Neural Network Integration for the Variational Multiscale Method

Master Thesis

Author

Niklas Perujo - 4534859

Supervisor

Dr. Steven Hulshoff

Date of Defence
January 28, 2025

Abstract

This thesis presents a data-driven closure model for the variational multiscale method. The model is trained and tested in the context of a turbulent channel flow with $Re_\tau = 180$. Focus is on predicting the closure terms of the momentum equations. The model is trained using norms which only take into consideration local flow accuracy. For this reason long term stability is not addressed explicitly and is not the focus of this work.
A convolutional neural network is chosen for its ability to efficiently make use of local spatial and temporal data. Different architectures and inputs are considered for the model. The performance is analyzed in an a priori sense by plotting correlations between the predicted closure terms and the exact closure terms. Using integrated forms of the momentum equation as input produces the highest a priori correlations. Taking multiple time steps as input also proves to be important. The model is then integrated into the simulation of a turbulent channel flow. The closure terms produced by the model are more accurate than those of a modern algebraic model. The flow field produced by the model is thus closer to that of the DNS than the flow field of the algebraic model.

Acknowledgments

This thesis has been a long time in the making and has taught me a lot about the usefulness of only biting off as much as you can chew. I had a lot of support throughout the entire process and could not have done this without a lot of people.
Firstly I would like to thank my parents for supporting me the entire time and never doubting that I would finish. I should also thank my siblings, especially my sister, for encouraging me (yelling at me) to keep going and to pick up the pace. Without them I would truly be lost.
I would also like to thank my friends that I met here in Delft. You truly have made the last few years enjoyable and have made this whole process worth it. The memories I have made are worth as much as finishing this degree. Of course this cannot be said without mentioning my dodgeball group. Through them I have met so many wonderful people and can look forward to every week.
Lastly I thank my supervisor Dr. Hulshoff for helping me at every step and having endless patience for all the things I wanted to do. His guidance and willingness to go along with my ideas has helped me tremendously and this project would not be possible without him.

Niklas Perujo, 2025

Contents

List of Figures

List of Tables

1 Introduction

Turbulence modeling has been an evolving area of research for decades. A core issue is making sure the entire range of turbulent scales is well represented in a simulation or model. Large Eddy Simulations (LES) tackle this problem by filtering out unresolved scales and modeling them with a turbulence model. In the past few years as computing technology has advanced and access to large data sets has become easier, data driven turbulence models have become a new area of research. The hope is that machine learning tools can help improve existing models or create entirely new ones [1]. Neural networks specifically have the benefit of being very flexible and are able to model a wide array of phenomena. If these tools can be implemented successfully a lot can be learned about turbulence modeling in the context of LES.

The focus of this project will be on the variational multiscale method, VMS, applied to the 3D Navier-Stokes equations. This is a type of LES that defines the solution of a system of partial differential equations as the combination of coarse and fine scales [2, 3]. VMS, unlike LES, does not us a filter but assumes a priori that the resolved scales exist as a projection in some finite-dimensional space and that the unresolved scales exist as a projection in some infinite-dimensional space. This is advantageous as it results in a set of equations that are exact for the resolved and unresolved scales. The issue that the LES filter is not commutative with spatial differentiation is avoided.

Neural networks have been used in the past to improve existing turbulence models or completely replace them as described by Beck [4] and Duraisamy [5]. The focus of the project will be on completely replacing LES models with a neural network. The benefit is that neural networks are able to model highly non-linear phenomena and could in theory be more flexible that other turbulence models. The networks are trained on direct numerical simulation (DNS) data that is projected to a coarser grid. A priori tests of trained neural nets show correlations of over 0.9 between actual closure terms from DNS data and predicted closure terms from neural networks shown by both Beck and Bettini [1, 6]. Janssens [7] also shows that a major issue is that these neural nets introduce instabilities that cause a simulation to diverge. Another issue is applying these neural networks to flows they have not been trained on. If a net is trained on data at a certain Reynolds number or only for channel flow it might not be able to produce correct closure terms for other flows.

The research objective of this project is to study if a neural network within a VMS framework can recreate a locally accurate flow velocity field of the 3D Navier-Stokes equations. Previous studies have shown that several complications arise from introducing a data driven approach to a numerical solver [8], and implementations have to be tailored to this setup. This project is a continuation of the work done by Bettini [6] and will focus on a certain type of neural network, convolutional neural networks or CNNs.

1.1 Previous Work in Data Driven Turbulence Modeling

There are a wide array of different approaches to incorporate machine learning into LES modeling. There include using machine learning to improve existing models and using machine learning to completely replace turbulence models with new ones.

In the LES space machine learning is being used in conjunction with direct numerical simulation data to train neural nets to predict closure terms. CNNs have become more common than other networks because they generally produce better results when spatial patterns are important [9]. Beck, Flad, and Munz [1] use a type of CNN called an RNN trained on DNS data to directly predict closure terms in an LES setup for homogeneous isotropic turbulence. RNNs, residual neural networks, are normal networks that include skip connections that are able to skip hidden layers such as convolution layers. This helps prevent the degradation of accuracy that is sometimes seen with deeper neural nets. The RNN is trained on DNS data that is projected onto polynomial space, P_5, using an L_2 projection. The input into the neural network are all the polynomial weights of one cell. The LES setup comes from the theory of 'Perfect LES' by Stefano and Vasilyev [10]. This combines the modeling error and numerical error to create a consistent model where the closure term can perfectly close the LES equations. This gives the neural network an opportunity to learn the perfect closure terms from the DNS data. The a priori correlations for the closure terms are about 0.3 for the outer surface of the cell and 0.73 for the inner part of the cell. This difference is attributed to the non-isotropy of the data and the filter kernel at the boundaries. Beck, Flad, and Munz [1] also prove that deeper networks increase the correlation of the terms. When the neural network is implemented in the LES setup it produces decent results for the first few time steps but eventually diverges. The model is dissipative at first but high frequency errors accumulate and the setup lacks long-term stability. Beck, Flad, and Munz introduce a mixed scale-similarity model and a Smagorinsky model which is then stable.

Shin, Ge, Lampmann, and Pfitzner [11] use a neural network to predict the subgrid scale closure term for the flame surface density in a turbulent premixed combustion simulation. They also use an LES setup and train on filtered DNS data provided by a third party. They test several different networks and produce correlations of over 0.9 with both CNNs and mulitlayer perceptrons, MLPs, and show that smaller MLPs that use only local data are also good enough for this simulation. A posteriori LES tests also show good results and can produce better results than other flame surface density models. The SHAP values are also used to interpret the results of the network. This helps them see which regions of the simulation affect the outputs. Shin, Ge, Lampmann, and Pfitzner do mention that generalization is a problem since the network is only able to produce good results for this specific type of combustion simulation. This highlights a general problem with machine learning for physics simulations. Neural nets trained on data from a specific type of simulation can only be implemented for those specific simulations. For fluid simulations the Reynolds number at which a neural net is trained can have a large effect on the usefulness of that net at other Reynolds numbers.

The combination of neural networks and the variational multiscale method has been stud-

ied in multiple projects at TU Delft. In works by Janssens [7] and Robijns [12] a neural network is coupled to a VMS form of the Burgers equation. Robijns studied the various terms that the neural net outputs, and Janssens studied the errors and instabilities that the network introduces to the problem. Janssens found two different types of instabilities named the Corrector-Pass Instability (CPI) and Long-Term Instability (LTI). The CPI comes from spurious roots that the neural net introduces when solving for the weak residual of the Navier-Stokes equations for the next time step. LTI comes from the neural networks failure to produce good results after many time steps. Pusuluri [13] improved the model by adding white and Gaussian noise to the training data. This helped stabilize the simulation with Gaussian noise outperforming white noise. Rajampeta [8] used a feature set that depended only on the previous time step to avoid the spurious roots being a problem which eliminated the CPI. The LTI were addressed by limiting the backscatter that the neural network introduces. This made the network less accurate but stabilizes the simulation. Bettini [6] used the CPI and LTI solutions for the three dimensional Navier-Stokes equations.For setup in three dimensions, a stencil was chosen for both the momentum and continuity equations based on results from Hughes and Sangalli [14]. Eight different networks were trained, one for each shape function in each cell since each shape function has a different correlation to the closure terms and skewness. Using an MLP network, Bettini produced a priori correlations for the closure terms of over 0.9. However, Krochk [15] observed that including the exact continuity closure terms in the full 3D Navier-Stokes LES simulation leads to slow convergence. To avoid this problem, for now, this thesis will follow the work of Rajampeta [8] and focus only on the momentum equations. To circumvent the continuity equation the exact pressure will be injected into the solution.

1.2 Research Questions and Justification

This thesis will continue the projects done at TU Delft. The objective is to setup a VMS-CNN model for the 3D Navier-Stokes equations. A VMS implementation is chosen since this unifies the concepts of discretization and modeling error, and in the current context results in a formation where machine learning error is the only source of error. The challenge lies in creating a neural network that is able to predict accurate closure terms that create a locally accurate flow field and is also stable in the long term. Previous work with LTI proves that long-time approaches based on matching statistics will be required for stability. These are expensive because many time steps are required to apply reinforcement learning techniques, and finding gradients also becomes extremely expensive. They also do not ensure the turbulence dynamics are reproduced well locally in space and time. The problem of long term instabilities are therefore left to later work and the focus of this work is on the design of machine learning models to achieve local dynamic accuracy. The machine learning model will be compared to an algebraic model for reference [16]. While not the focus of this research, the implications of LTI will be briefly examined.

A CNN is chosen over an MLP because CNNs are known to be better at using of patterns in the input data [9, 17]. This is useful since terms like the derivative of the spatial data can be implicitly calculated by the network. Different hyperparameters will be tested, this will

help determine what features are most important maintaining local accuracy.

The following research questions will help facilitate the completion of the objective.

1. How should the CNN be designed so that it produces closure terms with high correlations to the exact ones?
   (a) What data from the simulation should be used as inputs to the CNN?
   (b) How large does a stencil have to be?
   (c) What kind of hyperparameters are needed?
2. How effective are CNNs when used for the momentum equations in terms of producing accurate local flow statistics in the simulation?
   (a) Are CNN models more accurate than algebraic models for local dynamics?

2 Methodology

2.1 Variational Multiscale Approach to the Navier-Stokes Equations

This paper continues the work of Bettini [6] which uses the variational multiscale method applied to the incompressible Navier-Stokes equations shown in equation 1.

$$\begin{aligned} \nabla \cdot u &= 0 \\ \frac{\partial u}{\partial t} + (u \cdot \nabla)u &= -\nabla p + \nabla \cdot (2\nu \nabla^s u) + f \end{aligned} \tag{1}$$

Here ρ is the medium density, u is the medium velocity, p is the pressure (divided by constant density), ν is the medium kinematic viscosity, $\nabla^s u = \frac{1}{2}(\nabla u + (\nabla u)^T)$ is the symmetric velocity gradient tensor (strain-rate tensor), and f are the body accelerations.

The variational multiscale method is an LES method that assumes separation of the scales beforehand and does not use a filter. This avoids commutation errors. Additionally no modeling has to be done until its time to solve for the unresolved scale solution. The procedure is to define the resolved scales as a projection onto a finite dimensional space and the unresolved scales as a projection onto infinite-dimensional space. This a priori projection means the discretized forms of the resolved and unresolved scale equations exactly satisfy the Navier-Stokes equations (in a weak sense). Modeling error is thus introduced only when approximating the effect of the unresolved scales on the resolved scales.

The VMS equations used are heavily inspired by those of Hughes, Scovazzi, and Franca [2] and Bazilevs et al. [3].

We take a space-time domain $Q = \Omega \times [0, T] \subset \mathbb{R}^3 \times \mathbb{R}^+$ with boundaries $P = \Gamma \times [0, T]$. The domain is then sliced into N slabs $Q_n = \Omega \times [t_n, t_{n+1}], n = 0, 1, 3, ..., N$. The solution is $\boldsymbol{U} = \{\boldsymbol{u}, p\} \in \mathcal{V}_s(Q_n)$ for all Q_n, with an initial condition of $\boldsymbol{u}_0$ in Q_0. Figure 2.1 shows the domain and the slabs.

