Convolutional Neural Network for Fluid Dynamics

Thesis Project

Sylle Hoogeveen
4445082

MSc Applied Mathematics Student

Department of Electrical Engineering,
Mathematics & Computer Science

April 12, 2022

Contents

Nomenclature

- NN - Neural network
- ROM - Reduced Order Model
- PINN - Physics Informed Neural Network
- PRNN - Physics Reinforced Neural Network
- CNN - Convolutional Neural Network
- POD - Proper Orthogonal Decomposition
- PCA - Principal Component Analysis
- PGD - Proper Generalized Decomposition
- RD - Reduced Basis
- ML - Machine Learning
- LSTM - Long-Short Term Memory
- SVD - Singular Value Decomposition
- CAE - Convolutional Autoencoder
- RNN - Recurrent Neural Network
- GRU - Gated Recurrent Unit
- CFD - Computational Fluid Dynamics

1 Introduction

1.1 Relevance

1.2 Aim and Objectives

The aim of this study is to construct a single reduced order model for time-dependent incompressible flow that can account for different boundary conditions, material parameters and geometries (computational domains). As test case, blood flow during one heartbeat will be used. The objectives are retaining accuracy of simulated velocity field by OpenFoam, and obtaining speed-up between the OpenFoam simulations and machine learning predictions.

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 different domains of computation. To create a more accurate model, usually the computational load increases and the generalizability decreases. This leads to the research question:

"What is possible in terms of computation time, accuracy and generalizability, for time-dependent flow simulations with use of the state-of-the-art machine learning techniques?"

This question can be divided into sub-questions:

1. Which ML network architecture can predict consecutive time steps of time-dependent flow the most accurate?
2. How accurate are ML predictions compared to OpenFoam simulations?
3. How much speed-up can be achieved between OpenFoam simulations and ML predictions?
4. How much can computational domains differ while retaining accuracy?

2 | Preliminaries & Related work

In this chapter relevant theory and related research is 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 given in section 2.3.

2.1 Reduced order methods

Reduced order models (ROMs) aim to capture the most important features of a physical phenomena being simulated, whilst reducing the computational load compared with full order/high-fidelity models. The increase in complexity of mathematical models in an attempt to approximate reality and desire to have near real-time simulations, for example in medical image-guidance [1] or creating digital twins [2], have emphasized the need for such strategies. There are two approaches to this task. 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 [3].

Some well known projection-based methods are proper orthogonal decomposition (POD), reduced basis (RD) 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 though eigendecomposition of the covariance matrix or singular value decomposition (SVD) of the data matrix. The data set my origin from various sources, for example numerical simulations or physical problems. The process of finding principal components is called principal component analysis (PCA) [4]. RD methods find an approximate solution to a parametrized 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 [5]. 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 [6].

The drawback of using these methods is that they usually rely on linear basis functions [7]. In contrast, the underlying dynamics are often non-linear. Also, due to discretizing, they depend on fixed computational domain (fixed geometry). 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 [8], [9].

Although many interesting research can be found on the use of classic ROMs, this research is limited to discussing them further only when used in combination with machine learning (ML) techniques. Especially in the 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 [10], ML can be of help.

2.2 Machine learning framework

Instead of using classical numerical methods, 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 network that could be used for creating a reduced order representation are convolutional neural networks and autoencoders. For time evolution, plain 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 CFD is given.

2.2.1 Neural Network

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_i) and summed, the bias (b) is added, and the result is passed though an activation function (act) to determine the output of a neuron (y), as represented in equation 2.1. The output of a neuron is then again the input of a neuron in the next layer, until the output layer is reached.

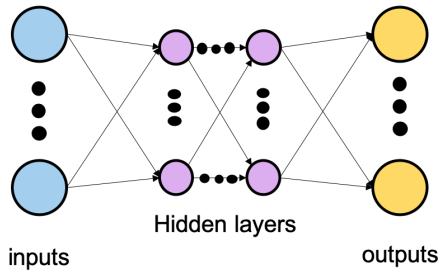


Figure 2.1: Neural Network

$$y = act(\sum_i (w_i * x_i) + b) \tag{2.1}$$

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 drop-out probability to layers of neurons [11].

The activation functions used in the network can 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 for are [12]:

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

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

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

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

$$\text{sigmoid function: } \sigma(x) = \frac{1}{1 + e^{-x}}$$

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

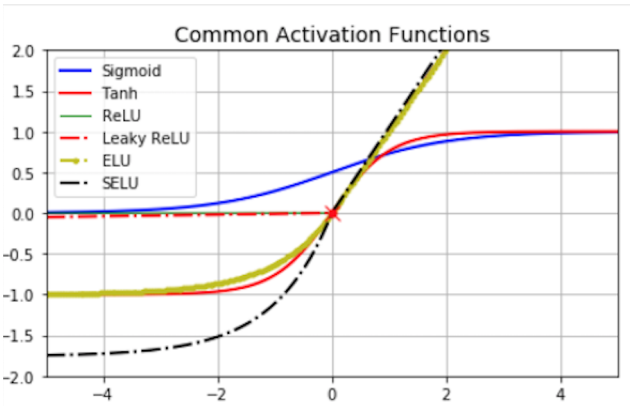


Figure 2.2: Activation functions[1]

[1] figure from: https://medium.com/@kshitijkhurana3010/activation-functions-in-neural-networks-ed88c56b611b

