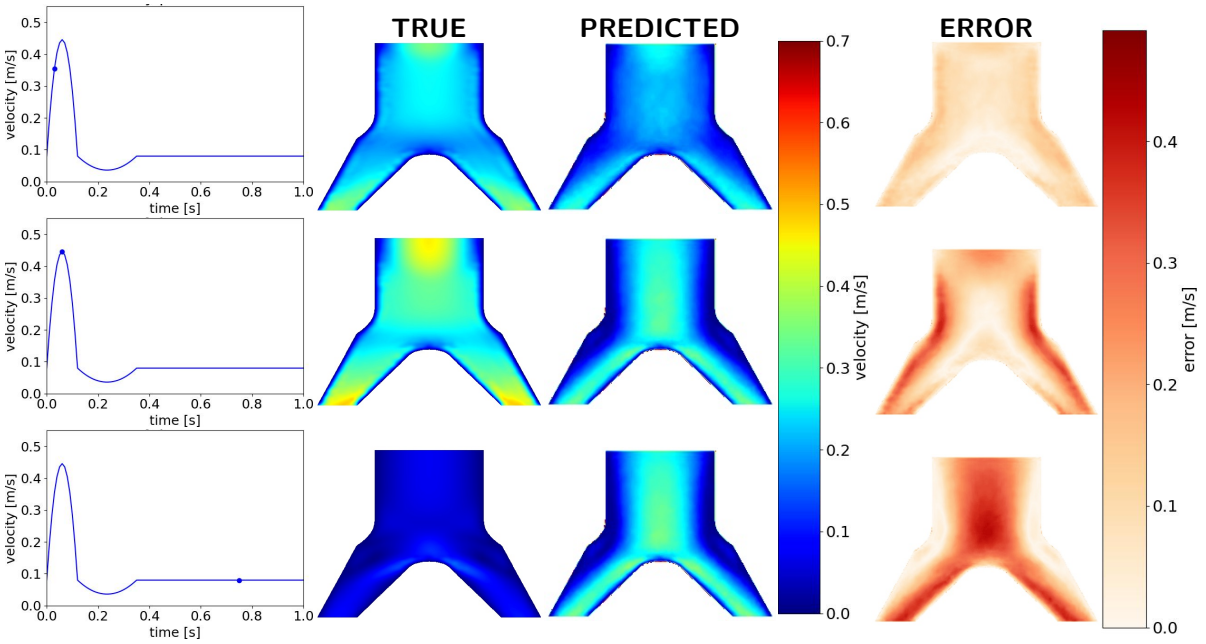


Figure 4.15: Three time steps (from left to right: t=0.03, t = 0.06, and t = 0.75) in predicting one heartbeat for vp5.10 using histogram equalization. The first column shows the moment in the time series, the second column the true velocity fields, the third column the predicted velocity fields and the last column shows the difference between true and predicted values.

To see if adding more data would improve the performance of the network, it was trained on various data set sizes. The training data set size was increased from including 15 time series to including the full set of 67 time series. The errors drop significantly when including 28 versus 15 time series, then plateau for including 41 and 53 time series but drop again when including all data. This is shown in figure 4.16, with on the left axis the MSE and on the right axis the MAE. The figure gives no definitive answer to the question if adding even more time series would improve the performance, as the data points confirm neither a plateau nor a linear decrease in error values. Nevertheless, the general trend shows increasing the data set size would improve the performance.

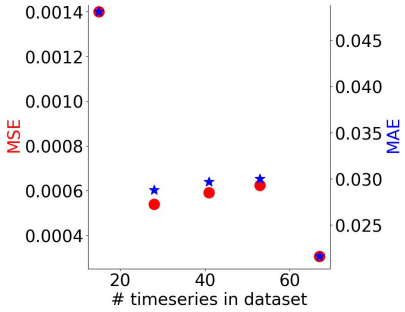


Figure 4.16: MSE (left, red dots) and MAE (right, blue stars) when increasing data set size.

Lastly, an experiment was done to see if predicting more time steps ahead, giving the model more information, would improve the performance. The results of training the model to predict ten time steps ahead are shown in tables 4.13 and 4.14.

			t+1	t+2	t+3	t+4	t+5
RELATIVE	AVG	boundary	0,25949	0,24073	0,23930	0,23667	0,23422
		domain	0,15254	0,15043	0,15772	0,16394	0,16923
	MAX	boundary	48,0	45,3	45,0	45,0	46,7
		domain	21,7	21,3	20,3	21,7	22,0
ABSOLUTE	AVG	boundary	0,01706	0,01596	0,01563	0,01555	0,01554
		domain	0,00417	0,00451	0,00483	0,00508	0,00528
	MAX	boundary	0,16562	0,15647	0,16196	0,16471	0,16562
		domain	0,07229	0,07137	0,08327	0,08144	0,08418
	domain RMSE		0,00768	0,00821	0,00871	0,00908	0,00938
net flow error			0,000073	0,000090	0,000103	0,000101	0,000103

Table 4.13: Error metrics when training the model to predict 10 time steps ahead, averaged over three variations of vp5. Columns t+1 to t+5 show the average error metrics for individual input images.

			t+6	t+7	t+8	t+9	t+10	One beat
RELATIVE	AVG	boundary	0,23132	0,22863	0,22572	0,22191	0,21767	0,35508
		domain	0,17391	0,17850	0,18355	0,18896	0,19479	1,19212
	MAX	boundary	46,7	46,3	42,7	40,7	37,3	42,3
		domain	23,0	23,7	25,7	30,7	35,3	70,7
ABSOLUTE	AVG	boundary	0,01555	0,01556	0,01559	0,01561	0,01563	0,01888
		domain	0,00545	0,00562	0,00579	0,00592	0,00600	0,04399
	MAX	boundary	0,16379	0,16562	0,16562	0,16745	0,16196	0,14915
		domain	0,09242	0,10065	0,11621	0,14183	0,16105	0,30288
	domain RMSE		0,00964	0,00990	0,01018	0,01040	0,01057	0,07069
net flow error			0,000106	0,000110	0,000114	0,000117	0,000120	0,000648

Table 4.14: Error metrics when training the model to predict 10 time steps ahead, averaged over three variations of vp5. Columns t+6 to t+10 show the average error metrics for individual input images and the last column shows the average error metrics for one whole heartbeat.

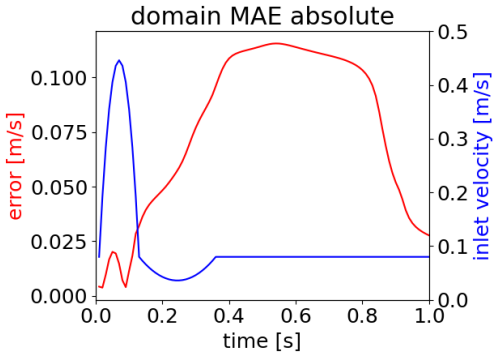


Figure 4.17: MAE in the domain for vp5.10 using the model trained on predicting 10 time steps. The red line shows the absolute error in [m/s] and the blue line the inlet velocity in [m/s] versus time in [s].

The tables show the errors do not significantly increase when predicting ten time steps ahead versus predicting five time steps ahead. The largest relative domain MAE for five time steps model is 16% at t+5, whilst this is 19% for the ten time steps model at t+10. However, this is not true when predicting one whole heartbeat, for which the errors are much higher. Figure 4.17 shows absolute MAE per time step when predicting one heartbeat for vp5.10. The model generally predicts too much velocity, which is shown in more detail in appendix H.

4.4.3 Three heartbeats

Finally, the network was tested by repeatedly taking the prediction of t+1 and using it as new input for the network until three heartbeats, or three seconds, were predicted. In figure 4.18, the blue line again shows the inlet velocity pattern in [m/s] and the red line the absolute MSE in [m/s] versus time in [s]. As can be seen, the mean absolute error for the boundary as well as for the domain shows the same pattern at the start of each heartbeat, following the the high velocity peak. Thereafter the boundary error decreases and becomes constant until the next heartbeat, whilst the domain error shows a significant peak when the inlet velocity drops, before gradually decreasing. It can also be noted, that the boundary MAE as well as the domain MAE are stable for three heartbeats, i.e., they show the same magnitude and pattern for each heartbeat. Hence, the model seems to be stable for a periodic inlet velocity pattern. The final prediction is equal for each heartbeat, as shown in figure 4.19.