The solution is defined such that $\forall \boldsymbol{W} = \{\boldsymbol{w}, q\} \in \mathcal{V}_w(Q_n)$:

$$B_1(\boldsymbol{W}, \boldsymbol{U}) + B_2(\boldsymbol{W}, \boldsymbol{U}, \boldsymbol{U}) - L(\boldsymbol{W}) = 0 \tag{2}$$

where $\mathcal{V}_s$ is the trial solution space and $\mathcal{V}_w$ is the weighted function space defined over the

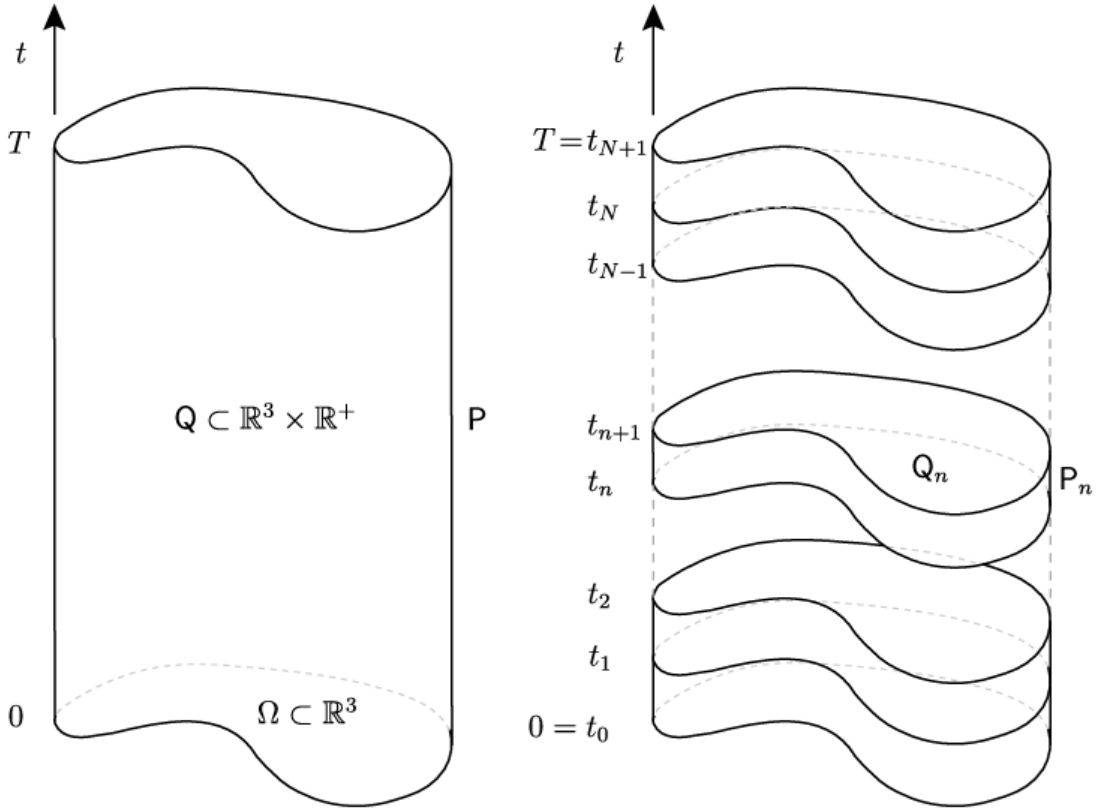


Figure 2.1: Space time domain on the left, space-time slabs on the right. Figure taken from Bazilev [3].

domain Q_n. $\boldsymbol{B}_1$ is a bilinear form and $\boldsymbol{B}_2$ is a trilinear form defined as:

$$
\begin{aligned}
B_1(\boldsymbol{W},\boldsymbol{U}) &= \left(\boldsymbol{w}, \frac{\partial \boldsymbol{u}}{\partial t} + \frac{1}{\rho}\nabla p - \nabla\cdot(2\nu\nabla^s u)\right)_{Q_n} + \Big(q, \nabla\cdot u\Big)_{Q_n} \\
B_2(\boldsymbol{W},\boldsymbol{U},\boldsymbol{V}) &= \Big(\boldsymbol{w}, (\boldsymbol{u}\cdot\nabla)\boldsymbol{v}\Big)_{Q_n} \\
L(\boldsymbol{W}) &= \Big(\boldsymbol{w}, \boldsymbol{f}\Big)_{Q_n}
\end{aligned}
\tag{3}
$$

where $\boldsymbol{V} = \{\boldsymbol{v}, \cdot\}$. We define the inner product $(f,g) \equiv \int_{Q_n} f\ g\ dV$. The trial space, $\mathcal{V}_s$, is separated into resolved scales, $\overline{\mathcal{V}_s}$, and unresolved scales $\mathcal{V}_s'$, through direct sum decomposition.

$$
\mathcal{V}_s = \overline{\mathcal{V}_s} \oplus \mathcal{V}_s' \tag{4}
$$

$\overline{\mathcal{V}_s}$ is a finite-dimensional space while $\mathcal{V}_s'$ is an infinite-dimensional space. By further decomposition $\boldsymbol{U} = \overline{\boldsymbol{U}} + \boldsymbol{U}'$, $\boldsymbol{u} = \overline{\boldsymbol{u}} + \boldsymbol{u}'$,and $p = \overline{p} + p'$, where $\overline{\boldsymbol{U}} = \{\overline{\boldsymbol{u}}, \overline{p}\} \in \overline{\mathcal{V}_s}$, and

$\boldsymbol{U}' = \{\boldsymbol{u}', p'\} \in \mathcal{V}_s'$. Equation 2 can be rewritten into:

$$B_1(\boldsymbol{W}, \overline{\boldsymbol{U}} + \boldsymbol{U}') + B_2(\boldsymbol{W}, \overline{\boldsymbol{U}} + \boldsymbol{U}', \overline{\boldsymbol{U}} + \boldsymbol{U}') - L(\boldsymbol{W}) = 0 \tag{5}$$

which in turn can be rewritten using the linearity of the forms into:

$$\begin{aligned} B_1(\boldsymbol{W}, \overline{\boldsymbol{U}}) + B_2(\boldsymbol{W}, \overline{\boldsymbol{U}}, \overline{\boldsymbol{U}}) - L(\boldsymbol{W}) = \\ -B_1(\boldsymbol{W}, \boldsymbol{U}') - B_2(\boldsymbol{W}, \overline{\boldsymbol{U}}, \boldsymbol{U}') - B_2(\boldsymbol{W}, \boldsymbol{U}', \overline{\boldsymbol{U}}) - B_2(\boldsymbol{W}, \boldsymbol{U}', \boldsymbol{U}') \end{aligned} \tag{6}$$

The weighted function space, $\mathcal{V}_w$, is also separated using direct-sum decomposition:

$$\mathcal{V}_w = \overline{\mathcal{V}_w} \oplus \mathcal{V}_w' \tag{7}$$

again $\overline{\mathcal{V}_w}$ is the resolved scale finite-dimensional space while $\mathcal{V}_w'$ is the unresolved scale infinite-dimensional space. By further decomposition $\boldsymbol{W} = \overline{\boldsymbol{W}} + \boldsymbol{W}'$, $\boldsymbol{w} = \overline{\boldsymbol{w}} + \boldsymbol{w}'$,and $q = \overline{q} + q'$, where $\overline{\boldsymbol{W}} = \{\overline{\boldsymbol{W}}, \overline{q}\} \in \overline{\mathcal{V}_w}$, and $\boldsymbol{W}' = \{\boldsymbol{W}', q'\} \in \mathcal{V}_w'$. Plugging in this decomposition into equation 6 and rearranging we get two coupled systems of equations, the course-scale:

$$\begin{aligned} B_1(\overline{\boldsymbol{W}}, \overline{\boldsymbol{U}}) + B_2(\overline{\boldsymbol{W}}, \overline{\boldsymbol{U}}, \overline{\boldsymbol{U}}) - L(\overline{\boldsymbol{W}}) = \\ -B_1(\overline{\boldsymbol{W}}, \boldsymbol{U}') - B_2(\overline{\boldsymbol{W}}, \overline{\boldsymbol{U}}, \boldsymbol{U}') - B_2(\overline{\boldsymbol{W}}, \boldsymbol{U}', \overline{\boldsymbol{U}}) - B_2(\overline{\boldsymbol{W}}, \boldsymbol{U}', \boldsymbol{U}') \end{aligned} \tag{8}$$

and the unresolved scale:

$$\begin{aligned} B_1(\boldsymbol{W}', \boldsymbol{U}') + B_2(\boldsymbol{W}', \boldsymbol{U}', \boldsymbol{U}') - L(\boldsymbol{W}') = \\ -B_1(\boldsymbol{W}', \overline{\boldsymbol{U}}) - B_2(\boldsymbol{W}', \overline{\boldsymbol{U}}, \boldsymbol{U}') - B_2(\boldsymbol{W}', \boldsymbol{U}', \overline{\boldsymbol{U}}) - B_2(\boldsymbol{W}', \overline{\boldsymbol{U}}, \overline{\boldsymbol{U}}) \end{aligned} \tag{9}$$

Equations 8 and 9 are exact but non-unique since $\overline{\boldsymbol{U}}$, $\boldsymbol{U}'$, $\overline{\boldsymbol{W}}$, and $\boldsymbol{W}'$ are only defined in their spaces. To uniquely define them a projection is chosen with a corresponding norm, $\hat{N}$:

$$\begin{aligned} \overline{\boldsymbol{U}} &= proj_{\hat{N}}^{\overline{\mathcal{V}_s}} \boldsymbol{U} \\ \overline{\boldsymbol{W}} &= proj_{\hat{N}}^{\overline{\mathcal{V}_w}} \boldsymbol{W} \end{aligned} \tag{10}$$

Once the projector is chosen, the unresolved scales are easily defined as:

$$\begin{aligned} \boldsymbol{U}' = \boldsymbol{U} - \overline{\boldsymbol{U}} &= \left(iden - proj_{\hat{N}}^{\overline{\mathcal{V}_s}} \right) \boldsymbol{U} \\ \boldsymbol{W}' = \boldsymbol{W} - \overline{\boldsymbol{W}} &= \left(iden - proj_{\hat{N}}^{\overline{\mathcal{V}_w}} \right) \boldsymbol{W} \end{aligned} \tag{11}$$

where *iden* is the identity operator. Bazilevs et al. [14] rewrite equation 9 as:

$$B_1(\boldsymbol{W}', \boldsymbol{U}') + B_2(\boldsymbol{W}', \boldsymbol{U}', \boldsymbol{U}') = -B_2(\boldsymbol{W}', \overline{\boldsymbol{U}}, \boldsymbol{U}') - B_2(\boldsymbol{W}', \boldsymbol{U}', \overline{\boldsymbol{U}}) - (\boldsymbol{W}', Res_s(\overline{\boldsymbol{U}})) \tag{12}$$

where $(\boldsymbol{W}', Res_s(\overline{\boldsymbol{U}})) = B_1(\boldsymbol{W}', \overline{\boldsymbol{U}}) + B_2(\boldsymbol{W}', \overline{\boldsymbol{U}}, \overline{\boldsymbol{U}}) - L(\boldsymbol{W}')$, the resolved scale residual "lifted" to the dual of the unresolved space. Now the formal solution of equation 12 can be

represented as a functional $\overline{\boldsymbol{U}}$ and $Res_s(\overline{\boldsymbol{U}})$, i.e. $\boldsymbol{U}' = F(\overline{\boldsymbol{U}}, Res_s(\overline{\boldsymbol{U}}))$. This means if the functional exists the unresolved scale system becomes unnecessary for the resolved scales. Practical calculations require that the functional is approximated as $\widetilde{F}$, which means the solution is also an approximation, $\widetilde{\overline{\boldsymbol{U}}}$.

$$\begin{aligned} B_1(\overline{\boldsymbol{W}}, \widetilde{\overline{\boldsymbol{U}}}) + B_2(\overline{\boldsymbol{W}}, \widetilde{\overline{\boldsymbol{U}}}, \widetilde{\overline{\boldsymbol{U}}}) - L(\overline{\boldsymbol{W}}) = \\ -B_1(\overline{\boldsymbol{W}}, \widetilde{\boldsymbol{U}}') - B_2(\overline{\boldsymbol{W}}, \widetilde{\overline{\boldsymbol{U}}}, \widetilde{\boldsymbol{U}}') - B_2(\widetilde{\overline{\boldsymbol{U}}}, \widetilde{\boldsymbol{U}}', \widetilde{\overline{\boldsymbol{U}}}) - B_2(\overline{\boldsymbol{W}}, \widetilde{\boldsymbol{U}}', \widetilde{\boldsymbol{U}}') \end{aligned} \tag{13}$$

Where $\widetilde{\boldsymbol{U}}' = \widetilde{F}(\widetilde{\overline{\boldsymbol{U}}}, Res_s(\widetilde{\overline{\boldsymbol{U}}}))$ is an approximation of the unresolved scales. Equation 13 is the equation solved and represents a two-scale solution of the incompressible Navier-Stokes equations. The right hand side represents the closure terms and are predicted by a neural network in this paper. When all the terms are fully written out the full set of equations that are solved are shown in equation 14. The right hand side terms are predicted by the neural network. Note that the continuity equation is not solved directly in this work since it causes convergence problems. The projection of the DNS pressure is injected into the solution and the continuity equation is sidestepped.

$$\begin{aligned} \left(\overline{\boldsymbol{w}}, \frac{\partial \overline{\boldsymbol{u}}}{\partial t} + \nabla \overline{p}_{DNS} - \nabla \cdot (2\nu \nabla^s \overline{\boldsymbol{u}})\right) + \left(\overline{\boldsymbol{w}}, (\overline{\boldsymbol{u}} \cdot \nabla)\overline{\boldsymbol{u}}\right) - \left(\overline{\boldsymbol{w}}, \boldsymbol{f}\right) = \\ - \left(\overline{\boldsymbol{w}}, \frac{\partial \boldsymbol{u}'}{\partial t} + \nabla p' - \nabla \cdot (2\nu \nabla^s \boldsymbol{u}')\right) - \left(\overline{\boldsymbol{w}}, (\overline{\boldsymbol{u}} \cdot \nabla)\boldsymbol{u}'\right) - \left(\overline{\boldsymbol{w}}, (\boldsymbol{u}' \cdot \nabla)\overline{\boldsymbol{u}}\right) \\ - \left(\overline{\boldsymbol{w}}, (\boldsymbol{u}' \cdot \nabla)\boldsymbol{u}'\right) \end{aligned} \tag{14}$$

2.2 Simulation Setup

For the present work, we will consider the application of the CNN models to turbulent channel flow with $Re_\tau = 180$. The presented results were generated using a domain length of 6 in the streamwise, x, direction, 4 in the spanwise, y, direction, and 2 in the wall normal, z, direction.