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 [13]. 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 are adjusted, through the backpropagation algorithm [14], such that a loss function is minimized. 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 option of choice [15]. Adam is based on stochastic gradient descent (SGD) and utilizes the concepts of momentum and adaptive learning. Here, momentum is defined as using the previous gradients for determining current gradient (equation 2.2), such that there is less oscillation in finding the minimum compared to SGD. Thus the learning rate can be increased, leading to faster convergence [16]. Adaptive learning refers to having a different the learning rate (λ) per parameter at each update time (equation 2.3) [17]. Together, they result in an update of the weights and biases determined by equation 2.4. 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.

$$M_{dw_t} = \beta \frac{\partial L_{t-1}}{\partial w_{t-2}} + (1-\beta)\frac{\partial L_t}{\partial w_{t-1}} \tag{2.2}$$

$$V_{dw_t} = \gamma \frac{\partial L_{t-1}}{\partial w_{t-2}} + (1-\gamma)\big(\frac{\partial L_t}{\partial w_{t-1}}\big)^2 \qquad \lambda = \frac{\eta}{\sqrt{V_{dw_t}} + \epsilon} \tag{2.3}$$

$$w_t = w_{t-1} - \lambda * M_{dw_t} \tag{2.4}$$

The most naive way of predicting future time steps with machine learning is using a plain neural network. With a NN, no assumption on the input data is made (as assumed spatial relation in convolutional neural networks) and there is no mechanism to retain information of data on which previous prediction were made (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 plain NN could have less complexity, which is desirable as it is computationally and memory-wise more efficient.

Using a NN for time evolution in combination with using ML techniques for reducing order was explored by Kim et al. [7]. 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 Network

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 filter operations (or filters) are the weights of the network, and are learned during training phase. Different filters uncover different relations in the input data. For example the filter $\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$ 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 [18].

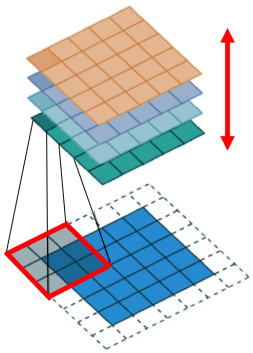
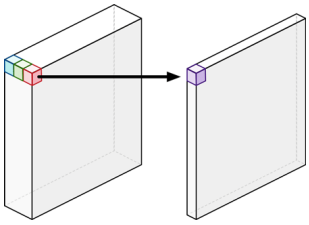
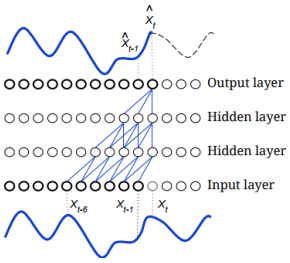


Figure 2.3: Convolution[2]

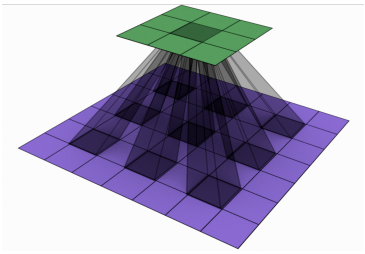
There exist multiple types of convolutions. The first type, normal convolutions, move over each channel of an input image or one feature map of the previous layer separately, as depicted in figure 2.3. In contrast, pointwise convolutions are 1x1 convolution kernels that perform a filter operation on the entire depth of the previous layer. Hence the depth of the successive layer can be controlled, as shown in figure 2.4a. 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 [19]. Dilatation is a method to increase the receptive field whilst decreasing the number of layers and thus the number of parameters to train. Dilatation can also be applied to normal convolutions. This is done by skipping over certain steps, as depicted in figure 2.4c



(a) Pointwise convolution[3]



(b) Causal convolutions for 1D sequence[4]



(c) Dilated convolutions[5]

Figure 2.4: Different types of convolution

A CNN can consist of multiple convolutional layers, each reducing the dimension of the input by kernel-size -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.

Lopez-Martin et al. [21] created one deep CNN with 3D convolutional layers to reduce dimensionality of velocity fields generated by a synthetic jet simulation. 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 tensor is reshaped such that it can be broken into the number of time steps to be predicted vectors. These vectors are then used as input to the fully connected layers, which perform the time evolution .

[2] Adaptation of original figure from [18]
[3] Figure from [20]
[4] Figure from: https://discuss.pytorch.org/t/causal-convolution/3456
[5] 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 Autoencoder

An autoencoder 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 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.

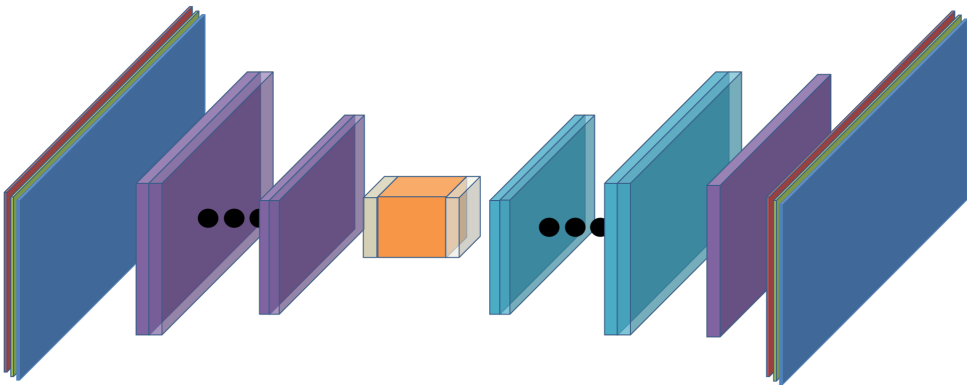


Figure 2.5: Autoencoder structure for RGB image

Also for the purpose of creating reduced order models autoencoders have been used. Simpson et al. [22] used a simple autoencoder, consisting of layers of normal neuron, 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 sections above. Kim et al. [7] and Eichinger et al [23] 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 [23].