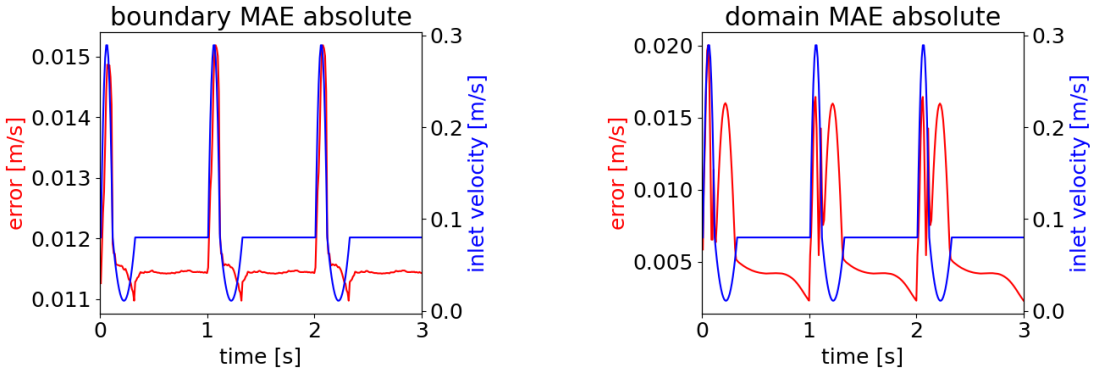


Figure 4.18: Absolute MAE in three heartbeats of vp5.9 for boundary (left) and domain (right). The red line shows the absolute error in [m/s] and the blue line the inlet velocity in [m/s] versus time in [s].

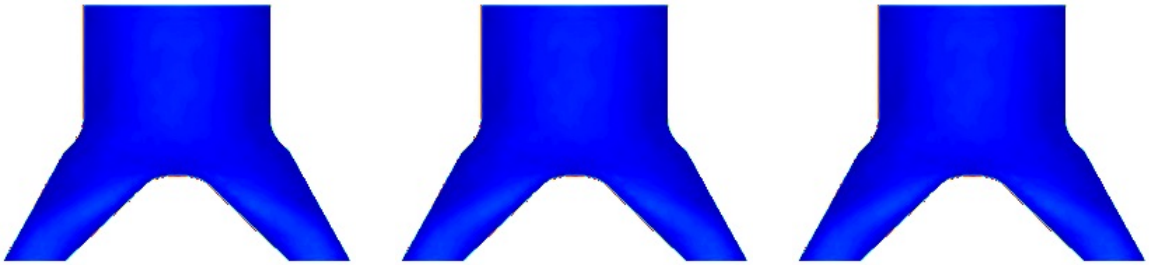


Figure 4.19: Last prediction for each heartbeat (from left to right: t=1, t=2 and t=3)

When predicting three heartbeats, the prediction time can be compared to the simulation time with OpenFoam [2], as all simulations were also run for three heartbeats. A simulation time of approximately 1600s was found, whilst the prediction time was approximately 70s, using the same computer (CPU). Hence the prediction of three heartbeats can be done in around one minute and is about 20 times faster than the OpenFoam simulation. Predictions of one heartbeat took around 20s. However, it should be noted the simulations with OpenFoam were not optimized for time and further decrease of simulation time could be achieved.

5 Conclusions and recommendations

The aim of this study was to construct a single reduced order model, based on NNs, for time-dependent incompressible blood flow through the aorta that can account for varying inlet velocity conditions, material parameters and geometries (computational domains). The objectives were to minimize the reduction of accuracy of simulated time series with OpenFoam [2], and to obtain speedup compared to the OpenFoam simulations. Three research questions were proposed to support this aim and objective. Firstly, which ML network architecture is most accurate at predicting consecutive time steps of time-dependent aortic blood flow? Secondly, can the ML network produce accurate results for a range of physically relevant boundary conditions and material parameters? Thirdly, how much can computational domains differ while retaining accuracy? All these questions will be discussed in the following paragraphs.

The best performing architecture found from all tested networks and variations was the encoder, RNN with LSTM units, difference variant, decoder model. This network consists of a convolutional encoder, which reduced the dimensions of the input image, being 192x256, to a latent vector of size 150. The latent vector is then repeated five times and each vector is concatenated with the difference in inlet velocity between the current time step and one to five time steps ahead. These vectors are provided as input to an RNN with LSTM units, which returns the latent vectors for five consecutive time steps. These are all decoded by the same convolutional decoder, which is symmetrical to the encoder with exception of the skip connection.

The hyperparameters of the RNN with LSTM units were not tuned, which could optimize the architecture. Additionally, of course, not all possible architectures have been tested. Other interesting architectures that could be investigated in future research are using 3D convolutions over two spatial and one time dimension instead of separate spatial and time networks. In the opposite direction, the explainability and generalizability might improve if further separation of geometry and flow features can be achieved, for example by forcing certain latent variables to remain constant during time evolution. In addition, novel methods for sequence-to-sequence problems (time evolution) could be implemented, such as the transformer models with attention mechanism. Transformer models have led to significant improvements in the field of natural language processing and show promising results for time series forecasting [83].

For all networks, it was observed that all error metrics increased when moving from the task the network was trained on, i.e. predicting five time steps ahead, to predicting one whole heartbeat. In general, from the tested architectures, the networks that showed overestimation of the velocity performed worse on predicting one whole heartbeat. As aforementioned, there is no control mechanism implemented when predicting one entire heartbeat, meaning one wrong prediction could alter the rest of the sequence. Implementing a control mechanism, for example a Kalman-filter, using new observations whenever they are available, might significantly improve the results for one heartbeat. Nevertheless, the error metrics stabilized when extending the prediction of one heartbeat to multiple heartbeats. The same pattern was used for three consecutive heartbeats, varying the inlet velocity pattern per heartbeat could still be explored.

The generalizability for the tested material parameter, blood viscosity, while keeping the inlet velocity pattern fixed, appeared quite good for various geometries. For a straight channel, an average relative MAE of 8% was obtained in the domain. For a bent and a bifurcated channel, the MAE was 9% and 15%, respectively. The relative errors were considerably higher in comparison to the absolute errors, and mainly increased by low velocity parts of the domain and time series. Also, large maximum errors occurred at the boundaries of the domain, where the velocities are low for every times step.

When testing the network on varying inlet velocity patterns, the relative MAE in the domain increased considerably to 26% for variations of vp5, 27% for variations of vp6 (excluding vp6.8) and 28% when looking at a more realistic inlet velocity pattern of the aorta. The network showed the same shortcomings as before, that is, under-performing on low velocity areas of the domain and time series. This could be explained by the fact that in high velocity areas, the pixel intensities are also around one order of magnitude higher, thus the loss in these areas is relatively higher. The use of MSE as loss function amplifies this behaviour, due to squaring values below one.

In addition, the network did not always capture the full height of the velocity peak and a spike in the domain MAE was seen for rapidly decreasing inlet velocities, originating from the network overestimating the velocity field. This could be caused by the combination of supplying the differences in inlet velocity between time steps as information to the RNN and using a batch normalization layer. Negative differences, when the inlet velocity is decreasing, become values close to zero. Hence, the network might give more importance to increasing velocities.

Various attempts to improve the performance, including altering the loss function and augmenting the data, were not successful. However, many choices of loss functions, including loss functions incorporating physics, and other data augmentations were not tested. Besides loss functions incorporating physics, a loss function comparing the mean of the velocity fields could lead to a performance increase, as it was found to be a key indicator for long-term prediction accuracy [84]. Other data transformations that may improve the results are transformations, that unlike histogram equalization, have an inverse, such as the log transform. With such a transformation the network might attribute more importance to low velocity parts of the time series and learn how to relate the transformed velocity fields to the original velocity fields. As decreasing velocities during the prediction of one heartbeat were challenging for the networks, other normalization methods (besides batch normalization between 0 and 1) could additionally improve the performance. Also, supplying information of the change in velocity in more than one way and the hard enforcement of boundary conditions were not tested.

Another approach to create a model that can predict both high and low velocities accurately may be using an ensemble model. The ensemble model could include separate networks for high and low velocity fields or even separate networks for certain areas of the computational domain, such as the center of the channel or the boundary. Depending on the application, one might also be interested in a specific case, for example high velocity near the boundary, to determine the maximum wall shear stress, for which a model could be finetuned.