The LES simulation were performed on a $32x32x32$ rectangular grid. The time step of the LES simulation was $\Delta t_{LES} = 0.0072$. The initial condition of the simulation was obtained using a projection of the DNS solution. The domain was periodic in spanwise and streamwise directions. The spatial discretization used trilinear basis functions centered on the element nodes. These basis functions are 1 at the element nodes and 0 at all neighboring nodes. Because of the way the code assembles the element matrices, the basis functions are split up into 8 shape functions, one for each of the element the basis function affects. This means each element is affected by 8 shape functions, one from each basis function centered around the element's 8 nodes. The time discretization was defined by a second order finite-difference

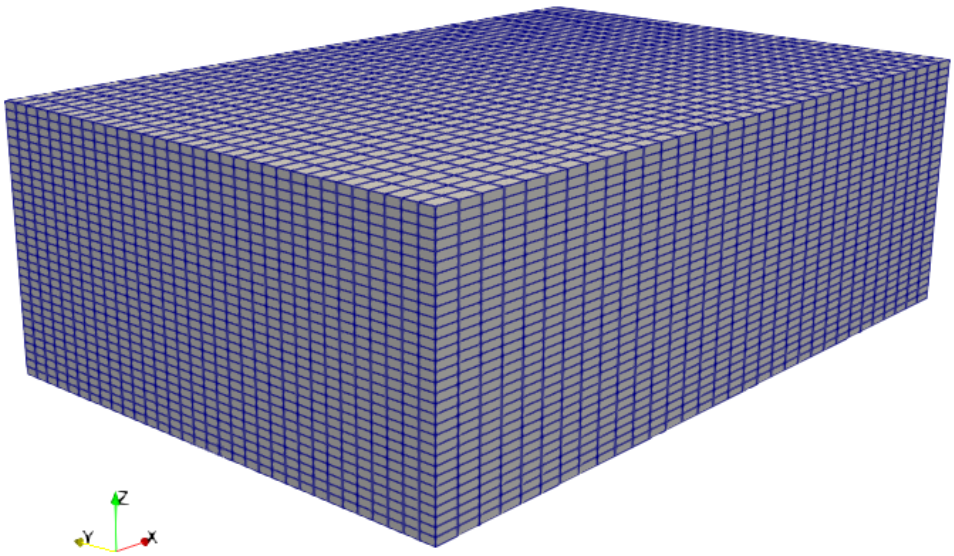


Figure 2.2: Domain of simulation.

method, as shown in equation 15.

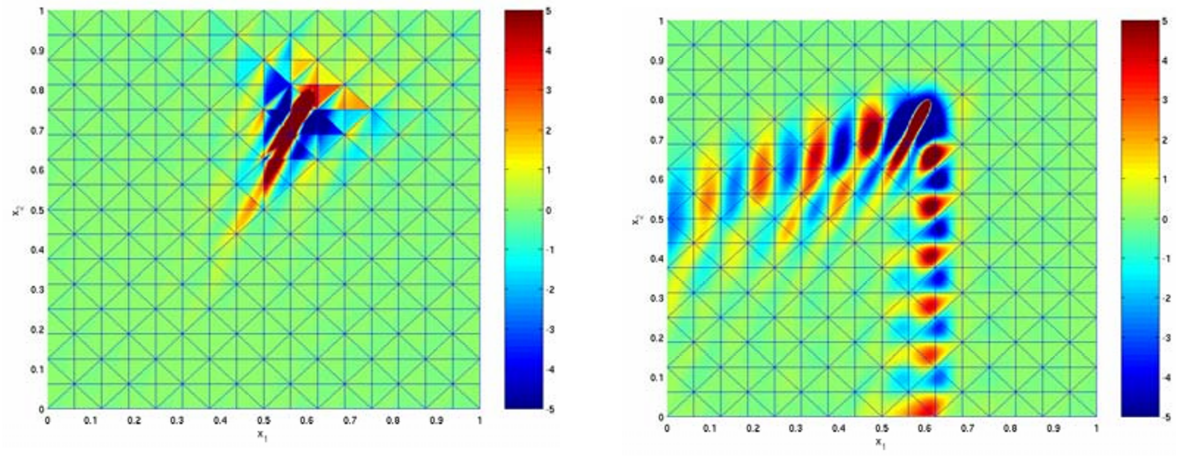
$$\frac{\partial u_n}{\partial t} = \frac{1}{2\Delta t}(3u_n - 4u_{n-1} + u_{n-2}) \tag{15}$$

The DNS simulations were performed using 1/60th of the timestep of the LES, $\Delta t_{DNS} = 0.00012$, 128 spectral modes in both the streamwise and spanwise direction, and 164 finite elements in the wall normal direction.

The initial condition is labeled L0 and starts at t=0. It was created by running the DNS until statistical convergence. The neural networks were trained using the 120 LES time steps from t=1.0 to t=1.8928, and tested in the simulation using the L0 initial condition. This means the models are tested on flow which it has not trained on.

2.2.1 Projector

To obtain training data for the CNN, the DNS was run first and the exact closure terms computed. This was done by projecting the DNS to the LES grid and using equation 11 to calculate $\boldsymbol{u}'$. Using $\boldsymbol{u}'$ the weak forms in equation 14 were calculated and written to a file. The neural network used these terms to train and learn how to predict the closure terms (the terms involving $\boldsymbol{u}'$). To do this, a projector has to be chosen. The choice of projector is often motivated by the effect it has on the fine-scale Green's function, g'. Equation 12 shows that the unresolved scales, $\boldsymbol{U}'$, are driven by the residual of the resolved scales. Hughes [2]



(a) Using H_0^1 Projector.

(b) Using L_2 Projector.

Figure 2.3: Unresolved scale Green's functions for the 2D advection-diffusion equation using both H_0^1 and L_2 projectors. The equation is advection dominated in this case. Figures taken from Hughes [14].

proves this and also shows that $\boldsymbol{U}'$ can be expressed as:

$$\boldsymbol{U}' = -\int_{\Omega} g' Res_s(\overline{\boldsymbol{U}})\, d\Omega \tag{16}$$

where g' is the unresolved scale Green's function. Figure 2.3 shows the findings from Hughes and Sangalli for the advection dominated 2D advection-diffusion equation. The projection using the H_0^1 seminorm produces a very local fine-scale Greens function while the L_2 seminorm projection produces a very global fine-scale Greens function. This means $\boldsymbol{U}'$, and thus the closure terms, is more influenced by the global solution if the L_2 projector is used, and that is why the H_0^1 projection is a better choice for the full Navier-Stokes problem. A smaller area where the solution influences the local closure terms is very useful since it means a smaller stencil size can be used by the machine learning model. However these findings only apply to the 2D advection diffusion equation and it is only an assumption that this also applies to the full 3D Navier-Stokes equations. The other problem is that using the H_0^1 projection requires solving an additional system which can be expensive. The nodal projector is the same as the H_0^1 projector in one dimension, and is also much faster to evaluate. For these reasons a nodal projector was used in this work.

2.2.2 Closure Terms Definition

The neural networks were trained by using a feature set derived from the projected solution and the objective of predicting the exact closure terms from equation 14. This was done by using the DNS to generate the exact velocity field U, projecting to get the resolved scale velocity, $\overline{U}$, and then subtracting them, $U' = U - \overline{U}$. Using a fine quadrature mesh the individual integrals of equations **??** and 14 can be calculated. Adding all these terms

together creates the closure term that the CNN model learned to predict. In theory the exact closure terms and the resolved scale weak forms satisfy equations **??** and 14 perfectly. And if the LES simulation is run using the perfect closure terms the exact weak form of the equations will be resolved. In practice this requires substantial computational effort, due to the slow corrector-pass convergence also observed by Krochak. In addition the DNS and LES codes express continuity in different norms (one spectral and finite volume, the other finite element) which can lead to small errors in mass conservation even without machine learning approximation error. Another issue is that both the DNS and LES used finite-difference in time but the DNS used a much smaller time step. Because of this it is difficult to define an equivalent projection to compute $(\overline{w}, \frac{\partial U'}{\partial t})$. The solution Krochak implemented, and the one implemented in this work, is that the unresolved scale weak forms for continuity and time derivative are not calculated by integration but are defined as the residual of all other terms. This forces equation 14 to be true at all times. Equation 17 shows how the terms are redefined for the momentum equations considered here.

$$\begin{aligned}\left(\overline{\boldsymbol{w}}, \frac{\partial \boldsymbol{u}'}{\partial t}\right) \equiv &- \left(\overline{\boldsymbol{w}}, \frac{\partial \overline{\boldsymbol{u}}}{\partial t} + \nabla \overline{p} - \nabla \cdot (2\nu \nabla^s \overline{\boldsymbol{u}})\right) - \left(\overline{\boldsymbol{w}}, (\overline{\boldsymbol{u}} \cdot \nabla)\overline{\boldsymbol{u}}\right) + \left(\overline{\boldsymbol{w}}, \boldsymbol{f}\right) \\ &- \left(\overline{\boldsymbol{w}}, \nabla p' - \nabla \cdot (2\nu \nabla^s \boldsymbol{u}')\right) - \left(\overline{\boldsymbol{w}}, (\overline{\boldsymbol{u}} \cdot \nabla)\boldsymbol{u}'\right) - \left(\overline{\boldsymbol{w}}, (\boldsymbol{u}' \cdot \nabla)\overline{\boldsymbol{u}}\right) \\ &- \left(\overline{\boldsymbol{w}}, (\boldsymbol{u}' \cdot \nabla)\boldsymbol{u}'\right) \quad (17)\end{aligned}$$

2.2.3 Implementation

In the current work Tensorflow was combined with the MEX library which provided the implementation of VMS with the neural network and the DNS. Tensorflow is a software library developed by Google for training neural networks. MEX is a C++ library created and maintained by Dr. Steven Hulshoff for setting up and running finite element simulations. MEX uses three other libraries, MFEM, Hypre, and Metis. MFEM is used to compute terms of the finite-element method. Hypre is used for large scale linear solvers. Metis is used to partition finite-element meshes for efficient parallel computing. MEX uses these three libraries and allows the user to create finite element discretizations with high level commands. MEX includes an implementation for a MLP neural network, but linking with Tensorflow provided access to the CNN as well as a range of other advanced machine learning models.

The Tensorflow library is most easily used in Python. This means the CNNs were trained and run using Python. Since MEX is written in C++ the models had to be converted to be run in C++. Tensorflow is available in C++ but it is not as flexible as the Python version and documentation is severely lacking. To make the jump from Python to C++ a library called CppFlow was used [18]. This library allows models to be saved by Tensorflow using Python and run using C++. This makes the use of models much easier since the models can be read in and run automatically instead of recreating the model by hand in C++, an arduous task that can easily lead to implementation errors.

This thesis used a finite-difference time discretization and a finite-element spatial discretization. The time discretization is substituted into the weak form, see equation 15, to define the weak residual. This way the time discretization error is compensated by the exact closure terms. The weak residual, $\mathcal{R}_w$ was calculated as a sum of the residual of all the cells.

The system of equations defined by the weak form has as unknowns the amplitudes of the solution bases of the next time step, a_i^{n+1}. This nonlinear system was solved using a Newton method:

$$\begin{aligned} \frac{\partial \mathcal{R}_w}{\partial a} \Delta a^p &= -\mathcal{R}_w \\ a^{p+1} &= a^p + \Delta a^p \end{aligned} \tag{18}$$

In equation 18 each update to a^{p+1} counts as a corrector pass.

2.3 Artificial Neural Networks

Artificial neural networks are machine learning models that are made up of layers of neurons. A neuron takes as input the data from the previous layer and passes it through a function, the output is used as input in the next layer. The first layer is the input layer, here the user data is fed into the network. The last layer is called the output layer, this data is what the user wants. The layers in between are called hidden layers.

2.3.1 Multilayer Perceptrons (MLP)

MLP artificial neural networks take an input and propagate the information forward through fully connected layers of neurons. Each neuron takes in all the previous layers output and uses trainable weights for each input. The weighted sum plus a trainable bias is then put through an activation function and then the output of the neuron is used as the input for the next layer. Figure 2.4 shows a neural network with one hidden layer. The output, h_i, of a neuron in the hidden layer is calculated by taking a weighted sum of the inputs plus a bias. The sum is then put through an activation function, f.

$$h_i = f\Big(\sum_{j=1}^{n_i} w_{ij} x_j + b_i\Big) \tag{19}$$

Figure 2.5 shows the inner workings of a neuron.

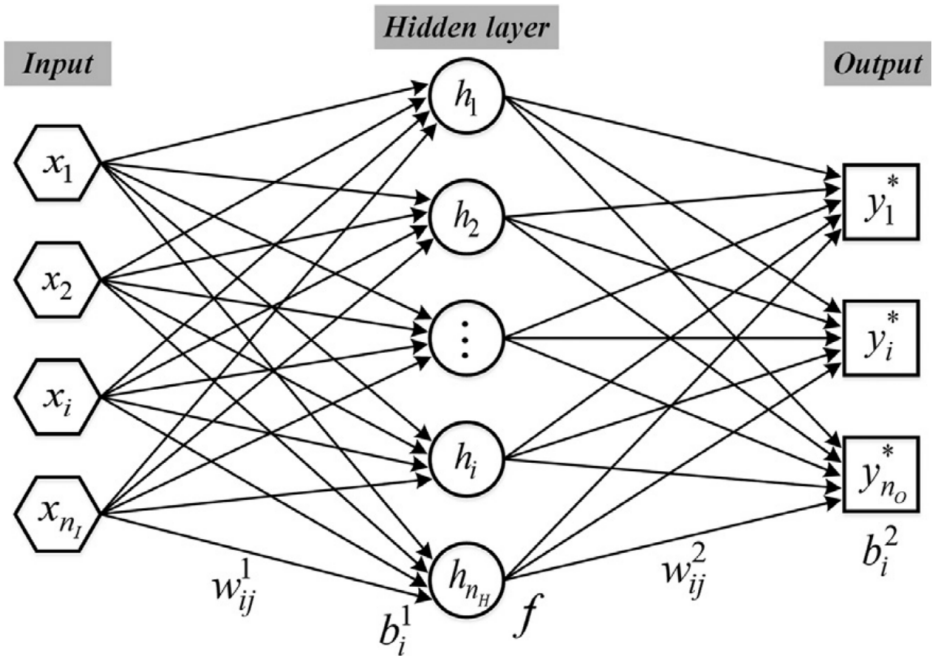


Figure 2.4: A neural network with one hidden layer.