Kim et al. [7] used an CAE network to generate smoke and fluid simulations for graphical implementations (games, videos). Their approach led to 700x speedup compared to MantaFlow simulations while maintaining accuracy (on visual inspection) for a variety of fluid behaviour. However, the ability to reconstruct physically accurate scenarios depended heavily on the how closely the scenarios matched the input training data.

Due to the depth of the networks, the vanishing gradient problem can occur. The large number of steps in backpropagation though 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. [24] as residual networks (ResNets).

A (different) Kim et al. [25] 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. This was possible as they developed a hyper-reduction technique exploiting classic numerical methods for solving PDEs/ODEs. 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 Network

Recurrent NNs (RNNs) can be built with different types of units: standard units, long short-term memory (LSTM) units or gated recurrent units (GRUs). (Also RNNs can have a specific architecture called an Encoder-Decoder network with or without attention mechanism.) All these techniques will be discussed in the following sections.

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 one way 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 an unit (explained in the following sections).

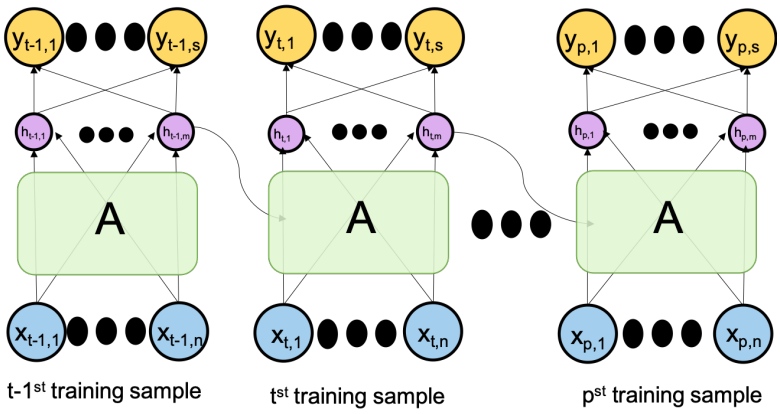


Figure 2.6: RNN with n input neurons, m hidden state variables, s output neurons and p time steps

Standard Unit

A standard unit or standard RNN cell has two inputs: x_t, 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 results is passed through an activation function, such as the hyperbolic tangent function. The outcome is a new hidden state: $h_t = act(W * x_t \frown h_{t-1} + b)$ 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 [26].

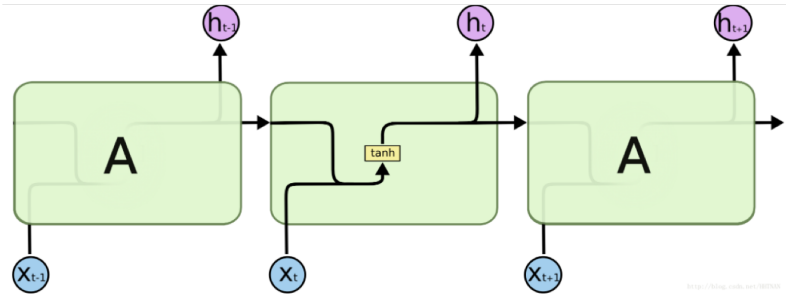


Figure 2.7: Standard unit RNN[6]

However, the vanishing gradient problem occurs when the sequence is very long [26]. 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 to no weight updates and learning halts.

[6] 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 [27]. The LSTM unit has three inputs instead for 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 [28]. Similar to the idea of adding residual connections in deep NNs, this helps preserve the gradient. In the LSTM unit, the three gates are computed from the concatenation $x = x_t^\frown h_{t-1}$, each having their own weight matrix:

- Forget gate $f = \sigma(W_f x + b_f)$
- Update gate $u = \sigma(W_u x + b_u)$
- Output gate $o = \sigma(W_o x + b_o)$

Here, σ is the sigmoid function which returns values between 0 and 1, killing input close to 0 and letting input 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 (update gate times $tanh(W_c * x + b_c)$): $c_t = f * c_{t-1} + u * tanh(W_c * x + b_c)$. 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): $h_t = o * tanh(c_t)$ [26]. 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 [28]. From the hidden state again the output of the network can be computed by multiplying it with weight matrix V.

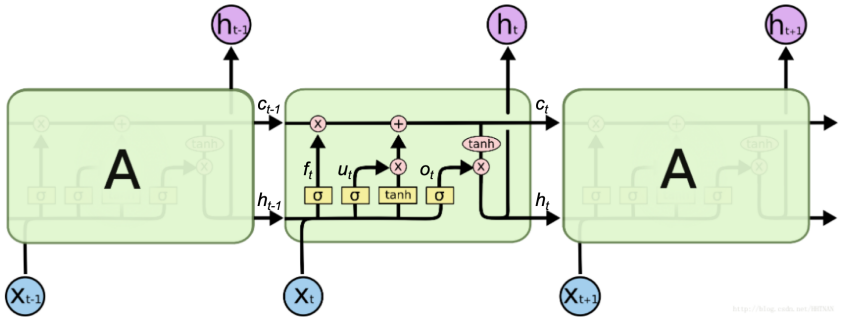


Figure 2.8: LSTM unit RNN[7]

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. [29] 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.[30] also merged a classical reduced order approach, POD with Galerkin projection, with LSTM network for time evolution in fluid mechanics simulations. Moreover, RNNs with LSTM units have been used in combination with machine learning techniques for order reduction. Simpson et al. [22] 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.