In addition, the network was unable generalize to new geometries. This could be improved by adding more geometries and more variation to the geometries in the data. This could be done by making the inlet and outlet widths of the domain random instead of choosing values from a predetermined set of values. Moreover, including more realistic geometries, such as geometries with more wiggles, multiple inlets, more outlets, channels containing aneurysms or stenosis, could be an interesting extension of the data set. Research continuing in this direction could also incorporate geometries of smaller arteries, trees of arteries or possibly real medical data. One network might not suffice for the suggested level of complexity, thus the use of multiple networks interacting and depending on each other could also be explored.

Besides improvements that can be made to the network, there are certain aspects that were not investigated in this research. For example, variations of the image resolutions (in relation to the time step size), using multiple images as input and having a time step size larger than one between inputs, input and predictions and between predictions, were not analyzed in this research. Moreover, the errors could be analyzed further, for example the divergence of the flow was not evaluated in the domain. The error metrics could also be split in a different way than boundary and domain, for example in low and high velocity areas.

Furthermore, the number of simplifications made could be reduced in future research. One can think of varying the pressure outlet boundary condition, including fluid-structure interaction to account for deformation of the artery walls, allowing for turbulent flow, incorporating non-Newtonian properties of blood and extending the model from 2D to 3D.

In conclusion, this research should be seen as a basis for time-dependent blood flow predictions in the aorta, from which a multitude of different paths can be explored. What should also not be forgotten, is that although a speedup of 20 (using a CPU) was achieved between OpenFoam [2] simulations and the predictions, the time to train the final network was about 12 hours. On top of that, a total of 168.429 CPU minutes were used on a supercomputer during this research. This is a staggering 117 days. Hence, the implementation of machine learning algorithms should be considered with care and should not be assessed without the underlying cost of experimentation and training.

Appendix

A | Details numerical schemes

$$\frac{\partial \mathbf{u}}{\partial t} + \nabla \cdot (\mathbf{uu}) - \nabla \cdot (\nu \nabla \mathbf{u}) = -\nabla p$$

Evaluate for each volume :

$$\int_{v_i} \frac{\partial \mathbf{u}}{\partial t} dv + \int_{v_i} \nabla \cdot (\mathbf{uu}) dv - \int_{v_i} \nabla \cdot (\nu \nabla \mathbf{u}) dv = -\int_{v_i} \nabla p dv$$

Use Gauss theorem and trick constant vector pressure gradient, average u over volume for time derivative:

$$\frac{\partial \bar{\mathbf{u}}_\mathbf{i}}{\partial t} v_i + \int_{s_i} \mathbf{u}(\mathbf{u} \cdot \mathbf{n}) ds - \int_{s_i} \nu (\nabla \mathbf{u} \cdot \mathbf{n}) ds = -\int_{s_i} p \mathbf{n} ds$$

Split surface integral over the K faces of the control volume:

$$\frac{\partial \bar{\mathbf{u}}_\mathbf{i}}{\partial t} v_i + \sum_{j=1}^{K} \int_{s_j} \mathbf{u_j}(\mathbf{u_j} \cdot \mathbf{n_j}) ds_j - \sum_{j=1}^{K} \int_{s_j} \nu_j (\nabla \mathbf{u_j} \cdot \mathbf{n_j}) ds_j = -\sum_{j=1}^{K} \int_{s_j} p_j \mathbf{n_j} ds_j$$

Using Gauss integration and assuming linear variation on the surface, to evaluate the surface integrals:

$$\frac{\partial \bar{\mathbf{u}}_\mathbf{i}}{\partial t} v_i + \sum_{j=1}^{K} \mathbf{u}_{f_j}(\mathbf{u}_{f_j} \cdot \mathbf{n}_j) s_j - \sum_{j=1}^{K} \nu_{f_j} (\nabla \mathbf{u}_{f_j} \cdot \mathbf{n}_j) s_j = -\sum_{j=1}^{K} p_{f_j} \mathbf{n}_j s_j$$

Using Euler implicit method for the time derivative and assuming the value at the centre of the volume $\mathbf{u}_i$ is the average value of the volume $\bar{\mathbf{u}}_i$:

$$\frac{\mathbf{u}_i - \mathbf{u}_i^{t-\Delta t}}{\Delta t} v_i + \sum_{j=1}^{K} \mathbf{u}_{f_j}(\mathbf{u}_{f_j} \cdot \mathbf{n}_j) s_j - \sum_{j=1}^{K} \nu_{f_j} (\nabla \mathbf{u}_{f_j} \cdot \mathbf{n}_j) s_j = -\sum_{j=1}^{K} p_{f_j} \mathbf{n}_j s_j$$

In which $\mathbf{u}_i$ is the velocity for volume i in the current time step at time t and $\mathbf{u}_i^{t-\Delta t}$ is the velocity in the previous time step , with Δt the time step size. The terms containing an f_j as subscript denote the value at the centre of the corresponding face j and s_j is the area of the surface. To evaluate the values presented in the equation above we need interpolation schemes. We will look at each term separately.

Linear interpolation was used to obtain the value of the kinematic viscosity, ν_{f_j}, for face j (of volume i and neighbour volume k) in the diffusion term and the value of the kinematic pressure, p_{f_j} in the pressure gradient term.

$$\nu_{f_j} = \frac{\nu_i - \nu_k}{\mathbf{d}} \tag{A.1}$$

$$p_{f_j} = \frac{p_i - p_k}{\mathbf{d}} \tag{A.2}$$

In which $\mathbf{d}$ is the vector from the centre of volume i to the centre of a neighbour volume k. LinearUpwind was used to obtain the value of the velocity on the face centre for face j, $\mathbf{u}_{f_j}$, between volume i and a neighbour volume k in the convection term.

$$\mathbf{u}_{f_j} = \begin{cases} \mathbf{u}_i + (\nabla \mathbf{u})_i \cdot \mathbf{r} & F_{f_j} > 0 \\ \mathbf{u}_k + (\nabla \mathbf{u})_k \cdot \mathbf{r} & F_{f_j} < 0 \end{cases} \tag{A.3}$$

In which $\mathbf{r}$ is the vector from the centre of volume i to face j. F_{f_j} is the mass flux over face j defined as $F_{f_j} = \rho_{f_j} * s_j(\mathbf{u}_{f_j} \cdot \mathbf{n}_j)$. The second part of this equation, $(\nabla \mathbf{u}) \cdot \mathbf{r}$, and the mass flux are computed explicitly from the initial guess or the value from the previous iteration. The gradient term $\nabla \mathbf{u}$ is again computed by linear interpolation as shown in equations A.1 and A.2. To obtain the value of $\nabla \mathbf{u}_{f_j}$ orthogonal to the face normal vector $\mathbf{n}_j$ for face j between volume i and neighbour volume k in the diffusion term, the 'corrected' scheme was used (correcting for non-orthogonality in the mesh). Corrected uses linear interpolation for the part which is orthogonal to the vector between the centre of volume i and neighbour volume k, $\mathbf{d}_j$, and an explicit correction (using the value of the initial guess or previous iteration of $\nabla \mathbf{u}_{f_j}$) to account for the non-orthogonality.

$$\nabla \mathbf{u}_{f_j} \cdot \mathbf{n}_j = \frac{1}{cos(\theta)} \frac{\mathbf{u}_i - \mathbf{u}_k}{|\mathbf{d}_j|} + (\mathbf{n}_j - \frac{1}{cos(\theta)} \mathbf{d}_j) \cdot \nabla \mathbf{u}_{f_j} \tag{A.4}$$

B | Details numerical solvers

All discretized terms described in appendix A are combined in the matrix M. This reflects connectivity between all volumes, gathered in the vector $\mathbf{U}$. The discretized pressure gradient and explicitly calculated parts, see equations A.3 and A.4, are contained in the vector $\mathbf{P}$, such that

$$M\mathbf{U} = \mathbf{P} \tag{B.1}$$

All coefficients in M are known from the discretization schemes and an initial guess or values from the previous iteration of $\mathbf{P}$ are used to solve this equation for $\mathbf{U}$. This is called the momentum predictor stage. Note that we have completely ignored the continuity equation in the Navier-Stokes equations until now. Thus we obtain an solution $\mathbf{U}$ of equation B.1 which does not satisfy the continuity equation.