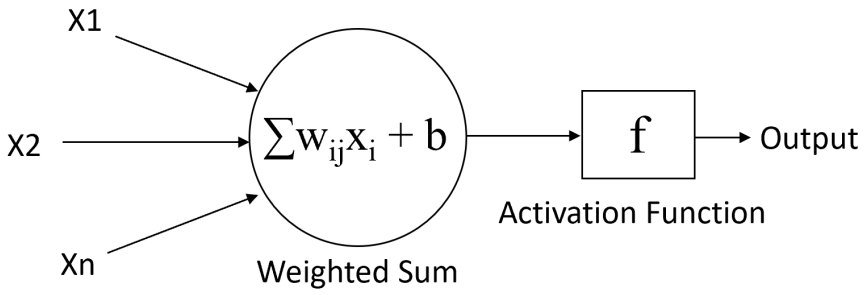


Figure 2.5: A neuron of a neural network.

The weights and biases of each neuron are trained using a loss function and backpropagation. MLPs have been shown to approximate an extremely wide array of functions and the non linearity is useful for when simple models are not good enough. Training data is needed, which can be expensive [4, 5].

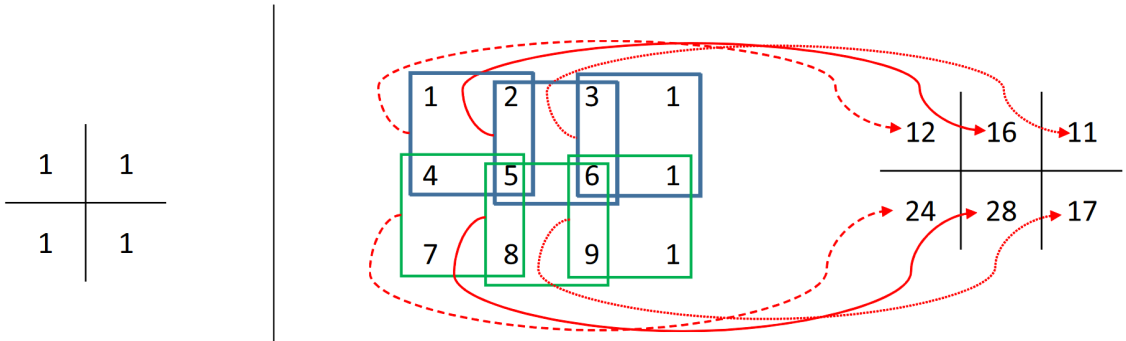
2.3.2 Convolutional Neural Networks

Convolutional neural networks are ANNs that contain convolutional layers. In these layers a convolution filter, or kernel, slides over the input and produces some output, usually an activation function is applied directly after a convolution. The convolution filter has a user defined size and is moved over the input with a user defined step size. Each convolutional layer uses a pre-defined number of filters. Figure 2.6 shows a $2x2$ filter with step size 1

moving over an input and producing some output. The filter produces an output only based on local data which makes CNNs very useful for image recognition [9].

The convolution filter has weights that are trained, so only the size and step are hyperparameters.

Pooling layers are also applied, common ones include max and average pooling layers, these decrease the amount of inputs and combine the information from previous layers. Convolutional layers are combined with layers of fully connected neurons, the same as those from an MLP, to create networks that can recognize local features and then combine those features to produce a useful output. The CNN networks used in this thesis start with one or more convolutional layers and end with at least one dense layer.



(a) A 2×2 kernel

(b) The convolution input and output

Figure 2.6: Convolution operation with a 2×2 filter and a step size of 1 [17].

CNNs are very useful in image processing, they are good at detecting features since the filters learn which part of the inputs are useful for the loss function. The theory behind using CNNs for closure models is that the convolutional layers can efficiently learn what features in the flow are important to the closure terms [1, 11].

3 Neural Network Design

There are many choices for how to setup a CNN. The ones used in this work are detailed here. The choices made include choosing the size of the spatial stencil. To predict the closure terms for one element the features are taken from the neighboring elements. The spatial stencil defines which neighboring elements are used. The temporal stencil is also important. In this work, a maximum of two previous time steps are taken to predict the closure terms. Another important choice is what kind of features to use as inputs. The most commonly used feature used as input for CFD simulations are the velocity components [1]. In this work both the velocity components and the unresolved scale integrated features are used. Lastly the hyperparameters of the CNN, the amount of filters and their sizes, must be chosen. The sensitivity of the results to those parameters is examined in this section. All networks contain a hidden layer with 200 neurons and an output layer.

3.1 2D vs 3D Spatial Convolutions

In theory one could convolve over however many dimensions are needed. However the Tensorflow library only supports up to three-dimensional convolutions. Any higher requires the development of custom layers which is beyond the scope of this work. That leaves a maximum of three dimensions for the input. 3D spatial stencils are useful since spatial data from all directions is used. This is rather expensive since slightly larger stencils increase the amount of data by a power of three. 2D spatial stencils have the advantage that the inputs are a lot smaller and thus there is significantly less data to train on. The other benefit of choosing a 2D spatial stencil is that it allows the addition of a temporal convolution. If the spatial stencil is 2D and two time steps are used as input then a 3D convolution can be done, with the first two axes in space and the third axis in time. In this work the 2D spatial convolutions are performed in the streamwise and spanwise direction. Since the domain is periodic in these directions, the 2D convolutions do not need boundary conditions (at the boundary the input can always be taken from the the opposite boundary). In 3D there is a need for a boundary condition at the wall since there is no flow data beyond the wall. In this case beyond the wall the values are zeroed out. In this thesis the both 2D and 3D convolution layers are tested.

3.2 Input Features

There are many different choices for input features, the most commonly chosen ones for the current work have been the unresolved-scale integrated features, the left-hand side of equation 14. Also tested in this work are the velocity amplitudes of the shape functions. The x, y, z, directions of the features chosen are input into the convolution layer as channels (analogous to the rgb values in a color image). Many different types of features can be added as channels but this is only briefly explored here.

Another choice is whether to average the features across the element and predict all closure terms for all shape functions at the same time, or whether to take as input the features per

shape function and predict the closure terms for only one shape function at a time. When averaging the amount of input data decreases by a factor of 8 which is very appealing but the question is whether too much information is lost when averaging. On the other hand only using the information from one shape function at a time could limit what the network can learn.

3.3 Multiple Time Steps

Using more than one previous time step as input seems to give a significant advantage [15], and there are different ways to insert this data into the convolution layers. One way is to append it as extra channels. This works for both the 2D and 3D spatial stencils. An example input of this for a $9x9$ spatial stencil would be of shape $9x9x6$. Another way is to add it as extra convolutional dimension, the 2D convolution becomes a 3D convolution with the first two axes being space and the 3rd being time. For example, for a spatial stencil of $9x9$ the input into the CNN would be of shape $9x9x2x3$. A 3D spatial convolution layer would turn into a 4D one which is only possible using a custom layer in Tensorflow, thus this is only implemented for the 2D case.

3.4 Hyperparameters

The amount of filters is an important hyperparameter. Here, the amount of filters in each layer was increased the deeper the network goes so that the first layers learn the simple patterns and the subsequent layers learn the complex patterns. The filter size should not be too big, since then some information can be overlooked, and not too small as to not make the network too large. Here, a 3x3 spatial filter was used.

The amount of convolution layers was also varied. The amount of convolutional layers in the model affects whether the input has to be padded. If the input to the convolution layer is small, for example $5x5$, and the input is not padded then with a $3x3$ filter only a single convolution layer can be used. This is because after 1 layer the output is $3x3$, the same size as the filter. If the input is padded (Tensorflow pads with zeros) then the output is the same size and this problem is avoided. The effect of padding is important for the current work because usually only 2 time steps were used which means there had to be padding in the time direction for the convolution layer to work in time. The effects of padding the input were also tested.

Two hyperparameters that are independent of the network are the choice of optimizer and training rate. For simplicity the Adam optimizer was chosen with its default learning rate of 0.001. The Adam optimizer is a very common first choice since it is fairly fast and robust. Other optimizers would have to be considered if there was a problem with convergence but no such problem was encountered in this work.

3.5 Data Size

The amount of input data has a significant effect on how the training is carried out. If the entire dataset fits onto the GPU then the data is transferred only once to the GPU. If the entire dataset fits in RAM but not in VRAM then the CPU loads the entire dataset and loads subsections of it into VRAM to train. If the dataset is too large to fit into RAM then batches are either read from disk memory or preprocessed on the fly and then loaded into VRAM for training. The loading of data from CPU to GPU is very time consuming and the loading from disk or preprocessing is even more expensive. This means there is a large incentive to make sure the dataset fits in VRAM in its entirety. The GPU used in this thesis was the Tesla P40 with 24 Gb of VRAM. It should also be noted that Tensorflow uses a lot of memory and is known to often leak memory as well. Tensorflow also appears to use several internal buffers when copying data from the CPU to the GPU. The result is a process which typically uses twice as much memory as the dataset.

The basic 2D stencil for averaged terms (a $9x9x3$ input into the model) required about 5 Gb of training data if 90 training time steps were used. The table below shows approximately how large datasets were for different input sizes. Table 1 shows that a 2D stencil for averaged features is very desirable for fast training. This is one of the reasons that the hyper parameter study was conducted with this choice of network input.

3.6 Training Setup

The training of the neural networks was done using gradient descent of the networks parameters to minimize a loss function. The amount and type of training data is very important, and determines the quality of the network. It is common to feed the entire data to the network after which the gradient of the loss function for each parameter is calculated and the parameters are updated. One run through the entire data set is called an epoch. There is usually a large amount of data and updates to the network parameters are usually done for a smaller subsection of the data, several times per epoch. These are called batches.

The largest challenge in training a network efficiently is making sure that the GPU is being used at full capacity. However, there are several things that can slow down training and prevent a GPU from being used at full power. The first is the time it takes for data to be transferred from the CPU to the GPU. This is by far the biggest time sink and if done inefficiently it can be the bottleneck during training. If the entire data set fits onto the GPU then it is not a problem as the transfer from CPU to GPU only takes place once. If that is not the case then the data has to be loaded onto the GPU in batches and trained per batch. Tensorflow has a good solution for this in their dataset pipeline. The idea is that batches are prepared on the CPU and loaded onto the GPU while the GPU is still training the last batch. This way the GPU never has to wait for data to arrive from the CPU and is constantly working.

The problem is that the Tensorflow library is not perfect and the data pipeline often does not work quite as intended. So if there is a slowdown when communicating and if each batch is loaded onto the GPU every time its needed then there is a lot of communication and this

Description	**Network Input Shape**	**Data size [Gb]**
2D 9x9 space stencil, average features, one time step	(9, 9, 3)	5.34
2D 9x9 space stencil, average features, two time steps	(9, 9, 2, 3)	10.68
2D 5x5 space stencil, average features, two time steps	(5, 5, 2, 3)	3.30
2D 9x9 space stencil, independent features, one time step	(9, 9, 3)	42.72
2D 9x9 space stencil, independent features, two time step	(9, 9, 2, 3)	85.44
2D 7x7 space stencil, independent features, two time step	(7, 7, 2, 3)	51.68
2D 5x5 space stencil, independent features, two time step	(5, 5, 2, 3)	26.37
3D 9x9x9 space stencil, average features, one time step	(9, 9, 9, 3)	48.06

Table 1: Dataset size for 90 training files.

slows down the training. A way to speed up training is to load sections of the data and train on those sections repeatedly a certain amount of times and then load in the next section. An example would be splitting your entire data into three sections, loading the first section into the GPU, training on that section for a certain amount of epochs (also in batches), and then continuing with the next section of the data. This is done until all the sections are trained on and then repeated for a certain number of epochs. This reduces the amount of times data is transferred from the CPU to the GPU.

In this thesis the models with averaged inputs and 2D spatial stencils all had data sizes less than 15 Gb. This meant that all the data was loaded onto the GPU at the same time and no special data pipeline had to be created. The models with independent inputs, and the models with 3D spatial stencils both had too much data. For these cases the data was split into multiple chunks, at least 3, and each chunk was trained on.

4 A Priori Results

The networks were tested a priori on a set of testing data that was distinct from the training data. The main metric that was evaluated was the correlation of the closure terms predicted by the model with the real closure terms. When plotted using 2D histograms, it is easy to see how accurate the model is. This is better than just calculating correlations, as a high correlation does not necessarily mean low errors. The histogram shows whether the model produced closure terms that are accurate. The predictions are shown for all 8 shape functions and the x, y, y directions. This means there are 24 outputs considered. The correlation plots for all 24 outputs are cluttered and reveal that there are two subsets of shape functions that always have nearly the exact same correlation plots. Shape functions 1, 4, 5, 8 always show the same correlations as do shape functions 2, 3, 6, 7. This was also noted by Bettini [6], and for this reason only the correlation plots for shape functions 1 and 2 are shown, as the other plots give no additional information. Table 2 shows the five networks that were tested to determine the effects of a 2D or 3D spatial stencil, velocity or momentum inputs, average or independent inputs and the temporal stencil. Hyperparameters of the CNN are studied separately.

Network Description	**Name**
9x9x9 spatial stencil, average velocity input, 1 time step	3D9vel
9x9 spatial stencil, average velocity input, 1 time step	2D9vel
9x9 spatial stencil, average momentum input, 1 time step	2D9mom
9x9 spatial stencil, average momentum input, 2 time steps	2D9momt
5x5 spatial stencil, independent momentum input, 2 time step	2D5momti

Table 2: Networks Tested.