[7] figure 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 [28]. By dropping the output gate only two gates are retained:

- Reset gate $r = \sigma(W_r x + b_r)$
- Update gate $z = \sigma(W_z x + b_z)$

The reset gate is similar to the forget gate in the LSTM unit and is to decide what to forget. The update gate is to decide what should be passed to the output. First a proposed new hidden state 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: $\hat{h}_t = tanh(W\widehat{x_r}(r * h_{t-1})$. 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 reset gate determines how much of the hidden state is updated: $h_t = (1 - z_t)h_{t-1} + z_t\hat{h}_t$ [28].

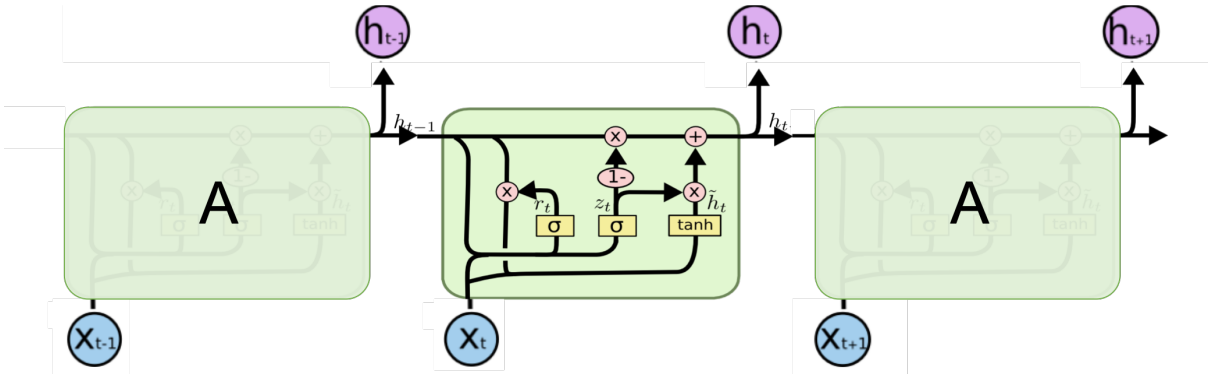


Figure 2.9: GRU RNN [8]

With only two gates, there is no mechanism to control which parts of the state is exposed to the output, but the GRU has a less complex structure. This makes the units computationally more efficient while it was proven to have comparable performance to LSTM units [31].

RNN with GRUs for time evolution have been used in combination with classical reduced order techniques. Wu et al. [32] created a latent representation of flow past a cylinder with POD-Galerkin projection and implemented a CNN with causal convolutional layers and RNNs with LSTM and GRU for comparison. The CNN achieve better performance and needed fewer parameters.

Also, GRU RRN it has been used in the prediction of blood flow characteristics by Jamil et al. [33]. In this research, inlet velocity and percentage lumen openings (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 the 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.

[8] Adaptation of figure 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

The idea of making neural networks for solving problems involving PDEs 'physics informed' was explored by Raissi et al. [34]. Information about the underlying physics of the problem can be inserted into the NN by constructing activation and loss functions specific for the underlying differential operator. All network types before mentioned thus can be physics informed, if the activation and/or loss function is tailored to the PDE.

Chen et al. [35] created a neural network to predict the projection coefficients of the POD-Galerkin projection (PDNN), a neural network to predict the reduced order solution (PINN) and a neural network to predict the projection of high-fidelity solution on the reduced space (PRNN) by incorporating appropriate terms into the loss function.

To ensure conservation of mass (thus non-divergence) for incompressible fluid dynamics, Kim et al. [7] introduced a stream function based loss function in the decoder part of their network. In the stream loss function as described in equation 2.5, $G(\mathbf{c})$ is the output of the network and $\mathbf{u_c}$ is a simulation sample from the data set.

$$L_G(\mathbf{c}) = \lambda_{\mathbf{u}}||\mathbf{u_c} - \nabla \times G(\mathbf{c})||_1 + \lambda_{\nabla \mathbf{u}}||\nabla \mathbf{u_c} - \nabla(\nabla \times G(\mathbf{c}))||_1 \tag{2.5}$$

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 standard loss functions 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. [36] took a different approach, embedding hard physical constraints in the neural network architecture. This was done by adding non-trainable layers with physical input after a CAE structure. The decoder in this network has the vector potential as output. The non-trainable layers consists of a layer enforcing BCs with ghost cells, a layer which computes all spatial derivatives and a last layer to compute the curl on the vector potential field.

A combination of hard and soft physical enforcement was implemented by Sun et al. [37] 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, IC and Dirichlet BC are enforced by constructing the NN ansatz $\hat{\mathbf{u}}$ and $\hat{p}$ with a particular solution as described in equation 2.6,

$$\begin{aligned} \hat{\mathbf{u}}(t, \mathbf{x}) &= \mathbf{u}_{par}(t, \mathbf{x}) + D(t, \mathbf{x})\tilde{\mathbf{u}}(t, \mathbf{x}) \\ \hat{p}(t, \mathbf{x}) &= p_{par}(t, \mathbf{x}) + D((t, \mathbf{x})\tilde{p}(t, \mathbf{x}) \end{aligned} \tag{2.6}$$

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 on 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.