To rectify this, first a diagonal matrix A is formed with the diagonal elements of M and the residual matrix H is defined such that

$$H = A\mathbf{U} - M\mathbf{U} \tag{B.2}$$

As $M\mathbf{U} = \mathbf{P}$, $A\mathbf{U} - H = \mathbf{P}$, which can be rewritten as

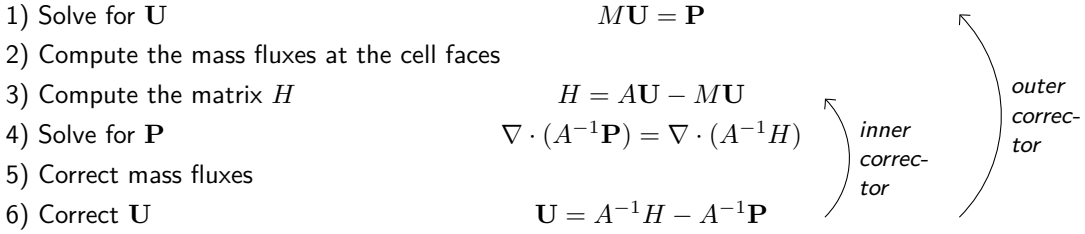
$$\mathbf{U} = A^{-1}H - A^{-1}\mathbf{P} \tag{B.3}$$

and substituted in the continuity equation to obtain an equation for the vector $\mathbf{P}$

$$\nabla \cdot (A^{-1}H) = \nabla \cdot (A^{-1}\mathbf{P}) \tag{B.4}$$

Now the velocity field can be corrected with the computed pressure field (satisfying the continuity equation) from equation B.4, by solving equation B.3. However, $\mathbf{U}$ is now different, thus the matrix H is different, changing the solution of equation B.4. Hence, the pressure field needs updating.

As mentioned in 3.1.5, the PIMPLE algorithm is a combination of the SIMPLE algorithm and PISO algorithm. The difference between these algorithms is how they correct the pressure field. The SIMPLE algorithm returns all the way to the momentum predictor stage, equation B.1 to correct the pressure field and continues from there. this is defined in the PIMPLE algorithm as outer corrections. The PISO algorithm starts from recalculating the matrix H, equation B.2, and continues from there, which is defined as inner corrections in the PIMPLE algorithm.



C | Changing latent variables

To see the influence of one specific latent variable over time, the feature was multiplied by a factor 100 in the latent space before decoding. This was done for multiple time steps throughout one heartbeat (10 time steps from t=0 to t=1 with time step size 0,1s). The results were checked for all 150 latent variables, all of which changed over time. Hence, no latent variable contained only information on the geometry, as this was fixed over time, and all latent variables contained information on the flow (and possibly geometry). Some examples of the results of this experiment are shown in figures C.1, C.2 and C.3.

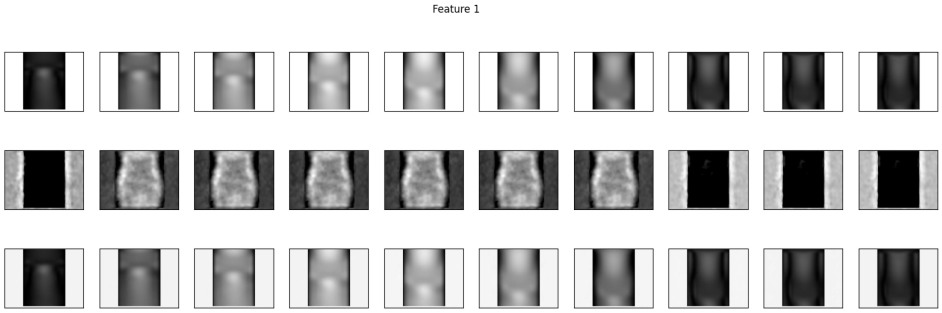


Figure C.1: Results of multiplying feature 1 by 100 and decoding. The top row contains the original input image, the second row the decoded image with alteration of the latent feature and the bottom row the decoded image without alteration of the latent feature.

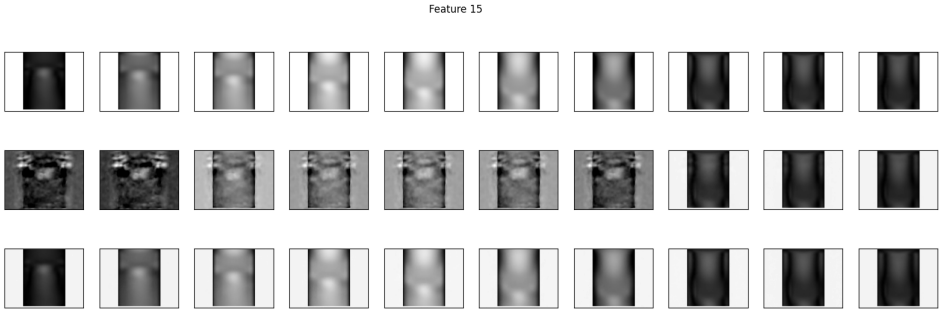


Figure C.2: Results of multiplying feature 15 by 100 and decoding. The top row contains the original input image, the second row the decoded image with alteration of the latent feature and the bottom row the decoded image without alteration of the latent feature.

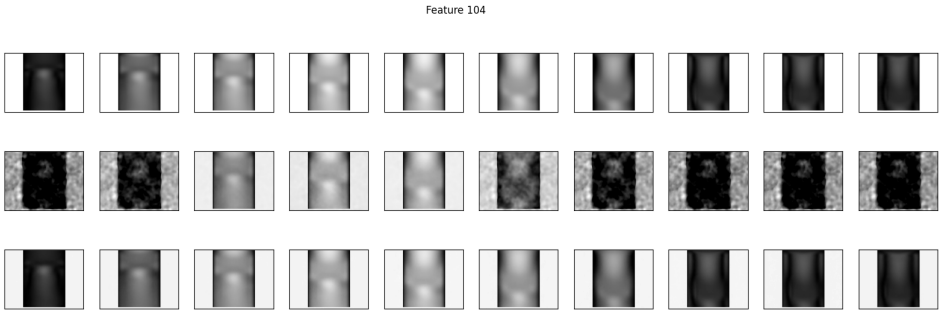


Figure C.3: Results of multiplying feature 104 by 100 and decoding. The top row contains the original input image, the second row the decoded image with alteration of the latent feature and the bottom row the decoded image without alteration of the latent feature.

D | One heartbeat using various time step sizes

As described in section 3.2.2, one whole heartbeat was predicted using t+1. All networks were also tested by using t+5 to predict one whole heartbeat for the straight channel test set. This implies one heartbeat contains 20 time steps, versus 100 time steps when using t+1. As shown in table D.1, the network with LSTM units also performs best on this task. This is in agreement with the results on each individual prediction shown in tables 4.1, 4.2 and 4.3.

			NN	LSTM	GRU
RELATIVE	AVG	boundary	0,29225	0,16860	0,27700
		domain	0,10049	0,05145	0,07660
	MAX	boundary	8,0	5,0	9,0
		domain	17,0	18,0	21,0
ABSOLUTE	AVG	boundary	0,00213	0,00104	0,00183
		domain	0,00534	0,00216	0,00307
	MAX	boundary	0,03137	0,04902	0,02745
		domain	0,08431	0,05882	0,08431
	domain RMSE		0,00986	0,00445	0,00639
net flow error			0,000166	0,000094	0,000118

Table D.1: One heartbeat predictions using t+5 for straight channel

For the network with the RNN containing LSTM units for time evolution, predicting one heartbeat was also done using the remaining time steps (t+2, t+3 and t+4) for the straight channel test time series. The results are shown in table D.2. No clear pattern can be seen in the errors when increasing the time step size. However, using t+5 does give the lowest errors on nearly all metrics.

			t+1	t+2	t+3	t+4	t+5
RELATIVE	AVG	boundary	0,19805	0,16866	0,15205	0,15616	0,16860
		domain	0,08784	0,07701	0,09486	0,13214	0,05145
	MAX	boundary	7,0	5,0	5,0	4,5	5,0
		domain	20,0	22,0	46,0	63,0	18,0
ABSOLUTE	AVG	boundary	0,00153	0,00155	0,00108	0,00109	0,00104
		domain	0,00454	0,00306	0,00331	0,00399	0,00216
	MAX	boundary	0,02353	0,02157	0,01961	0,02941	0,04902
		domain	0,40196	0,10000	0,10196	0,14902	0,05882
	domain RMSE		0,00900	0,00657	0,00629	0,00725	0,00445
net flow error			0,000095	0,000080	0,000103	0,000100	0,000094

Table D.2: Encoder - RNN LSTM - decoder network, straight data set, one heartbeat prediction using different time steps

E Maximum error locations bent and bifurcated channel

The maximum error locations for the bent and bifurcated channel were also investigated. Again the orange dots represent the relative ME in the domain and the blue dots the ME error on the boundary. The size of the dots represent the number of occurrences of the maximum error in that exact location. For all time step predictions (t+1, ... t+5) the maximum error most frequently occurred at the edges of the domain, as shown in figure E.1.