4.1 2D and 3D

First a basic input of averaged velocity amplitudes was tested. A 9x9 and 9x9x9 input was used. The 2D spatial stencil was found to outperform the 3D spatial stencil. Upon closer inspection the model with a 3D spatial stencil was found to be predicting zero for many different inputs. This could be due to the zeroing out of the features beyond the wall. Without a lot of features on nearly half the input it seems that the model does not have enough information to make a good prediction.

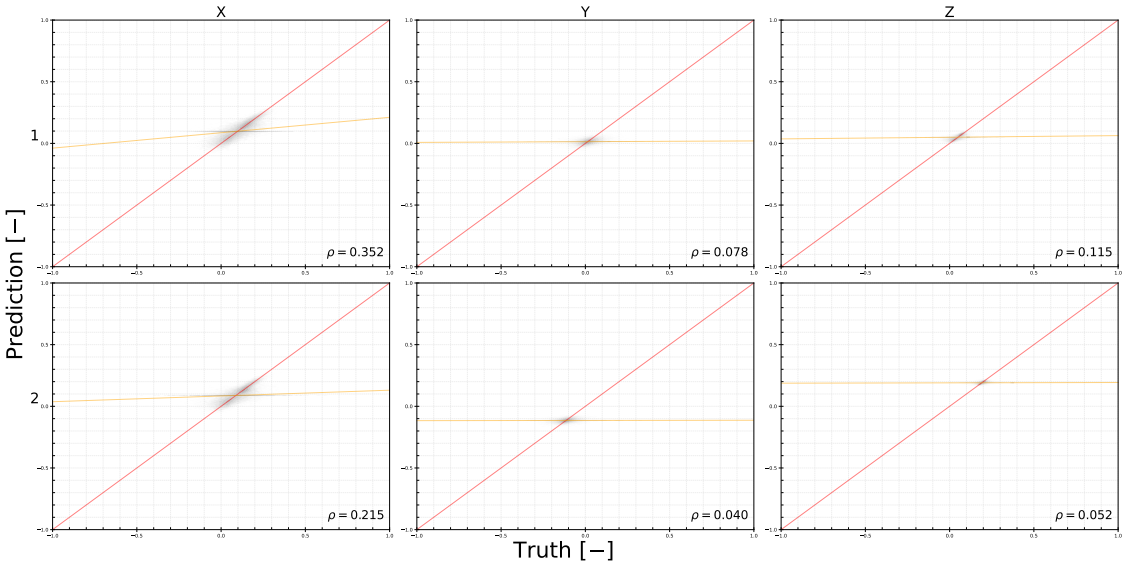


Figure 4.1: Correlations for a CNN with 9x9x9 input of averaged velocity amplitudes, model 3D9vel.

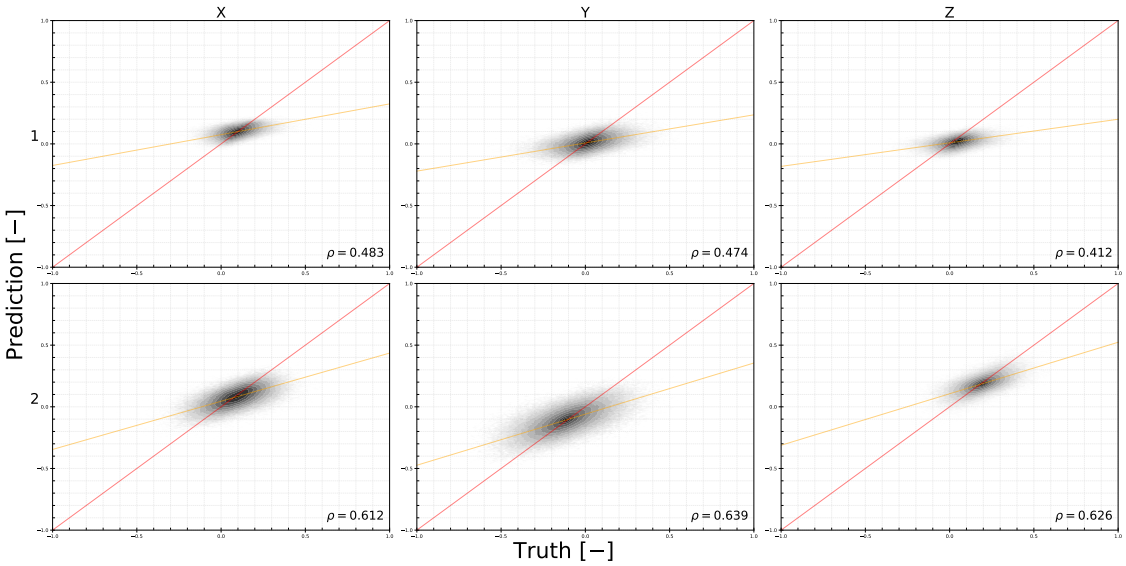


Figure 4.2: Correlations for a CNN with 9x9 input of averaged velocity amplitudes, model 2D9vel.

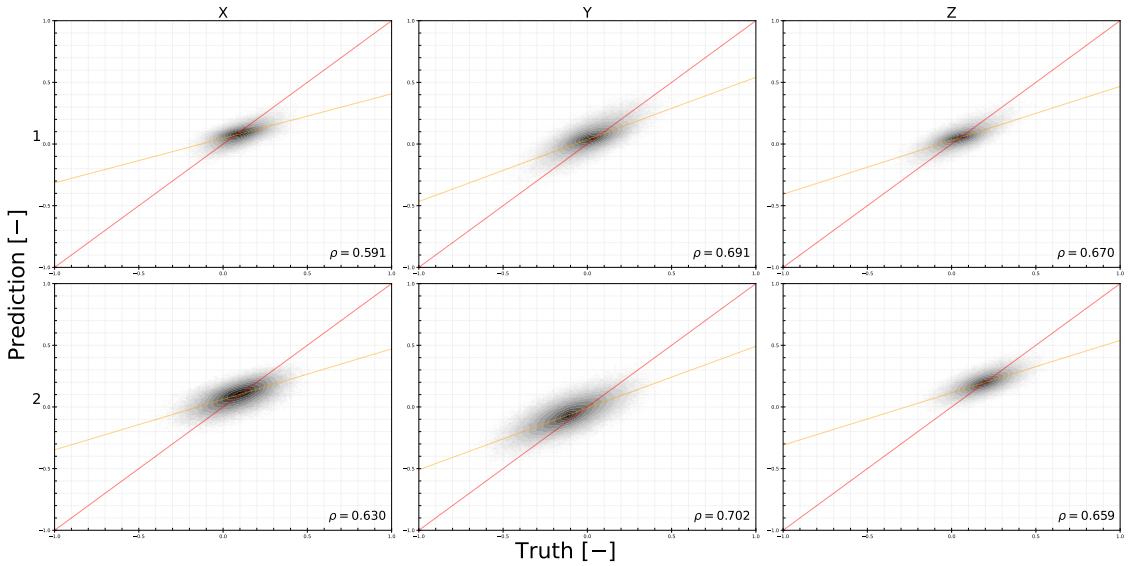


Figure 4.3: Correlations for a CNN with 9x9 input of averaged coarse scale momentum sums, model 2D9mom.

4.2 Inputs

Using 2D spatial stencils the difference between the the averaged velocity amplitudes and averaged momentum coarse sum is shown. The momentum terms do slightly better. This should not be a surprise since the momentum coarse sums show up in the equation for the closure terms.

4.3 Extra Time Steps

Next both t_{n-1} and t_{n-2} time steps were used, in a 3D convolution setup (1 convolution in time). The extra time step increases the accuracy of the model significantly, as was also shown by Krochak [15]. One reason the extra time step increases the accuracy is that the model is able to learn a version of the time derivative of the closure terms. This is also related to how information flows into the element. Where the frozen turbulence, the upstream neighbors of previous time steps contain closure terms similar to that of the element under consideration.

4.4 Averaging vs Independent

If the inputs are averaged across the shape functions the amount of input training data decreases by a factor of 8 which is very appealing but could lead to worse models. Figure 4.5 shows the correlations for when the coarse scale momentum sums are taken as input and not averaged. Since not averaging increases the data size the spatial stencil was reduced to $5x5$, which also means the input was padded. The correlations are all over 0.9, meaning this model performed the best out of all the models. This demonstrates that averaging does remove some important information for determining closure terms at the shape function

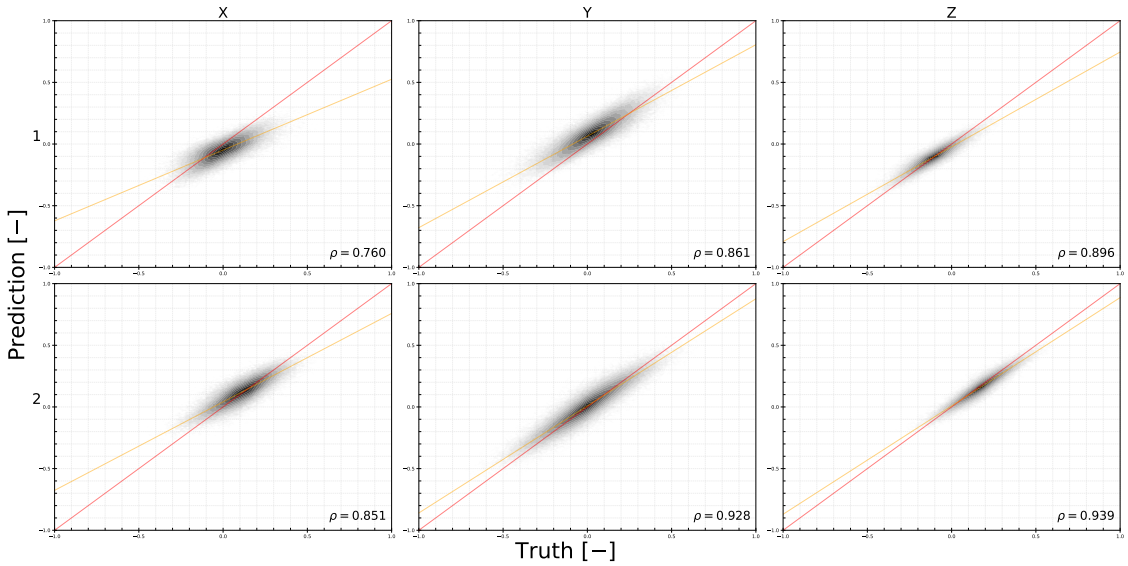


Figure 4.4: Correlations for a CNN with 9x9x2 input of averaged coarse scale momentum sums with and extra time step, model 2D9momt.

level, something also shown for MLPs by Krochak [15]. Note that this model has 3 outputs instead of 24 and would have to be called 8 times per element. This means the model is 8 times as expensive as a model that takes in averaged inputs and predicts all the outputs at once.

4.5 Hyperparameters

A hyperparameter study was perfomed using models with average momentum inputs and two time steps. First, a $5x5$ spatial stencil was used. This stencil is quite small considering the $3x3$ filter size. Without padding the input the size of the data would shrink to $3x3$ after a single convolution layer and thus no further convolutions can be done. For this reason the entire hyperparameter study was done with padding on the input. The first parameter that was changed is the spatial stencil size, from $5x5$ to $9x9$. In theory, more spatial data will help the model notice patterns. The second parameter is the amount of convolution layers, from 1 to 3. In theory the more layers the more complicated patters the model is able to learn. The last parameter is the amount of filters. The amount of filters used in the first layer is listed in table 3. The subsequent convolution layers had twice the amount of filters as the previous layer.

Table 3 shows the results of the hyperparameter study. Only the correlations for shape functions 1 and 2 are given. Figure 4.6 shows the correlations for the models with different amounts of convolutional layers. Figure 4.7 shows the correlations for different amount of filters. Figure 4.8 shows the correlations for different stencil sizes. Padding the input helps the model performance. In general the larger the stencil and deeper the model is the more accurate it becomes. However, the performance increase from larger stencils is very small.

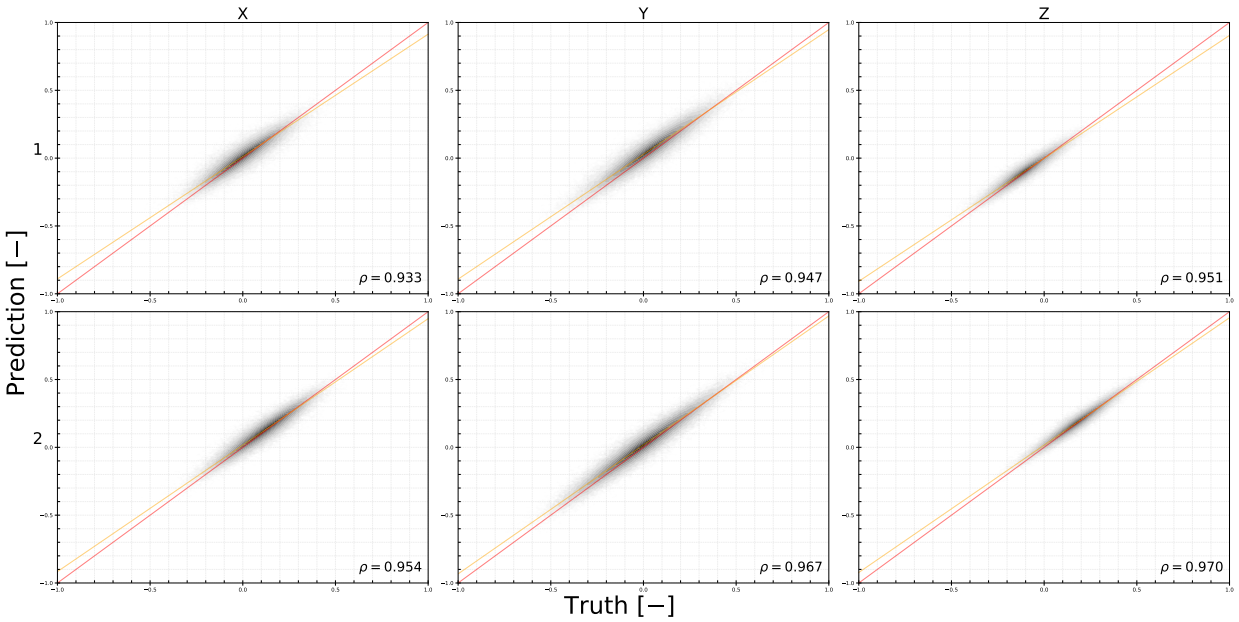


Figure 4.5: Correlations for a CNN with 5x5x2 input of independent coarse scale momentum sums with an extra time step, model 2D5momti.