Physics can also be injected in a time-series forecasting networks. 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 [30].

2.2.6 Other Machine Learning use in CFD

Machine learning has not only been used to produce reduced order representations and perform time evolution. Other attempts have been made using ML to combat 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 [38]. Also, ML was used for hyperreduction, in situations were dimensionality reduction by projection-based ROMs does not lead to computation speed-up. This occurs when the computation of reduced-order operators is expensive [39]. 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 [40]. Similarly, ML was also used to reconstruct missing or faulty flow fields in CFD simulations [41].

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 [42] and estimating the output error and performing mesh adaptation for CFD simulations [43].

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 given in this section. Also, a brief discussion is presented on other ML applications in cardiovascular modelling.

2.3.1 Mathematical representation & material parameters

Blood is a non-Newtonian fluid, however, it was found that the non-Newtonian impact for Reynold numbers between 110 and 850 in big arteries is negligible [44]. 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 [45], adapted from 3D to 2D (equation 2.7).

$$\begin{aligned} \nabla \cdot \mathbf{u} &= 0 \\ \frac{\partial}{\partial t}(\mathbf{u}) + (\nabla \mathbf{u})\mathbf{u} + \nabla \frac{p}{\rho} - \nabla \cdot (2\nu_2 D(\mathbf{u})) &= 0 \end{aligned} \tag{2.7}$$

In which $\mathbf{u}$ is the velocity, p is the mean normal stress ($p = -\frac{1}{2}(tr\mathbf{T})$ with $\mathbf{T} = -p\mathbf{I} + 2\mu\mathbf{D}$ the fluid stress), ν is the kinematic viscosity ($\nu = \frac{\mu}{\rho}$, with μ the dynamic viscosity) and $\mathbf{D}$ the strain-rate tensor ($\mathbf{D} = \frac{1}{2}(\nabla \mathbf{u} + (\nabla \mathbf{u})^T)$). The dynamic viscosity is reported between 3.5 and 5.5 cP, which is equal to 0.003 and 0.0055 $\frac{Ns}{m^2}$. However, in reality, the viscosity of blood has a much larger range dependent on hemodynamic conditions [46]. Similarly, blood density differs depending on gender and moreover, body position. It was set to the average blood density of 1060 $\frac{kg}{m^3}$ [47]. The artery walls are modeled as rigid, which is again simplification of reality. To model artery walls more accurately, hyperelastic and viscoelastic material models can be incorporated [48].

2.3.2 Geometry dimensions & inlet velocities & pressures

The aorta can roughly be divided into four segments; the ascending aorta and aortic arch (1), descending thoracic aorta (2), abdominal aorta (3) and bifurcation of the aorta into iliac arteries (4), as shown in figure 2.10. This research will constrict itself to modeling these segments with one inlet and one or two outlets. The abdominal aorta will be modelled with the renal artery as second outlet and the bifurcation of the aorta will have the two common iliac arteries as outlets.The diameter of the aorta differ 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 [49],[50],[51] and [52] are shown in the second column.

Aorta segment	diameter range in mm	length in mm
Aortic arch (1)	22 - 36	
Descending aorta (2)	20 - 30	
Abdominal aorta (3)	15.2 - 22.5	
Bifurcation aorta (4)	10 - 23	
Renal artery (3)	6.3 - 7.1	
Iliac common artery (4)	6.5 - 16.5	

Table 2.1: Aorta dimensions

The inlet velocity during one heartbeat ranged from 0 - 71 cm/s, as measured in [53]. 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 [54].

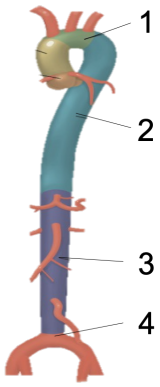


Figure 2.10: Anatomy aorta[9]

2.3.3 ML for 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 [55]. Also hemodynamic parameters, such as static pressure, wall-shear-stress, secondary flow degree and specific kinematic energy, were predicted using NNs [56]. 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.

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

3 | Methodology

3.1 Data Generation

To solve the mathematical problem stated in the previous section numerically, OpenFoam 9 software was used. In particular the IcoFoam solver, for incompressible, Newtonian fluids was chosen. All code to generate data can be found in [GITHUB REPO]. In each section the corresponding file will be mentioned.

3.1.1 Equations

The IcoFoam solver solves the following equation:

$$\begin{aligned} \nabla \cdot \mathbf{u} &= 0 \\ \frac{\partial}{\partial t}(\mathbf{u}) + \nabla \cdot (\mathbf{u} \otimes \mathbf{u}) - \nabla \cdot (\nu_1 \nabla \mathbf{u}) &= -\nabla p_k \end{aligned} \tag{3.1}$$

In which $\mathbf{u}$ is the velocity in $\frac{m}{s}$ and p_k is the kinematic pressure ($p_k = \frac{p}{\rho}$, with ρ the density) in $\frac{m^2}{s^2}$. Equations 2.7 and 3.1 are equivalent as shown below.