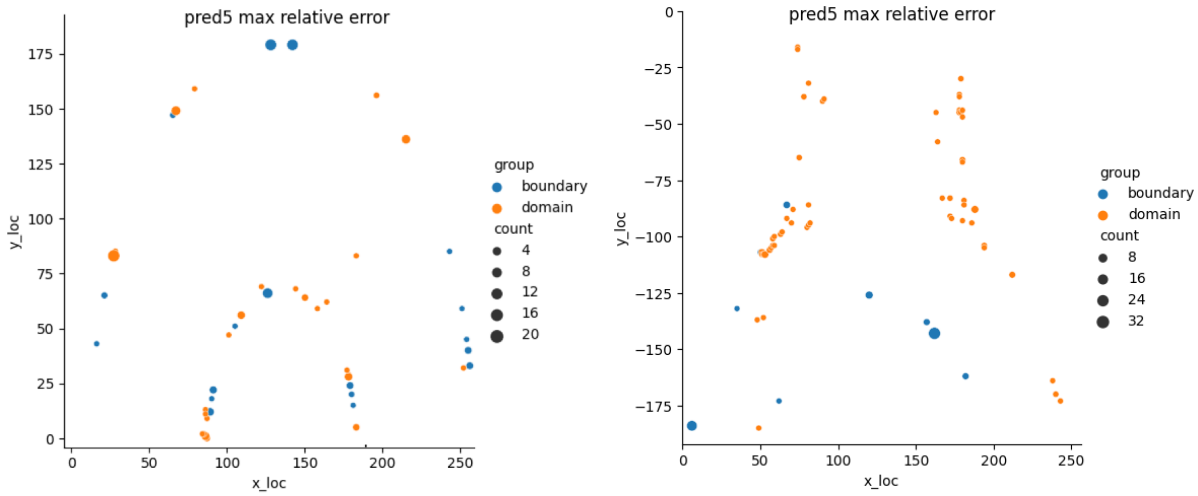


Figure E.1: Maximum error locations for bent channel (left) and bifurcated channel (right)

F | Predictions on vp6.8

The full results on predicting the time series with inlet vp6.8 and the average of the remaining two variations on vp6 are shown in the tables below.

			t+1	t+2	t+3	t+4	t+5	One beat
RELATIVE	AVG	boundary	0,26537	0,25904	0,25815	0,26218	0,26637	0,24468
		domain	0,12710	0,14158	0,15546	0,16742	0,17812	0,27223
	MAX	boundary	32,0	35,5	44,5	52,5	58,0	25,5
		domain	22,5	22,5	21,0	21,5	24,0	23,5
ABSOLUTE	AVG	boundary	0,01503	0,01435	0,01428	0,01444	0,01462	0,01479
		domain	0,00349	0,00443	0,00539	0,00616	0,00671	0,01189
	MAX	boundary	0,17020	0,14137	0,15098	0,16882	0,17980	0,15235
		domain	0,11941	0,11941	0,12078	0,11941	0,11804	0,18667
	domain RMSE		0,00652	0,00796	0,00938	0,01056	0,01142	0,02108
net flow error			0,000031	0,000014	0,000015	0,000025	0,000038	0,000203

Table F.1: Average error metrics on vp6.10 and vp6.11

			t+1	t+2	t+3	t+4	t+5	One beat
RELATIVE	AVG	boundary	0,24830	0,24846	0,24481	0,24444	0,24501	0,29611
		domain	0,13923	0,14697	0,16095	0,17091	0,18050	0,84738
	MAX	boundary	37,0	31,0	29,0	27,0	25,0	32,0
		domain	43,0	36,0	37,0	38,0	40,0	53,0
ABSOLUTE	AVG	boundary	0,01198	0,01249	0,01200	0,01192	0,01201	0,01313
		domain	0,00320	0,00354	0,00418	0,00478	0,00533	0,02647
	MAX	boundary	0,15922	0,15922	0,14824	0,14824	0,15098	0,18118
		domain	0,14549	0,14000	0,13177	0,13177	0,13451	0,16196
	domain RMSE		0,00578	0,006275	0,007221	0,008118	0,00893	0,04103
net flow error			0,000041	0,000011	0,000014	0,000032	0,000051	0,000379

Table F.2: Error metrics on vp6.8

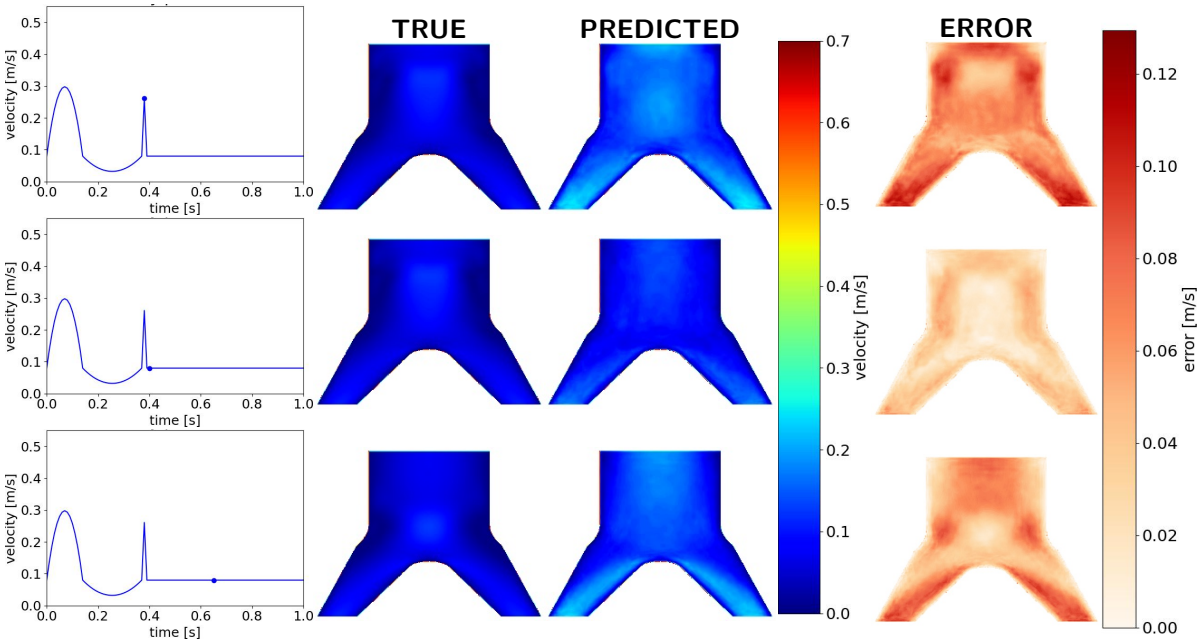


Figure F.1: Three time steps (from left to right: t=0.38, t = 0.40, and t = 0.65) in predicting one heartbeat for vp6.8. The first column shows the moment in the time series, the second column the true velocity fields, the third column the predicted velocity fields and the last column shows the difference between true and predicted values.

G | MSE+MAE loss function

The average error metrics for vp5.9, vp5.10 and vp5.11 are shown in table G.1.

			t+1	t+2	t+3	t+4	t+5	One beat
RELATIVE	AVG	boundary	0,46177	0,47846	0,46985	0,47482	0,47647	0,48089
		domain	0,09862	0,10254	0,10875	0,11510	0,12209	0,63772
	MAX	boundary	90,0	95,7	87,7	87,3	91,7	73,3
		domain	22,3	22,7	24,0	25,0	27,3	56,0
ABSOLUTE	AVG	boundary	0,03516	0,03507	0,03472	0,03494	0,03514	0,03534
		domain	0,00283	0,00315	0,00355	0,00380	0,00404	0,01886
	MAX	boundary	0,31294	0,31386	0,32118	0,33399	0,34954	0,31111
		domain	0,07778	0,07869	0,09150	0,09791	0,10157	0,23608
	domain RMSE		0,00527	0,00579	0,00637	0,00677	0,00719	0,03202
net flow error			0,000051	0,000069	0,000084	0,000093	0,000099	0,000220

Table G.1: Average error metrics on variations of vp5 using MSE+MAE loss function

H | Predicting ten time steps

Various moments in the time series for vp5.10 are shown using the model trained on predicting 10 time steps in figure H.1. It can be concluded the model largely overestimates the flow at the end of the heartbeat.