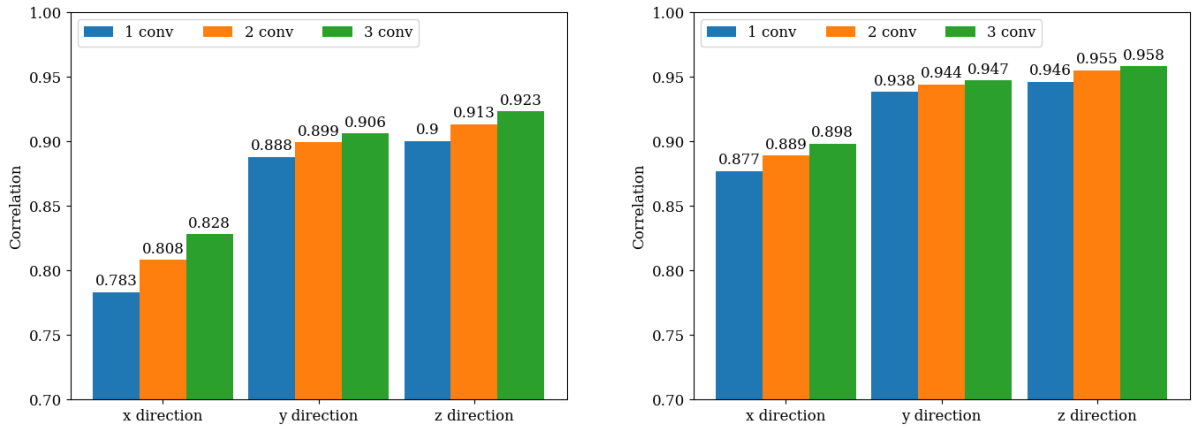
The increases in correlation are mostly in the x momentum closure term, in both y and z direction the improvements are marginal if any. The model with a $5x5$ spatial stencil, 2 convolution layers, and 10 filters performs well for its size. This model is named 2D5momt.

Parameters	**X Momentum**		**Y Momentum**		**Z Momentum**	
	1	2	1	2	1	2
5x5, 2 conv, 10 filters	0.808	0.889	0.899	0.944	0.913	0.955
5x5, 1 conv, 10 filters	0.783	0.877	0.888	0.938	0.900	0.946
5x5, 3 conv, 10 filters	0.828	0.898	0.906	0.947	0.923	0.958
5x5, 2 conv, 5 filters	0.778	0.876	0.890	0.939	0.905	0.948
5x5, 2 conv, 20 filters	0.830	0.895	0.906	0.948	0.919	0.958
7x7, 2 conv, 10 filters	0.816	0.892	0.901	0.945	0.919	0.954
9x9, 2 conv, 10 filters	0.829	0.896	0.905	0.946	0.924	0.956

Table 3: Correlations of hyperparameter study.

4.6 Correlation Over Channel Width

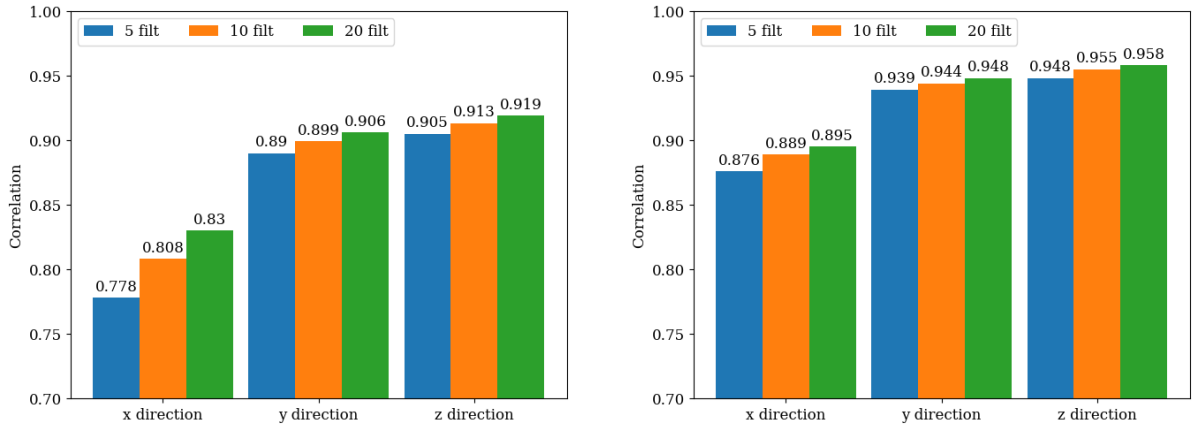
The correlation plots are a good indicator of how well the model performs but it averages over the entire channel and thus does not show if the model performs better in certain regions, specifically near the wall where the closure terms behave differently from the center of the channel. Models 2D5momt and 2D5momti are used to examine the correlation near the walls. Figures 4.9 and 4.10 show the correlations of the predicted closure terms and the real closure terms as a function of the channel height. The model that uses averaged



(a) Shape function 1.

(b) shape function 2.

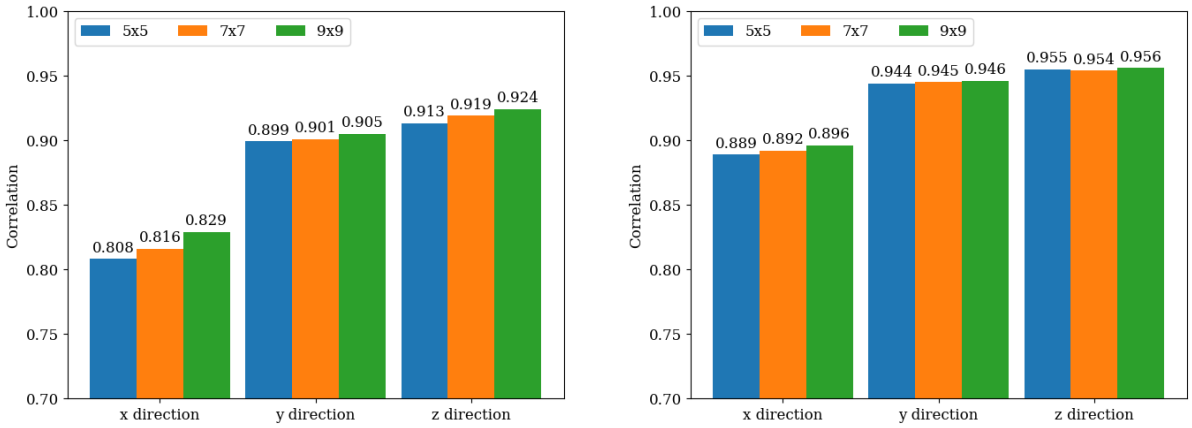
Figure 4.6: Correlations of model with varying number of convolutional layers, for shape functions 1 and 2 in all directions.



(a) Shape function 1.

(b) shape function 2.

Figure 4.7: Correlations of model with varying number of filters, for shape functions 1 and 2 in all directions.



(a) Shape function 1.

(b) shape function 2.

Figure 4.8: Correlations of model with varying stencil sizes, for shape functions 1 and 2 in all directions.

momentum terms, 2D5momt, performs well in the center of the channel but is significantly worse near the walls. This can be a problem in simulations since near the wall is where a lot of the turbulence generation occurs. Because the grid used in this work is fairly coarse it is important that in the first element near the wall the model predicts closure terms accurately.

Figure 4.10 shows the correlation in the channel when the model uses independent momentum terms, 2D5momti. The correlations are higher especially near the wall in comparison to the momentum average model, 2D5momt. This is likely due to the shape functions at the wall being very important for the closure terms at the wall, and when they are averaged this information is lost. Both models do better in the middle of the channel. This can be due to the more complicated flow phenomena near the wall.

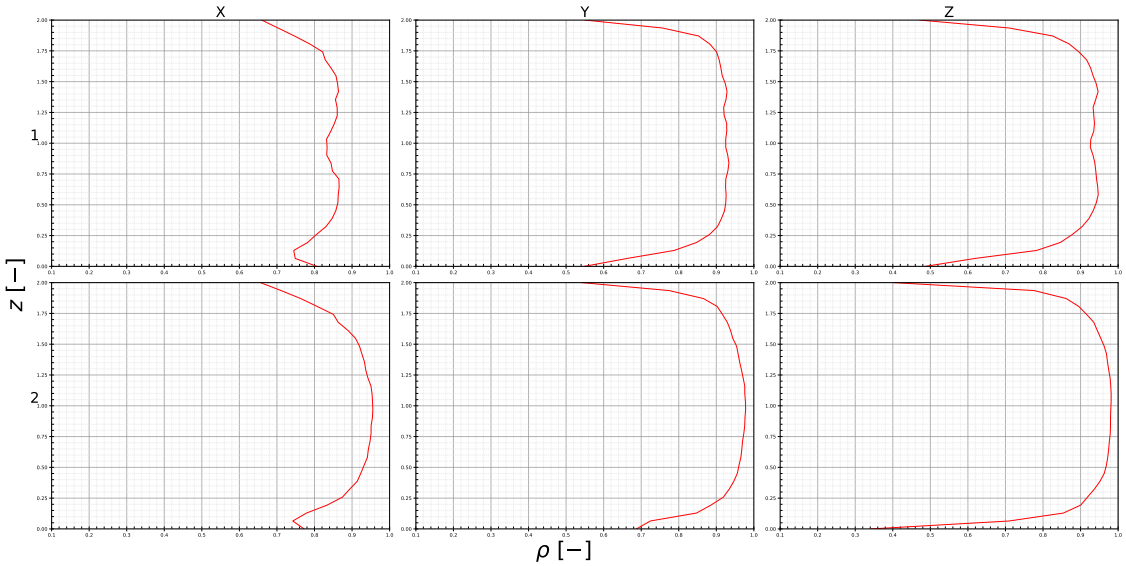


Figure 4.9: Correlations for a CNN with 5x5x2 input of averaged coarse scale momentum sums with and extra time step across the channel height.

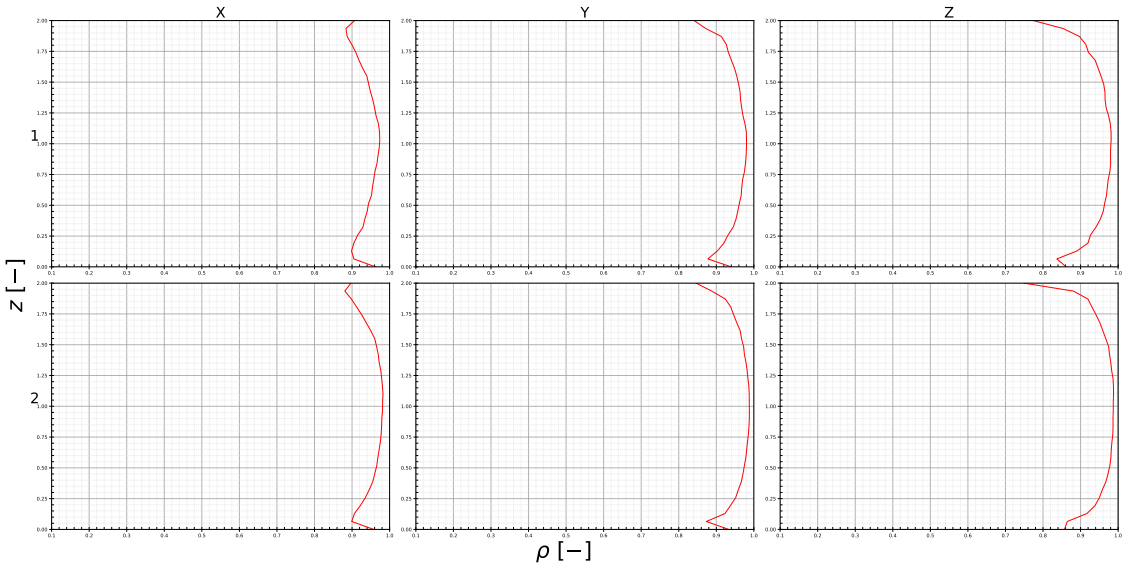


Figure 4.10: Correlations for a CNN with 5x5x2 input of independent coarse scale momentum sums with and extra time step across the channel height.

5 A Posteriori Results

The trained CNN model 2D5momti was used for a turbulent channel flow simulation with initial condition L0, one second prior to the training data. The results that are most interesting are the velocity fields in the first few LES time steps. The model was tested in a flow on which it has not trained but if the model is trained sufficiently it should produce accurate local flow dynamics. The CNN model used for a posteriori testing was the best model from section 4. This is the model with 2D inputs of independent coarse scale momentum sums with 2 time steps. This model produced correlations of over 0.9 for all shape functions and momentum terms and performed better near the wall than the model that used averaged momentum terms as input. The results for the model using averaged momentum terms as input, 2D5momt, are similar to the ones of 2D5momti and are presented in the appendix. A simulation using an algebraic model was also run as a baseline for comparison.
The CNN model was trained only to predict accurate closure terms per element for one time step at a time, and not to produce good long term flow statistics. Nonetheless some longer term flow statistics are also presented to show how the model behaves.