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

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

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

$$\nabla \cdot (\nu_1 \nabla \mathbf{u}) = \nabla \cdot (\nu_1 \begin{bmatrix} \frac{\partial u_x}{\partial x} & \frac{\partial u_x}{\partial y} \\ \frac{\partial u_y}{\partial x} & \frac{\partial u_y}{\partial y} \end{bmatrix}) = \nu_1 \begin{bmatrix} \frac{\partial^2 u_x}{\partial x^2} + \frac{\partial^2 u_y}{\partial x \partial y} \\ \frac{\partial^2 u_y}{\partial y^2} + \frac{\partial^2 u_x}{\partial x \partial y} \end{bmatrix} \overset{2}{=} \nu_1 \begin{bmatrix} \frac{\partial^2 u_x}{\partial x^2} \\ \frac{\partial^2 u_y}{\partial y^2} \end{bmatrix} \overset{3}{=} \nu_2 \begin{bmatrix} 2\frac{\partial^2 u_x}{\partial x^2} \\ 2\frac{\partial^2 u_y}{\partial y^2} \end{bmatrix}$$

$$\overset{2}{=} \nu_2 \begin{bmatrix} 2\frac{\partial^2 u_x}{\partial x^2} + \frac{\partial^2 u_y}{\partial x \partial y} + \frac{\partial^2 u_x}{\partial y^2} \\ 2\frac{\partial^2 u_y}{\partial y^2} + \frac{\partial^2 u_x}{\partial x \partial y} + \frac{\partial^2 u_y}{\partial x^2} \end{bmatrix} = \nabla \cdot (\nu_2 \begin{bmatrix} 2\frac{\partial u_x}{\partial x} & \frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x} \\ \frac{\partial u_y}{\partial x} + \frac{\partial u_x}{\partial y} & 2\frac{\partial u_y}{\partial y} \end{bmatrix}) = \nabla \cdot (\nu_2 (\begin{bmatrix} \frac{\partial u_x}{\partial x} & \frac{\partial u_x}{\partial y} \\ \frac{\partial u_y}{\partial x} & \frac{\partial u_y}{\partial y} \end{bmatrix} + \begin{bmatrix} \frac{\partial u_x}{\partial x} & \frac{\partial u_y}{\partial x} \\ \frac{\partial u_x}{\partial y} & \frac{\partial u_y}{\partial y} \end{bmatrix}))$$

$$= \nabla \cdot (2\nu_2 * \frac{1}{2}(\nabla \mathbf{u} + (\nabla \mathbf{u})^T))) = \nabla \cdot (2\nu_2 \mathbf{D}(\mathbf{u}))$$

2) Use $\frac{\partial u_x}{\partial y} = \frac{\partial u_y}{\partial x} = 0$
3) Take $\nu_1 = 2\nu_2$

3.1.2 Boundary conditions

For the velocity component a time dependent function was generated to replicate the velocity pattern during one heartbeat of the measurements in [53]. The blue function in graph 3.1 was converted to tabular values, exported to the inlet_v.csv file and referred to as 'UniformFixedValue' type boundary condition (BC) for the inlet boundary in the 0/U file. The outlet condition for velocity was set to zero gradient, defined as the Neumann BC: $\frac{\partial \mathbf{u}}{\partial t} = 0$. *Note that here the assumption is made that the velocity is fully developed and constant at the outlet boundary.* No slip boundary conditions were implemented for all walls. This is defined as the Dirichlet BC: $\mathbf{u} = 0$.

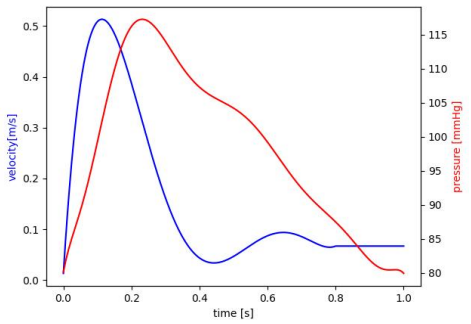


Figure 3.1: time dependent velocity inlet and pressure outlet boundary condition

The pressure boundary conditions were all defined as zero gradient, 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. *Also the assumption is made that the pressure is constant at the inlet boundary.* The outlet BC is a again a function of time, as shown in graph 3.1 by the red line, created to simulate the aortic pressure during one heartbeat. The graph was again converted to tabular values, exported to the outlet_p.csv file and referred to as 'UniformFixedValue' type BC in 0/p file.

3.1.3 Geometries

Four different geometries, shown in figure 3.2, were created to approximate the four aorta segments described in section 2.3. The meshes were made using the GMSH python library and by performing the following steps:

1. define 2D mesh by creating points, lines and a surface in the xy-plane
2. extrude the surface in the z-direction by one cell (this is necessary as OpenFoam only takes 3D meshes)
3. add all surfaces to appropriate Physical Groups and name these groups
4. save the mesh as .msh2 file (this is compatible with gmshToFoam function)
5. in openfoam, run gmshToFoam on mesh file
6. set front and back patches to empty type in constant/polyMesh/boundary file, *set inflow and outflow boundaries to patch type and set other boundaries to wall type*

The code to create the meshes can be found in create_mesh.py and all OpenFoam commands are listed in the Allrun executable file.

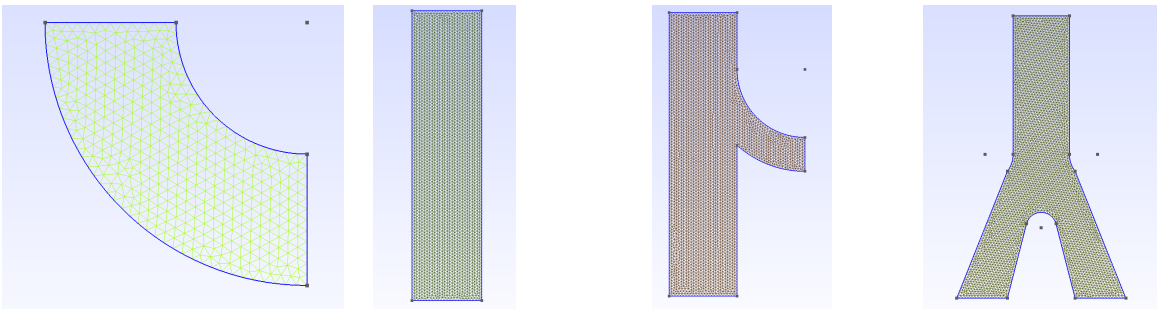


Figure 3.2: Meshes of four geometries representing four aortic segments. From left to right: aortic arch = channel bend, descending aorta = channel straight, abdominal aorta = channel branch and bifurcation aorta = channel bifurcation

Note that the inflow boundary is always at the top.