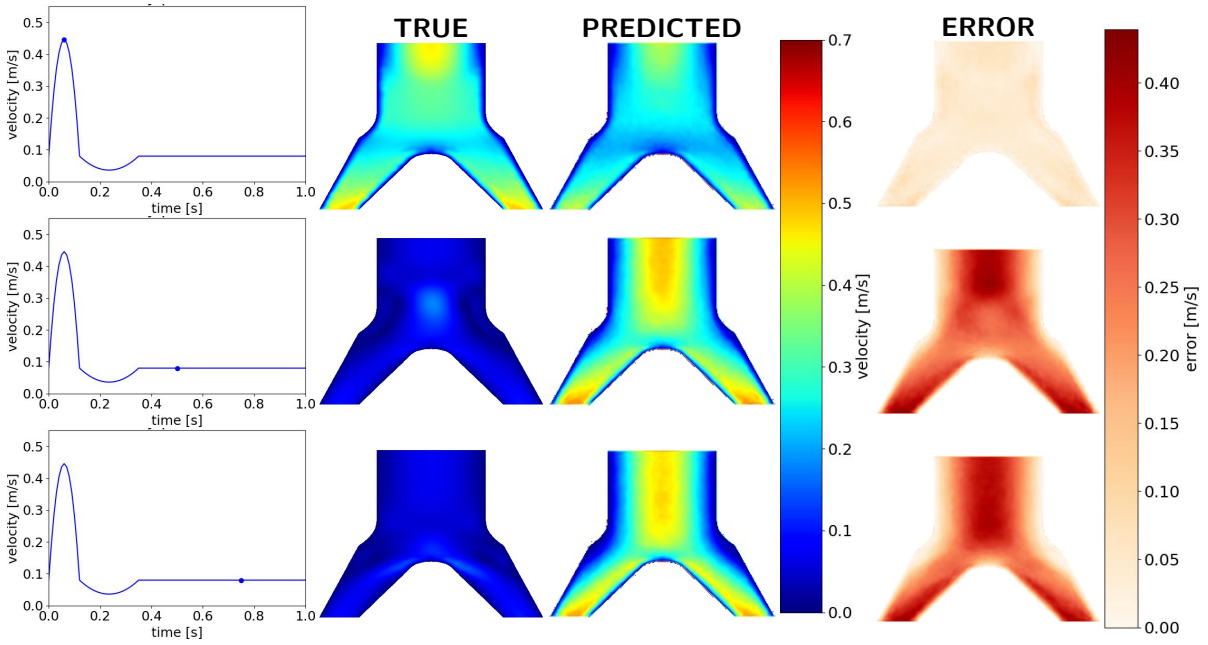


Figure H.1: Three time steps (from left to right: t=0.06, t = 0.50, and t = 0.75) in predicting one heartbeat for vp5.10. The first column shows the moment in the time series, the second column the true velocity fields, the third column the predicted velocity fields and the last column shows the difference between true and predicted values.

Bibliography

[1] T. Lassila et al. "Model Order Reduction in Fluid Dynamics: Challenges and Perspectives". In: *Reduced Order Methods for Modeling and Computational Reduction*. Ed. by A. Quarteroni and G. Rozza. Cham: Springer International Publishing, 2014, pp. 2–4. ISBN: 978-3-319-02090-7. DOI: 10.1007/978-3-319-02090-7_9. URL: https://doi.org/10.1007/978-3-319-02090-7_9.

[2] H. G. Wellera et al. *OpenFoam 9*. Version 9.0. July 20, 2021. URL: https://openfoam.org/version/9/.

[3] Z. Taylor, S. Crozier, and S. Ourselin. "Real-Time Surgical Simulation Using Reduced Order Finite Element Analysis". In: *Medical image computing and computer-assisted intervention : MICCAI, International Conference on Medical Image Computing and Computer-Assisted Intervention* 13 (Sept. 2010), pp. 388–95. DOI: 10.1007/978-3-642-15745-5_48.

[4] Dirk D. Hartmann, M. Herz, and U. Wever. "Model Order Reduction a Key Technology for Digital Twins". In: *Reduced-Order Modeling (ROM) for Simulation and Optimization: Powerful Algorithms as Key Enablers for Scientific Computing*. Apr. 2018, pp. 167–179. ISBN: 978-3-319-75318-8. DOI: 10.1007/978-3-319-75319-5_8.

[5] B. Kim et al. "Deep Fluids: A Generative Network for Parameterized Fluid Simulations". In: *Computer Graphics Forum* 38 (2 May 2019), pp. 59–70. ISSN: 14678659. DOI: 10.1111/cgf.13619.

[6] A. Quarteroni and G. Rozza, eds. *Reduced Order Methods for Modeling and Computational Reduction*. Springer, Cham, 2014. Chap. Preface. DOI: 10.1007/978-3-319-02090-7.

[7] J. Weiss. "A Tutorial on the Proper Orthogonal Decomposition". In: 2019. URL: https://depositonce.tu-berlin.de/handle/11303/9456.

[8] A. Quarteroni, A. Manzoni, and F. Negri. *Reduced Basis Methods for Partial Differential Equations*. 2016. Chap. Reduced Basis Methods for PDEs at a Glance. DOI: https://doi.org/10.1007/978-3-319-15431-2.

[9] F. Chinesta, R. Keunings, and A. Leygue. *The Proper Generalized Decomposition for Advanced Numerical Simulations*. 2014. Chap. The Proper Generalized Decomposition at a Glance. DOI: https://doi.org/10.1007/978-3-319-02865-1.

[10] M.H. Dao. *Projection-Based Reduced Order Model for Simulations of Nonlinear Flows with Multiple Moving Objects*. URL: https://arxiv.org/pdf/2106.02338v1.pdf.

[11] A. Treuille, A. Lewis, and Z. Popovic. "Model reduction for real-time fluids". In: *ACM Trans. Graph.* 25 (July 2006), pp. 826–834. DOI: 10.1145/1141911.1141962.

[12] Y. Tian and Y. Zhang. "A comprehensive survey on regularization strategies in machine learning". In: *Information Fusion* 80 (2022), pp. 146–166. ISSN: 1566-2535. DOI: https://doi.org/10.1016/j.inffus.2021.11.005. URL: https://www.sciencedirect.com/science/article/pii/S156625352100230X.

[13] *Activation Functions in Neural Networks*. https://medium.com/@kshitijkhurana3010/activation-functions-in-neural-networks-ed88c56b611b. Accessed: 2022-03-21.

[14] D. Wang. "Unsupervised Learning: Foundations of Neural Computation". In: *AI Mag.* 22 (2001), pp. 101–102.

[15] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. "Learning representations by back-propagating errors". In: *Nature* 323.6088 (Oct. 1986), pp. 533–536. DOI: 10.1038/323533a0.

[16] D.P. Kingma and J. Ba. "Adam: A Method for Stochastic Optimization". In: *International Conference on Learning Representations* (Dec. 2014). DOI: 10.48550/arXiv.1412.6980.

[17] N. Qian. "On the momentum term in gradient descent learning algorithms". In: *Neural Networks* 12.1 (1999), pp. 145–151. ISSN: 0893-6080. DOI: https://doi.org/10.1016/S0893-6080(98)00116-6. URL: https://www.sciencedirect.com/science/article/pii/S0893608098001166.

[18] M.D. Zeiler. "ADADELTA: An Adaptive Learning Rate Method". In: *CoRR* abs/1212.5701 (2012). arXiv: 1212.5701. URL: http://arxiv.org/abs/1212.5701.

[19] V. Dumoulin and F. Visin. "A guide to convolution arithmetic for deep learning". In: (2019). DOI: https://doi.org/10.48550/arXiv.1603.07285. URL: https://arxiv.org/abs/1603.07285.