5.1 Local Dynamics

To determine the accuracy of the simulation when using the CNN to close the momentum equations, streamwise slices were analyzed and the difference between the projected DNS and the CNN velocity flow field was computed. Figure 5.1 shows the difference between the projected DNS velocity field form the CNN and algebraic simulation after the first LES time step at $x = 0$. The CNN error is on the top row and the algebraic error is on the bottom row. The CNN simulation has lower errors over the slice, although it does commit its errors in similar locations as the algebraic model. Figure 5.2 shows the difference of the absolute value of the velocity errors between the two models. Here the green color means the CNN model has lower errors than the algebraic model. There are only a few cells where the algebraic model is better than the CNN model. The algebraic model is based on results from linearized multiscale theory and has been observed to produce good results in less complex regions of the flow [7]. The CNN model, however, is not limited by theoretical assumptions. Its error are only due to architecture and training limitations.

Figure 5.3 shows the averaged absolute value of the velocity errors and 5.4 shows the difference between the CNN average error and the algebraic average error after the first LES time step. Figure 5.3 shows that especially near the walls the algebraic model produces a lot of errors. For figure 5.4 the green color again represents the CNN model performing better. These figures show that over the entire domain the CNN model is better than the algebraic model in every cell.

While after one LES time step the CNN model outperforms the algebraic model this is to be expected since the CNN is trained specifically to predict accurate closure terms for single time steps. The algebraic model was made to stabilize the simulation over long time scales but cannot accurately predict local flow dynamics. Over long time scales it is to be expected that the CNN, which is trained only on projected DNS data, will ingest its own errors and

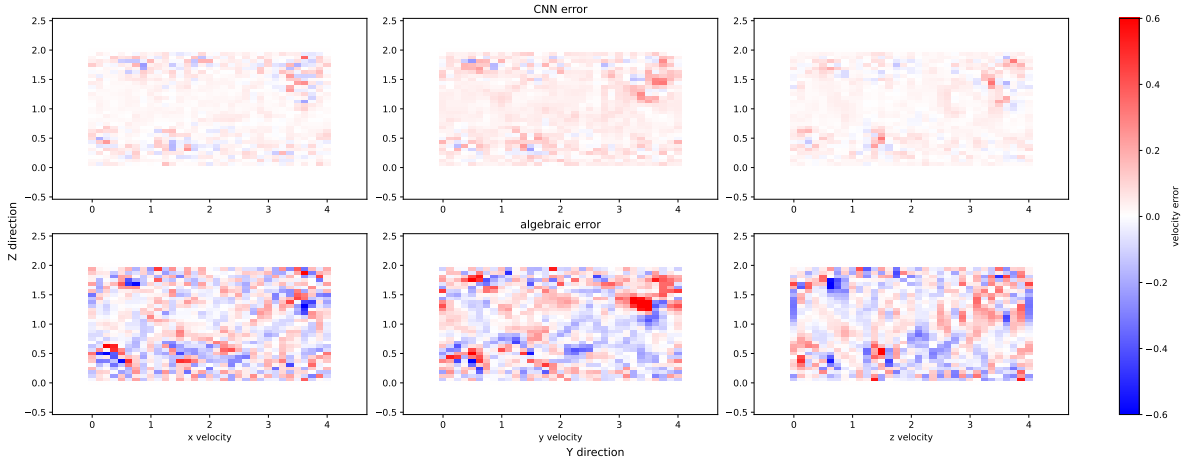


Figure 5.1: Velocity error after 1 LES time step for CNN simulation and algebraic simulation at x=0.

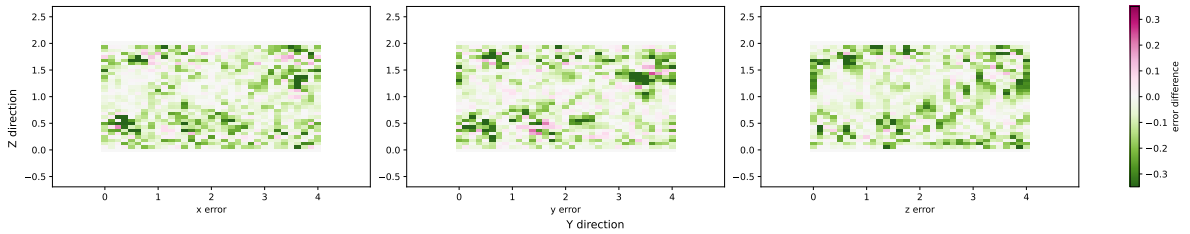


Figure 5.2: Difference in velocity error after 1 LES time step for CNN simulation and algebraic simulation at x=0.

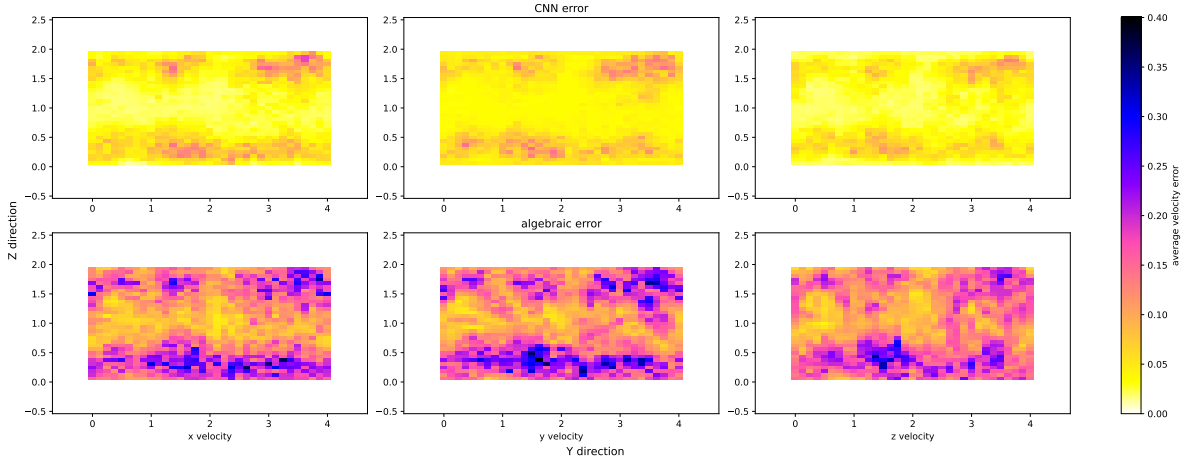


Figure 5.3: Average velocity error after 1 LES time step for CNN simulation and algebraic simulation over the whole domain.

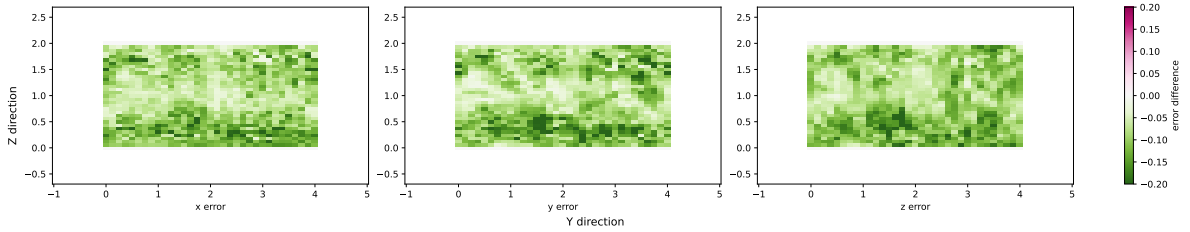


Figure 5.4: Difference in average velocity error after 1 LES time step for CNN simulation and algebraic simulation over the whole domain.

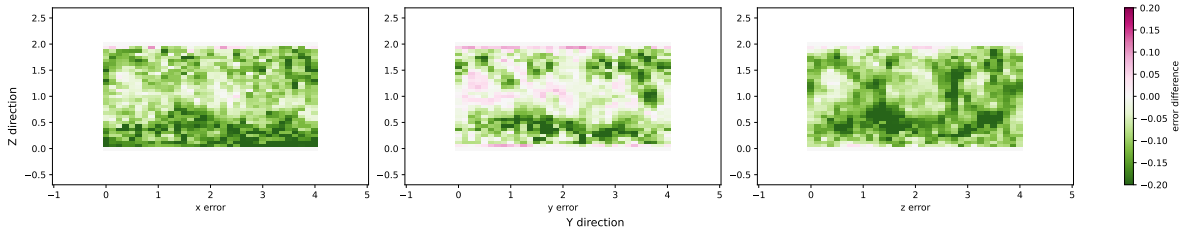


Figure 5.5: Difference in average velocity error after 3 LES time steps for CNN simulation and algebraic simulation over the whole domain.

start to produce worse closure terms.

Figure 5.5 shows the difference between the CNN and algebraic simulation error after three time steps and figure 5.6 shows it after five time steps. After five time steps it is clear that the CNN model can no longer produce closure terms at a highly accurate level, in the v direction especially the CNN creates significant error.

5.2 Long Term Stability

The a priori tests are useful to prove that the model is able to fairly accurately predict closure terms but it says nothing about long term behavior and stability of the model in a simulation. The a posteriori tests aim to show how the model affects the overall stability and statistics of the simulation. The kinetic energy, minimum and maximum of velocity terms, and the error of the velocity are all plotted. It should be noted that long term stability was

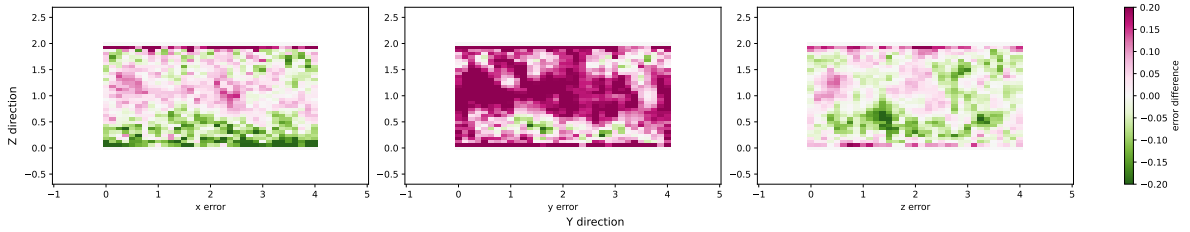


Figure 5.6: Difference in average velocity error after 5 LES time step for CNN simulation and algebraic simulation over the whole domain.

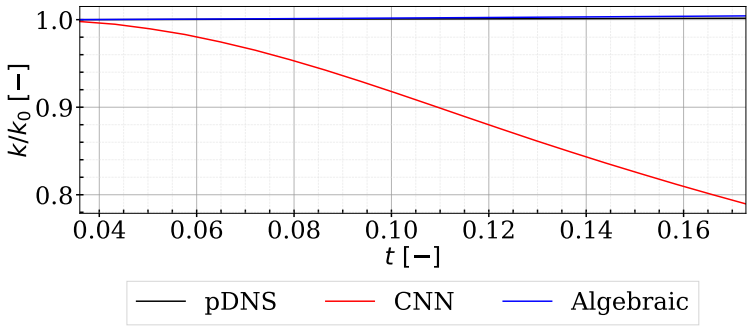


Figure 5.7: Kinetic energy of flow.

not the focus of this research and the models were not trained for this. These results are an additional analysis to determine if the models are lacking and how they can perhaps be improved.

Figure 5.7 shows the kinetic energy of the flow for the first 20 time steps. In the first LES simulation the momentum equations are closed using the CNN. The second simulation uses a full algebraic model. The CNN model takes energy out of the simulation at an alarming rate. After 20 time steps more than 20 percent of the energy has already been drained. The algebraic model adds kinetic energy consistently but at a very small rate. The algebraic model is designed to make sure the energy of the simulation is stable, while the CNN model is only trained to predict closure terms for next time step and is not informed about long term behavior of the flow or how to make predictions with data corrupted by its own errors.

The minimum and maximum velocity, shown in figure 5.8, show if the flow is diverging and which velocity components are contributing to error.

Figure 5.9 shows the root mean square error of the velocity over the entire domain. The CNN model does much better initially but after about five time steps the errors of the CNN model overtake the error of the algebraic model.

5.3 Network Cost

The computational cost for the network to make a prediction is important if the network is ever to work efficiently in a simulation. The time it takes to run the network was not a priority when training the models.

This depends heavily on how large and complex the model is and how large the input is. The model with a $9x9$ spatial stencil, 2 time step stencil, and all 3 directions as input takes between 0.5 and 0.7 milliseconds to make a prediction. The model with only a $5x5$ spatial stencil takes between 0.2 and 0.4 milliseconds to make a prediction. These models predict for all shape functions of an element at the same time so are called only once per element. Note that model 2D5momti predict closure terms for each shape function independently meaning for a single element it takes 8 times longer than model 2D5momt.

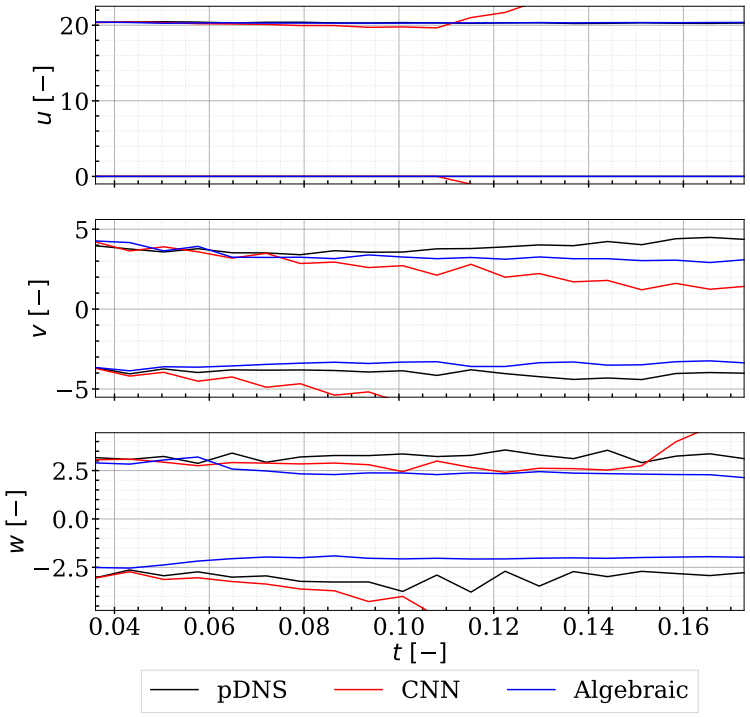


Figure 5.8: Min and max of velocity components.