3.1.4 Schemes

To numerically represent the equations, the Finite Volume Method (FVM) is used. FVM is based on integration over a 3D control volume (small volume around a point in the unstructured mesh), converting divergence terms in the equation to surface integrals by the divergence theorem and evaluating these terms as fluxes at the boundary of each volume. The method is conservative.

To discretise the equations multiple schemes are used and specified in the system/fvSchemes file. There are many options, which can be found in OpenFoam documentation [57]. For this research, the default schemes supplied by OpenFoam were not altered, and are listed below.

- Time derivative : Euler implicit method
- Gradient : Gauss theorem
- Divergence : Gauss theorem
- LaPlacian : Gauss theorem

Besides these schemes for terms in the equations, an interpolation scheme is needed for transforming cell-centre quantities to surface centres. This was supplied as linear with the exception of the divergence

term. There linear-upwind was chosen [58]. In addition, the LaPlacian term needs an surface-normal gradient scheme, which was set to 'corrected' [59]. *For turbulence models the distance to the nearest wall is needed. This distance was calculated with mesh-wave* **[OF_meshwave]**

3.1.5 Solver

As mentioned IcoFoam from Openfoam 9 was chosen as solver, which implements the Pressure Implicit with Splitting Operators (PISO) algorithm. This technique is characterized by decoupling the operations on pressure from those on velocity in the solving process [60]. For solving the linear equations obtained after applying the discretization schemes iterative methods were used. The preconditioned conjugate gradient (PCG) with simplified Diagonal-based Incomplete Cholesky (DIC) as preconditioner was used to solve the pressure equations and smooth solver with symmetric Gauss-Seidel as smoother was used to solve the velocity equations. These options are specified in the system/fvSolution file. The solver is run by the icoFoam command.

3.1.6 Data Generation Pipeline

Combining all sections above, an automated data pipeline was created such that input for the NNs could be generated swiftly. The pipeline takes scenario to run (bend, straight, branch or bifurcation), geometry parameters (channel width) *and bend angles, secondary channel width if applicable, and boundary condition parameters, for example peak velocity and blood pressure*, as input. The geometry input parameters were used to create an unstructured mesh and with the boundary condition input parameters the time dependent velocity inlet and pressure outlet values were generated. These were all loaded into the openfoam Docker image. In the Docker image, the GMSH mesh was converted to a Foam polymesh and the dict files were adjusted as described in above section. Next IcoFoam solver was run creating Foam output. This Foam output was converted to VTK files with the "foamToVTK -allPatches" command and thereafter converted again to PNG image files. The PNG image files are the output of the data generation pipeline and input for the NNs, an example for each scenario is shown in figure 3.4. The whole data pipeline is depicted in figure 3.3 below.

Figure 3.3: Data Generation Pipeline, icons from: Flaticons.com

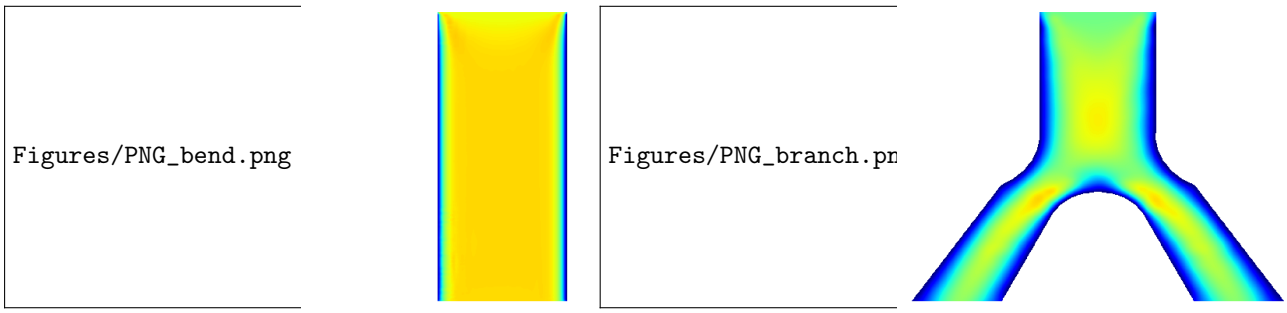


Figure 3.4: Output of four scenarios. From left to right: bend, straight, branch and bifurcation

3.2 Network Architectures

Taking into account the possibilities discussed in chapter 2, first of all a convolutional autoencoder was chosen for order reduction, given the high dimensionality of the input data and desired flexibility in the computational domain. Two network architectures are proposed: an autoencoder with a NN for time evolution and an autoencoder with a RNN using GRUs for time evolution.

Secondly, a 3D convolutional network with dimensions (x,y,t) was explored.