[20] S. Bai, J.Z. Kolter, and V. Koltun. *An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling*. 2018. arXiv: 1803.01271 [cs.LG].

[21] F. Yu and V. Koltun. "Multi-scale context aggregation by dilated convolutions". In: *Proceedings of ICLR 2016*. URL: https://arxiv.org/abs/1511.07122.

[22] F. Yu, V. Koltun, and T.A. Funkhouser. "Dilated Residual Networks". In: *CoRR* abs/1705.09914 (2017). arXiv: 1705.09914. URL: http://arxiv.org/abs/1705.09914.

[23] N. Duong Trung, L.D. Quach, and C.N. Nguyen. "Learning Deep Transferability for Several Agricultural Classification Problems". In: *International Journal of Advanced Computer Science and Applications* 10 (Feb. 2019).

[24] M. Lopez-Martin, S. Le Clainche, and B. Carro. "Model-free short-term fluid dynamics estimator with a deep 3D-convolutional neural network". In: *Expert Systems with Applications* 177 (2021). ISSN: 09574174. DOI: 10.1016/j.eswa.2021.114924.

[25] T. Simpson, N. Dervilis, and El. Chatzi. "Machine Learning Approach to Model Order Reduction of Nonlinear Systems via Autoencoder and LSTM Networks". In: *Journal of Engineering Mechanics* 147 (10 2021). ISSN: 0733-9399. DOI: 10.1061/(asce)em.1943-7889.0001971.

[26] M. Eichinger, A. Heinlein, and A. Klawonn. *Technical Report Series Center for Data and Simulation Science Surrogate Convolutional Neural Network Models for Steady Computational Fluid Dynamics Simulations*. 2016.

[27] K. He et al. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[28] Y. Kim et al. "A fast and accurate physics-informed neural network reduced order model with shallow masked autoencoder". In: *Journal of Computational Physics* 451 (2022). ISSN: 10902716. DOI: 10.1016/j.jcp.2021.110841.

[29] R.M. Schmidt. "Recurrent Neural Networks (RNNs): A gentle Introduction and Overview". In: *CoRR* abs/1912.05911 (2019). arXiv: 1912.05911. URL: http://arxiv.org/abs/1912.05911.

[30] S. Hochreiter and J. Schmidhuber. "Long Short-term Memory". In: *Neural computation* 9 (Dec. 1997), pp. 1735–80. DOI: 10.1162/neco.1997.9.8.1735.

[31] J. Chung et al. "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling". In: *CoRR* abs/1412.3555 (2014). arXiv: 1412.3555. URL: http://arxiv.org/abs/1412.3555.

[32] R. Hu et al. "Rapid spatio-temporal flood prediction and uncertainty quantification using a deep learning method". In: *Journal of Hydrology* 575 (Aug. 2019), pp. 911–920. ISSN: 00221694. DOI: 10.1016/j.jhydrol.2019.05.087.

[33] S. Pawar et al. "Model fusion with physics-guided machine learning: Projection-based reduced-order modeling". In: *Physics of Fluids* 33 (6 2021). ISSN: 10897666. DOI: 10.1063/5.0053349.

[34] S. Yang, X. Yu, and Y. Zhou. "LSTM and GRU Neural Network Performance Comparison Study: Taking Yelp Review Dataset as an Example". In: June 2020, pp. 98–101. DOI: 10.1109/IWECAI50956.2020.00027.

[35] P. Wu et al. "Data-driven reduced order model with temporal convolutional neural network". In: *Computer Methods in Applied Mechanics and Engineering* 360 (2020), p. 112766. ISSN: 0045-7825. DOI: https://doi.org/10.1016/j.cma.2019.112766. URL: https://www.sciencedirect.com/science/article/pii/S0045782519306589.

[36] I.R. Jamil and M. Humaira. "Modeling and Predicting Blood Flow Characteristics through Double Stenosed Artery from Computational Fluid Dynamics Simulations using Deep Learning Models". In: *International Journal of Advanced Computer Science and Applications* 13.1 (2022). ISSN: 2158-107X. DOI: 10.14569/ijacsa.2022.0130197. URL: http://dx.doi.org/10.14569/IJACSA.2022.0130197.

[37] M. Raissi, P. Perdikaris, and G. E. Karniadakis. "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations". In: *Journal of Computational Physics* 378 (2019). ISSN: 10902716. DOI: 10.1016/j.jcp.2018.10.045.

[38] G. Pang, L. Lu, and G.E. Karniadakis. "fPINNs: Fractional physics-informed neural networks". In: *SIAM Journal on Scientific Computing* 41.4 (2019), A2603–A2626.

[39] E. Kharazmi, Z. Zhang, and G.E. Karniadakis. "Variational physics-informed neural networks for solving partial differential equations". In: *arXiv preprint arXiv:1912.00873* (2019).

[40] A. Mohan et al. "Embedding Hard Physical Constraints in Neural Network Coarse-Graining of 3D Turbulence". In: (Jan. 2020).

[41] L. Sun et al. "Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data". In: *Computer Methods in Applied Mechanics and Engineering* 361 (2020), p. 112732. ISSN: 0045-7825. DOI: https://doi.org/10.1016/j.cma.2019.112732. URL: https://www.sciencedirect.com/science/article/pii/S004578251930622X.

[42] D. Kochkov et al. "Machine learning accelerated computational fluid dynamics". In: *Proceedings of the National Academy of Sciences* 118.21 (2021), e2101784118. DOI: 10.1073/pnas.2101784118. eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.2101784118. URL: https://www.pnas.org/doi/abs/10.1073/pnas.2101784118.

[43] S. Grimberg and C. Farhat. "Hyperreduction of cfd models of turbulent flows using a machine learning approach". In: *AIAA Scitech 2020 Forum*. Vol. 1 PartF. 2020, pp. 1–13.

[44] K.T. Carlberg et al. "Recovering missing CFD data for high-order discretizations using deep neural networks and dynamics learning". In: *Journal of Computational Physics* 395 (2019), pp. 105–124. ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2019.05.041. URL: https://www.sciencedirect.com/science/article/pii/S0021999119303857.

[45] S. Lee, I.G. Kevrekidis, and G.E. Karniadakis. "Resilient algorithms for reconstructing and simulating gappy flow fields in CFD". In: *Fluid Dynamics Research* 47.5 (2015).

[46] H.J. Bae and P. Koumoutsakos. "Scientific multi-agent reinforcement learning for wall-models of turbulent flows". In: *Nature Communications* 13.1 (2022).

[47] G. Chen and K. J. Fidkowski. "Output-based error estimation and mesh adaptation using convolutional neural networks: Application to a scalar advection-diffusion problem". In: *AIAA Scitech 2020 Forum*. Vol. 1 PartF. 2020.

[48] T. Sochi. "Non-Newtonian Rheology in Blood Circulation". In: (June 2013). URL: https://arxiv.org/pdf/1306.2067.pdf.

[49] K. Beenutty and A. Mariamichael. "Simulation and analysis of aortic bypass surgery using computational fluid dynamics". In: *ICECT 2011 - 2011 3rd International Conference on Electronics Computer Technology* 3 (Apr. 2011). DOI: 10.1109/ICECTECH.2011.5941768.

[50] H. Švihlová et al. "Determination of pressure data from velocity data with a view toward its application in cardiovascular mechanics. Part 1. Theoretical considerations". In: *International Journal of Engineering Science* 105 (2016). ISSN: 00207225. DOI: 10.1016/j.ijengsci.2015.11.002.

[51] E. Nader et al. "Blood Rheology: Key Parameters, Impact on Blood Flow, Role in Sickle Cell Disease and Effects of Exercise". In: *Frontiers in Physiology* 10 (2019). ISSN: 1664-042X. DOI: 10.3389/fphys.2019.01329. URL: https://www.frontiersin.org/article/10.3389/fphys.2019.01329.

[52] M. Shmukler. *The Physics Factbook*. 2004. Chap. Density of Blood. URL: https://web.archive.org/web/20060919051122/http://hypertextbook.com/facts/2004/MichaelShmukler.shtml.

[53] A. Heinlein. *Parallel Overlapping Schwarz preconditioners and multiscale discretizations with applicationsto fluid-structure interaction and highly heterogeneous problems*. 2016.