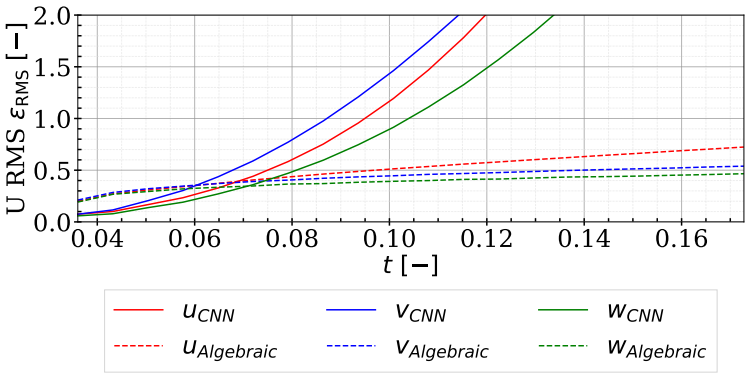


Figure 5.9: Root mean square error of velocity.

6 Conclusion

In this thesis a new turbulence model for the variational multi-scale method was created by using convolutional neural networks. The model was trained on data from a turbulent channel flow. Different inputs and network structures were tested to determine what is important in creating a new closure model that makes good local predictions. The networks are tested in both an a priori sense, correlations of model predictions against real closure terms, and an a posteriori sense, used in an online simulation. In an a priori sense the CNN was shown to produce high correlations.

6.1 How should the CNN be designed so that it produces closure terms with high correlations to the exact ones?

There are many choices when it comes to setting up the CNN and all of them affect the performance. The choice of input data is one of the most important factors and several choices were tested. The stencil size and hyperparameters of the network also change the performance and a few parameters are varied to determine the affect on the performance.

6.1.1 What data from the simulation should be used as inputs to the CNN?

The inputs to the neural network are very important and determine how well the model works. Several different types of inputs were tested. The first was the averaged velocity amplitudes of the shape functions of each element. The second was the integral forms of the resolved-scale momentum equation. The resolved-scale forms were tested as an average and without averaging. With the resolved-scale momentum forms the model was able to much better predict the closure terms. This is not a surprise as the resolved-scale momentum sums contain much more information about unresolved-scale momentum sums than the velocity amplitudes. The independent resolved-scale sums are also more informative than the averaged ones which can be attributed to a loss of information, especially at the walls.

6.1.2 How large does a stencil have to be?

The size of the input was also varied. Both 3D and 2D spatial stencils were tested, and in time both the current time step and an extra previous time step were tested as inputs. The 3D spatial stencil did not produce good results which was attributed to problems at the wall boundary. The stencil was padded with zeroes when it went beyond the wall boundary but this means large portions of the stencil contain no relevant information which hinders the model. A 2D spatial stencil, in the periodic streamwise and spanwise directions, produced higher correlations since there was no need to pad with zeros at the wall. Using two previous time steps, t_n and t_{n-1}, proved to be the most useful. Models that used both time steps performed significantly better than ones that only used one time step. This is most likely because the model is able to calculate a sort of time derivative of the closure terms which it can use to predict future closure terms.

6.1.3 What kind of hyperparameters are needed?

Convolutional neural networks have several hyperparameters which affect the accuracy of the network. Several of these were varied. The size of the input was varied between $5x5$, $7x7$, and $9x9$. The larger stencils produced better correlations but only slightly, and mainly in the x direction. For this reason the $5x5$ stencil was deemed enough. The amount of convolutional layers was also varied between 1 and 3. The accuracy of the model went up with the amount of convolutional layers, with the greatest affect again in the x direction momentum terms. The amount of filters per layer was also changed. The first layer of the network was tested with 5, 10, and 20 filters with the second layer containing twice as many filters as the first. The amount of filters had the most effect out of all hyperparameters with the x momentum terms again receiving the most benefits.
The effects of the hyperparameters were noticeable but not as significant as the effect of the inputs. For this reason a model with a $5x5$ stencil, 2 convolutional layers, and 10 filters in the first layer was chosen to be tested in a simulation. The smaller stencil significantly reduced the size of the training data, and the lower amount of filters lowered the training time.

6.2 How effective are CNNs when used for the momentum equations in terms of producing accurate local flow statistics in the simulation?

A CNN is able to close the momentum terms with good correlation in an a priori sense. Correlations of over 0.9 were achieved for all momentum terms.

6.2.1 Are CNN models more accurate than algebraic models for local dynamics?

The model was also tested in an online simulation and compared to an algebraic model. For the first few time steps the CNN predicts more accurate closure terms and the local flow dynamics are better than for the algebraic model.

6.3 Recommendations

There are several topics that should be investigated in the future.
To stabilize the simulation the model could be trained on data that is augmented with noise. This could produce a model that is able to eat its own error and still produce good closure terms. White noise or Gaussian noise could be used. Also gradient based techniques which consider large time intervals should be explored.
The effect of different projector is also not explored in this work. The nodal projector is quite fast but other projectors could produce help to improve the training data and stabilize the simulation.

References

[1] Andrea Beck, David Flad, and Claus-Dieter Munz. "Deep neural networks for data-driven LES closure models". In: *Journal of Computational Physics* 398 (2019). DOI: 10.1016/j.jcp.2019.108910.

[2] Thomas J. R. Hughes, Guglielmo Scovazzi, and Leopoldo P. Franca. "Multiscale and Stabilized Methods". In: *Encyclopedia of Computational Mechanics Second Edition* (2017). DOI: 10.1002/9781119176817.ecm2051.

[3] Y. Bazilevs, V.M. Calo, J.A. Cottrell, T.J.R. Hughes, A. Reali, and G. Scovazzi. "Variational multiscale residual-based turbulence modeling for large eddy simulation of incompressible flows". In: *Computer Methods in Applied Mechanics and Engineering* 197.1 (2007), pp. 173–201. ISSN: 0045-7825. DOI: https://doi.org/10.1016/j.cma.2007.07.016.

[4] Andrea Beck and Marius Kurz. "A perspective on machine learning methods in turbulence modeling". In: *GAMM-Mitteilungen* 44 (2021). DOI: 10.1002/gamm.202100002.

[5] Karthik Duraisamy. "Perspectives on machine learning-augmented Reynolds-averaged and large eddy simulation models of turbulence". In: *Physical Review Fluids* 6 (2021). DOI: 10.1103/PhysRevFluids.6.050504.

[6] Andrea Bettini. "Energy-Conservative Data-Driven Modelling for the two-scale Navier-Stokes Equations". MA thesis. TU Delft, 2023.

[7] Martin Janssens. "Machine Learning of Atmospheric Turbulence in a Variational Multiscale Model". MA thesis. TU Delft, 2019.

[8] Sharath Rajampeta. "Stabilization Strategies for a Variational Multiscale Method coupled with an Artifficial Neural Network". MA thesis. TU Delft, 2021.

[9] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going deeper with convolutions". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015). DOI: 10.1109/CVPR.2015.7298594.

[10] Giuliano De Stefano and Oleg V. Vasilyev. "'Perfect' modeling framework for dynamic SGS model testing in large eddy simulation". In: *Theoretical and Computational Fluid Dynamics* 18 (2004). DOI: 10.1007/s00162-004-0146-0.

[11] Junsu Shin, Yipeng Ge, Arne Lampmann, and Michael Pfitzner. "A data-driven subgrid scale model in Large Eddy Simulation of turbulent premixed combustion". In: *Combustion and Flame* 231 (2021). DOI: 10.1016/j.combustflame.2021.111486.

[12] Michel Robijns. "A Machine Learning Approach to Unresolved Scale Modeling for Burgers' Equation". MA thesis. TU Delft, 2019.

[13] Abhinand Pusuluri. "Noise-augmented offline training of ANN unresolved scale models". MA thesis. TU Delft, 2020.

[14] T. J. R. HUGHES and G. SANGALLI. "Variational Multiscale Analysis: the Fine-scale Green's Function, Projection, Optimization, Localization, and Stabilized Methods". In: *SIAM Journal on Numerical Analysis* 45 (2007). DOI: 10.1137/050645646.

[15] Oleksandr Krochak. "Data-Driven Closure Modelling of the Navier-Stokes Momentum Equations". MA thesis. TU Delft, 2024.

[16] I. Akkerman, Y. Bazilevs, V. M. Calo, T. J. R. Hughes, and S. Hulshoff. "The role of continuity in residual-based variational multiscale modeling of turbulence". In: *Computational Mechanics* 41 (2007), 371â378. DOI: 10.1007/s00466-007-0193-7.

[17] Jianxin Wu. "Introduction to convolutional neural networks". In: *National Key Lab for Novel Software Technology. Nanjing University. China* (2017).

[18] Sergio Izquierdo. *cppflow: Run TensorFlow models in C++ without installation and without Bazel*. Version 2.0.0. May 2019. DOI: 10.5281/zenodo.7107618. URL: https://github.com/serizba/cppflow.

Appendices

A A Posteriori Results for Averaged CNN Model

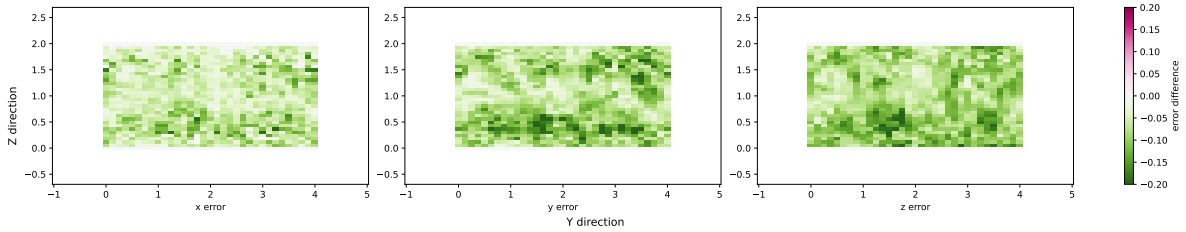


Figure A.1: Difference in average velocity error after 1 LES time step over the whole domain for a simulation using a CNN with averaged inputs and one using an algebraic model. Green color signifies the CNN performs better and purple signifies the algebraic model performs better.

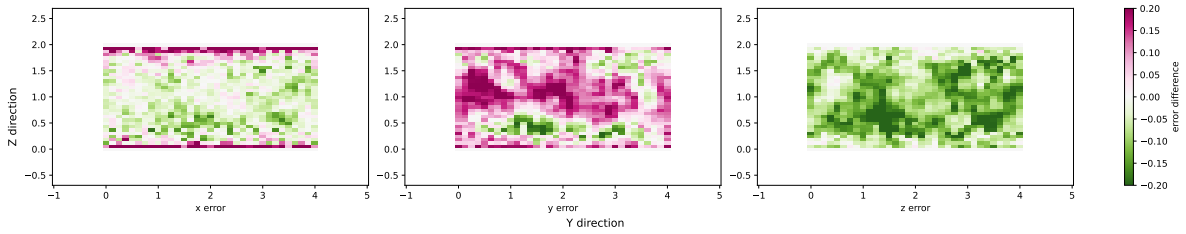


Figure A.2: Difference in average velocity error after 5 LES time steps over the whole domain for a simulation using a CNN with averaged inputs and one using an algebraic model. Green color signifies the CNN performs better and purple signifies the algebraic model performs better.

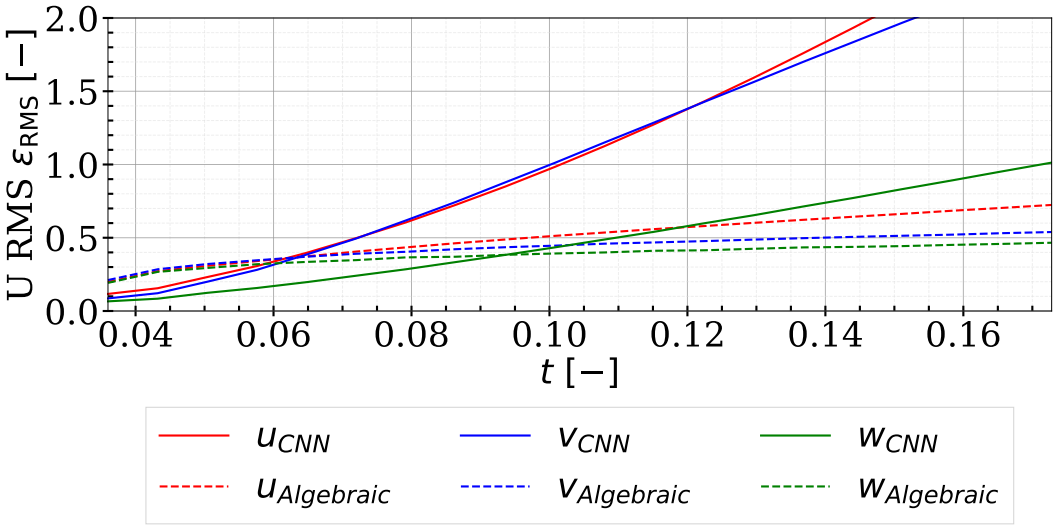


Figure A.3: Root mean square error of velocity comparing a CNN model using averaged inputs with an algebraic model.