3.2.1 Auto-encoder

3.2.2 Neural Network

3.2.3 Recurrent Neural Network with GRUs

3.3 Tuning, Training and Testing protocols

4 | Results

5 | Discussion

6 | Conclusions and Recommendations

Bibliography

[1] Zeike Taylor, Stuart Crozier, and Sébastien Ourselin. "Real-Time Surgical Simulation Using Reduced Order Finite Element Analysis". In: vol. 13. Sept. 2010, pp. 388–95. ISBN: 978-3-642-15744-8. DOI: 10.1007/978-3-642-15745-5_48.

[2] Dirk Hartmann, Matthias Herz, and U. Wever. "Model Order Reduction a Key Technology for Digital Twins". In: Apr. 2018, pp. 167–179. ISBN: 978-3-319-75318-8. DOI: 10.1007/978-3-319-75319-5_8.

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

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

[5] Federico Negri Alfio Quarteroni Andrea Manzoni. *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.

[6] Adrien Leygue Francisco Chinesta. Roland Keunings. *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.

[7] Byungsoo 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.

[8] My Ha Dao. *Projection-Based Reduced Order Model for Simulations of Nonlinear Flows with Multiple Moving Objects*.

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

[10] Toni Lassila et al. "Model Order Reduction in Fluid Dynamics: Challenges and Perspectives". In: *Reduced Order Methods for Modeling and Computational Reduction*. Ed. by Alfio Quarteroni and Gianluigi 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.

[11] Yingjie Tian and Yuqi 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.

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

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

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

[15] Jimmy Ba Diederik P. Kingma. "Adam: A Method for Stochastic Optimization". In: 2015. DOI: 10.48550/arXiv.1412.6980.

[16] Ning 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.

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

[18] Vincent Dumoulin and Francesco 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.

[19] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. *An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling*. 2018. arXiv: 1803.01271 [cs.LG].

[20] Nghia Duong Trung, Luyl Da Quach, and Chi-Ngon Nguyen. "Learning Deep Transferability for Several Agricultural Classification Problems". In: *International Journal of Advanced Computer Science and Applications* 10 (Feb. 2019).

[21] Manuel Lopez-Martin, Soledad Le Clainche, and Belen 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.

[22] Thomas Simpson, Nikolaos Dervilis, and Eleni 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.

[23] Axel Klawonn Matthias Eichinger Alexander Heinlein. *Technical Report Series Center for Data and Simulation Science Surrogate Convolutional Neural Network Models for Steady Computational Fluid Dynamics Simulations SURROGATE CONVOLUTIONAL NEURAL NETWORK MODELS 1 FOR STEADY COMPUTATIONAL FLUID DYNAMICS 2 SIMULATIONS*. 2016.

[24] Kaiming 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.

[25] Youngkyu 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.

[26] Robin 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.

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

[28] Junyoung 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.

[29] 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.

[30] Suraj 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.

[31] Shudong Yang, Xueying Yu, and Ying 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.

[32] Pin 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.

[33] Ishat Raihan Jamil and Mayeesha 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.

[34] 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.

[35] Wenqian Chen et al. "Physics-informed machine learning for reduced-order modeling of nonlinear problems". In: *Journal of Computational Physics* 446 (2021). ISSN: 10902716. DOI: 10.1016/j.jcp.2021.110666.

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

[37] Luning 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.

[38] Dmitrii 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.

[39] S. Grimberg and C. Farhat. "Hyperreduction of cfd models of turbulent flows using a machine learning approach". English. In: *AIAA Scitech 2020 Forum*. Vol. 1 PartF. Cited By :5. 2020, pp. 1–13. URL: www.scopus.com.

[40] Kevin 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.

[41] S. Lee, I. G. Kevrekidis, and G. E. Karniadakis. "Resilient algorithms for reconstructing and simulating gappy flow fields in CFD". English. In: *Fluid Dynamics Research* 47.5 (2015). Cited By :2. URL: www.scopus.com.

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

[43] G. Chen and K. J. Fidkowski. "Output-based error estimation and mesh adaptation using convolutional neural networks: Application to a scalar advection-diffusion problem". English. In: *AIAA Scitech 2020 Forum*. Vol. 1 PartF. Cited By :4. 2020. URL: www.scopus.com.

[44] K. Beenutty and Anburajan 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.

[45] 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.

[46] Elie 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.

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

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

[49] Arturo 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.

[50] Moawia 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.

[51] Maria 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.

[52] Harald Vik-Mo Ole Martin Pedersen Aslak Aslaksen. "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.

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

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

[55] Liang Liang, Wenbin Mao, and Wei Sun. "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.

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

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

[58] R. F. Warming and Richard 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.

[59] 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).

[60] 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.

[61] Amirhossein Arzani and Scott T.M. Dawson. *Data-driven cardiovascular flow modelling: Examples and opportunities*. 2021. DOI: 10.1098/rsif.2020.0802.

[62] H. Švihlová et al. "Determination of pressure data from velocity data with a view towards its application in cardiovascular mechanics. Part 2. A study of aortic valve stenosis". In: *International Journal of Engineering Science* 114 (2017). ISSN: 00207225. DOI: 10.1016/j.ijengsci.2017.01.002.

[63] Amir Barati Farimani, Joseph Gomes, and Vijay S Pande. *Deep Learning the Physics of Transport Phenomena*. 2017.

[64] M. Cheng et al. "An advanced hybrid deep adversarial autoencoder for parameterized nonlinear fluid flow modelling". In: *Computer Methods in Applied Mechanics and Engineering* 372 (Dec. 2020). ISSN: 00457825. DOI: 10.1016/j.cma.2020.113375.