[54] A. Evangelista et al. "Echocardiography in aortic diseases: EAE recommendations for clinical practice". In: *European Journal of Echocardiography* 11.8 (Sept. 2010), pp. 645–658. ISSN: 1525-2167. DOI: 10.1093/ejechocard/jeq056. eprint: https://academic.oup.com/ehjcimaging/article-pdf/11/8/645/7137659/jeq056.pdf. URL: https://doi.org/10.1093/ejechocard/jeq056.

[55] M. Gameraddin. "Normal abdominal aorta diameter on abdominal sonography in healthy asymptomatic adults: impact of age and gender". In: *Journal of Radiation Research and Applied Sciences* 12.1 (2019), pp. 186–191. ISSN: 1687-8507. DOI: https://doi.org/10.1080/16878507.2019.1617553. URL: https://www.sciencedirect.com/science/article/pii/S1687850721000170.

[56] M. Mohiuddin et al. "Analysis of renal artery morphometery in adults: A study conducted by using Multidetector computed Tomography Angiography". In: *Pakistan Journal of Medical Sciences* 33 (Aug. 2017). DOI: 10.12669/pjms.334.13063.

[57] O.M. Pedersen, A. Aslaksen, and H. Vik-Mo. "Ultrasound measurement of the luminal diameter of the abdominal aorta and iliac arteries in patients without vascular disease". In: *Journal of Vascular Surgery* 17 (Mar. 1993). DOI: https://doi.org/10.1016/0741-5214(93)90161-E.

[58] M. Beraia. "Arterial pulse impact on blood flow". In: *Health* 2.6 (June 2010). DOI: 10.4236/health.2010.26080.

[59] T.D. Homan, S. Bordes, and E. Cichowski. *Physiology, Pulse Pressure*. https://www.ncbi.nlm.nih.gov/books/NBK482408/. Accessed: 2022-03-22.

[60] L. Liang, M. Wenbin, and S. Wei. "A feasibility study of deep learning for predicting hemodynamics of human thoracic aorta". In: *Journal of Biomechanics* 99 (2020), p. 109544. ISSN: 0021-9290. DOI: https://doi.org/10.1016/j.jbiomech.2019.109544. URL: https://www.sciencedirect.com/science/article/pii/S0021929019308012.

[61] P. Yevtushenko et al. "Deep Learning Based Centerline-Aggregated Aortic Hemodynamics: An Efficient Alternative to Numerical Modelling of Hemodynamics". In: *IEEE Journal of Biomedical and Health Informatics* (2021).

[62] *PimpleFoam solver*. URL: https://www.openfoam.com/documentation/guides/latest/api/pimpleFoam_8C_source.html.

[63] G. K. Batchelor. *An Introduction to Fluid Dynamics*. Cambridge University Press, 1967, pp. 142–148. ISBN: ISBN 978-0-521-66396-0.

[64] R.E. Klabunde. *Cardiovascular Physiology Concepts*. URL: https://www.cvphysiology.com/Blood%5C%20Pressure/BP002.

[65] M.S.M. Watanabe, P.J. Blanco, and R.A. Feijóo. "Mathematical model of blood flow in an anatomically detailed arterial network of the arm". In: *ESAIM - Mathematical Modelling and Numerical Analysis* 47.4 (2013), pp. 961–985. DOI: 10.1051/m2an/2012053.

[66] P.J. Blanco, M.S.M. Watanabe, and R.A. Feijóo. "Identification of vascular territory resistances in one-dimensionalhemodynamics simulations". In: *Journal of Biomechanics* 45.12 (2012), pp. 2066–2073. DOI: 10.1016/j.jbiomech.2012.06.002.

[67] M.S.M. Watanabe. "ADAN: an human anatomically detailed arterial network for computational hemodynamics". PhD thesis. National Laboratory for Scientific Computing, 2013. URL: http://www.lncc.br/tdmc/tde_busca/arquivo.php?codArquivo=273.

[68] P.J. Blanco et al. "Trends in the computational modeling and numerical formulation of the cardiovascular system". In: *Scientific Computing Applied to Medicine and Healthcare. Current State and Future Trends at the INCT-MACC the Brazilian National Institute of Science and Technology in Medicine Assisted by Scientific Computing*. Ed. by R.A. Feijóo, A. Ziviani, and P.J. Blanco. INCT-MACC - National Institute of Science and Technology in Medicine Assisted by Scientific Computing, 2012. Chap. 2, pp. 29–77. URL: http://macc.lncc.br/relatorio/HTML/LivroAnnualReportMACC2011.php.

[69] R.A. Feijóo, A. Ziviani, and P.J. Blanco, eds. *Scientific Computing Applied to Medicine and Healthcare. Current State and Future Trends at the INCT-MACC the Brazilian National Institute of Science and Technology in Medicine Assisted by Scientific Computing*. INCT-MACC - National Institute of Science and Technology in Medicine Assisted by Scientific Computing, 2012. URL: http://macc.lncc.br/relatorio/HTML/LivroAnnualReportMACC2011.php.

[70] C.Geuzaine and J.F. Remacle. *Gmsh*. Version 4.6.0. June 22, 2020. URL: http://http://gmsh.info/.

[71] OpenCFD Ltd. *Schemes*. 2016. URL: https://www.openfoam.com/documentation/guides/latest/doc/guide-schemes.html (visited on 04/05/2022).

[72] R. F. Warming and R.M. Beam. "Upwind Second-Order Difference Schemes and Applications in Aerodynamic Flows". In: *AIAA Journal* 14.9 (1976), pp. 1241–1249. DOI: 10.2514/3.61457. eprint: https://doi.org/10.2514/3.61457. URL: https://doi.org/10.2514/3.61457.

[73] OpenCFD Ltd. *Corrected surface-normal gradient scheme*. 2016. URL: https://www.openfoam.com/documentation/guides/latest/doc/guide-schemes-sngrad-corrected.html (visited on 04/05/2022).

[74] R.I Issa. "Solution of the implicitly discretised fluid flow equations by operator-splitting". In: *Journal of Computational Physics* 62.1 (1986), pp. 40–65. ISSN: 0021-9991. DOI: https://doi.org/10.1016/0021-9991(86)90099-9. URL: https://www.sciencedirect.com/science/article/pii/0021999186900999.

[75] L.S. Caretto et al. "Two calculation procedures for steady, three-dimensional flows with recirculation". In: *Proceedings of the Third International Conference on Numerical Methods in Fluid Mechanics*. Ed. by H. Cabannes and R. Temam. Berlin, Heidelberg: Springer Berlin Heidelberg, 1973, pp. 60–68. ISBN: 978-3-540-38392-5.

[76] W.F. Boron and E.L. Boulpaep. *Medical Physiology E-Book*. Elsevier Health Sciences, 2016. ISBN: 9781455733286. URL: https://books.google.nl/books?id=6QzhCwAAQBAJ.

[77] R. Nave. *Fluid velocity profile*. URL: http://hyperphysics.phy-astr.gsu.edu/hbase/pfric.html#vel.

[78] W. Schroeder, K. Martin, and B. Lorensen. *The Visualization Toolkit*. 4th ed. Kitware, 2006. ISBN: 978-1-930934-19-1.

[79] L. Li et al. "Efficient Hyperparameter Optimization and Infinitely Many Armed Bandits". In: *CoRR* abs/1603.06560 (2016). arXiv: 1603.06560. URL: http://arxiv.org/abs/1603.06560.

[80] S.J. Reddi, S. Kale, and S. Kumar. "On the Convergence of Adam and Beyond". In: *International Conference on Learning Representations*. 2018. URL: https://openreview.net/forum?id=ryQu7f-RZ.

[81] Delft High Performance Computing Centre (DHPC). *DelftBlue Supercomputer (Phase 1)*. https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase1. 2022.

[82] R.C. Gonzalex and R.E. Woods. "Intensity Transformations and Spatial Filtering". In: *Digital Image Processing*. Upper Saddle River: Pearson Prentice Hall, 2008, pp. 144–150. ISBN: 978-0131687288.

[83] R. Farsani E., Pazouki, and J. Jecei. "A Transformer Self-Attention Model for Time Series Forecasting". In: *Journal of Electrical and Computer Engineering Innovations* 9 (Jan. 2021), pp. 1–10. DOI: 10.22061/JECEI.2020.7426.391.

[84] M. Stender et al. "Up-Net: A generic deep learning-based time stepper for parameterized spatio-temporal dynamics". In: (2022). URL: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4053304.